Суханов Владимир Иванович

Введение в Zope3

Уральский государственнй технический университет
Факультет ускоренного обучения
Кафедра программных средств и систем
http://fat.ustu.ru

Екатеринбург 2006

© Суханов 2006

Содержание

ВВЕДЕНИЕ **6**

1 РАЗРАБОТКА ПРЕЗЕНТАЦИЙ **8**

1.1 Интерфейсы менеджера сайта 8
1.2 Основные файлы Zope3 9
1.3 Создание презентаций 10
1.4 Заимствование и Zope3 12

2 ОСНОВЫ DTML **14**

2.1 Синтаксис тегов DTML 14
2.2 Стандартные объекты и атрибуты 16
2.3 Справочник по тегам DTML 17

3 СТРАНИЧНЫЕ ШАБЛОНЫ **29**

3.1 Операторы языка TAL 29
3.2 Выражения TAL 36
3.3 Макросы 39
3.4 Использование стандартных макросов 41

4 РАБОТА С БАЗАМИ ДАННЫХ **45**

4.1 Использование DTML страниц 46
4.2 Использование ZPT страниц 48

5 ОСНОВЫ РАЗРАБОТКИ ПРИЛОЖЕНИЙ В ZOPE3 ..... **50**

5.1 Интерфейсы 51
5.2 Объявление классов 54
5.3 Шаблоны страниц для отображения объектов 55
5.4 Конфигурирование приложения 57
5.5 Интернационализация приложения 61

6 АРХИТЕКТУРА ПРИЛОЖЕНИЙ ZOPE3 ..... **64**

6.1 Сервисы 64
6.2 Адаптеры 64
6.3 Пример адаптера для штата и города 75
6.4 Компоненты для просмотра 79
6.5 Тестирование приложений 81

7 СХЕМЫ И ФОРМЫ **83**

7.1 Схемы и интерфейсы 83
7.2 Основные поля схем 84
7.3 Словари в задании полей 86
7.4 Формы и виджеты 87

8 ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ КОНФИГУРАЦИЙ **92**

8.1 Система конфигурирования 92
8.2 Общие директивы пространства имен 100
8.2.1 configure 102
8.2.2 include 102
8.2.3 includeOverrides 103
8.3 Директивы группы zope 103
8.3.1 adapter 103
8.3.2 allow 104
8.3.3 authenticatedGroup 104
8.3.4 class 104
8.3.5 content 104
8.3.6 defaultLayer 105
8.3.7 defaultView 105

8.3.8 everybodyGroup ... *105*
8.3.9 factory ... *105*
8.3.10 grant ... *106*
8.3.11 grantAll ... *106*
8.3.12 interface ... *106*
8.3.13 localService ... *106*
8.3.14 localUtility ... *107*
8.3.15 module ... *107*
8.3.16 modulealias ... *107*
8.3.17 permission ... *107*
8.3.18 preferenceGroup ... *108*
8.3.19 principal ... *108*
8.3.20 require ... *108*
8.3.21 resource ... *108*
8.3.22 role ... *108*
8.3.23 securityPolicy ... *109*
8.3.24 subscriber ... *109*
8.3.25 unauthenticatedGroup ... *109*
8.3.26 unauthenticatedPrincipal ... *109*
8.3.27 utility ... *109*
8.3.28 view ... *109*
8.3.29 vocabulary ... *110*
8.4 Директивы группы BROWSER ... 110
8.4.1 addMenuItem ... *110*
8.4.2 addform ... *110*
8.4.3 addview ... *111*
8.4.4 addwizard ... *111*
8.4.5 containerViews ... *112*
8.4.6 defaultSkin ... *112*
8.4.7 defaultView ... *112*
8.4.8 editform ... *112*
8.4.9 editwizard ... *113*
8.4.10 form ... *113*
8.4.11 i18n-resource ... *113*
8.4.12 icon ... *114*
8.4.13 layer ... *114*
8.4.14 menu ... *114*
8.4.15 menuItem ... *114*
8.4.16 menuItems ... *114*
8.4.17 page ... *115*
8.4.18 pages ... *115*
8.4.19 resource ... *116*
8.4.20 resourceDirectory ... *116*
8.4.21 schemadisplay ... *116*
8.4.22 skin ... *116*
8.4.23 subMenuItem ... *117*
8.4.24 subeditform ... *117*
8.4.25 tool ... *117*
8.4.26 view ... *117*
8.5 Директивы группы DAV ... 118
8.5.1 provideInterface ... *118*
8.6 Директивы групп HELP ... 118
8.6.1 register ... *118*
8.7 Директивы группы I18N ... 118
8.7.1 registerTranslations ... *118*
8.8 Директивы группы META ... 118
8.8.1 complexDirective ... *118*
8.8.2 directive ... *119*
8.8.3 directives ... *119*
8.8.4 groupingDirective ... *119*
8.8.5 provides ... *119*
8.8.6 redefinePermission ... *119*

8.8.7 subdirective ... *119*
8.9 Директива renderer ... 120
8.10 Директива tales ... 120
8.11 Директивы xmlrpc ... 120

9. МЕТАДАННЫЕ ... 121

10 ЗАЩИТА КОМПОНЕНТ ... 124

10.1 Объявление ролей и разрешений ... 124
10.2 Декларация принципалов ... 126

11 ИНТЕРНАЦИОНАЛИЗАЦИЯ ПАКЕТОВ ... 128

11.1 Интернационализация кода Питона ... 128
11.2 Интернационализация шаблонов страниц ... 129
11.3 Интернационализация ZCML ... 129
11.4 Интернационализация объектов ... 130

12 КОНТЕЙНЕРЫ И ИХ СОДЕРЖИМОЕ ... 131

12.1 Связывание компонент по включению ... 131
12.2 События и подписчики ... 133
12.3 Обеспечение встроенной помощи ... 149
12.4 Разработка нового обличья ... 150
Шаг I: Подготовка ... 150
Шаг II: Создание новой формы ... 151
Шаг III: Модификация базового шаблона ... 151

13 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЙ ... 154

13.1 Объект приложения ... 154
13.2 Управление пользователями ... 159
13.3 Управление просмотрами ... 161
13.4 Отладчик Zope 3 ... 163

ГЛОССАРИЙ ... 166

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ... 181

ПРИЛОЖЕНИЕ А ... 182

А1 Основы языка Питон ... 182

А1.1 ЛЕКСИЧЕСКИЕ СОГЛАШЕНИЯ ... 183

А1.2 Выражения и порядок их вычисления ... 185
А1.3 Последовательности ... 186
А1.4 Строковые данные ... 189
А1.5 Множества ... 195
А1.6 Дополнительные типы ... 195

А2 ОПЕРАТОРЫ ... 196

А2.1 Присваивание ... 196
А2.2 Управление выполнением программы ... 197
А2.3 Пространство имен ... 199
А2.3.1. Определение функций ... 200
А2.3.2. Определение классов ... 201
А2.3.3. Встроенные функции ... 205

Сокращения

ООП – Объектно-ориентированное программирование

DTML – Document Template Markup Language, язык разметки шаблонов документов

IDL – Interface Definition Language, язык определения интерфейсов

IDLE – Integrated Development Environment for Python, интегрированная оболочка для
HTML – HyperText Markup Language, язык гипертекстовой разметки
METAL – Macro Expansion Template Attribute Language, макро язык для ZPT

TAL – Template Attribute Language, язык атрибутов шаблонов
TALES – Template Attribute Language Expression Syntax, язык выражений TAL

TTW – Through The Web, через сеть

URI – Unified Resource Identifier, Унифицированный идентификатор ресурса

W3C – WWW Consorcium

XML – Extensible Markup Language, расширяемый язык разметки

ZCML – Zope Configuration Markup Language, язык разметки конфигураций Zope

работы в Питоне

ZMI – Zope Manager Interface, интерфейс управления Zope

ZODB – Zope Object DataBase

ZPT – Zope Page Template, страничные шаблоны

Введение

Развитие сетевых информационных технологий создает предпосылки для разработки приложений для конечного пользователя с использованием вэб-интерфейсов. В отличие от клиентских приложений, выполненных в виде exe-модулей, веб-клиенты имеют существенные преимущества. Наиболее важным отличием признаком является отсутствие на стороне клиента специального программного обеспечения для взаимодействия с сервером, достаточно наличие обычного браузера, например, Internet Explorer, Netscape Navigator, Mozilla и других. Это существенно повышает мобильность как пользователя (можно иметь доступ к приложению из любой точки мира), так и программно-аппаратных платформ. Доступ можно обеспечить как со стационарного, так и с мобильного компьютера, снабженного любой операционной системой и выполненного на любой элементной базе. Большие преимущества такой организации корпоративных информационных систем и приложений имеет администрирование систем. Централизованное хранение на сервере функционального ядра системы позволяют обеспечить масштабирование, требуемый уровень защиты данных, оперативную корректировку программ, управление версиями, регистрацию и управление полномочиями пользователей, интернационализацию (автоматический перевод сообщений на язык конкретного пользователя) и ряд других полезных свойств системы. Поэтому на передний план среди инструментария программистов выходят средства разработки интегрированных информационных систем на основе вэб-интерфейсов.

Для этих целей на рынке программных средств существуют различные технологии. Примерами могут быть Internet Information Services (IIS) для MS Windows, многоплатформенные технологии Java, Apache + PHP, Zope + Python и др. Использование первых двух продуктов требует приобретения лицензий, третья поставляется в зависимом от платформы исполнении, последняя свободно распространяемая многоплатформенная среда, интегрирующая как средства исполнения, так и средства разработки приложений.

Технологии Zope [12] прошли длительный путь эволюционирования со сменой средств и концепций построения интегрированных систем. Последняя версия Zope3 позволяет разрабатывать вэб-приложения с использованием компонентной архитектуры с небольшим объемом базового программирования на языке Python [1, 2], HTML и средств конфигурирования программных фрагментов в единую систему на языке ZCML. Решительным преимуществом технологии Zope3 [3. 12] является ее независимость от платформы. Построенная на основе интерпретируемого языка Python сама система и разработанные в ней приложения не нуждаются в каких либо доработках при переносе в другую операционную систему и на другую архитектуру серверной ЭВМ. Следует отметить, что вся информация для формирования страниц, направляемых клиенту, сосредоточена во внутренних объектно-ориентированных базах данных с тщательно контролируемыми полномочиями доступа, не допускающих ее несанкционированное использование. Отличительной особенностью системы Zope является ее поставка с открытым кодом, позволяющая при необходимости дорабатывать или модифицировать ее функциональность. Учитывая многоплатформенность системы и широкое распространение ОС MS Windows, далее приводится описание использования Zope3 на сервере для этой операционной системы.

Для создания среды разработки на стороне сервера необходимо последовательно установить следующие продукты на системный диск ОС MS

Windows 2000/XP: Все продукты можно найти на сайтах www.python.org и www.zope.org.

1. Python-2.4.msi (запуск из проводника щелчком по имени, параметры по умолчанию)
2. Zope-3.1.0.win32-py2.4.exe (параметры по умолчанию)
3. Из папки C:\Python24\Scripts запустить на исполнение командный файл mkzopeinstance.bat. На запрос программы задать папку для экземпляра Zope-сервера (например, C:\Zope3Inst), имя администратора и пароль. При правильной работе будет создана указанная папка с необходимыми папками и файлами.
4. Для запуска на исполнение сервера Zope3 можно использовать командный файл C:\Zope3Inst\bin\runzope.bat. В период отладки приложений запуск сервера лучше производить из окна, созданного командой cmd из меню «Пуск-Выполнить». Остановить сервер можно комбинацией клавиш ctrl-Break с последующим подтверждением символом 'Y' и Enter или из меню ZMI браузера Сервер – Управление – Остановить сервер. Для запуска сервера в режиме нормальной эксплуатации целесообразно создать ярлык на рабочем столе с той же командной строкой или обеспечить автозапуск.
5. В папке C:\Zope3Inst\lib\python создать папку с именем пакета приложения (например, ais) – резиденцию для всех файлов и папок нового проекта.
6. Разработка программ производится в оболочке Python24, запускаемой из меню Пуск – Python24 – IDLE. Для удобства работы лучше создать ярлык на рабочем столе для запуска Питона с рабочей папкой C:\Zope3Inst\lib\python\ais
7. После запуска сервера управление производится через ZMI – интерфейс управления Zope, доступный в любом интернет браузере по адресу http://localhost:8080/manage. Для отображения русского меню в настройках браузера необходимо первым установить русский язык, а вторым английский. Иначе меню будет английским.

Для понимания материала данного пособия читателю необходимо знать основы языка Питон [1, 2] и HTML [5], общие принципы организации вэб серверов и сервисов. Остальной материал будет изложен по мере необходимости. Основные справочные материалы по языку Питон версии 2.4 приведены в приложении. Приведем некоторые ссылки на полезные материалы, которые помогут Вам освоить основы разработки приложений в Zope3:

- Сайт Zope3 http://dev.zope.org/Zope3
- Сайт Python http://www.python.org
- Стиль кодирования http://dev.zope.org/Zope3/CodingStyle
- Конференция русскоязычных пользователей Zope и Python http://www.itconnection.ru/zopyrus

1 Разработка презентаций

1.1 Интерфейсы менеджера сайта

ZMI – основано на классическом формате из двух колонок с одним фреймом сверху (рисунок 1). Верхний фрейм использован для логотипа и основной

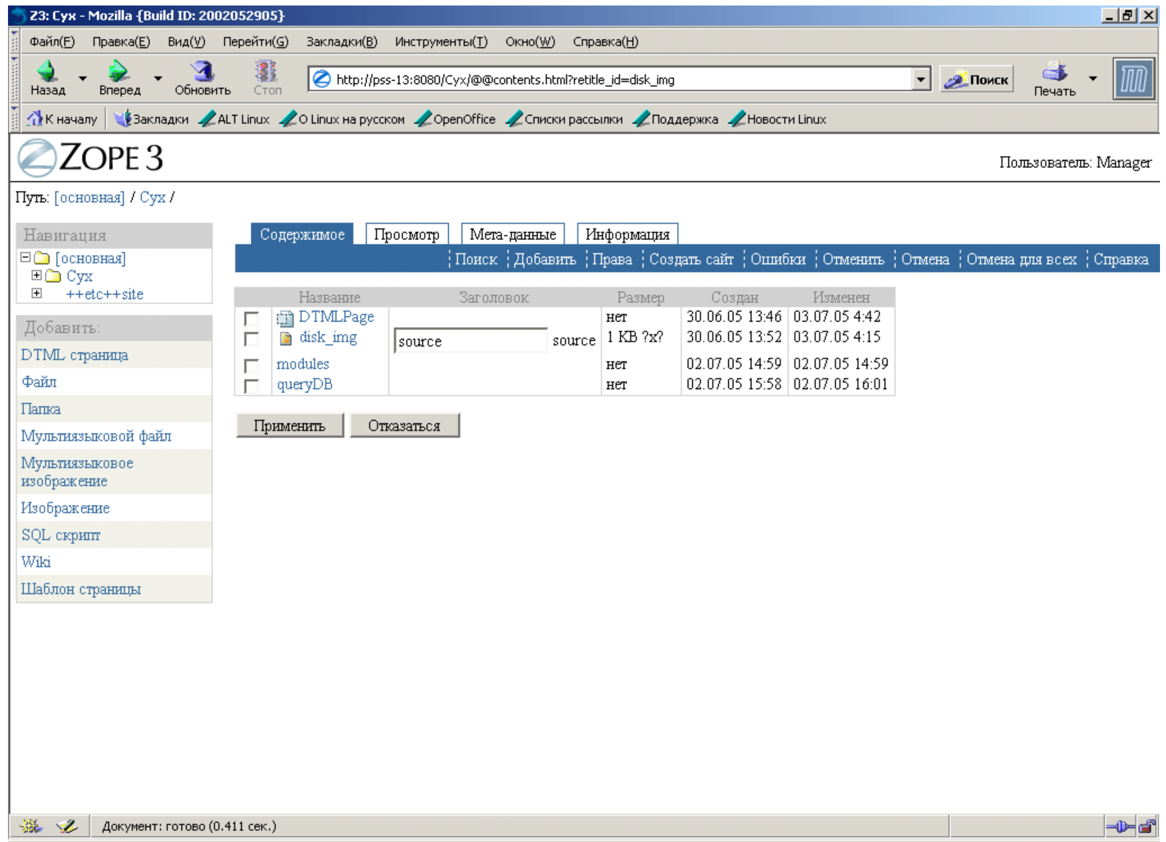

Рисунок 1 – Интерфейс менеджера сайта

информации о пользователе. Левый столбец – навигатор. Первый блок навигатора является деревом навигации, которое имеет вид иерархической структуры объектной базы данных. В нем представлены две категории объектов – простые объекты и контейнеры. Ниже дерева может быть помещено большое разнообразие других блоков, включая меню «Добавить».

Основная часть экрана – рабочая область. Сверху рабочей области размещается полный путь объекта, который является ссылками на каждый элемент пути. Ниже указателя адреса, помещается область закладок. Закладки известны как "ZMI views". Примером является вид "Contents"/"Содержание", демонстрирующий внутреннее содержание объекта. Ниже закладок расположен список действий, известный как "ZMI actions". Список действий зависит от объекта, но обычно он доступен для многих объектов. Общие действия включают "Undo", "Find", "Grant" and "Help" ("Отмена", "Найти", "Права" и "Помощь"), которые доступны для всех объектов.

Ниже меню действий – рабочая область ("viewspace"), где может содержаться несколько элементов, в зависимости от контекста. Все виды имеют закладку "Content"/"Содержимое", которая содержит форму с информацией выбранного

объекта. С правой стороны области может присутствовать дополнительный столбец – "Контекстная информация".

Добавление новых объектов производится выбором из меню действий или из левого бокового меню «Add/Добавить» ссылки на нужный класс объекта. В зависимости от выбранного класса необходимо заполнить форму с информацией об объекте и подтвердить согласие на добавление экземпляра с заполненными данными. Если есть намерение отказаться, то можно просто средствами браузера вернуться назад к предыдущему кадру или перейти по ссылке в нужную ветвь навигатора. Редактирование имеющихся объектов производится в формах, получаемых после щелчка по ссылке на имя экземпляра в таблице рабочей области. Объекты можно переименовывать, копировать, вырезать и удалять. К сожалению, в отличие от версии Zope 2, в Zope3 не предусмотрен экспорт и импорт объектов в формате, допускающем их перенос в другую среду разработки. Это обусловлено хранением части информации об объектах в файлах определения компонент.

Закладка «Meta-data/Метаданные» позволяет пополнить информацию об объекте, видимую только менеджеру сайта и не влияющую на функциональность экземпляра объекта. Сюда входит заголовок и описание объекта. Заголовок показывается в таблице рабочей области в строке объекта.

Для администрирования сайта в распоряжении менеджера есть ряд служебных операций, выполнение которых сопровождается сменой цветового оформления окон интерфейса. К таким операциям относятся «Manage Site» и «Manage Process». Операции «Manage Site» позволяют добавить папку управления сайтом и настроить параметры сайта. Обычно при установке системы параметры сайта задаются по умолчанию, что подходит для подавляющего числа применений, и не нуждаются в модификации. По мере необходимости менеджер сайта может добавлять утилиты и адаптеры для доступа, например, к базам данных. Операции «Manage Process» позволяют просматривать текущую информацию о сервере, перезапускать или останавливать сервер.

1.2 Основные файлы Zope3

После установки и создания экземпляра сайта на сервере создаются каталоги и файлы, необходимые для работы Zope3. В отличии от Zope 2 система Zope3 устанавливается как и все прикладные пакеты Питона в папку «C:\Python24\Lib\site-packages», часть инсталляционных файлов располагается в папке «C:\Python24\Scripts». К работе Zope3 имеют отношения папки BTrees, docutils, persistent, pytz, RestrictedPython, ThreadedAsync, transaction, ZConfig, zdaemon, ZEO, ZODB, zodbcode, zope. В директории экземпляра сайта имеются папки bin, etc, lib. log и var.

Папки в директории site-packages содержат программное обеспечение для работы различных подсистем Zope, Наиболее интересной с точки зрения программирования является папка «zope», содержащая код питона и другие ресурсы инструментальных средств системы. При изложении материала мы часто будем делать отсылки к некоторым ресурсам этой папки. При необходимости получения исчерпывающей информации о программной реализации тех или иных функций системы целесообразно просмотреть соответствующие ресурсы Zope. Просмотр кода позволяет получить отличные примеры реализации различных функций системы, которые можно использовать при разработке новых компонент.

Папка «bin» сайта содержит код и командные файлы для запуска системы на исполнение, в частности, «runzope.bat». Папка «etc» содержит конфигурационные

файла сайта. Файл «zope.conf» содержит наиболее интересные настройки системы. Секция server определяет порты протокола TCP/IP 8080 для HTTP сервера и 8021 для FTP сервера. Секция zodb определяет расположение файла объектной базы данных. Секции accesslog и eventlog определяют местоположение журналов сайта для регистрации событий.

Файлы principals.zcml и securitypolicy.zcml конфигурируют параметры допущенных пользователей сайта (принципалов) и их роли, в частности, менеджера сайта. Администратор сайта здесь может доопределить новых принципалов или сделать ссылку на локальный файл деклараций, используя директиву include. Файл site.zcml – главный конфигурационный файл сайта, содержащий ссылки на отдельные разделы, определяющие детали настроек системы.

Папка etc\package-includes содержит небольшие файлы на языке ZCML, определяющих включение различных пакетов в текущую конфигурацию сайта. Эта папка пополняется администраторами или разработчиками новых компонент при необходимости расширить функциональность сайта. Папка lib\python предназначена для размещения новых пакетов, определяющих особенности данного сайта. Для подключения пакета необходимо добавить ссылку на него в папку etc\package-includes. Папка log содержит файлы журналов, просмотр которых позволяет администратору или программисту проанализировать последовательность событий и диагностику ошибок, возникающих при отладке новых пакетов. Папка var содержит файлы ZODB.

1.3 Создание презентаций

Разработка простых презентаций ничем не отличается от общепринятых веб технологий. Менеджер должен создать папки и наполнить их соответствующими материалами: документами, шаблонами, файлами, рисунками, скриптами и связать их ссылками. Папки являются контейнерами для других объектов. Собственно HTML документы можно оформить как DTML или как ZPT страницы. Отличия между ними достаточно условные. Там и там основой является текст страницы на языке HTML. Но в DTML Page используются вставки программ генерации страниц на языке DTML, а в ZPT Page на языке TAL. Но следует заметить, что в DTML и ZPT страницах нельзя смешивать вставки на разных языках, они просто игнорируются при рендеризации. Как принято в веб-серверах, имя главной страницы сайта по умолчанию является «index.html».

Для примера создадим ZPT страницу с именем «index.html» следующего содержания:

```
<html>
 <body>
  <h1> Добро пожаловать в раздел сайта с именем
      <span tal:content="context/__name__"> Name </span>
  </h1>
 </body>
 </html>
```

Если теперь щелкнуть на закладке. "Preview/Просмотр", можно увидеть в тексте приветствия название папки, где расположен документ. Клиент может открыть страницу, используя в браузере адрес:

http://localhost:8080/

По аналогии, можно наполнить сайт любыми другими материалами, выполнить должным образом дизайн и выставить его для просмотра в интернет. Наиболее привлекательной характеристикой среды разработки и веб сервера является формирование динамического содержания публикуемых страниц. Этим вопросам будет посвящено дальнейшее изложение возможностей среды Zope3.

В Zope3 (рисунок 2) вся информация разделена на "пространство

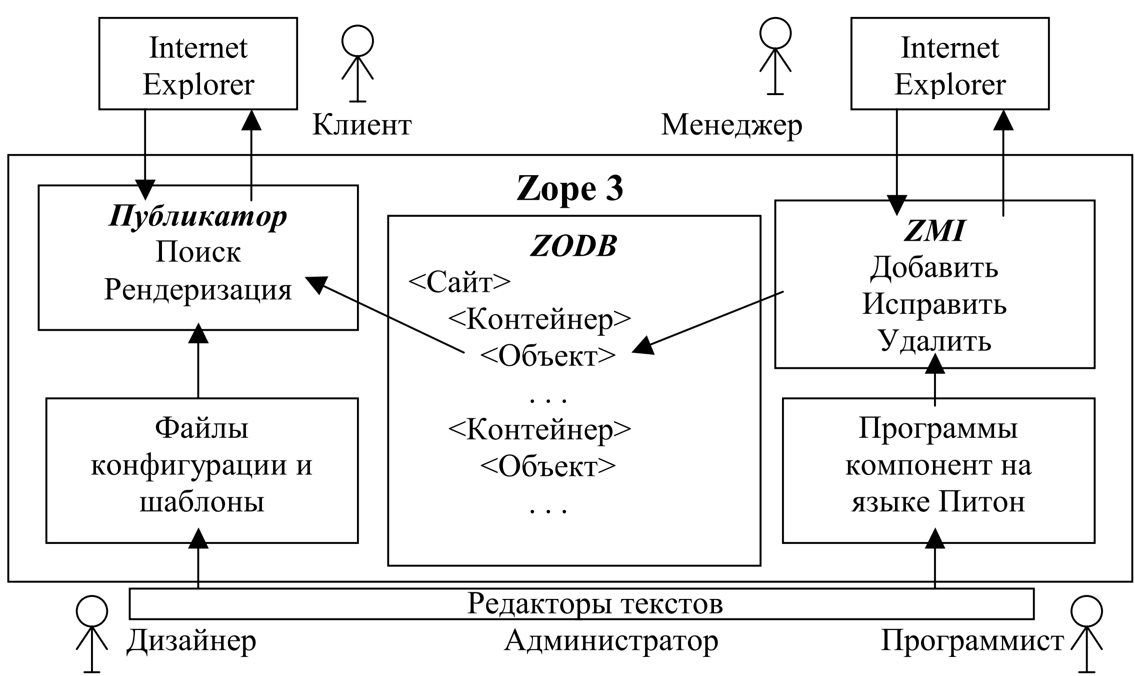

Рисунок 2 – Структура Zope 3

содержимого" и "пространство программ". Пространство содержимого хранит объекты сайта, которые используются для динамического формирования страниц. Вообще говоря, все страницы сайта формируются динамически из материалов, хранимых в объектно-ориентированной базе данных ZODB. Централизованное хранение, в отличие от хранения в файлах, позволяет тщательно контролировать доступ к ресурсам сайта со стороны «принципалов» – пользователей этих ресурсов. Контроль выполняется как при подготовке материалов, так и при их публикации. Структура части системы Zope3, отвечающей за ведение контента и его публикацию представлена на рисунке 2.

Каждый объект сайта имеет представителя в программном пространстве, которое может быть доступно через действие "Manage Site/Управлять сайтом". Вы можете видеть ссылку на эту операцию, когда просматриваете содержание корневой папки, так как корневая папка всегда является сайтом. Можно любую папку сделать сайтом, выполнив действие "Make a site/Сделать сайт".

Если щелкнуть на "Manage Site/Управлять сайтом", будет предоставлен список существующих локальных сервисов, включая их статус. Каждый сайт состоит из нескольких пакетов, включая встроенные пакеты, которые определены всегда. Пакеты используются для установки и хранения программного обеспечения.

Если щелкнуть на действии "Visit default folder/Переход в папку", Вы заканчиваете просмотр содержания пакета и переходите к добавлению локальных компонент, которые действует как программное обеспечение, например, переводчики на другие языки. Для удаления зарегистрированных утилит и адаптеров

нужно их деактивировать, зайти на вкладку «Registrations», удалить регистрацию соответствующих объектов, а далее удалить сами объекты.

1.4 Заимствование и Zope3

Технология заимствования (Acquisition) в Zope 2 позволяла реализовать динамический поиск объектов с требуемой функциональностью в контейнерах. Заимствование основной инструмент Zope 2, используемый в DTML, в Страничных Шаблонах, в сценариях на Питоне и даже в URL. Заимствование отличается от наследования в классическом ООП. Наследование – приобретение классом потомком атрибутов родителя [1]. В языке Питон допускается множественное наследование – класс может иметь более чем один суперкласс, и есть правила, чтобы определить, какой класс из суперклассов используется, чтобы определить поведение в любом контексте. Например:

```
class SuperA:
class SuperA:
  def amethod(self):
    print "I am the 'amethod' method of the SuperA class"
  def anothermethod(self):
    print "I am the 'anothermethod' method of the SuperA class"
class SuperB:
  def amethod(self):
    print "I am the 'amethod' method of the SuperB class"
  def anothermethod(self):
    print "I am the 'anothermethod' method of the SuperB class"
  def athirdmethod(self):
    print "I am the 'anothermethod' method of the SuperB class"
class Sub(SuperA, SuperB):
  def amethod(self):
    print "I am the 'amethod' method of the Sub class"
```

Если мы создадим экземпляр класса "Sub" и пытаемся вызывать один из его методов, то правила для определения принадлежности метода классу Sub, суперклассу SuperA или суперклассу SuperB сводятся к следующему. Если Sub класс имеет свое определение метода, то будет использовано локальный метод. В противном случае будет использована иерархия наследования: сначала он наследует атрибуты от SuperA, а потом от SuperB.

В Zope 2 объекты используют и другое средство поиска атрибутов – заимствование. Концепции заимствования таковы: объекты, расположенные внутри других контейнеров, могут заимствовать у них необходимые атрибуты. Заимствование предполагает, что объект может управлять своим поведением через иерархию контейнеров. В Zope 2 объектная иерархия наследования дополняется поиском атрибутов через иерархию заимствования. Если метод или атрибут не обнаруживается в объектной иерархии наследования, то поиск ведется в иерархии заимствования. Как следствие, объект может изменять свое поведение в зависимости от контекста, где он помещен. Это позволяло использовать дополнительные средства придания динамичности публикуемым страницам, касающиеся общего оформления, содержания окружения и другого.

В Zope3 концепция наследования ограничена в связи с использованием компонентной архитектуры и последовательным соблюдением принципов ООП. Заимствование из контекста по умолчанию делает неоднозначным поведение компонент, что противоречит полному программному контролю со стороны

разработчика. Заимствование в Zope3 управляемое только указаниями программиста, например, в ZPT шаблонах можно использовать ссылки как на контейнер, так и на контекст для определения пути получения объектов. Но способ выбирает сам программист, а не система. Для поиска объекта в родительском контейнере рекомендуется в тегах языка TAL использовать явное указание перехода вверх по уровням вложенности, например:

 Source

или

 Source

где /../ явно указывает системе, что искать объект index_html нужно в родительском контейнере для текущей папки, где помещен или вызван скрипт. Вместо символа «..» можно использовать эквивалентный атрибут «__parent__», например:

 Source

В Zope3 поиском запрошенных в URL объектов управляет механизм «traversal», который может быть настроен по желанию разработчика новых компонент на нужное ему поведение, в том числе и управление заимствованием.

2 Основы DTML

DTML (Document Template Markup Language) – средство создания страничных шаблонов, которые поддерживают динамический контент. Например, можно использовать DTML, чтобы создать страницу с таблицей, заполненной сведениями, выбранными из базы данных. DTML основан на тегах и скриптовом языке. Это означает, что теги <dtml-var …>, вставленные в HTML, вызывают появление на вашей странице фрагмента за счет замены тега на "вычисленное" значение переменной, конвертированное в строку. DTML – скриптовый язык, используемый на стороне сервера. Это означает, что команды DTML выполняются публикатором Zope на сервере, и результат этого выполнения будет послан в браузер. Для сравнения, языки описания "на стороне клиента" подобно JavaScript не обрабатываются сервером, а тексты на них посылаются клиенту и выполняются браузером.

2.1 Синтаксис тегов DTML

DTML теги поддерживают два формата: Extended Python format strings (EPFS) и HTML. Формат EPFS основан на заключении питоновских строк текста в специальные символы форматирования, ”(” и ”)” для задания границ блоков кода. Дополнительный параметр форматирования позволяет указать детали преобразования данных, например:

%(date fmt=DayOfWeek upper)s

позволяет преобразовать дату как день недели заглавными буквами.

Формат HTML использует синтаксис HTML на стороне сервера для кодирования команд вставки текстов в формируемый документ. Параметры можно использовать для задания формы представления информации, например:

<dtml-var total fmt=12.2f>

позволяет преобразовать значение атрибута или переменной total в соответствии с форматом «12.2f» языка программирования Си.

В зависимости от используемого формата шаблон документа может быть создан двумя способами:

- documenttemplate.String – создается из строк в формате EPFS;
- documenttemplate.HTML – создается из строк в формате HTML на стороне сервера.

Синтаксические формы следующие:

- `%(`*`var_name`*`)x`
- `<dtml-`*`tag_name`*` ...>`

где x – формат представления переменной var_name как строки Питона

Синтаксис DTML для указания текстовых подстановок основан на формате стандартных атрибутов тегов, использованный в аналогичных шаблонах. DTML тег имеет форму:

```
<dtml-tag_name attribute1="value1" attribute2="value2" ... >
```

`tag_name` идентифицирует тип тега. Следующие за именем тега один или более атрибутов указывают, где расположены данные тега и как данные должны быть обработаны. Можно опускать имя атрибута **name** (сокращенная форма),

значения атрибутов и кавычки вокруг них, если значения не содержат пробелов, табуляций, символов новой строки, знаков равенства или двойных кавычек. Примеры:

```
<dtml-var date fmt=Date>
<dtml-var name="header">
<dtml-with subfolder>
<dtml-var name="input_name" capitalize>
```

С опущенным именем атрибута **name** последний пример будет иметь вид:

```
<dtml-var "input_name" capitalize>
```

DTML содержит два типа тегов: одиночные и блочные теги. Одиночные состоят из одной последовательности символов, заключенной между символами «<» и «>». Тег var является примером одиночного тега:

```
<dtml-var "parrot">
```

Нет необходимости закрывать тег `var` тегом `</dtml-var>`, поскольку он одиночный тег.

Блочные теги состоят из двух частей: той, которая открывает блок, и той, которая закрывает блок, а содержимое помещается между ними, например:

```
<dtml-in "mySequence">
   <!-- this is an HTML comment inside the in tag block -->
</dtml-in>
```

Открывающий тег начинает блок и заключительный заканчивает его. Заключительный тег имеет то же имя, что и открывающий, но с предшествующим ему слэшем. То же соглашение используется в HTML и в XML.

Все теги DTML имеют имена. Имя является словом, следующим за «dtml-». Например, имя тега DTML **`dtml-var`** есть **`var`** , и имя тега **`dtml-in`** есть **`in`**.

Большинство тегов DTML содержат объекты или значения. Значения в DTML – слово или выражение, следующее после пробела за именем тега. Например, объект тега `<dtml-var header>` есть `header`. Объект тега `<dtml-in foo>` есть `foo`. Объект тега `<dtml-var "list(values())">` – выражение `list(values())`. Объект обычно задается именем или выражением Питона, понимаемыми как объект, который обрабатывается тегом.

Все теги DTML имеют атрибуты. Атрибут обеспечивает информацию о том, как тег должен работать. Некоторые атрибуты обязательные, некоторые дополнительные. Например, тег var включает обязательное значение, но у него может быть дополнительный необязательный атрибут missing, который определяет значение по умолчанию в случае, если переменная не может быть найдена, например:

```
<dtml-var "title" missing="Unknown title">
```

Если переменная или атрибут `title` не обнаружены в пространстве имен, то подставляется строка `"Unknown title"`.

Некоторые атрибуты не имеют параметров. Например, Вы можете преобразовать значение включаемой переменной в верхний регистр атрибутом `upper`:

```
<dtml-var title upper>
```

Значительную пользу при программировании интерактивных взаимодействий оказывает объект *REQUEST,* содержащий информацию о текущем запросе клиента. Этот объект есть в пространстве имен DTML. Для просмотра содержимого *REQUEST* можно создать страницу DTML следующего вида:

```
<html>
  <body>
    <dtml-var REQUEST html_quote>
  </body>
</html>
```

При просмотре результата получим нечто похожее на следующий фрагмент:

```
CHANNEL_CREATION_TIME: 1130389634.53
CONNECTION_TYPE: keep-alive
GATEWAY_INTERFACE: CGI/1.1
HTTP_ACCEPT:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,tex
t/plain;q=0.8,image/png,*/*;q=0.5
HTTP_ACCEPT_CHARSET: windows-1251,utf-8;q=0.7,*;q=0.7
HTTP_ACCEPT_ENCODING: gzip,deflate
HTTP_ACCEPT_LANGUAGE: ru-ru,ru;q=0.8,en-us;q=0.5,en;q=0.3
HTTP_HOST: localhost:8080
HTTP_KEEP_ALIVE: 300 HTTP_REFERER:
http://localhost:8080/Cyx/dtmlTest/@@preview.html
HTTP_USER_AGENT: Mozilla/5.0 (Windows; U; Windows NT 5.0; ru;
rv:1.8b5) Gecko/20051006 Firefox/1.4.1
PATH_INFO: /Cyx/dtmlTest/
QUERY_STRING:
REMOTE_ADDR: 127.0.0.1
REQUEST_METHOD:
GET SCRIPT_NAME:
SERVER_NAME: pss-7
SERVER_PORT: 8080
SERVER_PROTOCOL: HTTP/1.1
SERVER_SOFTWARE: zope.server.http (HTTP)
```

2.2 Стандартные объекты и атрибуты

При разработке презентаций в распоряжении дизайнера сайта и программиста имеются стандартные объекты, поставляемые с системой Zope3. При программировании работы с этими объектами важно знать перечень и типы атрибутов, соответствующих этим объектам. Полная информация о всех объектах может быть найдена в исходных кодах объявлений как интерфейсов, так и классов. Наиболее значимые сведения об объектах можно получить на закладке «Информация» интерфейса менеджера. Приведем выборку наиболее употребительных при разработке презентаций объектов и их атрибутов.

Объект Folder / Папка

keys() – метод, возвращающий последовательность имен внутренних объектов папки. Конкретный тип последовательности задается операцией приведения tuple или list, например, <dtml-var “tuple(keys())” html_quote>.

values() – метод, возвращающий последовательность самих внутренних объектов папки. Конкретный тип последовательности задается операцией приведения tuple или list, например, list(values()).

items() – метод, возвращающий последовательность пар (имя, объект) для внутренних объектов папки. Конкретный тип последовательности задается операцией приведения tuple или list, например, tuple(items()).

get(name, default=None) – метод, возвращающий объект с указанным именем или значение default, если объекта нет, например, <dtml-var "get('DTvarTest')" html_quote>.

__getitem__(key) – метод, возвращающий объект с указанным именем или исключение, если объекта нет, например, <dtml-var "getitem('DTvarTest')" html_quote>.

__len__() – метод, возвращающий количество объектов в папке, например, <dtml-var "__len__()" >

__name__ – атрибут, имя папки <dtml-var "__name__" html_quote>.

__parent__ – атрибут, родительская папка, например, <dtml-var "__parent__.__name__" html_quote>

Объект File / Файл

getSize() – метод, возвращающий число байт в файле, например, <dtml-var "get('readme').getSize()">.

contentType – атрибут, идентифицирует тип данных например, <dtml-var "get('readme').contentType">.

data – атрибут, содержимое файла, например, <dtml-var "get('readme').data">.

Объект Image / Образ

getSize() – метод, возвращающий число байт в файле, например, <dtml-var "get('logo').getSize()">.

getImageSize() – метод, возвращающий пару (x,y) – размер рисунка в точках растра, например, <dtml-var "get('logo').getImageSize()">.

contentType – атрибут, идентифицирует тип рисунка, например, <dtml-var "get('logo').contentType">.

data – атрибут, содержимое рисунка, например, <dtml-var "repr(get('logo').data)" html_quote>.

2.3 Справочник по тегам DTML

call: вызов метода

Тег call позволяет Вам вызывать метод, не включая результаты в выход DTML документа.

Синтаксис:

```
<dtml-call Variable|expr="Expression">
```

Если вызов использует переменную, аргументы методов передаются автоматически DTML подобно тегу var. Если вызываемый метод определяется выражением, тогда нужно самим передавать аргументы.

Примеры

Вызов с именем переменной:

```
<dtml-call UpdateInfo>
```

Это вызов объекта UpdateInfo с автоматической передачей аргументов.

Вызов выражением:

```
<dtml-call expr="RESPONSE.setHeader('content-type',
                  'text/plain')">
```

comment: комментарии в коде DTML

Тег комментария позволяет Вам документировать код DTML. Можно временно исключить из программы DTML-теги, комментируя их.

Синтаксис:

```
<dtml-comment>
    . . .
</dtml-comment>
```

Тег комментария является блочным. Содержимое блока не выполняются, ничего не включается в выход DTML. Примеры

Документирование DTML:

```
<dtml-comment>
  This content is not executed and does not appear in the
  output.
</dtml-comment>
```

Закомментированные DTML-теги:

```
<dtml-comment>
  This DTML is disabled and will not be executed.
  <dtml-call someMethod>
</dtml-comment>
```

функции языка DTML

DTML позволяет пользоваться некоторыми встроенными функциями Питона и некоторыми специальными DTML-функциями:

abs(number)

Возвратить абсолютную величину числа. Аргумент может быть простым или длинным целым или числом с плавающей запятой. Если аргумент - комплексное число, то возвращается его модуль.

chr(integer)

Возвратить строку из одного символа, чей код ASCII является целым, например, chr(97) возвращает строку a. Это инверсия функции ord(). Аргумент должен быть в области 0 - 255, включительно. Возбуждается исключение ValueError, если целое - за пределами этой области.

DateTime()

Возвращает Zope объект DateTime, дающий аргументы конструктора. Смотри DateTime ссылку API для полной информации об аргументах конструктора.

divmod(number, number)

Возвращайте пару чисел состоящую из их частного и остатка при использовании длинного деления. К смешанным типам операндов относятся правила для двоичных арифметических операторов. Для простых и длинных целых, результат такой же, как и `(a / b, a % b)`. Для плавающей точки результат – `(q, a % b )`, где q - обычно `math.floor(a / b)`, но может быть менее чем 1. В любом случае `q * b + a % b` очень близкое к `a`, если `a % b` не равным нулю, у него есть тот же знак как `b`, и `0 <= abs(a % b) < abs(b)`.

float(number)

Преобразует строку или число в число с плавающей точкой.

getattr(object, string)

Возвращает величину поименованного атрибута объекта. Имя должно быть строкой. Если строка является именем одного из объектных атрибутов, результат является величиной этого атрибута. Например, `getattr(x, "foobar")` - эквивалент `x.foobar`. Если поименованный атрибут не существует, возвращается умолчание, если предусмотрено, в противном случае возбуждается `AttributeError`.

getitem(variable, render=0)

Возвращает значение переменной DTML. Если `render` – истина, переменная предоставляется.

hasattr(object, string)

Аргументы - объект и строка. Результат – 1, если строка - имя одного из объектных атрибутов и 0, если нет. Пример: <dtml-var expr="hasattr(REQUEST, 'Form')">

hash(object)

Возвращайте hash величину объекта (если он ее имеет). Величины являются целыми. Они используются, чтобы быстро сравнивать словарные ключи при словарном поиске. Числовые величины, которые в результате сравниваются должны иметь ту же hash величину (даже если бы они - других типов, напр.. 1 и 1.0). . Пример: <dtml-var expr="hash('My string')">

has_key(variable)

Возвращается True, если пространство имен DTML содержит поименованную переменную. Пример: <dtml-var expr="dict(REQUEST).has_key('Form')" >

hex(integer)

Преобразовывает целое (любого размера) в шестнадцатеричную строку.

int(number)

Преобразование строки или числа в простое целое.

len(sequence)

Возвращает длину (количество элементов) объекта. Аргумент может быть последовательностью (строка, кортеж, список или словарь).

max(s)

С единственным аргументом s, возвращает самый большой пункт непустой последовательности (строка, кортеж или список). Более чем с одним аргументом, возвращает самый большой аргумент.

min(s)

С единственным аргументом s, возвращает самый меньший пункт непустой последовательности (строка, кортеж или список). Более чем с одним аргументом, возвращает самый меньший аргумент.

oct(integer)

Преобразование целого (любого размера) в восьмеричную строку

ord(character)

Возвращает код ASCII символа. Например, ord("a"), возвращает целое 97.

pow(x, y [,z])

Возвращает x в степени y; если присутствует z, то возвращается x в степени y по модулю z

range([start,] stop [,step])

Создает списки, содержащие арифметическое прогрессии. Аргументы должны быть простые целые.

round(x [,n])

Возвращает число с плавающей точкой x, округленное до n цифр в десятичной дроби. Если n опущен, он устанавливается по умолчанию в нуль.

render(object)

Представляет объект. Для DTML объектов возвращает результат вычисления кода DTML в текущем namespace. Для других объектов, это эквивалент str(object).

reorder(s [,with] [,without])

Переупорядочение пунктов в s согласно заданному порядку with и без пунктов, упомянутых в without. Пункты из s не упомянутые в with - удаляются. s, with, and without – все последовательности строк или последовательности ключевых кортежей, с упорядочением по ключам. Эта функция полезна для создания сортированных списков выбора.

str(object)

Возвращает строку, содержащую строковое представление объекта. Для строк возвращает саму строку.

unichr(number)

Возвращает строку уникода, представляющую число как символ уникода. Это - инверсия ord() для символов уникода.

unicode(string[, encoding[, errors]])

Декодирует строку, используя codec для кодирования. Пример: <dtml-var expr="unicode('Example')">

if: проверка условия

Тег if позволяет проверять условия и выбирать действия в зависимости от условий. Тег if – зеркало питоновского условного оператора `if/elif/else`.

Синтаксис

```
<dtml-if ConditionVariable|expr="ConditionExpression">
...
[<dtml-elif ConditionVariable|expr="ConditionExpression">]
...
[<dtml-else>]
...
</dtml-if>
```

Тег if является блочным тегом. Тег if и дополнительные теги elif используют в условии переменную или выражение. Если условие true, то блок выполняется. true – это все что не нуль, не пустая строка или не пустой список. Если переменная условия не найдена, тогда условие считается false.

Примеры

Условие для переменной:

```
<dtml-if snake>
   The snake variable is true
</dtml-if>
```

Условия для выражения:

```
<dtml-if expr="num > 5">
   num is greater than five
  <dtml-elif expr="num < 5">
   num is less than five
  <dtml-else>
   num must be five
</dtml-if>
```

in: цикл по последовательности

Тег in предоставляет мощные средства циклического выполнения блока для элементов последовательности с возможностью пакетной обработки.

Синтаксис

```
<dtml-in SequenceVariable|expr="SequenceExpression">
  . . .
  [<dtml-else>]
  . . .
</dtml-in>
```

Блок in повторяется для каждого пункта последовательности, заданной переменной или выражением. Текущий элемент выталкивается на вершину стека пространства имен DTML на период каждого выполнения блока in.

Если нет элементов в переменной последовательности или выражении, выполняется дополнительный блок else.

Атрибуты

reverse — Развернуть последовательность.

reverse_expr=expression — Разворачивает последовательность, если выражение true. Это позволяет выборочно разворачивать последовательность.

sort=string — Сортирует последовательность по значению атрибута с указанным именем.

sort_expr=expression — Сортирует последовательность атрибутом, названным значением выражения. Это позволяет сортировать по различным атрибутам.

Переменные тега

Текущие переменные элемента последовательности описывают текущий пункт.

sequence-item — Содержимое текущего пункта.

sequence-key — Текущий ключ. циклы над кортежами `(key,value)`, in интерпретирует их как `(sequence-key, sequence-item)`.

sequence-index — Индекс текущего пункта, начинаемый с 0.

sequence-number — Индекс текущего пункта, начинаемый с 1.

sequence-roman — Индекс текущего пункта в Римских Цифрах нижнего Регистра.

sequence-Roman — Индекс текущего пункта в Римских Цифрах Верхнего Регистра.

sequence-letter — Индекс текущего пункта в буквах нижнего регистра.

sequence-Letter — Индекс текущего пункта в буквах верхнего регистра.

sequence-start — Верно если текущий пункт является первым пунктом.

sequence-end — Верно если текущий пункт является последним пунктом.

sequence-length — Длина последовательности.

sequence-var-nnn — Переменная в текущем пункте. Например, `sequence-var-title` является переменной `title` текущего пункта. Обычно можно иметь доступ к этим переменным непосредственно, так как текущий пункт выталкивается в namespace DTML. Тем не менее, эти переменные могут быть полезными при отображении предшествующей и следующей пакетной информации.

Итоговые переменные

Эти переменные суммирует информацию о числовых переменных пункта. Чтобы получить итоговую статистику нужно использовать имя переменной в дополнение к имени статистики.

total-variable — Итог всех числовых значений переменной пункта.

count-variable — Количество случаев переменной пункта.

min-variable — Минимальная величина переменной пункта.

max-variable — Максимальная величина переменной пункта.

mean-variable — Средняя величина переменной пункта.

variance-variable — Вариация переменной пункта с count-1 степенями свободы.

variance-n-variable — Вариация переменной пункта с n степенями свободы.

standard-deviation-variable — Стандартное отклонение переменной пункта с count-1 степенями свободы.

standard-deviation-n-variable — Стандартное отклонение переменной пункта с n степенями свободы.

Примеры

Выполнение Цикла над подобъектами текущей папки:

```
<dtml-let lst="list(values())">
  <dtml-in lst>
    title: <dtml-var sequence-item html_quote> <br>
  </dtml-in>
 </dtml-let>
```

Выполнение Цикла над набором:

```
<dtml-let rows="(1,2,3)">
  <dtml-in rows>
   <dtml-var sequence-item><br>
   <dtml-if sequence-end>
       End of sequence
   </dtml-if>
  </dtml-in>
</dtml-let>
```

let: определение новых DTML переменных

Тег let определяет переменные в пространстве имен DTML.

Синтаксис

```
<dtml-let [Name=Variable][Name="Expression"]...>
  . . .
</dtml-let>
```

Пример:

```
  <dtml-let num="2"
            index="3"
            result="num*index">
    <dtml-var num> * <dtml-var index> = <dtml-var result>
  </dtml-let>
```

raise: поднимает исключение

Тег raise поднимает исключение, зеркальное отражение утверждения Питона `raise`.

Синтаксис

```
<dtml-raise ExceptionName|ExceptionExpression>
</dtml-raise>
```

Тег raise является блочным. Он поднимает исключение. Исключения могут быть исключительным классом или строкой. Содержимое тега передается как значение ошибки.

Примеры

Подъем KeyError:

```
<dtml-raise KeyError></dtml-raise>
```

Возбуждение HTTP 404 error:

```
<dtml-raise NotFound>Web Page Not Found</dtml-raise>
```

return: возврат данных

Тег `return` перестает выполнять DTML и возвращает данные. Это зеркало утверждения `return` Питона.

Синтаксис

```
<dtml-return ReturnVariable|expr="ReturnExpression">
```

Останавливает выполнение DTML и возвращает значение переменной или выражения. Ранее сформированный выход DTML страницы теряется. Для сценариев это устаревший тег.

Примеры

Возврат переменной:

```
<dtml-return result>
```

Возврат словаря Питона:

```
<dtml-return expr="{'hi':200, 'lo':5}">
```

try: Обработка ошибок

Тег **try** допускает обработку особой ситуации в DTML, зеркальное отражение Питон `try/except` и `try/finally`.

Синтаксис

Тег имеет два синтаксиса, `try/except/else` и `try/finally`.

Синтаксис try/except/else:

```
<dtml-try>
<dtml-except [ExceptionName] [ExceptionName]...>
...
[<dtml-else>]
</dtml-try>
```

`try` включает блок в котором исключения могут ловиться и обрабатываться. Может быть использован один или более `except` тегов, которые обрабатывают нуль или больше исключений. Если `except` тег не определил исключение, тогда, он обрабатывает все исключения.

Когда исключение поднимается, управление перескакивает на первый `except`, который обрабатывает исключение. Если нет `except`, чтобы обрабатывать исключение, то исключение поднимается как обычно.

Если никакое исключение не поднимается, и есть еще `else` тег, то `else` будет выполнено после тела `try` тега. e`xcept` и `else` - опциональны.

Синтаксис try/finally:

```
<dtml-try>
  . . .
<dtml-finally>
  . . .
</dtml-try>
```

`finally` не может использоваться в том же блоке `try` как `except` и `else`. Если есть `finally`, блок будет выполнен независимо от того, поднимается ли исключение в блоке `try`.

Атрибуты

except — Нуль или больше исключительных имен. Если никакие исключения не указываются, тогда тег обрабатывает все исключения.

Переменные тега

Эти переменные определены внутри блока `except`.

error_type — Тип исключения.

error_value — Значение исключения.

error_tb — Обратная трассировка вызовов.

Примеры

Отлов математической ошибки:

```
<dtml-try>
<dtml-var expr="1/0">
<dtml-except ZeroDivisionError>
You tried to divide by zero.
</dtml-try>
```

Возврат информации об обрабатываемом исключении:

```
<dtml-try>
  <dtml-call dangerousMethod>
<dtml-except>
  An error occurred.
  Error type: <dtml-var error_type>
  Error value: <dtml-var error_value>
</dtml-try>
```

Использование finally,, чтобы убедиться в выполнении разблокировки независимо от того, ошибка поднимается или нет:

```
<dtml-call acquireLock>
<dtml-try>
  <dtml-call someMethod>
<dtml-finally>
  <dtml-call releaseLock>
</dtml-try>
```

unless: Тестирует условие

`unless` тег обеспечивает сокращенное написание для отрицания условий. Для тестирования прямого условия используется тег `if`.

Синтаксис

<dtml-unless ConditionVariable|expr="ConditionExpression">

</dtml-unless>

Тег `unless` является блочным тегом. Если переменная условия или выражение оценивается false, то включаемый блок выполняется. По аналогии с тегом `if` для переменных, которые не находятся в пространстве имен, условие считаются false.

Примеры

Проверка переменной:

```
<dtml-unless testMode>
  <dtml-call dangerousOperation>
</dtml-unless>
```

Блок выполнится, если testMode не существует или существует, но равно false.

var: Включение переменной

Тег var включает значение переменных в выход генератора DTML.

DTML синтаксис

```
<dtml-var Variable|expr="Expression">
```

Тег var является одиночным тегом. Тег var находит переменную в DTML namespace, который обычно состоит из текущего объекта, текущих объектных контейнеров, и, наконец, запроса паутины. Если переменная обнаруживается, она включается на DTML выход. Если не обнаруживается, то Zope поднимает ошибку.

Entity синтаксис:

```
&dtml-variableName;
```

Entity синтаксис с атрибутами:

```
&dtml.attribute1[.attribute2]...-variableName;
```

Можно включить нуль или больше атрибутов, ограниченные точками. Нельзя обеспечить аргументы для атрибутов, используя синтаксис элемента. Пример: &dtml.html_quote-colors;

Атрибуты

html_quote —Преобразовать символы, имеющие специальное значение в HTML в символьные объекты HTML.

missing=string — Определить значение по умолчанию в случае, если Zope не может найти переменную.

fmt=string —Форматирование переменной. Zope обеспечивает некоторые встроенные форматы, включая C-стиль форматных строк. Если строка формата является не встроенным форматом, тогда она понимается как вызов метода объекта.

whole-dollars — Форматирует переменную как доллары.

dollars-and-cents — Форматирует переменную как доллары и центы.

collection-length — Длина переменной добавляется к последовательности.

structured-text — Форматирует переменную как Structured Text (Структурный Текст).

null=string — Значение по умолчанию, если переменная равна None.

lower — Верхний регистр символов преобразуется к нижнему регистру.

upper — Преобразовывает строчные символы к верхнему регистру.

capitalize — Превращает первый символ включенного слова в заглавный.

spacify — Изменение подчеркивания во включенной величине в пробел.

thousands_commas — Включает запятые через каждые три цифры слева от знака десятичной дроби в величинах, содержащих числа например 12000 становится 12,000.

url — Включает URL объекта, вызывая метод `absolute_url`.

url_quote —Преобразование символов, которые имеют специальное значение в URLs в символьных объектах HTML.

url_quote_plus —Подобен `url_quote`, но также преобразование пробелов в знаки плюс.

sql_quote — Преобразовывает единичные кавычки в пары кавычек. Это - нужно для включения величины в строки SQL.

newline_to_br — Преобразование Конца строки (включая возврат каретки), в прерывания строки HTML.

size=arg — Обрезает переменную по заданной длине (Примечание: если пробел встречается во второй половине усеченной строки, то строка в дальнейшем усекается по всем пробелам).

etc=arg — Определяет строку для добавления к концу строки, которая усечена (установкой атрибута размера, указанного выше). По умолчанию, это – «...»

Примеры

Включение простой переменной в документ:
```
<dtml-var "name">
<dtml-var "REQUEST" html_quote>
```

Включение содержимого файла в документ:
```
<dtml-var "get('readme').data" >
```

Списки и словари объектов текущей папки:
```
<dtml-var "tuple(values())" html_quote>
<dtml-var "tuple(values())" url_quote>
<dtml-var "str(len(tuple(keys())))">
<dtml-var "dict(items())" html_quote>
```

Первый объект текущей папки:
```
<dtml-var "str((list(values())[0]))" html_quote>
```

Имя родительской папки:

<dtml-var expr="__parent__.__name__" html_quote>

Запрос атрибута объекта:
```
<dtml-var "getattr(get('img'), '__name__', 'Title')" >
<dtml-var "(get('img')).getImageSize()" html_quote>
```

Округление:
```
<dtml-let colors="['red', 'green', 'blue']">
  colors = <dtml-var "colors" size=15 etc=", etc.">
</dtml-let>
```

генерирует следующий выход: `colors = ['red', 'green', etc.`

С-стиль форматирования строки:
```
<dtml-var expr="23432.2323" fmt="%.2f">
```

генерируется как «23432.23»

Использование entity синтаксиса:
```
colors = "&dtml-colors;
&dtml.html_quote-colors;
```

with: Управление видимостью DTML переменной

Тег with вталкивает объект в namespace DTML. Переменные будут видны на вытолкнутом объекте первыми.

Синтаксис

```
<dtml-with Variable|expr="Expression">
  . . .
</dtml-with>
```

Тег with является блочным. Он выталкивает поименованную переменную или значение выражения в пространство имен DTML на время работы блока with, что делает имена видимыми на вытолкнутом объекте первыми.

Атрибуты

only — ограничивает пространство имен DTML, чтобы видеть только свое определение объекта тега with. Предотвращает заимствование атрибутов и методов из окружения.

mapping — Указывает, что переменная или выражение - словарь объекта. Это гарантирует, что переменные – правильно видимы для объекта.

Примеры

Просмотр переменной в REQUEST:

```
<dtml-with REQUEST only>
  <dtml-if id>
    <dtml-var id>
  <dtml-else>
    'id' was not in the request.
  </dtml-if>
</dtml-with>
```

Проталкивание первого дочернего объекта на namespace DTML:

```
<dtml-with expr="tuple(values())[0]" only>
  <dtml-if __name__>
    First child's name: <dtml-var __name__>
  <dtml-else>
    '__name__' was not in the object.
  </dtml-if>
</dtml-with>
```

3 Страничные шаблоны

Страничные шаблоны являются средством генерации страниц на стороне сервера. Они позволяют программистам и дизайнерам работать совместно над созданием динамических страниц для приложений Zope. Дизайнеры могут использовать их в инструментальных средствах разработки документов WYSIWYG, вставлять созданный программистами код в свои страницы без нарушения грамматики языка HTML. Цель страничных шаблонов – предоставить инструмент для разработки интерфейсов с пользователем как для дизайнера, отвечающего за оформление документа, так и для программиста, обеспечивающего программное управление динамически формируемым контентом. Соблюдая оговоренные соглашения при совместной работе, дизайнер и программист не могут разрушить целостности приложения, отвечая каждый за свою часть.

Страничные шаблоны подчиняются трем принципам:

1. Совместимость с инструментальными средствами редактирования.

2. WYSIWYG – что Вы видите в редакторе, то Вы и получите в браузере.

3. Отделение кода приложения от презентационной логики.

ZPT страницы являются альтернативой DTML страниц, которые не предназначены для дизайнеров, работающих с языком HTML. Как только на HTML странице появляется код DTML, то результат обычно становится непригодным для редакторов и браузеров (например, Dreamweaver). В DTML не совсем удачно разделены представление информации, логика формирования документа и контент, из которого документ формируется. Это может затруднить масштабирование содержимого и разработку самого сайта. Наконец, модель пространства имен (namespace) в DTML имеет слишком много скрытых нюансов при работе с объектами, не допуская полного программного управления поиском. Значительные трудности (в существующей версии Zope 3.2.0) вызывает использование в выражениях встроенных переменных, содержащих в имени символ «-», интерпретируемый как операция минус. Следует ожидать, что эта коллизия в будущем будет устранена. По прогнозам авторов Zope3 язык DTML не имеет в будущем перспектив и будет удален или заменен более функциональным аналогом.

3.1 Операторы языка TAL

При использовании шаблонов страниц разработчикам приходится иметь дело с языками TAL, TALES и METAL, являющимися расширениями языка HTML/XML. Как это принято в XML, комментарии в шаблоне можно оформить блочными тегами начала <!-- и окончания -->, что позволят делать тексты шаблонов понятными для участников разработки приложений. Общие правила записи шаблонов ничем не отличаются от стандартов языка HTML. Операторы языка TAL подчиняются общему синтаксису тегов

```
<тег  атрибут=значение  атрибут=значение  . . .  >
```

Отличием операторов состоит в том, что атрибут тега начинается с ключевого слова **tal** и отделяется от имени оператора двоеточием «:», например, `tal:replace=request/SERVER_NAME`. Слева и справа от символа равенства можно оставлять пробелы. Значение атрибута заключается в одинарные или двойные кавычки, если оно содержит специальные символы. Обычно для структуризации текста значение записывают в двойных кавычках. Этот стиль по

существу стал стандартом записи выражений в операторах TAL. Следует иметь в виду, что в пределах одного тега нельзя использовать один и тот же оператор более одного раза. Само имя tal может использоваться в блочном теге **`<tal>`** … **`</tal>`**.

Включение текста может производиться двумя способами: заменой тега и заменой содержимого тега. Замена тега на значение производится оператором

tal:replace = выражение

Например, в следующем фрагменте

```
The URL is <span tal:replace="request/URL">
        http://www.example.com </span>
```

блочный тег span будет целиком заменен на локатор ресурса вида `http://localhost:8080/DTML/ZPTmySQL`.

Если необходимо включить текст внутри тега, но оставить сам тег, то используется оператор

tal:content. = выражение

Например, во фрагменте

```
<head>
  <title tal:content="template/title">
        The Title
  </title>
</head>
```

блочный тег title будет оставлен, но его содержимое «The Title» будет заменено значением атрибута «template/title».

Обычно операторы tal:replace и tal:content преобразовывают теги HTML и объекты в текст, который они включают в результирующий документ как простые символы, а не как разметку HTML. Если необходимо включить HTML-текст как часть структуры вашего документа без преобразования (как есть), то нужно перед выражением поставить ключевое слово «**structure**». Этот прием полезен, когда включаемый фрагмент HTML или XML кода является значением некоторого атрибута или генерируется другим объектом Zope. Например, пусть имеются файлы новостей в папке «news» приложения, которые сами являются HTML-страницами и содержат внутри разметку HTML (например, bold, italic и др.). Необходимо эти выделения сохранить при включении файлов в страницу "Top News". В этом случае, можно написать:

```
   <table border=1>
     <tr tal:repeat="item container/news/values">
       <td tal:content="structure item/source"> Новость дня </td>
      </tr>
    </table>
```

Без ключевого слова **structure** символы разметки исходных файлов будут преобразованы в специальные коды, например, «`<b> </b>`», что приведет к их отображению в браузере строкой « ».

Можно оставить содержание или сам элемент, используя в tal:content или tal:replace выражение **`default`**. Например: <p tal:content = "default"> Spam </p> интерпретируется, как: <p> Spam </p>. Если необходимо выборочно включать или нет некоторое содержимое, то используется комбинация значений атрибутов. Например:

```
<p tal:content="python:container.get('Obj') or default"> Spam </p>
```

Если метод `get('Obj')` возвращает не пустое значение, то этот результат будет включен в параграф, в противном случае он останется строкой «Spam».

Циклические повторения структур в результирующей странице создаются оператором

tal:repeat = "переменная последовательность"

Переменная последовательно пробегает значения элементов последовательности. Обычно оператор цикла помещается внутрь блочного тега, являющимся контейнером для других операторов динамического формирования текста документа для каждого повторяемого элемента последовательности. Например, цикл по динамически формируемым строкам таблицы может выглядеть так:

```
<table border=1>
  <tr tal:repeat="item container/values">
    <td tal:content="python:str(item.__name__)"> Имя </td>
    <td tal:content="python:repr(item.source)"> Тело </td>
  </tr>
</table>
```

Переменная item последовательно получает значение объектов текущей папки приложения. Ее значение может использоваться в пределах блочного тега, где употреблен оператор «tal:repeat». В нашем примере она передается в программы на языке Питон для вычисления нужных значений атрибутов «__name__» и «source» объектов папки, из которой вызван шаблон. Использование выражений питона, функций str и repr не обязательное и зависит от того, что и в каком формате должно быть помещено в документ.

При использовании оператора повторения в его области видимости доступны следующие переменные, обеспечивающие дополнительную информацию о текущем элементе:

- index – номер цикла, начинающийся с нуля;
- number – номер цикла, начинающийся с единицы;
- even – истина для четных индексов повторений (0, 2, 4,...);
- odd – истина для нечетных повторений (1, 3, 5,...);
- start – истина для начального повторения (индекс 0);
- end – истина для последнего повторения;
- length – длина последовательности, общее число повторений;
- letter - буквенный индекс повторения на нижнем регистре: "a" - "z", "aa" - "az", "ba" - "bz",..., "za" - "zz", "aaa" - "aaz", и так далее;
- Letter – то же самое на верхнем регистре.

В силу возможной вложенности операторов доступ к значению переменной повторения производится указанием трех разделов пути: имени оператора repeat, имени переменной повторения и имени специальной переменной, например, “repeat/item/number”. В выражениях Питона используется обычная нотация, например, python:repeat['item'].start. Пример использования переменных повторения:

```
<tr tal:repeat="item container/news/values">
```

```
  <td> <span tal:content="repeat/item/number"> # </span>).
       <span tal:content="structure item/source"> # </span>
  </td>
</tr>
```

Для повторения произвольного фрагмента шаблона можно использовать иные формы оператора цикла:

```
<tal:for tal:repeat="имя последовательность">
   тело
</tal:for>
```

или

```
<tal:loop tal:repeat="имя последовательность">
   тело
</tal:loop>
```

В заключение описания повторений несколько полезных практических рекомендаций. Иногда нужно повторять часть вашего шаблона, но не генерировать сам элемент, внутри которого использован оператор `repeat`. Это можно сделать с использованием атрибута **`tal:omit-tag=""`**, например:

```
<tr tal:repeat="item container/news/values" tal:omit-tag="">
  <td> <span tal:content="repeat/item/number"> # </span>).
       <span tal:content="structure item/source"> # </span>
  </td>
</tr>
```

Утверждения tal:repeat могут быть вложенными в друг друга. Каждое tal:repeat утверждение должно иметь внутри своего тега свое уникальное имя переменной повторения. Приведем пример, который генерирует таблицу умножения:

```
<table border="1">
  <tr tal:repeat="x python:range(1, 10)">
    <td tal:repeat="y python:range(1, 10)"
        tal:content="python:'%d x %d = %d' % (x, y, x*y)">
      X x Y = Z
    </td>
  </tr>
</table>
```

При использовании повторений необходимо внимательно следить за областью видимости переменных. Типичной ошибкой является следующий фрагмент:

```
<!-- ошибочный шаблон -->
<ul>
  <li tal:repeat="n python:range(10)"
      tal:condition="python:n != 3"
      tal:content="n">
    1
  </li>
</ul>
```

Здесь в одном теге объединены цикл и проверка условия для переменной цикла, что приведет к неверному результату. Для исправления ошибки следует написать следующее:

```
<ul>
  <div tal:repeat="n python:range(10)"
```

```
        tal:omit-tag="">
     <li tal:condition="python:n != 3"
         tal:content="n">
       1
     </li>
   </div>
</ul>
```

За счет tal:omit-tag="" тег «div» не попадет в результирующий документ, но исполнение цикла будет верным, что при проверке условия в теге «li» приведет к правильному результату.

К сожалению, в отличие от DTML, tal:repeat сам не поддерживает сортировку. Если необходимо отсортировать список, то нужно написать скрипт на Питоне. Другая проблема, связанная с циклической обработкой, – группирование. Группирование является процессом разбиения большого списка на меньшие списки. Утверждение tal:repeat не поддерживает группирование, но в Zope можно обойти эту трудность написанием шаблона с использованием параметров запроса. Примером решения такой задачи может быть следующий фрагмент:

```
1) <html>
2)  <body>
3)   <div tal:define="global next request/QUERY_STRING | nothing">
4)     <div tal:condition="not:  next">
5)       <div tal:define="global next python: 1"  />
6)    </div>
7)
8)    <div tal:condition="python: int(next) < 100">
9)       <table border="1">
10)           <tr tal:repeat="item python:range(int(next),
int(next)+10)">
11)              <td tal:content="item"> ii </td>
12)           </tr>
13)        </table>
14)        <a href="?00"  tal:condition="python: int(next) < 91"
               tal:attributes="href python:'?'+str(int(next)+10)">
               Next </a>
15)     </div>
16)    </div>
17)   </body>
18) </html>
```

Строки 3 – 6 управляют значением глобальной переменной next, используемой для задания начальной позиции в последовательности (от 1 до 100) при отображении порции. Значение поступает из атрибута request/QUERY_STRING, формируемого в запросе на следующую порцию при выборе клиентом ссылки Next. Обратите внимание на невозможность переопределения значения переменной в языке DTML в отличие от TAL, что иллюстрирует преимущества языка TAL.

Строки 9 – 13 формируют и отображают элементы порции последовательности.

Строка 14 формирует ссылку с указанием следующего начального номера последовательности, если вся последовательность не исчерпана.

Аналогично могут быть сформированы ссылки для движения в обратном направлении, к началу и в конец последовательности. Источниками формирования последовательностей могут быть запросы к базам банных, длинные тексты из файлов и др. Правила формирования самой порции существенно зависят от источника данных.

Условный оператор служит для проверки условий выполнения некоторых операций. Синтаксис условного оператора :

tal:condition = выражение

Он позволяет производить генерацию тега, в котором он размещен, только при выполнении заданного выражением условия в динамически формируемой странице. Следует иметь в виду, что в Питоне ложными считаются следующие результаты: константы None. False, 0. пустая строка “”, пустой тьюпл (), пустой список [] и пустой словарь {}. Остальные значения интерпретируются как истинные высказывания. В следующем примере параграф будет присутствовать в документе, если в запросе клиента будет найден ключ SERVER_PORT:

```
<p tal:condition="request/SERVER_PORT | nothing">
  The SERVER_PORT information in your request is :
<span tal:replace="request/SERVER_PORT"> Port </span> <br>
</p>
```

В следующем примере таблица будет создана, если текущая папка не пуста.

```
<table tal:condition="container/values"
         border="1" width="100%">
  <tr>
    <th>Имя</th>
    <th>Объект</th>
  </tr>
  <tr tal:repeat="item container/values">
    <td tal:content="python:str(item.__name__)"> # </td>
    <td tal:content="python:repr(item.source)"> # </td>
  </tr>
</table>
```

Условное управление выполнением произвольных фрагментов шаблона можно выполнить оператором **if** в теге

```
<tal:if condition = выражение >
   тело
</tal:if>
```

Например:

```
<tal:if condition="not:person">
   <span id="tools-zmi-principal" i18n:translate=""
         tal:content="user/title">ZMI Principal</span>
</tal:if>
```

Аналогично обход фрагмента выполняется оператором

```
<tal:disable condition = выражение >
   тело
</tal:disable>
```

Изменение атрибутов тега можно выполнить оператором

tal:attributes = ”имя выражение”

Например, атрибут «src» тега «img» будет заменен в выходном документе на значение «string:${context/img/@@absolute_url}», содержащее правильный URL изображения. Атрибут «src="http://localhost:8080/img"» в шаблоне выступает в качестве заполнителя.

```
<img src="http://localhost:8080/img" alt="Logo"
      tal:attributes="src string:${context/img/@@absolute_url}">
```

Определение новых атрибутов. Часто для сокращения длины выражений в атрибутах tal целесообразно в пределах некоторой области видимости задать новые имена переменных и связать их с некоторыми объектами приложения или значениями их атрибутов. Определение новых атрибутов, видимых внутри включающего тега (обычно блочного), производится оператором

tal:define = ”имя$_1$ значение$_1$; … ; имя$_n$ значение$_n$”

Например:

```
<div tal:define="qText    python: getQ['qTest'];
                 qImage   python: getQ['qImage'];
                 ansVars  python: getQ['ansVars']">
. . .
</div>
```

Обратите внимание на использование выражений языка Питон для вычисления присваиваемых новым атрибутам значений. Таким выражениям предшествует ключ «**python:**», за которым следует выражение на языке Питон, вычисленное значение которого связывается с переменной. В данном примере это элементы словаря getQ. Выражения на языке Питон можно использовать везде, где ожидается вычисленное значение атрибута или метода, например, `tal:condition = "python: parent not in view.parents()"`. В последнем примере `view.parents()` – метод, возвращающий коллекцию (список или тъюпл), а `parent` – переменная, значение которой не должно присутствовать в коллекции.

Устанавливая ключевое слово «**global**» перед именем переменной, можно сделать определение видимым от тега до конца шаблона:

```
<span tal:define="global items container/values"></span>
<h4 tal:condition="not:items">There Are No Items</h4>
```

Можно определить более чем одну переменную, разделяя их с точками с запятой. Например:

```
<p tal:define="ids container/values;
               name python:container.__name__">
```

В отличие от DTML страничные шаблоны можно использовать для **генерации XML** страниц. Приведем пример, в котором будет создан перечень имен в следующем формате XML:

```
<objectlist>
  <entry>
    <name>имя_объекта</name>
  </entry>
 . . .
</objectlist>
```

Это можно выполнить шаблоном следующего содержания:

```
<objectlist xmlns:tal="http://xml.zope.org/namespaces/tal">
  <entry tal:repeat="entry python:context.values()">
    <name tal:content="python:entry.__name__">
      It is object name
    </name>
  </entry>
</objectlist>
```

Отдельного внимания заслуживает использование атрибута «`tal:omit-tag`», позволяющего опускать тег, внутри которого он использован. При этом не генерируется открывающий и закрывающий теги блочного элемента, но его содержимое передается полностью. Например:

```
<b tal:omit-tag=""><i>this</i> stays</b>
```

преобразуется в

```
<i>this</i> stays
```

где отсутствует выделение жирным шрифтом.

Обработка ошибок выполняется оператором

tal:on-error = выражение

Например:

```
<span
  tal:on-error="string:Произошла ошибка"
  tal:define="global prefs here/scriptToGetPreferences">
    Пример ошибки
</span>
```

При генерации документа отыскивается первый, охватывающий точку возникновения ошибки, тег, имеющий оператор tal:on-error, и производится его исполнение, приводящее к генерации текста сообщения в документе вместо дефектного тега. Для большей гибкости можно вместо строки вызывать объект, который и вернет текст сообщения, например:

```
<span
  tal:on-error="structure python:str(container.get(
            'PyScript')(container))"
  tal:define="global prefs here/scriptToGetPreferences">
 Пример ошибки
</span>
```

3.2 Выражения TAL

Выражения задают значения в операторах TAL шаблона. Например, в утверждении TAL <td tal:content="request/form/age">Age</td> выражением является строка "request/form/age", задающая путь получения значения.

Встроенные переменные шаблонов задаются именами, которые можно использовать в выражениях операторов TAL. К ним относятся следующие.

nothing – ложное значение, подобное пустой строке, используемое в tal:replace или tal:content, чтобы удалять элемент или его содержание. Если атрибут задан значением nothing, то он будет удален из тега (или не включен). Пустая строка в этом случае должна включить тег с пустым значением, как в alt="".

default – специальное значение, которое не изменяет текст тега шаблона, когда использовано в tal:replace, tal:content или tal:attributes.

options – необязательные ключевые аргументы. Если присутствуют в вызове, то будут переданы в шаблон. Когда шаблон требуют клиенты сети, то параметры всегда отсутствуют. Такие аргументы доступны только тогда, когда шаблон назван из программы на Питоне или другим аналогичным способом. Например, когда шаблон **tt** вызван выражением Питона **tt(foo=1)** , то внутри шаблона выражение **options/foo** будет равняться 1.

attrs – Словарь атрибутов текущего тега в шаблоне. Ключи являются именами атрибутов, а данные являются значениями атрибутов в шаблоне.

container – контейнер (обычно папка) в котором содержится шаблон. Используется для получения объектов Zope относительно папки шаблона.

context – контейнер (обычно папка) из которого вызван шаблон. Используется для получения объектов Zope относительно текущей папки.

modules – набор модулей Питона доступных для шаблона.

Строковые выражения позволяют комбинировать текст с указанием пути к значению. Выражение с использованием синтаксиса пути должно следовать за знаком доллара ($), например:

```
"string:Just text. There's no path here."
"string:copyright $year by Fred Flintstone."
```

Во второй строке используется переменная **year** для вставки в окружающий текст. Если вставляемое выражение пути имеет более чем одну часть (если оно содержит слэш) или должно быть выделено из текста, внутри которого оно записано, то оно должно быть окружено скобами ({}). Например:

```
"string:Three ${vegetable}s, please."
"string:Path name is ${request/PATH_INFO}!"
```

Так как текст помещается внутри атрибута, то можно включить саму двойную кавычку, используя ее HTML код «"». Аналогично, знак доллара используется для выражения пути, но его можно записать как два знака доллара ($$).

Например:

```
"string:Please pay $${$dollars}"
"string:She said, &quot;Hello world.&quot;"
```

Некоторые сложные строки с использованием форматирующих операций (как, например, поиск/замена или изменение регистра) не могут выполняться в строковых выражениях. Для этого нужно использовать выражения Питона.

Выражения пути ссылаются на объекты способом, похожим на путь в URL. Путь описывает траекторию перемещения с объекта на объект. Все пути начинаются с некоторого предопределенного в системе или известного в текущем контексте объекта (например, встроенная переменная, переменная повторения, или определяемая в шаблоне переменная) и задают перемещение от нее до желаемого объекта. Приведем некоторые примеры выражения путей:

`context/title` – атрибут `title` папки, откуда вызван шаблон;

`container/files/values` – атрибут `values` папки `files`, где размещен шаблон;

`container/master.html/macros/header` – макрос `header` из шаблона `master.html`;

request/form/address – элемент с именем address формы;
view/standard_look.html – имя файла из папки browser приложения.

В выражениях пути для объекта можно просмотреть его подобъекты, включая атрибуты и методы. Вы можете использовать заимствование в выражениях пути. Zope организует объектное заимствование в выражениях пути точно так же, как и доступ к объектам с использованием URL. Клиент должен иметь требуемые разрешения на доступ ко всем объектам, перечисленным в выражении пути. Мы можем иметь доступ к видам в выражениях пути, используя операнд @@viewname. Например, выражение context/@@standard_macros ищет вид, обеспечивающий стандартные макросы.

В том случае, когда указанный объект не существует, можно задать альтернативное значение для его использования в операторе. Альтернативы задаются операцией «|», например: `<h4 tal:content="request/form/x | context/x"> Header </h4>`. Здесь при отсутствии объекта «x» в данных из формы будет использован локальный экземпляр «x» из контекста. Для предотвращения ошибок полезно использовать в качестве альтернативы объекты default и nothing.

В условных выражениях можно использовать операцию отрицания, например:

```
<p tal:condition="not:context/values">
    There are no contained objects.
</p>
```

Для предотвращения интерпретации объекта, например, документа или метода, используется операция «nocall». В этом случае будет использован сам объект, а не его содержимое, например:

```
<span tal:define="doc nocall:container/aDoc"
   tal:content="string:${doc/getId}: ${doc/title}">
     Id: Title
</span>
```

Для проверки существования объекта используется операция "**exists**», например:

```
<h4 tal:condition="exists:request/form/errmsg"
   tal:content="request/form/errmsg">Error!</h4>
```

Наибольшие возможности для исполнения намерений автора предоставляют выражения на языке Питон. Такому выражению предшествует ключ «python». Выражение возвращает вычисленный объект Питона. Это может быть результат операции сравнения (==, !=, >=, <= и другие), вызова атрибута или метода, вычисления произвольных выражений. Особенно полезны выражения на Питоне для доступа к данным сложной структуры: списки, словари, пространства имен, файлы и другие, например, содержимое текстового файла «fileofword» заменит содержимое тега:

```
<span tal:replace="python:context.get('fileofword').data">
   Словарь слов
</span>
```

При разработке приложений со сложной логикой полезно использовать модули, входящие в состав Zope3. Ниже приведен пример использования метода

«join» из стандартного модуля «string» и вычисления синуса функцией модуля «math»:

```
<div tal:define="global mstring modules/string;
                 global mmath modules/math"  >
    <span tal:replace="python:mstring.join('123456789', ':')">
          join()
    </span>
    <span tal:replace="python:mmath.sin(12)"> sin() </span>
</div>
```

Для поиска корневой папки сайта (ниже использована переменная root) можно предложить следующий фрагмент шаблона:

```
    <div  tal:define="global root  context" >
      <tal:loop tal:repeat = "n python:range(20)">
         <div tal:condition="root/__parent__" >
           <div tal:define="global root  root/__parent__" />
        </div>
      </tal:loop>
      <br>
    </div>
```

3.3 Макросы

Макросы позволяют многократно использовать фрагмент некоторого шаблона на разных страницах приложения. Макросы определяют фрагмент страницы, который может быть повторно использован на других страницах приложения. Макрос может быть целой страницей, или фрагментом страницы, например, заголовок или нижний колонтитул. После определения одного или более макросов в одном страничном шаблоне, их можно использовать в других шаблонах.

Вы можете определить макрос атрибутами тега подобно операторам языка TAL. Теги атрибутов макроса пишутся на языке METAL (Macro Expansion Tag Attribute Language). Вот пример определения макроса:

```
<p metal:define-macro="copyright">
  Copyright 2005, <em>Суханов В.</em> УГТУ-УПИ
</p>
```

и его использования

```
<b metal:use-macro="container/ZPTmodul/macros/copyright">
      Copyright
</b>
```

В этом шаблоне, элемент «b» при генерации страницы Zope будет полностью заменен на текст макроса:

```
<p>
 <b>
   Copyright 2006, <em> В.Суханов УГТУ-УПИ</em>
 </b>
</p>
```

При изменении текста макроса все макровызовы в будущем будут использовать его новое определение и учтут все изменения при генерации страниц. Это дает удобный механизм для унификации оформления интерфейсов с пользователями, называемый в Zope обличьем (skin).

Имя макроса должно быть уникальным в пределах страничного шаблона, в котором он определен. Можно определить в шаблоне любое число макросов, но все они должны иметь разные имена. Обычно ссылка на макрос в утверждении metal:use-macro выполняется с использованием пути. Тем не менее, можно использовать любой другой тип выражения, который возвращает макрос, например:

```
<p metal:use-macro="python:context.getMacro()">
   Текст заменится динамически определяемым макросом, который
   задан сценарием getMacro.
</p>
```

Специальное имя макроса «default» в metal:use-macro="default" приводит к копированию исходного тела макровызова без какой-либо замены по аналогии с утверждениями «`tal:content`» и «`tal:replace`».

При использовании макросов следует учитывать порядок обработки шаблона. Первыми производятся замены макровызовов, и только после этого производится вычисление других утверждений TAL. Это важно в том случае, когда в тексте макрорасширения имеются выражения, содержащие переменные, значение которых может зависеть от контекста, например, при использовании переменных «`context`» и «`container`». Заметим, что макросы **не отрабатываются** визуальными редакторами HTML, что вызывает необходимость их отладки в ZMI.

Использование слотов позволяет динамически вычислять тело макроса при его интерпретации. Макросы позволяют модифицировать макрорасширения за счет слотов, объявляемых в макроопределениях. Слот – часть тела макроса, добавляемая в макровызове, играет роль параметра в методах Питона. Рассмотрим пример.

```
<div metal:define-macro="sidebar">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Products</a></li>
  </ul>
  <span metal:define-slot="additional_info"></span>
</div>
```

Макрос «sidebar» кроме ссылок имеет заключительный слот «additional_info». При использовании этого макроса, например:

```
<p metal:use-macro="container/master.html/macros/sidebar">
 <b metal:fill-slot="additional_info">
  Make sure to check out our <a href="/specials">specials</a>.
 </b>
</p>
```

С учетом места и значения слота макровызов будет заменен на следующий фрагмент:

```
<div>
  Links
```

```
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Products</a></li>
  </ul>
  <b>
   Make sure to check out our <a href="/specials">specials</a>.
  </b>
</div>
```

3.4 Использование стандартных макросов

При разработке экранных форм для взаимодействия с пользователем безусловно полезным инструментом являются стандартные макросы системы, позволяющие выполнять много разных функций. Определения макросов стандартных обличий следует искать в папках zope\app\basicskin и zope\app\rotterdam. Приведем пример их использования в ZPT странице с именем «index.html», по умолчанию являющейся начальной страницей сайта. Для придания нужного обличья окнам используется шаблон с именем «macros», расположенный в папке «templates» сайта. Предлагаем читателю внимательно сопоставить вызовы и определения макросов и слотов. Используемые в примере рисунки, таблицы стилей и скрипты решающего значения не имеют и могут быть произвольными, например, взятые из какого либо приложения. Все они должны быть помещены в папку «resource». Имена папок тоже значения не имеют и могут быть заменены на произвольные.

Содержимое страницы «**index.htm**l» имеет вид.

```
<html  metal:use-macro="container/templates/macros/macros/page"
      i18n:domain="mydomain" >
   <tal metal:fill-slot="title">
    <!-- Загрузка java скриптов для работы навигатора -->
    <tal metal:use-macro=
"context/@@standard_macros/navigation_tree_js" />
   <title i18n:translate="">Окно для экспериментов</title>
 </tal>
<head metal:use-macro="context/@@standard_macros/head" />
 <div metal:use-macro= "context/@@standard_macros/logged_user" />
   <!-- Левая колонка -->
   <div metal:fill-slot="column-left">

    <!-- Макрос для навигационной панели -->
    <metal:tree use-macro=
         "context/@@standard_macros/navigation_tree_box" />

    <!-- Показать меню добавляемых объектов -->
    <div class="box" id="commonTasks"
         tal:define="view context/@@commonTasks|nothing"
         tal:condition="view/strip|nothing">
      <h4 i18n:translate="">Add:</h4>
      <div class="body">
        <span tal:replace="structure view" />
      </div>
    </div>
```

```
    <!-- Ссылки на объекты текущей папки -->
    <br>Goto ...<br>
    <tal:for tal:repeat="item context/values">
      <a  tal:attributes="href item/@@absolute_url"
          i18n:translate="">
          <span tal:content="python:item.__name__" />
      </a> <br>
    </tal:for>
   </div> <!-- Левая колонка -->
   <!-- Правая колонка -->
   <div>
    <h1 metal:fill-slot="content-header"
      i18n:translate="">Welcome to my server</h1> <br>

    <div metal:fill-slot="body">
      <!-- Стандартное меню горизонтальных закладок -->
      <div metal:use-macro=
           "context/@@standard_macros/zmi_tabs">Tabs </div>
    </div>
   </div>  <!-- Правая колонка -->

  <!-- Макрос для стандартного окончания страницы -->
  <div metal:use-macro="context/@@standard_macros/footer">
     Footer </div>

</html>
```

Содержимое шаблона с макросами «**template/macros**»

```
<metal:block define-macro="page" i18n:domain="mydomain">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <!-- Стандартные установки ресурсов -->
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
    <link rel="icon" type="image/png" href="favicon.ico"
          tal:attributes="href
string:${context/resource/favicon.ico/@@absolute_url}"/>
    <link rel="stylesheet" type="text/css" href="layout.css"
          tal:attributes="href
string:${context/resource/layout.css/@@absolute_url}"  />
    <link rel="stylesheet" type="text/css" href="style.css"
          tal:attributes="href
string:${context/resource/style.css/@@absolute_url}"  />
    <script type="text/javascript" src="myjs.js"
            tal:attributes="src string:${context/resource/
myjs.js/@@absolute_url}"  >
    </script>
    <metal:block define-slot="title">
```

```
      <!-- Определение слота для заголовка страницы -->
      <title i18n:translate="">My site</title>
    </metal:block>
  </head>
  <body>
    <div id="header">
      <div class="logo">
        <a href=""
          tal:define="url context/@@absolute_url"
          tal:attributes="href url">
          <img src="logo.png" alt=" My site "
i18n:attributes="alt"
            tal:attributes="src
string:${context/resource/logo.png/@@absolute_url}"/>
        </a>
      </div>
      <div class="tools">
        <div class="breadcrumbs">
      <!-- Нарезка из ссылок в верхней части страницы -->
        <tal:for tal:repeat="breadcrumb
context/@@absolute_url/breadcrumbs">
          <a tal:condition="repeat/breadcrumb/start"
             tal:attributes="href breadcrumb/url"
             i18n:translate="">top</a>
          <tal:if
condition="not:repeat/breadcrumb/start">&#187;</tal:if>
          <a tal:condition="not:repeat/breadcrumb/start"
             tal:attributes="href breadcrumb/url"

tal:content="breadcrumb/name|string:top">Breadcrumb</a>
        </tal:for>
        </div>
        <div
      <!-- Параметры регистрации пользователя -->
          tal:define="user request/principal;
                      url context/@@absolute_url"
          tal:on-error="nothing">

          <tal:if condition="not:user/authenticated">
            <a id="tools-login" i18n:translate=""
              tal:attributes="href string:${url}/@@login.html">Log
in</a>
          </tal:if>

          <tal:if condition="user/authenticated"
                  define="person user">
            <div metal:define-macro="logged_user">

              <tal:span i18n:translate="">User:</tal:span>

              <tal:if condition="not:person">
                <span id="tools-zmi-principal" i18n:translate=""
```

```
              tal:content="user/title">ZMI Principal</span>
          </tal:if>

          <tal:if condition="person">
            <a id="tools-principal" i18n:translate=""
              tal:attributes="href person/@@absolute_url"
              tal:content="user/title">Ais Person</a> |
            <a id="tools-logout" i18n:translate=""
              tal:attributes="href
string:${url}/@@logout.html"
              >Log Out</a>
          </tal:if>
        </div>
      </tal:if>
    </div>   <!-- user  -->
  </div>     <!-- tools  -->
</div>       <!-- header  -->
 <br>
 <TABLE  CELLSPACING=1 BORDERCOLOR="#000000" CELLPADDING=4
WIDTH="100%">
  <tr>
   <TD WIDTH="24%" VALIGN="TOP">
    <!-- Левая полоса страницы как колонка таблицы -->
    <metal:block define-slot="column-left" />
   </td>
   <td  WIDTH="74%" VALIGN="TOP">
    <div id="content-header">
      <!-- Заголовок правой колонки таблицы -->
      <metal:block define-slot="content-header"/>
    </div>
    <div>
      <!-- Содержимое правой колонки таблицы -->
      <metal:block define-slot="body"/>
    </div>
    <div class="clearer"> </div>
   </td>
  </tr>
</table>
<div id="footer" i18n:translate="">
  <p>
   My Server
  </p>
  <!-- Слот - подвал страницы -->
  <metal:block define-slot="footer"/>
</div>
 </body>
</html>
</metal:block>
```

4 Работа с базами данных

Разработка приложений редко обходится без использования массовых данных, хранимых в таблицах реляционных СУБД. Для этого в состав утилит Zope3 входят адаптеры баз данных к различным типам СУБД. Среди свободно распространяемых СУБД заметное место занимает MySQL, на примере которой можно проиллюстрировать приемы работы с базами данных в Zope3.

Вначале необходимо установить поддержку MySQL для Питона соответствующей Zope3 версии (сейчас 2.4). Для этого необходимо развернуть содержимое папки MySQLdb из архива "MySQL-python.exe-1.2.0.win32-py2.4.zip" в папку "C:\Python24\Lib\site-packages\MySQLdb". Следующим шагом нужно развернуть содержимое папки "mysqldbda" архива "mysqldbda-1.0.0.tgz" в папку "C:\<Zope3Instance>\lib\python\mysqldbda", где имя папки <Zope3Instance> определяет резиденцию экземпляра сайта, задаваемую при установке Zope3.

В существующей версии Zope 3.1.0 замечена неточность, приводящая к диагностике ошибок при работе с адаптером MySQL DA. Для ее устранения в интернет ресурсе «http://www.w3.org/TR/html4/loose.dtd [Zope3-dev] Suspected bug in package mysqldbda-1.0.0.tgz for Zope3» приведено изменение в тексте адаптера в файле «adaptor.py». Строку

```
return string.decode(self.encoding)
```

следует заменить строчкой

```
return string
```

Далее нужно скопировать или перенести файл "mysqldbda-configure.zcml" из папки "C:\<Zope3Instance>\lib\python\mysqldbda", в папку "C:\<Zope3Instance>\etc\package-includes". После этого необходимо перезапустить Zope.

Ниже предполагается, что у Вас уже есть таблица «modules» в базе данных «test» СУБД MySQL [1], и Вы имеете свои Login и Password. Чтобы получить доступ к таблицам нужно создать соединение с базой данных. Для этого нужно войти менеджером в ZMI, выбрать в навигационном окне ссылку "[top]/++etc++site" и в появившемся окне «Site Management» в разделе «Database Adapter» нажать кнопку «Add». В предложенной форме задать имя и тип адаптера СУБД и нажать кнопку «Add». После этого Вам будет предложена форма для задания параметров соединения. В первом окне нужно ввести строку следующего содержания

dbi://username:password@host:port/dbname;param1=value...

где username:password – имя и пароль пользователя СУБД MySQL;

host – имя сервера;

port – номер порта, обычно 3306;

dbname – имя базы данных, с которой будет производиться работа.

Примером может быть строка:

dbi://admin:ddd@localhost:3306/test

Во втором окне выбирается кодировка символов строк, необходимая для правильного отображения данных.

После задания соединения с СУБД можно приступить к созданию скриптов «SQL Script» из бокового меню, находясь в папке своего приложения. Далее выбором из списка задается имя соединения, записывается текст запроса на языке SQL и перечисляются параметры запроса, если они предусмотрены в его тексте. Здесь же можно протестировать запрос и просмотреть его результаты.

Например, имея таблицу «modules», можно создать два запроса: для получения шапки таблицы и данных самой таблицы. Первый запрос будет иметь имя «headTbl» и текст «`show columns from modules;`», второй имя «modules» и текст «`select * from modules;`».

4.1 Использование DTML страниц

Для тестирования работы всех компонентов создадим DTML страницу следующего содержания:

```
1)   <html>
2)   <body>
3)   <table border=1>
4)    <tr>
5)     <dtml-in "get('headTbl')()">
6)      <th>
7)       <dtml-var sequence-var-Field>
8)      </th>
9)     </dtml-in>
10)   </tr>
11)   <dtml-in "get('modules')()">
12)    <tr>
13)     <td>
14)      <dtml-var sequence-var-name>
15)     </td>
16)     <td>
17)      <dtml-var sequence-var-last_modification>
18)     </td>
19)     <td>
20)      <dtml-var sequence-var-size >
21)     </td>
22)     <td>
23)      <dtml-var sequence-var-lines_of_code >
24)     </td>
25)     <td>
26)      <dtml-var sequence-var-path >
27)     </td>
28)    </tr>
29)   </dtml-in>
30)  </table>
31)  </body>
32)  </html>
```

Строки 5 – 9 организуют цикл по полям заголовка таблицы и помещают в документ их имена с использованием оператора в строке 7. Это специальная форма тега для работы с элементами набора, имеющими атрибуты, в нашем случае – поле «Field» таблицы колонок (просмотрите результаты тестирования SQL-скрипта headTbl).

Строки 11 – 29 организуют цикл по записям таблицы «modules» и отображение их значений в документе с использованием тега <dtml-var sequence-var-*поле*>.

Для добавления новых записей в таблицу модулей создадим SQL-скрипт с именем «insRec» со строкой параметров «modName modTime modSize modLines modPath» и кодом

```
insert into modules values(
'<dtml-var modName>',
'<dtml-var modTime>',
'<dtml-var modSize>',
'<dtml-var modLines>',
'<dtml-var modPath>');
```

Для получения доступа к значениям параметров скрипта используется тег «dtml-var». В остальном текст запроса соответствует грамматике языка SQL.

DTML страница для тестирования вставки новых записей имеет вид:

```
<html>
<body>
<dtml-if "REQUEST.form">
 <dtml-comment>
   Значение элементов ввода формы находятся в словаре form
   объекта REQUEST. Ключами словаря являются значения
   атрибута name требуемого элемента (смотри ниже)
 </dtml-comment>
 <dtml-let name="REQUEST.form['name']"
           time="REQUEST.form['time']"
           size="REQUEST.form['size']"
           lines="REQUEST.form['lines']"
           path="REQUEST.form['path']">
   <dtml-call "get('insRec')(modName=name, modTime=time,
modSize=size, modLines=lines, modPath=path)">
</dtml-let>
</dtml-if>

<form method="post">
<table border=1>
 <tr>
  <dtml-in "get('headTbl')()">
   <th>
    <dtml-var sequence-var-Field>
   </th>
  </dtml-in>
 </tr>
 <tr>
  <td> <input type="text" name="name" value=""> </td>
  <td> <input type="text" name="time" value=""> </td>
  <td> <input type="text" name="size" value=""> </td>
  <td> <input type="text" name="lines" value=""> </td>
  <td> <input type="text" name="path" value=""> </td>
 </tr>
 <tr>
  <td>
   <input type="submit" name="AddRecord" value="Добавить модуль">
```

```
  </td>
 </tr>
</table>
</form>
</body>
</html>
```

Аналогично выполняется удаление или замена записей, для которых скрипт будет содержать соответствующие операторы языка SQL, а интерфейсы с пользователем можно организовать таким же способом.

4.2 Использование ZPT страниц

Для получения доступа к базам данных из шаблонов страниц существуют аналоги средствам языка DTML, использованным ранее, но уже в языке TAL. Например, формирование списка имен модулей из таблицы «modules» выполняется следующим фрагментом:

```
<li tal:repeat="module container/modules">
  <b tal:content="module/name" />
</li>
```

где modules – имя того же самого SQL скрипта для запроса данных из таблицы «modules» базы данных «test» СУБД MySQL;

module – имя переменной цикла, где хранится ссылка на очередную запись таблицы;

name – имя поля в таблице модулей.

Шаблон для приведенной выше задачи просмотра и добавления записей в таблицу модулей с использованием средств языка TAL может быть следующим:

```
1)  <html>
2)  <body>
3)  <div tal:condition="request/form">
4)   <div tal:define="name python:request.form['name'];
5)         time python:request.form['time'];
6)         size python:request.form['size'];
7)         lines python:request.form['lines'];
8)         path python:request.form['path']"
9)       tal:content="python:context.get('insRec')(modName=name,
            modTime=time, modSize=size, modLines=lines,
            modPath=path)">
10)    Record is added
11)  </div>
12) </div>
13) <table border=1>
14)  <tr>
15)   <th tal:repeat="field container/headTbl" >
16)    <span tal:replace="field/Field" />
17)   </th>
18)  </tr>
19)  <tr tal:repeat="module container/modules">
20)   <td> <span tal:replace="module/name" /> </td>
21)   <td> <span tal:replace="module/last_modification" /> </td>
22)   <td> <span tal:replace="module/size" /> </td>
```

```
23)    <td> <span tal:replace="module/lines_of_code" /> </td>
24)    <td> <span tal:replace="module/path" /> </td>
25)  </tr>
26) </table>
27) <br>
28) <!-- Форма -->
29) <table border=1>
30)  <tr>
31)   <th tal:repeat="field container/headTbl" >
32)    <span tal:replace="field/Field" />
33)   </th>
34)  </tr>
35)  <form method="post">
36)  <tr>
37)   <td> <input type="text" name="name" value=""> </td>
38)   <td> <input type="text" name="time" value=""> </td>
39)   <td> <input type="text" name="size" value=""> </td>
40)   <td> <input type="text" name="lines" value=""> </td>
41)   <td> <input type="text" name="path" value=""> </td>
42)  </tr>
43)  <tr>
44)   <td> <input type="submit" name="AddRecord"
45)      value="Добавить модуль"> </td>
46)  </tr>
47)  </form>
48) </table>
49) </body>
50) </html>
```

Строка 3 проверяет наличие данных из формы в запросе клиента.

Строки 4 – 8 создают локальные переменные тега «div» для их использования в вызове SQL скрипта «insRec» добавления записей в строке 9.

Строки 15 – 17 создают заголовок таблицы с использованием цикла по записям, возвращаемым запросом «container/headTbl».

Строки 19 – 25 формируют строки таблицы с использованием цикла по записям, возвращаемым запросом «container/modules».

Строки 29 – 48 создают форму для ввода значений полей записи добавляемого модуля.

Следует обратить внимание на более компактную запись операций и их параметров при использовании страничных шаблонов по сравнению с DTML.

5 Основы разработки приложений в Zope3

Zope3 основан на использовании компонент (есть некоторая аналогия с архитектурой компонент в Delphi, с классами в Smalltalk и в других языках), которым соответствует класс в языке Питон. Класс сопровождается дополнительной информацией о правилах использования компонента, сосредоточенной в различных файлах проекта (интерфейсы, конфигурации, адаптеры видов, переводы текстов на разные языки, тесты и др.)

Среда Zope3 является одновременно сетевым сервером – публикатором хранимых на сервере в объектной базе данных экземпляров компонент и средством разработки сетевых приложений. Сетевые приложения могут разрабатываться как в режиме TTW (через сеть) с использованием стандартного ZMI, так и в режиме создания файлов проекта средствами файловой системы ЭВМ. Последний режим обладает максимальными возможностями выражения намерений авторов и реализации функциональности.

Разработать пакет компонентов для создания приложений в Zope3 (рисунок 3)

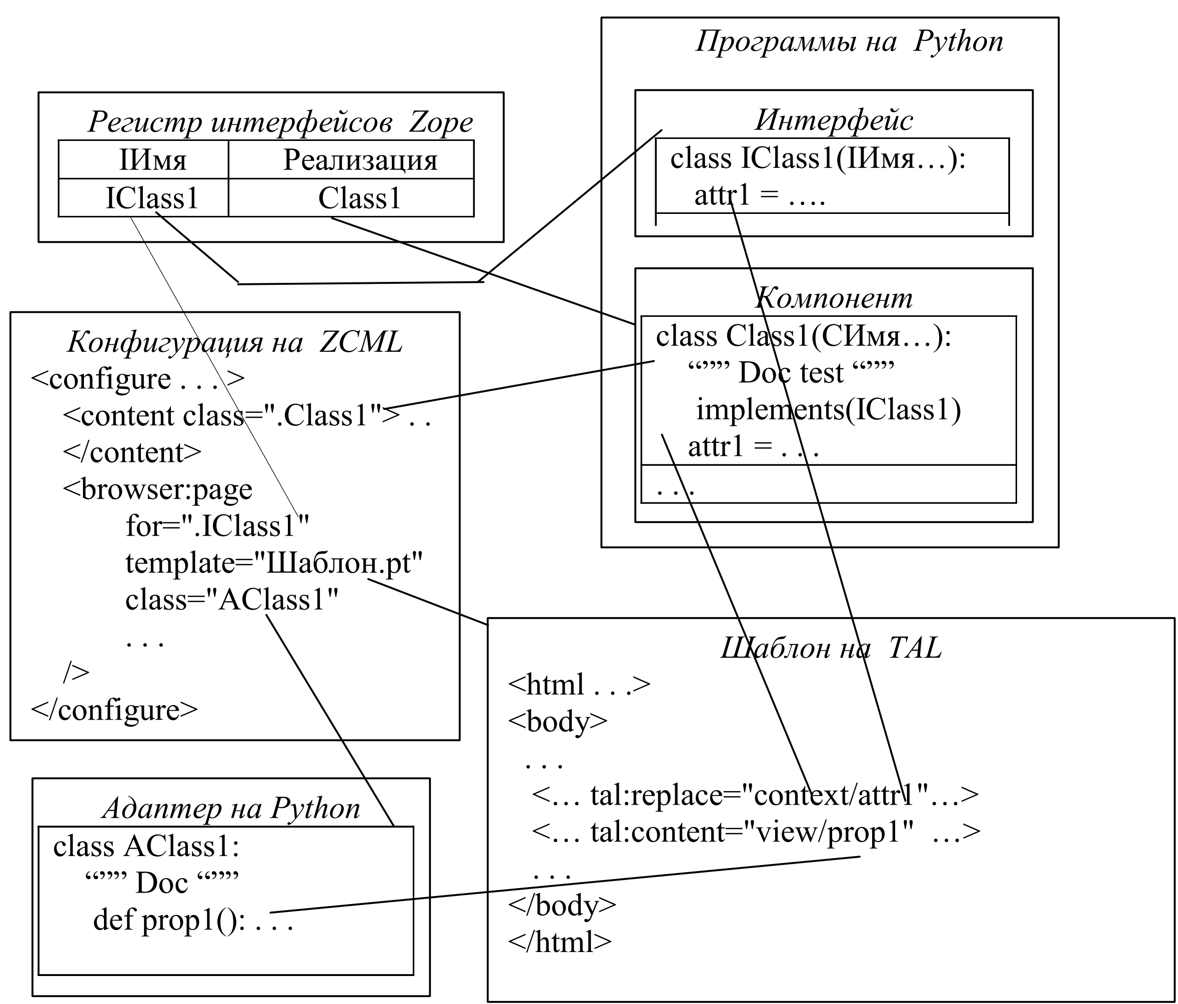

Рисунок 3 – Структура приложения в архитектуре Zope 3

с использованием программирования на Питоне это [3]:

- Описать интерфейсы компонент, определяющие состав данных – схему для внутреннего представления объектов в хранилищах среды Zope. Поскольку среда берет на себя заботу о создании экземпляров компонент по заказу менеджера сайта или уполномоченного пользователя при выборе им команды из меню «Добавить», то интерфейсы используются и при автоматической генерации оконных форм для запроса значений атрибутов и выступают здесь в роли схем данных, ожидаемых от менеджера при вводе. Здесь же задаются отношения вложенности контейнеров и компонент.
- Описать классы компонент, используя подходящие родительские классы и множественное наследование.
- Задать ограничения и правила использования компонент в конфигурационных файлах (права доступа, допустимость аннотаций и другие).
- Написать адаптеры – согласовать внутреннее представление данных компонент и способ их отображения при просмотре пользователем, написать шаблоны страниц и связать все вместе в конфигурационном файле приложения.
- Зарегистрировать пакет в системе.

При разработке пакетов необходимо придерживаться рекомендаций по стилю оформления текстов и размещению информации в папках сервера. Резиденцией всех файлов является папка с выбранным программистом именем с использованием латинских символов, расположенная в директории C:\<Zope3Inst>\lib\python, например, **buddydemo.** Все остальные файлы и папки размещаются в этой резиденции. По требованию языка Питон, чтобы содержимое папки рассматривалось как пакет, в ней должен находиться специальный файл инициализации с именем **__init__.py.** Обычно для разработки большинства приложений этот файл имеет пустое содержимое или только строки комментариев, например:

```
#
# init package
#
```

5.1 Интерфейсы

Современные технологии разработки сложного программного обеспечения, такие как CORBA, COM, Java и другие, для сегментации кода и организации взаимодействия участников проекта и инструментальных средств давно используют объявления интерфейсов. Для написания текста объявлений обычно используют специализированные языки IDL (Interface Definition Language). Отделение объявления интерфейсов от текста описания соответствующих классов объектов позволяет группе программистов независимо вести автономную разработку различных подсистем приложения.

Интерфейс часто рассматривают как соглашение (контракт) между поставщиком и потребителем некоторого программного сервиса. Одна сторона обязуется обеспечить предоставление корректных данных и правильное исполнение объявленных функций, другая сторона обязуется правильно ими пользоваться в соответствии с опубликованной спецификацией.

В отличии от других языков, в языке Питон для объявления интерфейсов используется тот же синтаксис, что и для написания обычных программ. Поэтому нет необходимости в специализированном языке IDL. Дополнительные средства введены за счет самодостаточности объектно-ориентированного языка Питон, путем подключения необходимых модулей, использования их классов, атрибутов и методов. В Zope3 все интерфейсы пакета описываются всегда в файлах с именем **interfaces.py.** Роль интерфейсов в Zope3 не ограничивается только синтаксическими соглашениями. Это центральное понятие, работающее в различных аспектах при разработке программ компонент на языке Питон.

Приведем пример объявления интерфейсов в файле **buddydemo/interfaces.py** [4]. В приведенном примере строки пронумерованы для удобства их комментирования. Реальные файлы должны починяться правилам записи соответствующих конструкций языка Питон без нумерации строк.

```
1)  import zope.interface
2)  import re
3)  from zope.schema import Text, TextLine
4)  from zope.i18nmessageid import MessageIDFactory
5)  _ = MessageIDFactory("buddydemo")
6)  class IBuddy(zope.interface.Interface):
7)    """Provides access to basic buddy information"""
8)    first = TextLine(title=_("First name"))
9)    last  = TextLine(title=_("Last name"))
10)   email = TextLine(title=_("Electronic mail address"))
11)   address = Text(title=_("Postal address"))
12)   postal_code = TextLine(title=_("Postal code"),
13)     constraint=re.compile("\d{5,5}(-\d{4,4})?$").match)
14)   def name():
15)     """ Gets the buddy name.
16)         The buddy name is the first and last name
17)     """
```

Строка 1 импортирует базовый пакет работы с интерфейсами, в том числе, содержащий класс `Interface`. Любой объект, который имеет этот мета-класс своим предком, является интерфейсом, а не обычным классом. Строка 2 импортирует базовый пакет работы с регулярными выражениями, которые используются для формирования ограничений на вводимые данные в строке 13. Строка 3 импортирует из пакета **schema** объявления классов **Text** и **TextLine**, используемые для описания характера поступающих от уполномоченного на то пользователя данных о приятелях. Строка 4 импортирует из базового пакета **zope.i18nmessageid**, обеспечивающего автоматический перевод текстов, отображаемых на дисплее клиента, на соответствующий язык (например, русский), метода формирования фабрики переводчиков **MessageIDFactory**. Для работы переводчика программист должен будет разработать необходимые файлы, которые будут описаны позднее.

В строке 5 принятой соглашениями Zope3 переменной с именем «_» присваивается экземпляр переводчика для работы с доменом, имя которого в нашем случае совпадает с именем разрабатываемого пакета **buddydemo**.

Начиная со строки 6, помещается описание класса **IBuddy**, являющимся потомком класса **Interface** из пакета **zope.interface**. Текст описания родительского класса с дополнительным именем **Interface** можно посмотреть в файле C:\Python24\Lib\site-packages\zope\interface\interface.py. Это имя объявлено в файле

zope\interface\init.py как атрибут пакета. Все потомки этого класса рассматриваются как интерфейсы, и по соглашениям Zope их имена должны начинаться с символа «I». Строка 7 является встроенной в текст документацией класса, содержимое которой доступно в период исполнения приложения через ссылку «IBuddy.__doc__».

Строки 8, 9, 10 и 11 определяют имена и значения атрибутов класса, являющихся полями в формах для ввода информации от пользователя об экземпляре добавляемого объекта класса, реализующего этот интерфейс. Это так называемая схема запроса данных, которая используется сервером для автоматической генерации формы на языке HTML, отправляемой браузеру клиента. Тип вводимой информации определяется вызываемым конструктором соответствующего класса **Text** или **TextLine** с параметрами, определяющими оформление полей ввода. С полным перечнем типов полей схемы можно познакомиться в разделе 7.2 или в материале Zope3Book [4].

Строки 14 – 17 объявляют метод **name**, который в будущем должен реализовать класс, для получения сцепления фамилии и имени приятеля в единую строку.

Интерфейсы описаны в пакете zope\interface. Пакет экспортирует непосредственно два класса, `Interface` и `Attribute`и несколько вспомогательных методов. Класс прародитель `Interface` используется для объявления частных интерфейсов, необходимых для разработки компонент. Пакет содержит несколько общих методов:

`declarations` обеспечивает утилиты для объявления интерфейсов экземпляров объектов и большее разнообразие полезных утилит, которые помогают управлять объявленными интерфейсами.

- document – обеспечивает утилиты для описания интерфейсов текстами.
- exceptions – имеет исключения специфические для интерфейсов
- interfaces – содержит список всех общих интерфейсов для этого пакета.
- verify – содержит утилиту для проверки реализации интерфейсов.

Вы можете использовать интерфейсы двумя способами:

I) В объявлениях того, что ваш объект реализует интерфейсы.

Есть несколько путей объявления, что объект реализует интерфейс:

- Вызовом метода zope.interface.implements в определении класса этого объекта.
- Вызовом метода zope.interfaces.directlyProvides для конкретного экземпляра объекта.
- Вызовом метода 'zope.interface.classImplements для утверждения, что все экземпляры класса реализуют интерфейс. Например:

```
from zope.interface import classImplements
classImplements(some_class, some_interface)
```

Этот способ полезен тогда, когда не нужно модифицировать исходный код класса. Заметим, что этот вызов не влияет на определение интерфейсов, принадлежащих классу, а только на реализуемые интерфейсы экземпляров указанного класса.

II) В запросах мета-данных интерфейса с использованием приведенных ниже методов.

Метод providedBy(object) тестирует реализацию интерфейса объектом. Возвращает истину, если объект реализует либо сам интерфейс, для которого вызван метод, либо его потомок. Например,

```
myBuddy = Buddy()
IBuddy.providedBy(myBuddy)
```

Метод implementedBy(class) тестирует реализацию интерфейса классом. Возвращает истину, если класс реализует либо сам интерфейс, для которого вызван метод, либо его потомок. Например, IBuddy.implementedBy(Buddy)

Метод names(all=False) получает имена атрибутов интерфейса. Возвращает список имен атрибутов и методов, включенных в определение интерфейса. Обычно включаются только непосредственно определенные атрибуты. Если позиционный аргумент True, то включаются атрибуты и методы, определенные в родительском классе. Например: IBuddy.names().

Метод namesAndDescriptions(all=False) получает имена атрибутов интерфейса и их описания. Формат представления результата – список пар вида (имя, значение).

Метод __getitem__(name) получает описание для имени атрибута. Если поименованный атрибут не определен, то возбуждается исключительная ситуация KeyError. Например: IBuddy.__getitem__('first').__name__ ➔ 'first'.

Метод direct(name) получает описание для имени атрибута, если оно было определено в интерфейсе, иначе возвращает None, например: IBuddy.direct('first').__name__ ➔ 'first'.

Метод validateInvariants(obj, errors=None) Проверить инварианты объекта. Если параметр errors=None, то поднимается первая ошибка; если errors является списком, добавляются все ошибки к списку, затем поднимается исключение Invalid с ошибками как первый элемент кортежа "args". Например: IBuddy.validateInvariants(buddy).

Метод __contains__(name) проверяет, содержится ли указанное имя в описании интерфейса, например: IBuddy.__contains__('first') ➔ True..

5.2 Объявление классов

Для работы с атрибутами и методами, предусмотренными в интерфейсе, необходимо описать класс, экземпляры которого будут реализовывать заявленный интерфейс. Обычно объявление классов помещается в файл с расширением «py», имя которого выбирает программист. Часто это имя совпадает с именем одного (обычно главного) класса пакета. Ниже приведен текст файла модуля **buddy.py**, который должен находиться в папке пакета и реализовывать приведенный ранее интерфейс **IBuddy**.

```
1)  import persistent
2)  import zope
3)  from buddydemo.interfaces import IBuddy
4)  class Buddy(persistent.Persistent):
5)  """Buddy information"""
6)    zope.interface.implements(IBuddy)
7)    def __init__(self, first='', last='', email='',
8)           address='', pc=''):
```

```
9)        self.first = first
10)       self.last = last
11)       self.email = email
12)       self.address = address
13)       self.postal_code = pc
14)     def name(self):
15)       return "%s %s" % (self.first, self.last)
```

Строки 1, 2, 3 производят импорт в адресное пространство модуля необходимых имен модулей (persistent, zope) и класса **IBuddy**. Строки с 4 – 15 объявляют класс **Buddy**, наследник класса **Persistent** из модуля **persistent**. Родительский класс определяет атрибуты и методы объектов, которые могут храниться в объектной базе данных ZODB и являются материалом для публикации по запросу клиента. Строка 5 – текст документации на метод.

Строка 6 является вызовом метода **implements** модуля zope.interface, сообщающего Zope о том, что класс реализует интерфейс с именем **IBuddy**. Это своеобразная регистрация интерфейса и реализующего его класса. Исполнение этого вызова приводит к присваиванию соответствующих атрибутов класса, которые потом становятся доступными для всех заинтересованных методов. Как отмечалось выше, реализуемые интерфейсы экземплярами объектов могут служить их своеобразными паспортами, проверку которых можно осуществить вызовом метода **IИнтерфейс.providedBy(объект)**, например:

if IBuddy.providedBy(anyBuddy): pass

Строки 7 – 13 объявляют конструктор класса **Buddy**, вызываемый при создании нового экземпляра объекта. Напомним, что создание экземпляра производится через ZMI из меню «Добавить» выбором нужного класса объекта пользователем, у которого есть на это право. Конструктор присваивает атрибутам создаваемого экземпляра значения параметров, передаваемых при вызове из формы, сгенерированной на основе ранее описанной в интерфейсе схемы. Всем параметрам предусмотрены значения по умолчанию, что удобно для автономной отладки текста модуля в интерпретаторе Питона и работы подсистемы тестирования, о которой речь пойдет позже.

Строки 14 – 15 объявляют метод **name** и его тело для сцепления фамилии и имени приятеля в одну строку использованием операции форматирования строк [1].

5.3 Шаблоны страниц для отображения объектов

При добавлении нового экземпляра Zope3 автоматически генерирует форму с необходимыми полями для ввода данных от пользователя. Но при просмотре гостем сайта этого экземпляра «казенная» форма выглядит не совсем эстетично. Поэтому разработчик должен предусмотреть свою форму окна для просмотра содержимого экземпляра объекта. Конечный пользователь вызывает эту форму при выборе из меню закладки «Просмотр», если конечно у него есть на это соответствующие разрешения. Для разработки шаблона страницы можно использовать встроенные в Zope языки DTML [6, 7] и TAL [8, 9]. Учитывая тенденции развития Zope технологий, рекомендуется использовать язык TAL, более приспособленный к инструментам разработки презентационных веб страниц, например, Dreamweaver. В целях лучшей структуризации текстов и различных ресурсов приложения рекомендуется придерживаться единообразного стиля формирования проекта. В частности, все

части приложения, отвечающие за отображение информации у клиента, целесообразно собрать в отдельном пакете с именем “browser”. Для этого создадим папку с таким же именем browser в папке пакета buddydemo, поместив в нее пустой файл __init__.py, описания необходимых интерфейсов, модулей и остальных ресурсов. Приведем и прокомментируем пример оформления презентационного шаблона для нашего примера приложения, написанного на языке TAL. Имя файла может быть произвольным, но расширение рекомендуется задавать «.pt». В нашем примере имя файла выбрано **info.pt**

```
1)   <html metal:use-macro="context/@@standard_macros/page"
2)         i18n:domain=.buddydemo.>
3)   <body>
4)    <div metal:fill-slot="body">
5)     <table>
6)      <caption i18n:translate="">Buddy information</caption>
7)      <tr>
8)       <td i18n:translate="">Name:</td>
9)       <td>
10)       <span tal:replace="context/first">First</span>
11)       <span tal:replace="context/last">Last</span>
12)      </td>
13)     </tr>
14)     <tr>
15)      <td i18n:translate="">Email:</td>
16)      <td tal:content="context/email">foo@bar.com</td>
17)     </tr>
18)     <tr>
19)      <td i18n:translate="">Address:</td>
20)      <td tal:content="context/address">1 First Street</td>
21)     </tr>
22)     <tr>
23)      <td i18n:translate="">Postal code:</td>
24)      <td tal:content="context/postal_code">12345</td>
25)     </tr>
26)    </table>
27)   </div>
28)  </body>
29)  </html>
```

Строки 1, 2 и 29 задают тег html, содержащий дополнительные атрибуты языка TAL, выделяемые в тексте наличием двоеточия за именем атрибута. Атрибут metal: определяет группу средств работы с макросами, а атрибут i18n: определяет группу средств, используемых для автоматического перевода текстов на различные языки будущих пользователей приложения. В частности, предполагается использование макроса context/@@standard_macros/page, задающего стандарт оформления пользовательской страницы (таблица стилей, композиция материала и другие параметры) и домена «buddydemo», где система будет искать переводы текстов.

Строки 3 и 28 задают границы блочного тега body – тела публикуемой страницы. Строки 4 и 27 задают границы блочного тега div –структурная часть тела

публикуемой страницы. Атрибут fill-slot="body" определяет имя заменяемого слота в макросе context/@@standard_macros/page. Строки 5 и 26 задают границы блочного тега table – табличного элемента. Строка 6 задает заголовок таблицы.

Строки 7 и 13 задают границы блочного тега tr – табличный ряд, содержащий две клетки, задаваемые тегами td. В первой клетке размещается текст Name, который необходимо подвергнуть переводу, на что указывает атрибут i18n:translate="". Во второй клетке отображаются два атрибута: фамилия и имя приятеля. Каждый атрибут оформлен как строчный элемент span с оператором tal:replace, задающим правило замены тега span и его содержимого «First» на значение атрибута first экземпляра объекта, для которого производится формирование страницы для публикации. На это указывает ссылка context. Таким образом, при формировании страницы фраза «First» буде заменена на значение атрибута first, где должна быть фамилия приятеля. Аналогично в строке 11 производится замена текста «Last» на значение атрибута context/last – имя приятеля. Необходимо иметь в виду, что оператор replace заменяет сам тег span на значение соответствующего атрибута. Если необходимо оставить в формируемой странице тег, но заменить его содержимое, то используется оператор tal: content, что мы и наблюдаем в строках 16, 20 и 24, служащих для включения в страницу адреса электронной почты, домашнего адреса и почтового индекса.

5.4 Конфигурирование приложения

Конфигурирование позволяет соединить вместе разработанные ранее фрагменты приложения. Для выполнения этой работы используется специальный язык ZCML, основанный на структурах языка XML. Файл конфигурации всегда имеет имя и расширение «**configure.zcml**» и располагается в папке проекта. Для включения проекта в состав действующих доступных пакетов Zope необходимо дополнительно в папку C:\Zope3Inst\etc\package-includes\ поместить файл с именем «buddydemo-configure.zcml», содержащий единственную строку:

<include package="buddydemo"/>

Язык ZCML теговый. Каждая директива конфигурирования является блочным или простым тегом. Подробное описание директив приведено в главе 8. Директивой самого верхнего уровня всегда является **configure**, служащая контейнером для всех остальных директив. Ниже приведено содержимое файла конфигурирования нашего пакета Buddy, который размещается в папке buddydemo.

```
1)  <configure
2)    xmlns="http://namespaces.zope.org/zope"
3)    xmlns:browser="http://namespaces.zope.org/browser"
4)    xmlns:i18n="http://namespaces.zope.org/i18n"
5)    i18n_domain="buddydemo">

6)    <i18n:registerTranslations directory="locales" />

7)    <content class=".buddy.Buddy">
8)      <implements
9)        interface="zope.app.annotation.IAttributeAnnotatable"
10)     />
11)     <require permission="zope.View"
```

```
12)        interface=".interfaces.IBuddy" />
13)      <require permission="zope.ManageContent"
14)        set_schema=".interfaces.IBuddy" />
15)    </content>

16)    <browser:addMenuItem
17)      class=".buddy.Buddy"
18)      title="Buddy"
19)      permission="zope.ManageContent"
20)      view="AddBuddy.html"
21)    />

22)    <browser:addform
23)      schema=".interfaces.IBuddy"
24)      label="Add Buddy information"
25)      content_factory=".buddy.Buddy"
26)      arguments="first last email address postal_code"
27)      name="AddBuddy.html"
28)      permission="zope.ManageContent"
29)    />

30)    <browser:editform
31)      schema=".interfaces.IBuddy"
32)      label="Change Buddy information"
33)      name="edit.html"
34)      menu="zmi_views"
35)      title="Edit"
36)      permission="zope.ManageContent"
37)    />
38)
39)    <include package=".browser" />
40)
41)  </configure>
```

Директива 1 – 5 конфигурации является группирующей директивой. Она не выполняет каких либо действий, но содержит указание пространства имен, влияющих на интерпретацию относительных имен и файловых путей. Она также задает область i18n, которую нужно определить для поиска переводов текстов сообщений клиенту. Эта информация используются вложенными поддирективами. Атрибут xmlns определяет пространство имен с заданным именем и локализацией. Если имя не задано, то это используется пространство по умолчанию.

Директива 6 регистрирует директорий, содержащий переводы строк.

Директива 7 – 15 регистрирует класс компонента содержимого сайта. Поддиректива **implements** объявляет, что данный класс контента реализует указанный интерфейс. Поддирективы **require** указывает список имен интерфейсов, которые для просмотра требуют разрешение "zope.View", и схему, требующую для ввода значений полей разрешение "zope.ManageContent". Точка, стоящая перед именем модуля, задает позицию файла “buddy.py” **относительно** папки, в которой расположен файл конфигурации.

Директива 16 – 21 **addMenuItem** класса **browser** (рисунок 4) определяет пункт

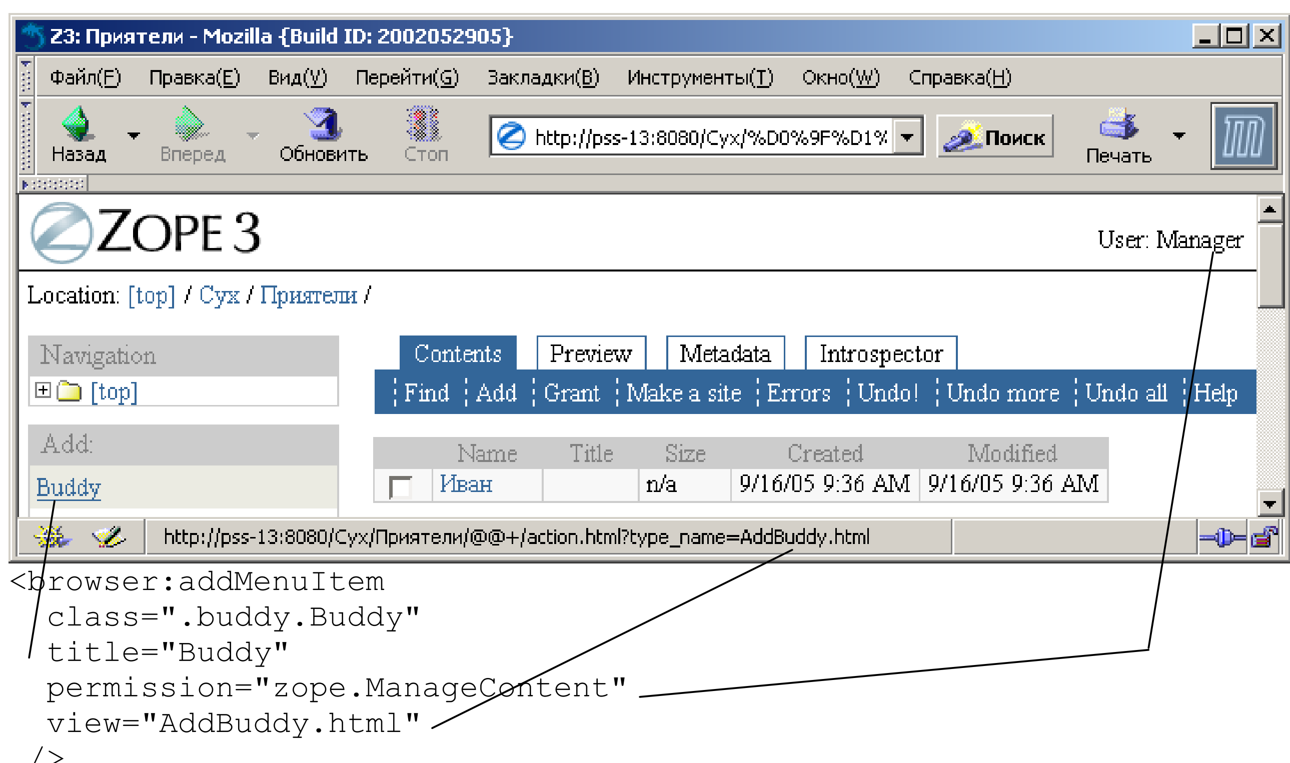

Рисунок 4 – Конфигурирование пункта «Buddy» в меню «Add»

меню в группе «Add/Добавить». Имя пункта задается атрибутом title="Buddy". Атрибут class=".buddy.Buddy" определяет класс, который нужно использовать как фабрику для создания новых объектов. Атрибут permission="zope.ManageContent" по прежнему задает уровень полномочий для пользователя, которому будет доступен этот пункт меню. Атрибут view="AddBuddy.html" задает имя автоматически сгенерированного ресурса, отправляемого браузеру на стороне клиента для добавления экземпляра объекта.

Директива 22 – 29 **addform** групы **browser** (рисунок 5) определяет вид и параметры страницы для ввода данных о приятеле при добавлении экземпляра объекта. Атрибут schema определяет схему, которая будет использована для автоматической генерации формы ввода. Атрибут arguments задает список значений полей ввода, передаваемых конструктору экземпляров класса, задаваемому атрибутом content_factory=".buddy.Buddy".

Директива 30 – 37 **editform** групы **browser** определяет вид и параметры страницы для корректировки данных о приятеле. Смысл атрибутов аналогичен предыдущей директиве.

Директива 39 включает в эту точку конфигурации содержимое конфигурационного файла вспомогательного пакета browser.

Файл конфигурации пакета browser тоже имеет имя «**configure.zcml**» и пока должен содержать следующее:

```
1)  <configure
2)    xmlns="http://namespaces.zope.org/zope"
3)    xmlns:browser="http://namespaces.zope.org/browser"
4)  >
```

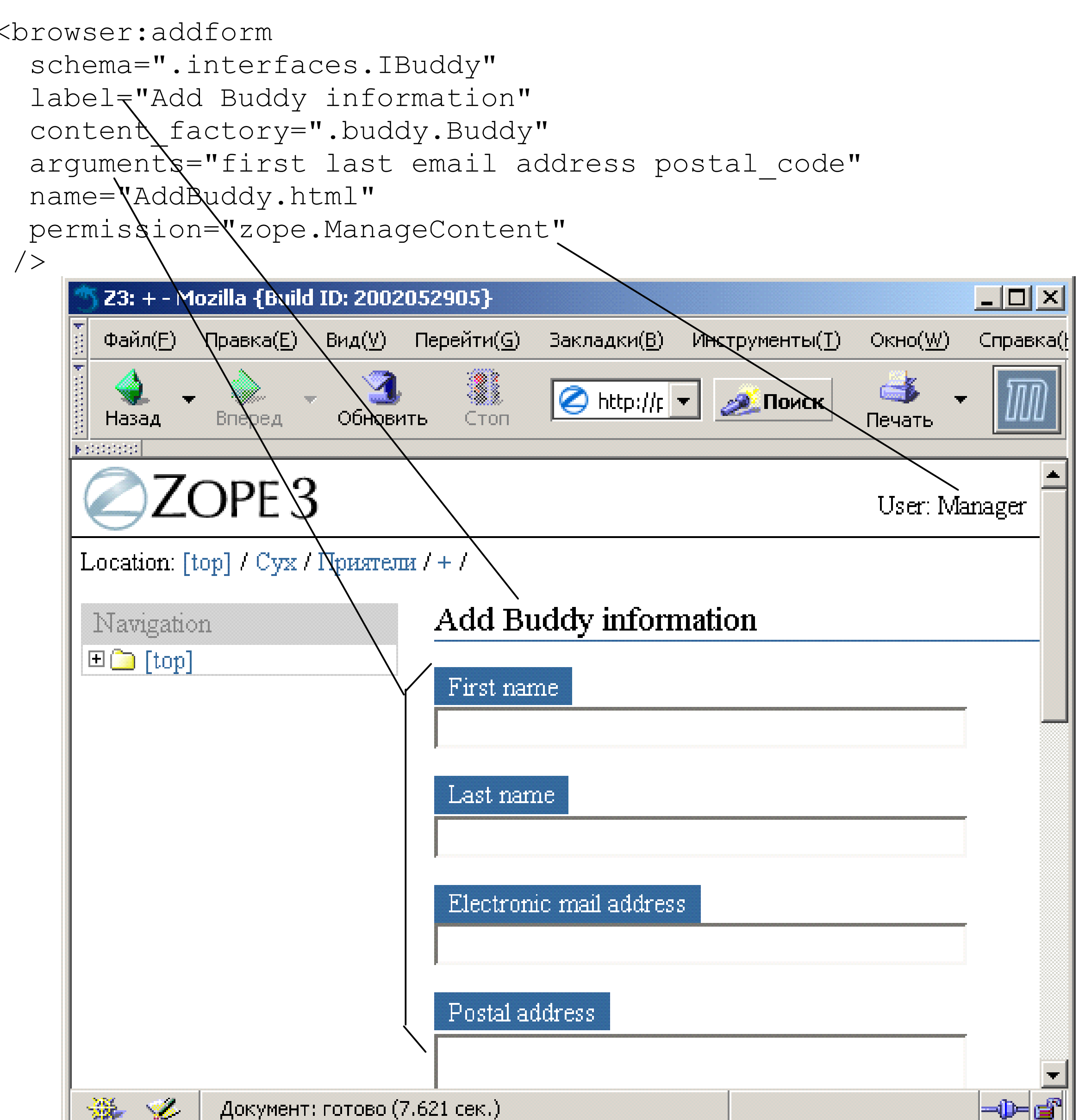

Рисунок 5 – Конфигурирование формы для добавления объекта

```
5)     <browser:page
6)       for="buddydemo.interfaces.IBuddy"
7)       name="index.html"
8)       template="info.pt"
9)       permission="zope.View"
10)      class=".buddy.Buddy"
11)    />
12)
13) </configure>
```

Директива 5 – 11 **page** группы **browser** определяет вид и параметры страницы для просмотра данных о приятеле неавторизованным пользователем, имеющим разрешения уровня permission="zope.View". Смысл атрибутов следует из структуры на рисунке 3. Атрибут for="buddydemo.interfaces.IBuddy" задает параметры

интерфейса, для которого предназначена страница. Атрибут name="index.html" задает имя ресурса, отправляемого пользователю на браузер. Атрибут template="info.pt" задает имя файла с шаблоном страниц, написанным с использованием языков HTML и TAL.

5.5 Интернационализация приложения

Интернационализация – автоматическая поддержка переводов текстов и форматирования даты/времени и денежных единиц в зависимости от страны пребывания клиента. Последнее устанавливается публикатором Zope анализом информации об окружении, направляемой браузером клиента на сервер вместе с запросом на ресурс (смотри атрибут tal:content=”request”). Система Zope3 при отправке сообщения на браузер клиента может предусмотренные заранее строки текста автоматически переводить на язык клиента. Для этого разработчики приложения должны произвести соответствующую подготовку. Подготовка приложения для интернационализации сообщений содержит следующие шаги.

I Шаг. Для автоматизации подготовки переводов всех нуждающихся в этом строк в состав дистрибутива Zope3 входит программа, извлекающая все помеченные для интернационализации тексты из файлов кода на Питоне, шаблонов страниц и конфигураций на ZCML. Исполнение программы производится командой

python i18nextract.py [options]

Опции:

-h | --help – печать сообщение о помощи и выход;

-d | --domain <domain> – определяет имя домена (папки), который должен быть сформирован извлечением строк;

-p | --path <path> – определяет пакет, который должен быть обследован для поиска строк;

-o <dir> – определяет директорий внутри заданного пакета, куда помещается результирующий шаблон переводов строк.

Для нашего примера, разрабатываемого под управлением операционной системы семейства Win32 нужно выполнить команду:

```
C:\> python C:\Zope3inst\bin\i18nextract -p
        "C:\Zope3inst\lib\python\buddydemo"
        buddydemo -d buddydemo -o locales
```

В результате будет создана папка с именем locales и в ней будет помещен файл buddydemo.pot.

II Шаг. В этой же папке нужно создать папку языка и папку локализованных сообщений, например, командами:

```
mkdir <src>\buddydemo\locales\ru
mkdir <src>\buddydemo\locales\ru\LC_MESSAGES
```

где `<src> = C:\Zope3inst\lib\python`

III Шаг. Скопировать файл «buddydemo.pot» в файл «buddydemo.po» в папку LC_MESSAGES.

IV Шаг. Дополнить переводы строк на русский язык (подходит программа WinWord с сохранением результата в кодировке Utf-8 или иной). Выбор кодировки производится указанием в атрибуте **charset=кодировка** файла с расширением «po» (смотри фрагмент файла ниже). Например, для ОС Windows более приемлема кодировка, задаваемая атрибутом **charset=cp1251**.

V Шаг. Компилировать файл «buddydemo.po» в файл «buddydemo.mo» командой

```
python msgfmt.py buddydemo.po
```

Программу msgfmt.py можно найти в C:\Python24\Tools\i18n и скопировать в текущую папку.

VI Шаг. Конфигурировать переводчик в ZCML

```
<configure
...
xmlns:i18n="http://namespaces.zope.org/i18n">
<i18n:registerTranslations directory="locales" />
```

Вот и все. Для реального отображения переводимых строк на русский язык не забудьте установить в браузере в разделе языки первым язык с меткой «ru» (смотри выше второй шаг).

Для иллюстрации приведем фрагмент содержимого файла «buddydemo.po».

```
# Copyright (c) 2003-2004 Zope Corporation and Contributors.
# All Rights Reserved.
# . . .
msgid ""
msgstr ""
"Project-Id-Version: X3 3.0.0\n"
"POT-Creation-Date: Sun May 29 16:21:37 2005\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: Zope3 Developers <zope3-dev@zope.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Generated-By: zope/app/translation_files/extract.py\n"
#: c:\Zope3instance\lib\python\buddydemo\configure.zcml:20
msgid "Buddy"
msgstr "Приятель"

#: c:\Zope3instance\lib\python\buddydemo\configure.zcml:34
msgid "Edit"
msgstr "Редактор"

. . .
```

Другой возможностью организации переводов является использование общих переводчиков сайта. Каждый сайт имеет программное пространство, которое может быть доступно через закладку Manage Site/Управление сайтом. Перейдите на закладку Visit Default Folder/Папка по умолчанию. Добавьте объект выбором из левого списка Translation Domain/Домен переводов и введите имя домена, например, myDom. На закладке Register/Регистрация задать этикетку, выбрать интерфейс с именем ILocalTranslationDomain, задать активный статус и нажать кнопку Add/Добавить. На закладке Translate/Переводы добавить языки для перевода сообщений, например, en и ru, которые появятся в левом списке. Выбрать в левом списке оба языка и нажать кнопку Edit/Редактировать. В таблице в нижней части формы заполнить в первом столбце идентификатор сообщения, например, mes1, в остальных столбцах тексты этого сообщения на соответствующих языках. Далее нажать кнопку Edit Messages/Редактировать сообщения. Для тестирования переводчика создайте ZPT шаблон следующего содержания:

```
<html>
   <body i18n:domain="myDom">
     <h1 i18n:translate="">mes1</h1>
      <span i18n:translate="mes1"> Ok </span>
   </body>
</html>
```

Убедитесь в работоспособности переводчика.

6 Архитектура приложений Zope3

Опыт разработки приложений в Zope 2 с раздутым кодом, постоянными переделками для совместимости с очередной версией системы или ее частей и рядом других проблем организационного и программистского толка привели разработчиков Zope3 к идее использования компонент по аналогии с уже существующими современными технологиями COM, CORBA, Mozilla и другим [4]. Основу технологии Zope3 составляют компоненты, интерфейсы, сервисы, адаптеры и утилиты.

6.1 Сервисы

Сервисы обеспечивают основную функциональность самой среды разработки Zope3, без которой прикладной сервер не может работать. Они хорошо согласуются с инструментальными средствами CMF (Content Management Framework – каркас для управления контентом в Zope 2), из которого была перенята основная концепция организации взаимодействия с пользователями. Сервисы не зависят от каких-либо других компонентов.

При работе в локальном окружении, если запрос не может быть обработан локально, сервисы должны всегда делегировать запрос наверх – вплоть до глобальной версии сервиса. При этом используется механизм «заимствования». Например, если необходимо найти утилиту с именем “hello1”, реализующую интерфейс IHello, и ее нет у локального компонента с учетом механизма наследования ООП, тогда Utility Service перешлет запрос контейнеру компонента. Запрос перемещается вверх до глобального сервиса Utility Service. Если глобальный сервис не сможет обеспечить нужный результат, возвращается ошибка.

Одним из фундаментальных сервисов являются регистрация компонентов. Всякий раз, когда регистрируется класс как утилита с использованием ZCML, то класс, зарегистрированный в “Utility Service”, может быть извлечен в последующем, используя этот сервис. Существует также “Service Service”, который управляет всеми зарегистрированными сервисами.

Другим сервисом Zope3 является «Error Reporting service», который записывает информацию обо всех ошибках, происходящих в течение процесса публикации страницы в журнал. Это позволяет разработчику просматривать детали состояния системы и запроса во время наступления ошибки.

Соглашения Zope3 об интерфейсах сервисов в том, что они содержат только методы доступа. Среди них есть постоянные, определенные в ядре Zope3 и мутируемые методы, обычно зависящие от реализации и предусмотренные дополнительными интерфейсами. Последствие этого в том, что сервисы обычно не модифицируются после старта системы и при их изменении требуется перезапуск Zope3. Заметьте, что авторы Zope3 настоятельно не рекомендуют разработчикам писать сервисы для приложений. Вместо них лучше использовать утилиты.

6.2 Адаптеры

Адаптеры можно считать своеобразным клеем в архитектуре компонентов, поскольку они соединяют одни интерфейсы с другими и допускают альтернативные ООП способы программирования, как, например, Aspect-Oriented Programming (ориентированное на результат программирование). Адаптер получает на вход компонент, осуществляющий один интерфейс, и на его основе создает объект с

другим интерфейсом. Важно, что использование адаптеров не затрагивает объявлений исходных классов входных интерфейсов и не порождает их потомков, как в ООП, но позволяют добавлять в прикладной пакет нужную функциональность. Адаптеры являются экземплярами нового создаваемого для этой цели класса, не зависящего напрямую от классов, реализующих адаптируемый интерфейс. Очень важно отметить, что речь идет об интерфейсах, а не реализующих их классах. Это центральный момент идеи использования адаптеров.

Например, можно написать адаптер, который позволяет интерфейс «IExample» компонента представлять как файл в FTP. Это может быть сделано посредством реализации интерфейсов «IReadFile» и «IWriteFile» для нового производного от входного объекта. Вместо дополнения этой функциональности непосредственно в класс «SimpleExample» при реализации нового интерфейса в исходном классе, создается адаптер, который преобразует «IExample» для того, чтобы на его основе сделать доступными для использования интерфейсы «IReadFile» и «IWriteFile». Как только адаптер будет зарегистрирован для обоих интерфейсов (обычно в конфигурациях ZCML), его экземпляр может быть получен следующим образом:

```
read_file = zapi.getAdapter(example, IReadFile)
write_file = zapi.getAdapter(example, IWriteFile)
```

Метод getAdapter(example, IReadFile) находит в реестре адаптер и создает его экземпляр, который преобразует любой из интерфейсов, реализованных в **example** (экземпляр SimpleExample), в интерфейс **IReadFile**. Дополнительный аргумент – контекст может быть передан как ключевой аргумент, определяющий место поиска адаптера. По умолчанию местом поиска является позиция, из которой был сделан запрос, а далее используется специальный механизм заимствования для адаптеров.

В этом конкретном примере, был описан адаптер для преобразования одного интерфейса в другой. Адаптеры могут отображать несколько интерфейсов в некоторый один. Они известны как мультиадаптеры (multi-adapters) и используются в разнообразных приложениях.

Наиболее важной особенностью использования адаптеров является то, что они не затрагивают первоначальной реализации исходного интерфейса в адаптируемом классе. Следовательно, в Zope3 можно использовать любой продукт Питона, внедряя его в разрабатываемый пакет при помощи написания адаптеров и регистрации их средствами языка ZCML.

Использование Реестра Адаптеров

Ниже приведена небольшая демонстрация работы адаптеров, обеспечивающая необходимые для понимания детали, но ограниченная использованием интерфейсов и адаптеров только в интерпретаторе Питона вне системы Zope3.

Сначала импортируем пакет интерфейсов.

```
>>> import zope.interface
```

Далее разрабатываем интерфейс для нашего объекта, который в нашем примере является файлом. Предусматриваем только один атрибут body, который является фактическим содержанием файла.

```
>>> class IFile(zope.interface.Interface):
...     body = zope.interface.Attribute('Contents of the file.')
...
```

Часто необходимо получать размер файла для различных расчетов объема памяти. Тем не менее, не совсем удобно реализовать хранение размера непосредственно в файловом объекте, так как размер действительно представляет собой метаданные для объекта. Для этого создаем другой интерфейс, который обеспечивает вычисление размера.

```
>>> class ISize(zope.interface.Interface):
...     def getSize():
...         'Return the size of an object.'
...
```

Теперь нам нужно реализовать класс файлов. Существенно то, что объекты класса реализуют интерфейс ***IFile***. Для примера чтобы упростить изложение зададим по умолчанию значение тела файла строкой константой.

```
>>> class File(object):
...      zope.interface.implements(IFile)
...      body = 'foo bar'
...
```

Затем реализуем адаптер, который будет обеспечивать интерфейс ***ISize***, получив любой объект, поддерживающий интерфейс ***IFile***. По соглашению Zope3 в теле адаптера мы используем атрибут «__used_for__» для задания входного интерфейса, который мы хотим преобразовывать адаптерным объектом; в нашем случае – IFile. Если у вас есть несколько исходных интерфейсов, для которых можно использовать адаптер, просто определите этот перечень интерфейсов как кортеж.

Кроме того, по соглашению, конструктор адаптера всегда получает один аргумент context. Контекст в данном случае – экземпляр класса File (с интерфейсом IFile), для которого определяется размер. Также по соглашению Zope3, контекст всегда хранится в атрибуте с именем context адаптера. Вы можете свободно использовать в конструкторе класса любое имя для обозначения формального параметра контекста, например, file, но фактическое значение всегда сохранять в атрибуте context

```
>>> class FileSize(object):
...      zope.interface.implements(ISize)
...      __used_for__ = IFile
...
...      def __init__(self, context):
...          self.context = context
...
...      def getSize(self):
...          return len(self.context.body)
...
```

Теперь, когда написан класс адаптера, нужно зарегистрировать его в реестре адаптеров, чтобы экземпляр адаптера мог быть автоматически создан, когда он потребуется для работы. В Zope не существует такого понятия как глобальный реестр, следовательно, для регистрации при работе в интерпретаторе Питона мы должны создать временный экземпляр реестра для нашего примера вручную.

```
>>> from zope.interface.adapter import AdapterRegistry
```

```
>>> registry = AdapterRegistry()
```

Реестр будет содержать словарь или карту адаптеров, с указанием входных и выходных интерфейсов. Далее мы должны зарегистрировать адаптер, который выполняет преобразование из IFile в ISize. Первый аргумент для метода register() является списком исходных интерфейсов. В нашем случае мы имеем только один входной интерфейс IFile. Список имеет смысл, тогда когда набор интерфейсов принадлежит мультиадаптеру, когда адаптер преобразует много интерфейсов в новый интерфейс. В этих ситуациях, конструктор адаптеров потребует аргументы для каждого определения интерфейса.

Второй аргумент метода register() является интерфейсом, который обеспечивает выход адаптера, в нашем случае ISize. Третий аргумент – имя адаптера. Сейчас мы оставляем его пустой строкой. Имена полезны, если есть разные адаптеры для того же набора интерфейсов, но они созданы для других нужд (по разному работают, написаны разными авторами и др.). Последний аргумент является в нашем случае именем адаптерного класса – конструктора экземпляров адаптера. В общем случае это может быть произвольный объект, согласованный с его последующим использованием в качестве фабрики адаптера.

```
>>> registry.register([IFile], ISize, '', FileSize)
```

После регистрации Вы можете для поиска адаптера в реестре использовать встроенные методы lookup и lookup1. Отличие состоит в том, что lookup требует первым параметром список входных интерфейсов, а lookup1 – имя одного единственного входного интерфейса (не списка). Проверим:

```
>>> registry.lookup1(IFile, ISize, '')
<class '__main__.FileSize'>
```

Давайте получим более полезный результат: создадим файловый экземпляр file и экземпляр адаптера sizeFile, используя поиск в реестре конструктора sizer. Затем мы увидим, возвращает ли адаптер правильный размер, вызывая getSize() для адаптера.

```
>>> file = File()             # Экземпляр файла
>>> sizer = registry.lookup1(IFile, ISize, '')  # Класс адаптера
>>> sizeFile = sizer(file)  # Конструктор экземпляра адаптера
>>> sizeFile.getSize()      # Метод экземпляра
7
```

Тем не менее, продемонстрированный пример не очень удобно использовать при программировании компонент, поскольку мы должны вручную передавать аргументы в метод поиска lookup1 и выполнять всю последовательность вызовов. Есть небольшой синтаксический прием, который позволяет нам получать экземпляр адаптера, просто вызывая ISize(file). Для того чтобы использовать эту функциональность, нам нужно произвести регистрацию в списке adapter_hooks, который является элементом модуля адаптеров. Этот список содержит набор, который автоматически вызывается, когда вызван I<OutClass>(<obj>). Цель в том, чтобы найти адаптер, который предоставляет запрошенный выходной интерфейс I<OutClass> для входного интерфейса у заданного фактическим параметром контекстного экземпляра <obj>.

От нас потребуется реализовать свой собственный метод поиска адаптера. Приведенный ниже пример показывает один из самых простых случаев использования реестра, но можно реализовать варианты с использованием кэша адаптеров или хранимых в ZODB объектов адаптеров. В нашем примере метод ожидает, что первый аргумент provided – желаемый выходной интерфейс (для нас ISize) и второй аргумент object – контекст адаптера (здесь file), и возвращает адаптер – экземпляр класса FileSize.

```
>>> def hook(provided, object):
...     adapter = registry.lookup1(zope.interface.providedBy(
            object), provided, '')
...     return adapter(object)
...
```

Теперь для регистрации возможности автоматического создания адаптера мы просто добавляем метод hook к списку adapter_hooks.

```
>>> from zope.interface.interface import adapter_hooks
>>> adapter_hooks.append(hook)
```

Как только метод поиска будет зарегистрирован, Вы можете использовать желаемый синтаксис для создания экземпляра адаптера.

```
>>> sizeFile = ISize(file)
>>> sizeFile.getSize()
7
```

Еще более короткой формой одноразового обращения к адаптеру может быть:

```
>>> ISize(file).getSize()
7
```

По окончании экспериментов мы можем очистить список adapter_hooks.

```
>>> adapter_hooks.remove(hook)
```

Простые адаптеры

Рассмотрим пример с использованием следующей спецификации интерфейсов:

```
>>> from zope.interface.adapter import AdapterRegistry
>>> import zope.interface

>>> class IR1(zope.interface.Interface):
...     pass
>>> class IP1(zope.interface.Interface):
...     pass
>>> class IP2(IP1):
...     pass

>>> registry = AdapterRegistry()
```

Зарегистрируем объект, который преобразует интерфейс IR1 в интерфейс IP2:

```
>>> registry.register([IR1], IP2, '', 12)
```

Задав регистрацию, мы можем найти его снова:

```
>>> registry.lookup([IR1], IP2, '')
12
```

Отметьте, что в приведенном примере использовано в качестве объекта с новым интерфейсом целое число. В реальных приложениях должны использоваться некоторые объекты, которые действительно обеспечивают заданные интерфейсы. Реестру все равно, что регистрировать, так что для демонстрации соглашений о регистрации и поиске адаптеров, чтобы наши примеры были проще, мы используем целые числа или строки. Есть одно исключение: регистрация значения None отменяет регистрацию любого прежде зарегистрированного адаптерного объекта с указанными входными и выходными интерфейсами, поэтому объект None не может выступать в качестве генератора адаптеров.

Если создается интерфейс потомок некоторого ранее зарегистрированного интерфейса, то родительский конструктор может быть найден по новой спецификации:

```
>>> class IR2(IR1):
...     pass
>>> registry.lookup([IR2], IP2, '')
12
```

При поиске адаптерного объекта мы можем найти спецификацию входного интерфейса указанием реализующего его класса:

```
>>> class C2:
...     zope.interface.implements(IR2)

>>> registry.lookup([zope.interface.implementedBy(C2)], IP2, '')
12
```

так как метод `zope.interface.implementedBy(C2)]` возвращает список реализуемых объектом `C2` интерфейсов.

Мы можем искать выходные интерфейсы, которые расширяют реализацию заданного интерфейса, например, если IP1 является родителем для IP2:

```
>>> registry.lookup([IR1], IP1, '')
12
>>> registry.lookup([IR2], IP1, '')
12
```

Но если Вы требуете объект со спецификацией входного интерфейса, который не является потомком зарегистрированного интерфейса, то Вы получите None:

```
>>> registry.lookup([zope.interface.Interface], IP1, '')
```

Для этих случаев можно задать объект по умолчанию, если поиск не дал результата:

```
>>> registry.lookup([zope.interface.Interface],IP1,'',default=42)
42
```

Если Вы пытаетесь получить выходной интерфейс, который объект не обеспечивает, то Вы также получите None:

```
>>> class IP3(IP2):
...     pass
>>> registry.lookup([IR1], IP3, '')
```

Вы также получите None, если задали неправильное имя:

```
>>> registry.lookup([IR1], IP1, 'bob')
>>> registry.register([IR1], IP2, 'bob', "Bob's 12")
>>> registry.lookup([IR1], IP1, 'bob')
"Bob's 12"
```

Вы можете не задавать имя при поиске:

```
>>> registry.lookup([IR1], IP1)
12
```

Если мы регистрируем объект, который обеспечивает IP1:

```
>>> registry.register([IR1], IP1, '', 11)
```

то этот объект при поиске будет предпочтен объекту 12:

```
>>> registry.lookup([IR1], IP1, '')
11
```

Также, если мы регистрируем объект для IR2, тогда он будет предпочтителен при использовании IR2:

```
>>> registry.register([IR2], IP1, '', 21)
>>> registry.lookup([IR2], IP1, '')
21
```

Поиск адаптера для одного входного интерфейса можно рассматривать как частный случай поиска для списка интерфейсов. Для этого можно использовать специализированный метод lookup1, который находит адаптер для единственного входного интерфейса:

```
>>> registry.lookup1(IR2, IP1, '')
21
```

```
>>> registry.lookup1(IR2, IP1)
21
```

Реестр адаптеров поддерживает транзитивность объектов и интерфейсов. Таким образом, служба регистрации поддерживает вычисление адаптеров. Пусть объект реализует интерфейс IR и нужно найти адаптер, преобразующий его в интерфейс IP1. Для этого необходимо зарегистрировать фабрику адаптеров Y:

```
>>> class IR(zope.interface.Interface):
...     pass

>>> class X:
...     zope.interface.implements(IR)

>>> class Y:  # Фабрика адаптеров для интерфейса IP1
...     zope.interface.implements(IP1)
...     def __init__(self, context):
...         self.context = context

>>> registry.register([IR], IP1, '', Y)
```

В этом случае, мы зарегистрировали класс “Y” как фабрику. Теперь мы можем вызвать метод queryAdapter, чтобы получить адаптирующий объект:

```
>>> x = X()
>>> y = registry.queryAdapter(x, IP1)
>>> y.context is x
True
>>> y.__class__.__name__
'Y'
```

Мы можем тоже регистрировать и искать адаптеры по заданному имени:

```
>>> class Y2(Y):
...     pass

>>> registry.register([IR], IP1, 'bob', Y2)
>>> y = registry.queryAdapter(x, IP1, 'bob')
>>> y.__class__.__name__
'Y2'
>>> y.context is x
True
>>> isinstance(y, Y)
True
>>> y.__class__.__name__
'Y2'
```

Альтернативный способ, который обеспечивает ту же функцию что и queryAdapter() – использование метода adapter_hook реестра адаптеров:

```
>>> y = registry.adapter_hook(IP1, x)
>>> y.__class__.__name__
```

```
'Y'
>>> y.context is x
True
>>> y = registry.adapter_hook(IP1, x, 'bob')
>>> y.__class__.__name__
'Y2'
>>> y.context is x
True
```

Метод adapter_hook используется для перехвата механизма вызова интерфейсов. Он просто переключает вызов на объект и связывает передаваемые аргументы.

Иногда Вы хотите обеспечить адаптер, который преобразовывает пустой интерфейс в заданный. Для этого нужно указать константу None как исходный интерфейс.

```
>>> registry.register([None], IP1, '', 1)
```

Затем мы можем использовать этот адаптер для объектов, у которых нет нужных адаптеров в указанный интерфейс `IP1`:

```
>>> class IQ(zope.interface.Interface):
...     pass
>>> registry.lookup([IQ], IP1, '')
1
```

Так как такой адаптер применим к входным интерфейсам любого класса, то их следует считать адаптерами по умолчанию в заданный выходной интерфейс.

Интерфейс при регистрации и поиске может быть задан любым доступным способом. Вы можете зарегистрировать входные интерфейсы адаптера ссылкой на декларацию некоторого класса, который их поддерживает, так же как и запросить поиск адаптера для интерфейсов класса:

```
>>> registry.register([zope.interface.implementedBy(C2)], IP1,
'', 'C21')
>>> registry.lookup([zope.interface.implementedBy(C2)], IP1, '')
'C21'
```

Вы можете отменить регистрацию, регистрируя «None» в качестве адаптерного объекта, например:

```
>>> registry.register([zope.interface.implementedBy(C2)], IP1,
'', None)
```

Теперь поиск адаптера возвратит ранее зарегистрированный объект:

```
>>> registry.lookup([zope.interface.implementedBy(C2)], IP1, '')
```

12

Как следствие, это означает, что значение «None» не может быть зарегистрировано адаптерным объектом. Это – исключение из сделанного раньше утверждения о том, что в реестре можно регистрировать любые объекты.

Мультиадаптеры

Вы можете преобразовывать несколько интерфейсов в один заданный:

```
>>> registry.register([IR1, IQ], IP2, '', '1q2')
>>> registry.lookup([IR1, IQ], IP2, '')
'1q2'
>>> registry.lookup([IR2, IQ], IP1, '')
'1q2'

>>> class IS(zope.interface.Interface):
...     pass
>>> registry.lookup([IR2, IS], IP1, '')

>>> class IQ2(IQ):
...     pass

>>> registry.lookup([IR2, IQ2], IP1, '')
'1q2'

>>> registry.register([IR1, IQ2], IP2, '', '1q22')
>>> registry.lookup([IR2, IQ2], IP1, '')
'1q22'
```

Если необходимо адаптировать несколько объектов, интерфейсы которых неизвестны, можно использовать при поиске адаптера ссылки на сами объекты.

```
>>> class Q:
...     zope.interface.implements(IQ)
```

Как и для единичных адаптеров, мы регистрируем фабрику, – часто просто класс:

```
>>> class IM(zope.interface.Interface):
...     pass
>>> class M:
...     zope.interface.implements(IM)
...     def __init__(self, x, q):
...         self.x, self.q = x, q
>>> registry.register([IR, IQ], IM, '', M)
```

И затем мы можем вызвать метод queryMultiAdapter, чтобы найти адаптер:

```
>>> x = X()
>>> q = Q()
>>> m = registry.queryMultiAdapter((x, q), IM)
>>> m.__class__.__name__
'M'
>>> m.x is x and m.q is q
True
```

Для персонификации адаптеров, мы можем использовать имена:

```
>>> class M2(M):
...     pass
>>> registry.register([IR, IQ], IM, 'bob', M2)
>>> m = registry.queryMultiAdapter((x, q), IM, 'bob')
>>> m.__class__.__name__
'M2'
>>> m.x is x and m.q is q
True
>>> m.x
<__main__.X instance at 0x00A9F530>
>>> m.q
<__main__.Q instance at 0x00AA7EB8>
```

Как и для простых адаптеров, Вы можете определить мультиадаптеры по умолчанию, определяя «None» для первой спецификации списка:

```
>>> registry.register([None, IQ], IP2, '', 'q2')
>>> registry.lookup([IS, IQ], IP2, '')
'q2'
```

Нуль-адаптеры

Вы можете зарегистрировать мультиадаптер с пустым списком входных интерфейсов – нуль-адаптер:

```
>>> registry.register([], IP2, '', 2)
>>> registry.lookup([], IP2, '')
2
>>> registry.lookup([], IP1, '')
2
```

Иногда полезно получать все именованные простые и мультиадаптеры для заданных интерфейсов:

```
>>> adapters = list(registry.lookupAll([IR1], IP1))
>>> adapters.sort()
>>> adapters
[(u'', 11), (u'bob', "Bob's 12")]

>>> registry.register([IR1, IQ2], IP2, 'bob', '1q2 for bob')
>>> adapters = list(registry.lookupAll([IR2, IQ2], IP1))
>>> adapters.sort()
>>> adapters
[(u'', '1q22'), (u'bob', '1q2 for bob')]
```

В том числе и для нуль-адаптеров

```
>>> registry.register([], IP2, 'bob', 3)
>>> adapters = list(registry.lookupAll([], IP1))
>>> adapters.sort()
>>> adapters
[(u'', 2), (u'bob', 3)]
```

6.3 Пример адаптера для штата и города

Пример пакета «buddydemo» позволяет продемонстрировать использование адаптеров и утилит. Ранее мы не стали включать названия города и штата в данные о приятеле, поскольку их можно «вычислить» из индекса, пользуясь почтовыми справочниками. Для этого создадим утилиту поиска информации о городе и штате по почтовому индексу. Наша утилита реализует интерфейс **IPostalLookup**. Интерфейс **IPostalInfo** описывает данные, не хранимые в объекте, а вычисляемые на основе других данных (рисунок 6). Ниже приведено содержимое файла интерфейсов

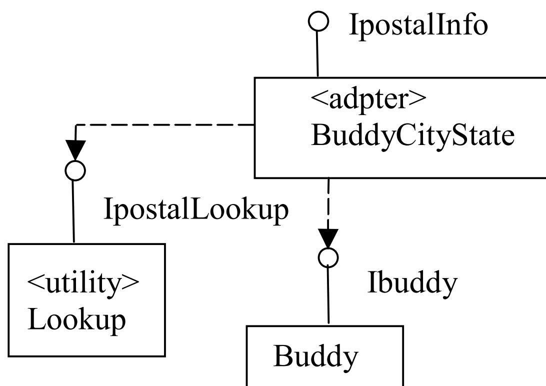

Рисунок 6 – Взаимодействие объекта, адаптера и утилиты

interfaces.py нашего приложения, реализующее эту идею. Этот файл мы поместим в пакет browser, так как эта информация относится к визуализации данных.

```
import zope.interface
from zope.schema import TextLine

class IPostalInfo(zope.interface.Interface):
  "Provide information for postal codes"
  city =  TextLine(title=u"City")
  state = TextLine(title=u"State")

class IPostalLookup(zope.interface.Interface):
  "Provide postal code lookup"
  def lookup(postal_code):
    """Lookup information for a postal code.
       An IPostalInfo is returned if the postal
       code is known. None is returned otherwise.
    """
```

Реализация соответствующей утилиты просмотра почтовых индексов может использовать некоторую базу данных для поиска города и штата по индексу. Мы не желаем сейчас связываться с деталями поиска в полных справочниках, поэтому создадим временную заглушку.

/buddydemo/browser/stubpostal.py

```
import zope.interface
from buddydemo.browser.interfaces import IPostalInfo,
IPostalLookup
class Info:
```

```
  zope.interface.implements(IPostalInfo)
  def __init__(self, city, state):
    self.city, self.state = city, state

class Lookup:
  zope.interface.implements(IPostalLookup)
  _data = {
    '224010': ('Fredericksburg', 'Virginia'),
    '448700': ('Sandusky', 'Ohio'),
    '900510': ('Los Angeles', 'California'),
    '620000': ('Ekaterinburg', 'Svk'),
    '620108': ('Ekaterinburg', 'Svk'),
    '127000': ('Moskow', 'Msk'),
    '445016': ('Magnitogorsk', 'Chlb'),
    }
  def lookup(self, postal_code):
    data = self._data.get(postal_code)
    if data:
      return Info(*data)
    else: return Info('', '')
```

Одно из важных преимуществ утилит в том, что их реализация может быть заменена без влияния на клиентов. В примере диалога с интерпретатором Питона, приведенном ниже, объект класса **Info** обеспечивает хранение информации о городе и штате:

```
>>> from buddydemo import stubpostal
>>> info = stubpostal.Info('Ekaterinburg', 'Svk')
>>> info.city, info.state
('Ekaterinburg', 'Svk')
```

Поиск объекта обеспечивает реализация метода **lookup** интерфейса IPostalLookup в небольшой собственной базе данных, являющейся атрибутом _data класса Lookup (напоминаем, что Питон различает буквы верхнего и нижнего регистров, будьте внимательны!).

```
>>> objLookup = stabpostal.Lookup()
>>> objInfo = objLookup.lookup('620000')
>>> objInfo.city, objInfo.state
('Ekaterinburg', 'Svk')
```

Добавление в browser/configure.zcml для регистрации утилиты

```
<content class=".stubpostal.Info">
  <allow interface=".interfaces.IPostalInfo" />
</content>

<utility
  factory=".stubpostal.Lookup"
  provides=".interfaces.IPostalLookup"
  permission="zope.Public"
/>
```

Как упомянуто раньше, IPostalInfo – тип интерфейса контента. Нам нужно задать декларации безопасности, чтобы пользователи могли иметь доступ к его атрибутам и методам. Для этого используется директива декларации безопасности **allow**, которая объявляет, что разрешение на доступ к интерфейсу или атрибуту имеют все. Директива **utility** регистрирует нашу утилиту с фабрикой для создания, являющейся классом Lookup. Мы могли взамен определить уже существующую утилиту, используя атрибут component. Это необходимо, если нам нужно задавать данные при инициализации компонента. В этом случае использование атрибута **factory** – более удобно, поскольку он позволяет нам избегать создания экземпляра в нашем модуле Питона.

Атрибут **provides** определяет интерфейс, обеспечиваемый утилитой. Мы могли бы также задать атрибут **name**. В этом примере, имя оставлено по умолчанию пустой строкой.

Мы определяем необходимое разрешение для использования утилиты. Разрешение – опционально, тем не менее, если бы оно не было установлено (permission="zope.Public"), тогда неавторизованный код не будет использован утилитой. В определении утилиты, мы использовали специальное разрешение zope.Public – оно обеспечивает безусловный доступ. Все, что требует zope.Public, всегда является безусловно доступным всем. Как можно догадаться, директива allow являться эквивалентом требования директивы с разрешением zope.Public.

Далее создадим в пакете browser модуль browser.py и добавим в него объявление класса BuddyCityState в

```
import zope.interface
from buddydemo.interfaces import IBuddy
from buddydemo.browser.interfaces import IPostalInfo,
        IPostalLookup
from stubpostal import Lookup

class BuddyCityState:
  zope.interface.implements(IPostalInfo)
  __used_for__ = IBuddy
  def __init__(self, buddy):
    lookup = zope.app.zapi.getUtility(IPostalLookup)
    info = lookup.lookup(buddy.postal_code)
    if info is None:
      self.city, self.state = '', ''
    else:
      self.city, self.state = info.city, info.state
```

Наш класс-адаптер реализует интерфейс IPostalInfo. По соглашению Zope атрибут __used_for__ должен содержать имя интерфейса IBuddy, который является входным для преобразования в выходной интерфейс IPostalInfo. Конструктор получает параметр – адаптируемый объект с интерфейсом типа IBuddy. В этом примере вся работа адаптера по вычислению атрибутов выходного интерфейса IPostalInfo выполняется в конструкторе: атрибутам объекта класса BuddyCityState, предоставляющего интерфейс IPostalInfo, присваиваются значения города и штата, найденные утилитой lookup.

Поиск утилиты lookup реализован при помощи метода zapi.getUtility. Модуль zapi включает ряд широко используемых общих программных модулей для приложений. Метод getUtility поднимает исключение, если запрошенная утилита не обнаружена. Обычно методы серии "get" возбуждают ошибки, если они не могут найти заданный объект, в отличии от обычных методов серии "query" (например, queryUtility), которые возвращают значение по умолчанию (часто None), если запрос нельзя обработать. Конструктор использует утилиту, чтобы найти информацию по почтовому индексу, сохраняя ее для использования получателем в атрибутах city и state.

Регистрация адаптера в browser/configure.zcml

```
<adapter
 factory=".buddy.BuddyCityState"
 provides=".interfaces.IPostalInfo"
 for=".interfaces.IBuddy"
 permission="zope.Public"
/>
```

Адаптерная директива подобно директиве для утилит определяет интерфейсы, фабрику и разрешение. Как и для утилиты, может быть задано имя. Атрибут for определяет входной интерфейс адаптера.

Для демонстрации страниц с новыми данными, получаемыми с использованием адаптера нужно описать класс BuddyInfo с новым составом атрибутов. Поместим его в файл **browser.py**.

```
class BuddyInfo:
     """Provide an interface for viewing a Buddy
     """
     zope.interface.implements(IPostalInfo)
     __used_for__ = IBuddy
     def __init__(self, context, request):
          self.context = context
          self.request = request
          info = IPostalInfo(context)
          self.city, self.state = info.city, info.state
```

Мы создали класс для просмотра, который обеспечивает город и штат как атрибуты, используемые нашим шаблоном ZPT. Объект для визуализации view использует адаптер в конструкторе, чтобы получать информацию о городе и штате. Мы получаем адаптер, просто вызывая интерфейс. Если переданный параметром context в интерфейс объекта (в нашем примере экземпляр BuddyInfo) реализует IPostalInfo, тогда будет возвращен нужный объект info, из которого извлекаются значения для атрибутов city и state объекта класса BuddyInfo.

Коррекция info.pt

```
<html metal:use-macro="context/@@standard_macros/page">
<body><div metal:fill-slot="body">
 <table>
```

```
  <caption i18n:translate="">Buddy information</caption>
  <tr><td i18n:translate="">Name:</td>
    <td><span tal:replace="context/first">First</span>
      <span tal:replace="context/last">Last</span></td>
  </tr>
  <tr><td i18n:translate="">Email:</td>
    <td tal:content="context/email">foo@bar.com</td>
  </tr>
  <tr><td i18n:translate="">Address:</td>
    <td tal:content="context/address">1 First Street</td>
  </tr>
  <tr><td>City:</td>
    <td tal:content="view/city | default">City</td>
  </tr>
  <tr><td>State:</td>
    <td tal:content="view/state | default">State</td>
  </tr>
  <tr><td i18n:translate="">Postal code:</td>
    <td tal:content="context/postal_code">12345</td>
  </tr>
 </table>
</div></body></html>
```

Мы откорректировали шаблон, чтобы включить в него отображение города и штата. Отметьте, что мы используем переменную **view**, чтобы ссылаться на объект просмотра и получать атрибуты города и штата. Когда при просмотре используется шаблон ZPT, для него определена переменная view верхнего уровня, чтобы обеспечить доступ к виртуальным атрибутам.

Коррекция директивы page в configure.zcml

Мы должны откорректировать директиву page, чтобы сделать доступным класс BuddyInfo для просмотра.

```
<browser:page
     for=".interfaces.IBuddy"
     name="index.html"
     template="info.pt"
     permission="zope.View"
     class=".browser.BuddyInfo"
/>
```

6.4 Компоненты для просмотра

Компоненты **view** для просмотра являются своеобразными специализированными адаптерами, создаваемыми средствами языка ZCML, и обеспечивают предоставление некоторого интерфейса для другого интерфейса пользователя или протокола (рисунок 7).

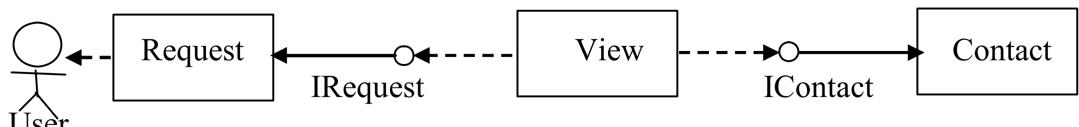

Рисунок 7 – Диаграмма обработки запроса

Компонент просмотра взаимодействует с пользователем, представленным объектом запроса Request с интерфейсом IRequest. Отметьте, что компоненты просмотра являются специальным случаем компонент представления. Компоненты представления ответственны за обеспечение интерфейсов пользователя (или интерфейсы в другие внешние сущности). Другим типом компонент представления являются ресурсы, подобные рисункам и таблицам стилей.

Обычно адаптер получает от среды публикации два параметра: контекст (объект для работы) и запрос. Конструктор экземпляра адаптера может поместить их в атрибуты с соответствующими именами для последующего использования методами, например:

```
def __init__(self, context, request):
     self.context = context
     self.request = request
```

Рассмотрим пример адаптера для поиска страниц (Traverser) в проекте zwiki, регистрируемый в файле конфигурации оператором:

```
  <view
       for="zwiki.interfaces.IWikiPage"
       type="zope.publisher.interfaces.browser.IBrowserRequest"
       provides="zope.publisher.interfaces.IPublishTraverse"
       factory="zwiki.traversal.WikiPageTraverser"
       permission="zope.Public"
       />
```

Адаптером является экземпляр класса WikiPageTraverser следующего вида.

```
1)   class WikiPageTraverser:
2)     implements(IPublishTraverse)
3)     __used_for__ = IWikiPage

4)     def __init__(self, page, request):
5)       self.context = page
6)       self.wiki = getParent(page)
7)       self.request = request

8)     def publishTraverse(self, request, name):
9)       page = self.wiki.get(name, None)

10)      # Check that page has self.context as parent
11)          if page is None or \
12)          not zapi.getName(self.context) in
                  IWikiPageHierarchy(page).parents:

13)          view = queryView(self.context, name, request)
14)          if view is not None:
15)              return view

16)          raise NotFoundError(self.context, name, request)


17)      return removeAllProxies(page)
```

```
18)    def browserDefault(self, request):
19)      c = self.context
20)      view_name = getDefaultViewName(c, request)
21)      view_uri = "@@%s" % view_name
22)      return c, (view_uri,)
```

Строки 1 – 3 соответствуют требованиям оформления адаптеров и определяют входные и выходные интерфейсы. Строки 4 – 7 объявление конструктора класса. Строки 8 – 17 задают метод поиска страниц приложения, начиная с контейнера для контекстного объекта. Строки 18 – 22 задают стандартный метод определения умолчаний.

6.5 Тестирование приложений

Тестирование – процесс установления соответствия авторского замысла и реального поведения программы. Прежде чем приступить к тестированию, необходимо четко выразить и зафиксировать ожидаемое поведение программы. Таким средством в Zope3 являются встроенные в строки документации на модуль так называемые «док-тесты» и тесты, помещенные в отдельные файлы. Внешне они выглядят как обычные тексты, но если среди них встречается строка, начинающаяся с символов «>>>», то она интерпретируется как команда, подлежащая обработке интерпретатором Питона. За оператором языка следуют строки с ожидаемыми результатами его выполнения. Наличие таких строк делает документацию более полной и понятной, поскольку исключаются неоднозначности, характерные для изложения работы программ фразами разговорного языка. Строки тестов являются примерами поведения программы, отклонение от которых свидетельствует о неверной ее работе. Когда производится тестирование программного обеспечения, стандартная подсистема Питона **unittest** запускает тесты автоматически и фактический вывод сравнивается с тем, который показан в примерах. Если вывод отличается, то тестирование неудачно. Ниже приведен пример оформления тестов для приложения «Приятели», размещенного в отдельном файле **buddy.txt**.

```
Buddies
=======
Buddies provide basic contact information:
- First name
- Last name
- Email address
- Street address, and
- postal code
Buddies can be created by calling the `Buddy` class:
>>> from buddydemo.buddy import Buddy
>>> bud = Buddy('Bob', 'Smith', 'bob@smith.org',
... '513 Princess Ann Street', '22401')
You can access the information via attributes:
>>> bud.first, bud.last, bud.email
('Bob', 'Smith', 'bob@smith.org')
>>> bud.address, bud.postal_code
('513 Princess Ann Street', '22401')
```

Поскольку тестирование выполняется посредством сравнения фактического и ожидаемого вывода, нужно быть осторожным в оформлении ожидаемого выхода и соблюдать следующие рекомендации:

- Выход должен быть одинаковым при каждом исполнении теста, избегайте вывода текущей даты, времени, адресов интернет-ресурсов, словарей и чисел с плавающей точкой.
- Выход не может превышать 79 символов. Это проблема связана с тем, что стандарт Zope допускают строки длиной не более 80 символов.
- Пустые строки используются в **doctest**, чтобы идентифицировать конец вывода, так что вывод не может содержать пустых строк. Если это необходимо, то используется специальный маркер: <BLANKLINE>
- Избегайте строк с конечными пробелами
- Избегайте использования обратной косой черты. Если они есть, то используйте их выделение.

Для проведения тестирования необходимо написать программу-задание **/buddydemo/tests.py** следующего содержания.

```
import unittest
from zope.testing.doctest import DocFileSuite
def test_suite():
 return DocFileSuite('buddy.txt')
```

Затем нужно запустить тест командой:

```
python test.py -s buddydemo
```

Интерпретатор тестов Zope **test.py** ищет в исходном дереве Zope модуль или пакет с именем **“tests”**. Если модуль тестирования найден, то он ищет в модуле функцию test_suite, которая возвращает блок теста **unittest**. Если он находит пакет, то ищет все модули в этом пакете, имена которых начинаются с **“test”**.

В нашем примере мы могли бы поместить тесты в модуль **buddy.py**. В этом случае функция **test_suite** должна использовать **DocTestSuite**, чтобы создать блок теста из док-строк модуля. Интерпретатор тестов обеспечивает массу полезных свойств. Для получения дополнительной информации о возможностях запустите тестер с аргументом -h: python test.py –h.

7 Схемы и формы

Еще на ранних этапах разработчики Zope3 пришли к мнению, что будет громоздким вручную писать формы HTML и программировать ввод данных при добавлении и редактировании объектов. Стало ясно, что если расширить объявления интерфейсов, то с их использованием можно автоматически генерировать формы HTML и автоматически проверять вводимые данные. К тому же, эти формы предназначены для менеджеров, а не для клиентов сайта, поэтому требования к их оформлению могут быть ослаблены до использования унифицированных шаблонов.

В основе идеи автоматической генерации форм лежат возможности, реализованные еще в продукте Formulator Zope 2, по созданию различных входных полей (целые или текстовые строки) в формах ввода и учету метаданных о входных областях, таких как максимальная и минимальная длина строки. Таким образом, процедурная составляющая представления была удалена совсем, поля стали соответствовать только декларациям атрибутов в интерфейсе. Был реализован учет метаданных в декларации атрибутов, что позволило выполнить автоматическую генерацию форм HTML и проверку данных. Эти расширенные атрибуты задаются при описании полей в интерфейсах компонент. Такой интерфейс, содержащий поля, обычно называют **схемой**.

7.1 Схемы и интерфейсы

Как упоминалось выше, схемы – просто расширенные в сторону описания форматов ввода данных интерфейсы и, следовательно, наследуют класс Interface пакета zope.interface. Поля в схемах являются экземплярами объектов соответствующих классов. Поля и методы в интерфейсах дополняют друг друга, поскольку они описывают разные аспекты компонента. Методы интерфейса описывают функциональное назначение компонента, а поля схемы представляют описание состава данных. Таким образом, нет необходимости разрабатывать новый синтаксис для описания схем, для этого используется обычная декларация интерфейса, например:

```
1)  from zope.interface import Interface
2)  from zope.schema import Text
3)
4)  class IExample(Interface):
5)
6)    text = Text(
7)      title=u"Text",
8)      description=u"The text of the example",
9)      required=True)
```

Строка 2 определяет, что все встроенное поле класса Text импортируются из пакета zope.schema.

Строки 6 – 9 определяют поле ввода многострочного текста. Название в строке 7 и описание в строке 8 используются как понятные пользователю надписи при генерации формы. Одновременно они также служат в качестве встроенной документации объявленной области. Опция required в строке 9 определяет должен

ли объект, реализующий IExample, обеспечивать обязательное задание текста или нет.

7.2 Основные поля схем

После рассмотрения простого примера схемы, приведем описание основных полей и их свойств.

Свойства всех полей:

title (тип: TextLine): Название, заголовок атрибута при отображении области.

description (тип: Text): Описание атрибута для конечных пользователей и службы помощи.

required (тип: Bool): Определяет обязателен атрибут или нет. В форме добавления требование является эквивалентом обязательным аргументам конструктора.

readonly (тип: Bool): Если для поля задано свойство readonly, то значение атрибута может устанавливаться только один раз и затем может только отображаться. Например, уникальный идентификатор объекта – хороший кандидат для отображения в области read only.

default (тип: зависит от поля): Значение по умолчанию для атрибута, если никакое значение не было введено. Это значение часто определяется для обязательных областей.

order (тип: Int): Определяет относительную позицию в списке полей. Обычно вручную это значение не устанавливается, поскольку автоматически назначается при инициализации интерфейса. Порядок областей в схеме по умолчанию такой же, как и порядок полей в описании интерфейса на Питоне.

Типы полей

Bytes, BytesLine Байты и строки байтов отличается тем, что BytesLine не может содержать символ перехода на новую строку. Байты ведут себя идентично типу **str** Питона. Области Bytes и BytesLine являются итерабельными (допускают цикл по символам строки). Атрибуты:

- min_length (тип: Int): После удаления пробелов длина строки должна быть не менее заданной. По умолчанию None, что соответствует отсутствию требования на минимальную длину.
- max_length (тип: Int) Аналогично для максимальной длины строки.

Text, TextLine Две области отличаются тем, что TextLine не может содержать символа перехода на новую строку. Текстовые области содержат символы уникода. Текст и TextLine являются итерабельными.

SourceText является специальной областью производной от Text, которая содержит исходный код любого типа.

Password – специальная производная от области TextLine для использования символов-заполнителей при вводе паролей.

Bool Область Bool не имеет атрибутов.

Int отображается непосредственно в объект **Int** Питона. Атрибуты:

- min (тип: Int): Определяет минимальное возможное значение, например, 0 для положительных чисел.
- max (тип: Int): Определяет самое большое возможное значение.

Float отображается непосредственно в объект Float Питона. Атрибуты:

- min (тип: Float): Аналогичен min для целых.
- max (тип: Float): Аналогичен max для целых.

Datetime Подобно Int и Float, Datetime имеет атрибуты min и max, которые определяют границы возможных значений. Приемлемые значения для этих полей должны быть экземплярами встроенного типа datetime.

Tuple, List Кортеж и список Питона соответственно. Кортеж и список являются итерабельными. Атрибуты:

- min_length (тип: Int): Длина последовательности не может быть менее чем указано. По умолчанию None – нет минимума.
- max_length (тип: Int): Длина последовательности не может быть более чем указано. По умолчанию None – нет максимума.
- value_type (тип: Field): Значения поля должны соответствовать этому ограничению области. Наиболее общая область выбора (смотри ниже) позволяет выбирать из фиксированного набора значений.

Dict – область, которая содержит пары: ключ – значение. Является итерабельной. Атрибуты:

- min_length (тип: Int): Определяет минимальное количество пунктов в словаре. По умолчанию None – отсутствие ограничения.
- max_length (тип: Int): Определяет максимальное количество пунктов в словаре. По умолчанию None – отсутствие ограничения.
- key_type (тип: Field): Тип всех ключей словаря.
- value_type (тип: Field): Тип всех значений словаря

Choice – область допускает выбор одного значения из предусмотренного набора. Вы можете задать набор как простую последовательность (список или кортеж) или определить словарь значений (ссылкой или именем). Словари обеспечивают гибкий список, допускающий изменение набора допустимых значений в системе. Атрибуты:

- vocabulary (тип: Vocabulary): экземпляр словаря для задания доступных значений. Этот атрибут – None, если имя словаря было определено и поле не было связано с контекстом. Сам словарь должен быть определен в файле конфигурации
- vocabularyName (тип: TextLine): Имя словаря, который используется для задания значений. Словарь с этим именем может просматриваться только, когда поле связано (имеет контекст). Конструктор получает аргумент, который определяет статический набор значений. Эти значения немедленно преобразовываются в статический словарь.

Object – область определяет объект, который должен реализовывать специфицированную схему. Допустимы только объекты, обеспечивающие указанную схему. Атрибуты:

- schema (тип: Interface): Эта область обеспечивает ссылку на схему, которая должна быть реализована объектами.

DottedName – производная от BytesLine область DottedName допускает только правильные точечные имена в стиле Питона (объектные ссылки). Эта область используется, когда необходимо задание правильного точечного имени Питона. Эта область не имеет других атрибутов.

URI – производная от области BytesLine, всегда гарантирующая правильный URI. Это особенно полезно, когда Вы хотите ссылаться на ресурсы (как например, CSS или рисунки) в удаленных компьютерах. Эта область не имеет других атрибутов.

Id – идентификатор. Области DottedName и URI, имеют поле Id. Любое точечное имя или URI представляет правильный id в Zope. Id используются для идентификации многих типов объектов, как например, разрешения, принципалы и для обеспечения ключей аннотирования. Это поле не имеет других атрибутов.

InterfaceField – поле интерфейса не имеет другие атрибуты. Значение должно быть объектом, который обеспечивает zope.interface.Interface и должно быть интерфейсом.

Для полного формального описания Schema/Field API, смотрите документацию по API в ***http://localhost:8080/++apidoc++*** или модуль ***zope.schema.interfaces***.

7.3 Словари в задании полей

Словарные области обеспечивают динамический состав позиций в элементах выбора, не зависящий от определения самой схемы. Словарь может быть создан как некоторый объект ZODB, как запрос к внешней базе данных, как содержание файла, созданным другим процессом или задан статически в некотором отдельном коде пакета.

Словарь является объектом, который содержит набор уникальных строковых значений. Уникальность касается как статических, так и динамически формируемых наборов. Словарный атрибут в схеме может содержать как словарный объект с интерфейсом IBaseVocabulary, так и имя словаря. Если использовано имя, то словарь должен быть предварительно зарегистрирован в системе. Например, в пакете ais для обеспечения словаря с данными в поле выбора, заданного в схеме оператором:

```
subjItem = Choice(
    title=_(u"Subject/Item"),
    description=_(u"Subject and Item."),
    vocabulary="itemTests",
    default=u"",
    required=True)
```

требуется в конфигурационном файле пакета ais поместить регистрационный тег:

```
<vocabulary
    name="itemTests"
    factory="ais.runtest.ItemTestVocabulary" />
```

В модуле runtest.py папки ais должен быть определен метод, указанный в регистрации атрибутом factory="ais.runtest.ItemTestVocabulary", например:

```
def ItemTestVocabulary(context):
    global vocab
    if hasattr(context, 'context'):
        # поиск корневого контейнера
        root = context.context
        while root.__parent__.__name__ <> None:
            root = root.__parent__
        lst = list(root.values())    # Список объектов корневой папки
        # Заполнение списка имен объектов для словаря
        vocab = []
        subj = None
        # просмотр списка
        while lst:
            item = removeSecurityProxy(lst[-1])
            del lst[-1]
            # Если объект нужного класса, то добавить в список пар имен и объектов
            if type(item).__name__ == 'ItemTest':
                subj = item.__parent__.title
                name = item.title
                vocab.append((subj+"."+name, item))
            elif (type(item).__name__ == 'Folder') or
                        (type(item).__name__ == 'SubjItem'):
                # Если объект папка, то добавить ее содержимое в список для просмотра
                for it in item.values():
                    lst.append(it)
            elif type(item).__name__ == 'Subj':
                subj = item.title
                vocab.append((subj+".*", subj+".*"))
                for it in item.values():
                    lst.append(it)
        # Формируем возвращаемый словарь только из имен объектов
        return SimpleVocabulary([SimpleTerm(name, name) for name, test in vocab]
)
    else:
        return SimpleVocabulary([SimpleTerm("", "")])
```

В приведенном примере словарь формируется динамически из имен объектов интересующего нас класса ItemTest, содержащихся во всех контейнерах сайта.

7.4 Формы и виджеты

Формы в Zope более частные, чем схемы, и находятся в пакете zope.app.forms. Видимые элементы полей схемы называются виджетами. Виджеты отвечают за данные, отображают и преобразовывают их в нужное представление. Виджеты главным образом используются для организации окон просмотра на языке HTML.

Виджеты разделены на две группы: отображения и ввода. Визуальные виджеты очень просты и отображают текстовое представление объекта Питона.

Входные виджеты более сложные и имеют большое число вариантов исполнения. Следующий список показывает все доступные базовые окна для ввода значений (описания находятся в zope.app.form.browser):

Текстовые элементы

Группа текстовых виджетов всегда требует ввод данных с клавиатуры. Представление поля всегда строка, которая затем преобразовывается в объект Питона, подобно целому или дате.

TextWidget: – самый простой текстовый входной элемент, используется в TextLine для строк в уникоде. Он служит в качестве базового для многих окон.

TextAreaWidget: – отображает текстовую область и принимает ввод некоторой строки уникода. (Publisher сам заботится о перекодировании символов).

BytesWidget, BytesAreaWidget: – потомки от TextWidget и TextAreaWidget, единственное различие то, что эти окна ожидают байты и не строки уникода, что соответствует кодировке ASCII.

ASCIIWidget: – основан на BytesWidget, гарантирует, что только символы ASCII являются входными данными.

PasswordWidget: – идентичен TextWidget, но отображает и вводит пароль вместо текстового элемента.

IntWidget: – производный от TextWidget, перекрывает метод преобразования, чтобы гарантировать преобразование в целое.

FloatWidget: – производный от TextWidget, перекрывает метод преобразования, чтобы гарантировать преобразование в плавающую точку.

DatetimeWidget: – простой TextWidget со строкой datetime. Есть также DateWidget, который оперирует с датой.

Булевские элементы

Группа виджетов (Boolean Widgets) для ввода двоичных значений True или False представлены следующими элементами.

CheckBoxWidget: – отображает один переключатель (checkbox), который может быть включен или нет, соответствуя логическому значению True и False.

BooleanRadioWidget: – две радиокнопки для задания истинного и ложного значения. Можно задать текстовые этикетки для двух состояний как аргументы конструктора (по умолчанию – “on” и “off” или их перевод на другие языки).

BooleanSelectWidget, BooleanDropdownWidget: – подобен BooleanRadioWidget с текстовым представлением состояния True и False для выбора значения. Смотрите SelectWidget и DropdownWidget, соответственно, если нужна дополнительная информация.

Элементы для одиночного выбора

Виджеты, которые допускают выбор одного элемента из списка значений и словаря запроса. Так называемые прокси используются для отображения области запроса в пары для словаря запроса. Например, ChoiceInputWidget берет область Choice и объект запроса, который ищет другой виджет, зарегистрированный для поля Выбора, его словарь и запрос. Ниже приведен список всех доступных виджетов, которые требуют ввода.

SelectWidget: – обеспечивает многострочный элемент выбора, где опции заполнены из записей словаря. Если область не задана, то будет доступен выбор “no value”. Пользователю разрешается выбирать только одно значение.

DropdownWidget: – производный от SelectWdiget с одной строкой и кнопкой, которая распахивает выпадающий список.

RadioWidget: – отображает радио кнопку для каждого ключа в словаре. Радио кнопки имеют небольшое преимущество, они всегда показывают все выборы и хорошо пригодны для небольших словарей.

Элементы для множественного выбора

Эта группа виджетов используется, чтобы отображать входные формы с коллекциями, например, списком или множеством. Подобно элементам одиночного выбора, используются прокси для показа. Первый шаг должен отображать из области запроса в область значений – CollectionInputWidget. Это позволяет использовать другие элементы, когда тип является Int или областью выбора. Второй используется, чтобы преобразовывать значение в пару для словаря запроса, когда тип значения является областью выбора.

MultiSelectWidget – создает элемент выбора с множеством атрибутов, устанавливаемыми в истину. Он создает элемент с множественным выбором и особенно полезен для словарей со многими элементами. Если словарь поддерживает интерфейс запроса, то можно отфильтровать его пункты с использованием запроса.

MultiCheckBoxWidget: – подобен MultiSelectWidget, но допускает множественный выбор из заданного списка и вместо списка использует переключатели. Он подходит для небольших словарей.

TupleSequenceWidget: – используется, чтобы добавлять новые значения к кортежу. Другие входные элементы используются для удаления значений.

ListSequenceWidget: – является эквивалентом предыдущему, за исключением того, что генерирует списки вместо кортежей.

Дополнительные виджеты

FileWidget – отображает элемент ввода имени файла и убеждается, что полученные данные соответствуют реальному файлу. Это поле является идеальным для быстрой загрузки байтовых потоков, что необходимо для области Bytes.

ObjectWidget – вид для объектной области. Использует схему объекта, чтобы создать входную форму. Объектная фабрика, которая задается как аргумент конструктора, используется для последующего построения объекта из вводимых данных.

Иже приведен простой пример диалога в интерпретаторе Питона, демонстрирующий предоставление и преобразование данных виджетами:

```
1  >>> from zope.publisher.browser import TestRequest
2  >>> from zope.schema import Int
3  >>> from zope.app.form.browser import IntWidget
4  >>> field = Int(__name__='number', title=u'Number', min=0, max=10)
5  >>> request = TestRequest(form={'field.number': u'9'})
6  >>> widget = IntWidget(field, request)
7  >>> widget.hasInput()
```

```
8  True
9  >>> widget.getInputValue()
10  9
11  >>> print widget().replace(' ', '\n  ')
12  <input
13    class="textType"
14    id="field.number"
15    name="field.number"
16    size="10"
17    type="text"
18    value="9"
19
20    />
```

Строки 1 и 5: Для видов, включая виджеты, всегда нужен объект запроса. Класс TestRequest является быстрым и легким способом создать запрос без значительных усилий. Класс берет аргумент формы, который является словарем значений в формате HTML. Виджет позже имеет доступ к этой информации.

Строка 2: Импортирует поле для целого.

Строки 3 и 6: Импортируют виджет для отображения и преобразования целого, получаемого из формы HTML. Инициализация виджета требует только поле и запрос.

Строка 4: Создает поле для целого с ограничением, что значение должно лежать между 0 и 10. Аргумент __name__ должен быть передан здесь, так как область не инициализирована внутри интерфейса.

Строки 7-8: Этот метод проверяет, содержала ли форма значение для этого виджета.

Строки 9-10: Если это так, то мы можем использовать метод getInputValue(), чтобы возвратить преобразованное и проверенное значение (в данном случае целое). Если мы получили целое за пределами этого диапазона, то будет поднято исключение WidgetInputError.

Строки 11-20: Отображает HTML представление виджета. Вызов метода replace() необходим для лучшей удобочитаемости выхода.

Отметим, что обычно Вы не должны иметь дело с этими методами вручную, так как генератор формы и преобразователь данных делает всю работу за Вас. Единственный метод, который Вы обычно перепишете – _validate(), который используется для проверки введенного значения. Это дает нам право использовать следующий виджет, модифицированный с учетом потребности приложения пользователя.

Есть два пути модификации виджетов для пользователя. Для небольших заданий некоторых параметров (свойств) виджета, можно использовать поддирективы browser:widget директивы browser:addform и browser:editform. Например, чтобы изменить виджет для области "name", может быть использован следующий код ZCML.

```
1  <browser:addform
2    ... >
3
```

```
4   <browser:widget
5       field="name"
6       class="zope.app.form.browser.TextWidget"
7       displayWidth="45"
8       style="width: 100%"/>
9
10  </browser:addform>
```

В этом случае, мы заставляем систему использовать TextWidget для поля **name**, устанавливая его ширину в 45 символов и добавляя атрибут стиля, который должен попытаться установить всю доступную ширину окошка для входных значений.

Другая возможность изменять виджет поля – написать свой класс CustomTextWidget для вида. Заказные виджеты легко реализуются с использованием суперкласса CustomWidget. Приведем небольшой пример:

```
1  from zope.app.form.widget import CustomWidget
2  from zope.app.form.browser import TextWidget
3
4  class CustomTextWidget(TextWidget):
5     ...
6
7  class SomeView:
8     name_widget = CustomWidget(CustomTextWidget)
```

Строка 1: Импорт CustomWidget – независимый тип представления, определенный в пакете zope.app.form.widget.

Строки 4-5: Вы просто расширяют существующий виджет. Здесь Вы можете перекрыть все, включая метод _validate().

Строки 7-8: Вы можете перекрыть стандартные средства ввода заказным виджетом, добавляя атрибут name_widget, являющийся именем области. Значение атрибута является экземпляром CustomWidget, который имеет только один аргумент конструктора – заказной виджет для поля. Также могут определяться и другие ключевые аргументы виджета.

8 Директивы определения конфигураций

8.1 Система конфигурирования

В силу имеющегося разделения труда между программистами, администраторами, менеджерами контента, дизайнерами и другими специалистами, отвечающих за функциональность вэб приложения необходимы средства интеграции в единое целое отдельных фрагментов. Таким средством в Zope3 является система конфигурирования. Система конфигурирования Zope3 основана на использовании директив конфигурации и обеспечивает расширяемый механизм для поддержки различных типов конфигураций. Система конфигурации расширяема за счет метаязыка, позволяющего определять новые директивы конфигурации. Новая директива определяется заданием метаданных о директиве и коде обработчика директивы. Предполагается, что язык конфигураций может быть даже сменным. По умолчанию сейчас используется декларативный язык, основанный на языке XML. Разработчики имеют в своем распоряжении полный набор директив, которые позволяют создать нужную конфигурацию приложения.

Процесс обработки файла конфигурации состоит из трех этапов. На первом этапе, директивы обрабатываются, чтобы создать план действий по конфигурированию. Действия по существу являются отложенными вызовами функций. Это позволяет произвести семантический анализ задания, в том числе обнаружить противоречия до его фактического исполнения. Два или большее количество действий противоречат друг другу, если у них одинаковый дискриминатор, определяющий характер назначений. Система конфигурации имеет внутренние правила для разрешения конфликтов. Если конфликты не могут быть разрешены, произойдет ошибка. Разрешение противоречия сводится обычно к игнорированию одного из противоречивых действий так, чтобы остальные первоначально противоречивые действия больше не конфликтовали. На последующих этапах непротиворечивые действия выполняются в таком порядке, который обеспечивает каждое последующее вызываемое действие передачей ему необходимых позиционных и ключевых аргументов.

Есть четыре типа директив.

- Простые директивы, определяющие действия конфигурации. Их обработчики являются обычными функциями, которые получают контекст, нуль или более ключевых аргументов и возвращают последовательность действий конфигурации. Чтобы посмотреть, как создаются простые директивы, загляните в файл test_simple.py пакета zope/configuration/tests.

- Групповые директивы, содержащие информацию, которая нужна вложенным в них директивам. Они вызываются с контекстным объектом, который они адаптируют для некоторого интерфейса, являющегося потомком интерфейса IConfigurationContext. Для того чтобы узнать, как создавать групповые директивы, посмотрите документацию в файле zopeconfigure.py пакета zope/configuration, который и обеспечивает реализацию директивы configure. Другие директивы могут быть вложены в групповую директиву. Для того чтобы узнать, как реализовать вложенные директивы, смотрите документацию в файле test_nested.py.

- Комплексные директивы, имеющие внутренние поддирективы. Поддирективы имеет обработчики, являющиеся методами комплексных директив. Комплексные директивы обрабатываются фабриками, обычно классами, которые

создают объекты, имеющие методы для обработки поддирективы. Эти объекты также имеют __call__ методы, которые вызываются при завершении обработки поддирективы. Сложные директивы существуют, чтобы поддерживать старые обработчики. Они в будущем, вероятно, исчезнут совсем.

- Вложенные поддирективы в комплексные директивы. Они похожи на простые директивы, но они определяют обработчики – методы сложных директив. Существуют поддирективы подобные сложным директивам, чтобы поддерживать старые обработчики директив. Они, вероятно, также исчезнут в будущем.

Разработка любого прикладного компонента выполняется в среде Питона, а не в оболочке Zope3. Поэтому существуют средства конфигурации, которые сообщают системе, как компоненты работают вместе, чтобы функционировал прикладной сервер. Это делается с использованием языка ZCML. При этом учитываются и роли каждого из участников разработки. Предполагается, что программист пишет код и некоторый начальный фрагмент конфигурации, а системный администратор его доопределяет, добавляя и удаляя функциональное назначение или управляя установками параметров сервера. Системные администраторы часто не являются программистами, поэтому для них нежелательно задавать конфигурацию на языке Питон. Администратору достаточно знать специализированный язык конфигурации и требования оболочки Zope3. Так как конфигурация пишется не на Питоне, очень важно, чтобы было хорошее взаимодействие с представлением информации в программах компонент на Питоне. Это касается ссылок на атрибуты в модулях Питона, запрос перевода нужных строк для конечного пользователя и др. Язык конфигурации декларативный, он не обеспечивает программирование логики действий, а лишь описание связей объектов приложения и их параметров. Разработка новых типов приложений и прикладных компонент иногда может потребовать расширение языка конфигурации.

Чтобы удовлетворить перечисленным требованиям, в системе Zope3 используется язык на основе XML (Extensible Markup Language). Преимущество XML в том, что это стандартный формат, известный многим специалистам. Кроме того, это дает возможность при обработке конфигураций для выполнения грамматического разбора текстов использовать стандартные модули языка Питон.

XML - это производный от SGML язык разметки документов, позволяющий структурировать информацию разного типа, используя для этого произвольный набор инструкций. XML-документ представляет собой обычный текстовый файл, в котором при помощи специальных тегов создаются элементы данных, последовательность и вложенность которых определяет структуру документа и его содержание. Номенклатура и правила вложенности тегов определяются в специальном файле DTD – определений структур допустимых данных.

Чтобы гарантировать уникальность вводимых в употребление имен директив, в XML должны использоваться уникальные универсальные неповторяющиеся префиксы (аналог GUID в COM-технологиях). Для этого применяется механизм Namespaces – пространств имен. Спецификация Namespaces была официально утверждена W3C и является частью стандарта XML. Согласно этой спецификации, для определения принадлежности тега необходимо определить уникальный атрибут, описывающий название элемента, по которому анализатор документа сможет определить, к какому пространству имен он относится. По правилам XML идентификаторы пространства имен могут применяться для описания уникальных названий, как элементов, так и их атрибутов, например:

```
<configure
```

```
    namespace="http://namespaces.org/my-namespace">

  <namespace:directive
      attr="value">

    <namespace:sub-directive
        attr1="value"
        attr2="This is a long value that
               spans over two lines."
        />

  </namespace:directive>
</configure>
```

Ниже приведен краткий список наиболее важных пространств имен, используемых в файлах конфигурации Zope3:

zope = "http://namespaces.zope.org/zope" – наиболее общее и фундаментальное пространство имен, поскольку позволяет регистрировать все основные структуры с компонентной архитектурой. Это пространство имен обычно используется по умолчанию, если у директивы не задано иное.

browser = "http://namespaces.zope.org/browser" – содержит все директивы, которые имеют дело с выходом HTML, включая управление общим видом и слоями, объявляет новые виды (страницы) и ресурсы, а также установки автоматически генерируемых форм.

meta = "http://namespaces.zope.org/meta" – для расширения перечня доступных директив ZCML.

xmlrpc = "http://namespaces.zope.org/xmlrpc" – эквивалент browser, за исключением того это допускает определение методов компонента, которые должны быть доступны через XML-RPC.

i18n – содержит всю специфику интернационализации и локализации для конфигурации. Используя registerTranslations можно зарегистрировать новые каталоги сообщений с доменом перевода.

help = "http://namespaces.zope.org/help" – регистрация новых страниц подсказок в системе помощи. Это дает контекстно-зависимую помощь для экранов ZMI ваших продуктов.

mail = "http://namespaces.zope.org/mail" – установки компонентов почтового отправления, которые ваше приложение может использовать для посылки писем.

Зарезервированными являются следующие пространства имен:

```
local          = ""
inh            = “inherit”
renderer       = "http://namespaces.zope.org/renderer"
code           = "http://namespaces.zope.org/code"
startup        = "http://namespaces.zope.org/startup"
workflow       = "http://namespaces.zope.org/workflow"
dav            = "http://namespaces.zope.org/dav"
rdb            = "http://namespaces.zope.org/rdb"
tales          = "http://namespaces.zope.org/tales"
server-control = "http://namespaces.zope.org/server-control"
```

```
fssync      = "http://namespaces.zope.org/fssync"
etc         = "http://namespaces.zope.org/etc"
event       = "http://namespaces.zope.org/event"
gts         = "http://namespaces.zope.org/gts"
```

Средства описания пространства имен XML помогают группировать тексты директив конфигурации по функциональному назначению.

Каждый шаг конфигурации определяется директивой, являющейся тегом XML. Все директивы конфигурационного файла помещены в блочный тег **configure**, который задает начало конфигурации. При открытии этого тега всегда указывается атрибуты пространства имен, которые используются в файле конфигурации. Имена интерфейсов и классов упоминаются с использованием соответствующей точечной нотации языка Питон. Конфигурационный тег может также содержать атрибут i18n_domain, который содержит указание домена, используемого для извлечения переводов строк при передаче их конечному пользователю, содержащихся в текстах файла конфигурации.

Как и везде в Zope3, в языке ZCML есть несколько соглашений об именах и способе кодирования. По умолчанию файл конфигурации должен иметь имя configure.zcml. В файле необходимо объявить атрибуты xmlns с заданием пространства имен, которые действительно будут использоваться в конфигурации. При написании директив рекомендуется логические группы директив записывать вместе и использовать комментарии. Комментарии пишутся с использованием общего синтаксиса языка XML: <!--.Это комментарий -->. Более подробную информацию о стиле записи конфигурационных файлов можно найти в электронном ресурсе http://dev.zope.org/Zope3/ZCMLStyleGuide.

Для будущих расширений областей применения Zope3 имеется возможность пополнения директив языка ZCML с использованием метадиректив. Каждая директива может быть полностью описана четырьмя компонентами: именем, принадлежностью к пространству имен, схемой и обработчиком (handler) директивы, например:

```
1  <meta:directive
2      namespace="http://namespaces.zope.org/zope"
3      name="adapter"
4      schema=".metadirectives.IAdapterDirective"
5      handler=".metaconfigure.adapterDirective" />
```

Эти метадирективы обычно помещаются в файле meta.zcml. Схема директивы обычно находится в файле metadirectives.py. Это обычные схемы Zope3, чьи области описывают доступные атрибуты для директивы. Система конфигурации использует области, чтобы преобразовывать и проверять значения параметров конфигурации при исполнении. Например, точечные имена автоматически преобразуются в объекты Питона. Есть несколько специализированных полей для конфигурации окружения:

PythonIndentifier – область описывает идентификатор языка Питон, например, простое имя переменной.

GlobalObject – может быть доступно как глобальный объект модуля, например, класс, функция или константа.

Tokens – последовательность, которая может быть прочитана из разделенной пробелами строки; value_type описывает тип признака.

Path – файловое имя пути, которое может быть использовано как относительный путь. Введенные пути преобразовываются в абсолютные пути.

Bool – Расширенное логическое значение. Значения могут быть введены (в нижнем или верхнем регистре) как любое слово из набора: yes, no, y, n, true, false, t или f.

MessageID – строка текста, которая должна быть переведена для пользователя. Следовательно, схема директивы является единственным местом, где нужно иметь дело с интернационализацией.

Обработчик обычно находится в файле metaconfigure.py – функциональный или другой вызываемый объект, который знает, что должно быть сделано с данной информацией директивы. Вот простой пример (упрощенный код):

```
1  def adapter(_context, factory, provides, for_, name=''):
2
3      _context.action(
4          discriminator = ('adapter', for_, provides, name),
5          callable = provideAdapter,
6          args = (for_, provides, factory, name),
7          )
```

Первый аргумент обработчика – всегда _context переменная, которая имеет аналогичную функцию, как self в классах. Он обеспечивает общие методы, необходимые для обработки директив. Следующие аргументы являются атрибутами директивы (и их имена должны быть соответствующими). Если имя атрибута совпадает с ключевым словом Питона, как в примере, тогда в имя атрибута добавляется подчеркивание.

Обработчик непосредственно сам не должен выполнять действие до тех пор, пока система не проверит всю конфигурацию и не обнаружит возможные конфликты и накладки. В связи с этим объект _context имеет метод, регистрирующий действие, которое нужно выполнить в исполнительной части процесса конфигурации. Первый аргумент является дискриминатором (discriminator, смотри примеры ниже), который однозначно определяет директиву. Другой важный параметр путь включения файлов для директивы (includepath). Он добавляется в каждое действие и используется при разрешении конфликтов. Атрибут includepath является кортежем имен включенных друг в друга файлов конфигурации.

Действия противоречат друг другу, если у них имеется одинаковый и не пустой дискриминатор. Противоречивые действия могут быть разрешены, если путь включения includepath одного из действий является префиксом, но целиком не совпадает с includepath другого из противоречивых действий. Когда мы имеем противоречивые директивы, мы можем исключить их, если одна из противоречивых директив была в файле, который включен во все другие пути (смотри zope\configuration\tests\test_xmlconfig.py). Такое поведение для правильного перекрытия требует, чтобы все перекрывающие директивы были расположены в одном файле, обычно в файле самого верхнего уровня по включению. Это не очень удобно. Эту трудность можно преодолеть с использованием директивы includeOverrides, позволяющей перекрыть действия других директив, вложенных в перекрываемые файлы. Перекрытие делает более приоритетными директивы из

перекрывающего файла при разрешении конфликтных ситуаций. Давайте посмотрим на примере, как это работает.

Пусть есть файл **bar.zcml**

```
<configure>
  <include file="bar1.zcml" />
  <include file="bar2.zcml" />
</configure>
```

Он включает файл **bar1.zcml**

```
<configure xmlns="http://namespaces.zope.org/test">
  <include file="configure.zcml" />
  <foo x="blah" y="1" />
</configure>
```

и файл **bar2.zcml**

```
<configure xmlns="http://namespaces.zope.org/test">
  <include file="bar21.zcml" />
  <foo x="blah" y="2" />
  <foo x="blah" y="1" />
</configure>
```

Файл bar1.zcml включает файл **configure.zcml**

```
<configure xmlns="http://namespaces.zope.org/zope"
               xmlns:meta="http://namespaces.zope.org/meta"
               xmlns:test="http://namespaces.zope.org/test"
  >
  <meta:directive
      namespace="http://namespaces.zope.org/test"
      name="foo"
      schema="zope.configuration.tests.samplepackage.foo.S1"
      handler="zope.configuration.tests.samplepackage.foo.handler"
      />
  <test:foo x="blah" y="0" />
</configure>
```

и имеет директиву foo.

Файл bar2.zcml включает **bar21.zcml**

```
<configure xmlns="http://namespaces.zope.org/test">
  <foo x="blah" y="0" />
  <foo x="blah" y="2" />
</configure>
```

и имеет директиву foo, которая противоречит такой же в bar1.zcml. Файл bar2.zcml также перекрывает foo директиву в bar21.zcml. Файл bar21.zcml имеет foo директиву, которая противоречит таковой в configure.zcml.

Давайте посмотрим, что случается, когда мы пытаемся обрабатывать файл bar.zcml.

```
>>> import unittest
>>> import os
>>> from zope.testing.doctestunit import DocTestSuite
>>> from zope.configuration import xmlconfig, config
>>> from zope.configuration.tests.samplepackage import foo
>>> from pprint import PrettyPrinter, pprint
```

```
>>> context = config.ConfigurationMachine()
>>> xmlconfig.registerCommonDirectives(context)
>>> path = 'c:\\python24\\lib\\site-
packages\\zope\\configuration\\tests\\samplepackage\\bar.zcml'
>>> xmlconfig.include(context, path)
>>> pprint=PrettyPrinter(width=70).pprint
>>> pprint(clean_actions(context.actions))
[{'discriminator': (('x', 'blah'), ('y', 0)),
  'includepath': ['tests/samplepackage/bar.zcml',
                  'tests/samplepackage/bar1.zcml',
                  'tests/samplepackage/configure.zcml'],
  'info': 'File "tests/samplepackage/configure.zcml", line 12.2-
12.29'},
 {'discriminator': (('x', 'blah'), ('y', 1)),
  'includepath': ['tests/samplepackage/bar.zcml',
                  'tests/samplepackage/bar1.zcml'],
  'info': 'File "tests/samplepackage/bar1.zcml", line 5.2-5.24'},
 {'discriminator': (('x', 'blah'), ('y', 0)),
  'includepath': ['tests/samplepackage/bar.zcml',
                  'tests/samplepackage/bar2.zcml',
                  'tests/samplepackage/bar21.zcml'],
  'info': 'File "tests/samplepackage/bar21.zcml", line 3.2-3.24'},
 {'discriminator': (('x', 'blah'), ('y', 2)),
  'includepath': ['tests/samplepackage/bar.zcml',
                  'tests/samplepackage/bar2.zcml',
                  'tests/samplepackage/bar21.zcml'],
  'info': 'File "tests/samplepackage/bar21.zcml", line 4.2-4.24'},
 {'discriminator': (('x', 'blah'), ('y', 2)),
  'includepath': ['tests/samplepackage/bar.zcml',
                  'tests/samplepackage/bar2.zcml'],
  'info': 'File "tests/samplepackage/bar2.zcml", line 5.2-5.24'},
 {'discriminator': (('x', 'blah'), ('y', 1)),
  'includepath': ['tests/samplepackage/bar.zcml',
                  'tests/samplepackage/bar2.zcml'],
  'info': 'File "tests/samplepackage/bar2.zcml", line 6.2-6.24'}]
```

Как Вы можете видеть, существует много конфликтов (действия с тем же дискриминатором). Некоторые из них могут быть решены, но многие не могут, если мы пытаемся исполнить действия:

```
>>> try:
...    v = context.execute_actions()
... except config.ConfigurationConflictError, v:
...    pass
>>> print clean_text_w_paths(str(v))
Conflicting configuration actions
  For: (('x', 'blah'), ('y', 0))
    File "tests/samplepackage/configure.zcml", line 12.2-12.29
        <test:foo x="blah" y="0" />
    File "tests/samplepackage/bar21.zcml", line 3.2-3.24
        <foo x="blah" y="0" />
```

```
  For: (('x', 'blah'), ('y', 1))
    File "tests/samplepackage/bar1.zcml", line 5.2-5.24
        <foo x="blah" y="1" />
    File "tests/samplepackage/bar2.zcml", line 6.2-6.24
        <foo x="blah" y="1" />
```

Отметьте, что конфликты для (('x', 'blah'), ('y', 2)), не включены в сообщения об ошибках, поскольку они могли бы быть разрешены по общим правилам. Давайте попробуем все заново, используя директиву includeOverrides. Мы включим **baro.zcml**,

```
<configure>
  <include file="bar1.zcml" />
  <includeOverrides file="bar2.zcml" />
</configure>
```

который включает bar2.zcml как Overrides – перекрытие.

```
>>> context = config.ConfigurationMachine()
>>> xmlconfig.registerCommonDirectives(context)
>>> path = 'c:\\python24\\lib\\site-
packages\\zope\\configuration\\tests\\samplepackage\\baro.zcml'
>>> xmlconfig.include(context, path)

>>> pprint(clean_actions(context.actions))
[{'discriminator': (('x', 'blah'), ('y', 0)),
  'includepath': ['tests/samplepackage/baro.zcml',
                  'tests/samplepackage/bar1.zcml',
                  'tests/samplepackage/configure.zcml'],
  'info': 'File "tests/samplepackage/configure.zcml", line 12.2-
12.29'},
 {'discriminator': (('x', 'blah'), ('y', 1)),
  'includepath': ['tests/samplepackage/baro.zcml',
                  'tests/samplepackage/bar1.zcml'],
  'info': 'File "tests/samplepackage/bar1.zcml", line 5.2-5.24'},
 {'discriminator': (('x', 'blah'), ('y', 0)),
  'includepath': ['tests/samplepackage/baro.zcml'],
  'info': 'File "tests/samplepackage/bar21.zcml", line 3.2-3.24'},
 {'discriminator': (('x', 'blah'), ('y', 2)),
  'includepath': ['tests/samplepackage/baro.zcml'],
  'info': 'File "tests/samplepackage/bar2.zcml", line 5.2-5.24'},
 {'discriminator': (('x', 'blah'), ('y', 1)),
  'includepath': ['tests/samplepackage/baro.zcml'],
  'info': 'File "tests/samplepackage/bar2.zcml", line 6.2-6.24'}]
```

Анализ запланированных действий конфигурации позволяет сделать следующие выводы:

- противоречивые действия между bar2.zcml и bar21.zcml разрешились;
- остальные действия (после разрешения противоречий) из bar2.zcml и bar21.zcml имеют ключ includepath, который они бы имели, если бы были определены в baro.zcml, что перекрывает действия из bar1.zcml и configure.zcml;

- можно выполнять действия без проблем, так как все конфликты являются разрешимыми.

8.2 Общие директивы пространства имен

В директивах конфигурации используются атрибуты – аргументы, позволяющие задать параметры создаваемой конфигурации приложения. Основное назначение и типы общих аргументов директив конфигурации следующие:

- alias – PythonIdentifier (по умолчанию = None), дополнительное имя.
- allowed_attributes – (по умолчанию = None) доступные атрибуты объекта, если получатель имеет соответствующее разрешение.
- allowed_interface – (по умолчанию = None) доступные интерфейсы, если получатель имеет соответствующее разрешение.
- attributes – (по умолчанию = None) атрибуты для обеспечения доступа.
- category – Bool (по умолчанию = False), является ли группа категорией.
- class – GlobalObject (по умолчанию = None), класс объекта.
- content_factory – GlobalObject (по умолчанию = None), фабрика содержимого –. вызываемый объект для создания новых объектов контента. Этот атрибут не используется, если определен класс, который реализует createAndAdd.
- content_factory_id – (по умолчанию = None) Id фабрики для создания новых объектов контента.
- component – GlobalObject (по умолчанию = None), компонент, который нужно использовать.
- description – TextLine (по умолчанию = None), обеспечивает описание для объекта. Обычно используется для организации подсказок в ZMI и помощи.
- factory – GlobalObject (по умолчанию = None), имя фабрики, которая может создать соответствующую реализацию объекта. Идентифицирует объект в модуле с использованием полного точечного имени.
- fields – (по умолчанию = None) определяют имена областей, которые необходимо отображать. Порядок в этом списке совпадает с отображением областей. Если этот атрибут не определен, все области будут отображены в порядке, определенном в схеме.
- filter – TextLine (по умолчанию = None), условие для отображения пункта меню. Пункт меню будет отображен, если есть фильтр и он равен True.
  Условие задается как выражение языка TALES. Выражение имеет доступ к следующим переменным:

    context – объект, для которого отображается меню;

    request -- запрос окна просмотра;

    nothing – None.
- for – GlobalInterface (по умолчанию = None), определяет интерфейсы, для которых объявлен объект.

- handler - GlobalObject (по умолчанию = None). Вызываемый объект, который обрабатывает события.
- icon – TextLine (по умолчанию = None), путь для ресурса иконки, представляющей пункт меню.
- id – (по умолчанию = None) Id использованного объекта.
- interface – (по умолчанию = None) интерфейсы для обеспечения доступа. Доступ будет предусмотрен ко всем именам, определенным интерфейсом. Можно задать несколько интерфейсов.
- keyword_arguments – (по умолчанию = None) ключевые аргументы фабрики.
- label – MessageID (по умолчанию = None), этикетка в заголовке формы.
- layer – GlobalObject (по умолчанию = None), слой.
- like_class – GlobalObject (по умолчанию = None), сообщает, что этот класс содержимого будет сконфигурирован так же как заданный класс. Если этот аргумент задан, то никакой другой аргумент не может быть использован.
- locate – Bool (по умолчанию = False), локальное расположение объекта.
- login – TextLine (по умолчанию = None), имя пользователя.
- menu – MenuField (по умолчанию = None), имя пункта меню.
- module – GlobalObject (по умолчанию = None), имя модуля.
- name – TextLine (по умолчанию = None), имя объекта.
- order – Int (по умолчанию = 0), относительная позиция пункта в меню.
- password – TextLine (по умолчанию = None), пароль пользователя.
- permission – Permission (по умолчанию = None), уровень разрешений на доступ к объекту для принципала.
- principal – (по умолчанию = None) Id принципала.
- provides – GlobalInterface (по умолчанию = None), задает интерфейс, который должен обеспечиваться экземпляром объекта.
- role – (по умолчанию = None) Id роли пользователя.
- set_after_add – (по умолчанию = None) установки после добавления. Список областей, которые нужно задать для вновь созданного объекта после того, как он будет добавлен.
- set_attributes – (по умолчанию = None) список атрибутов, которые могут быть модифицированы.
- set_before_add – (по умолчанию = None) установки перед добавлением. Список областей, которые нужно назначить на вновь созданном объекте прежде, чем он будет добавлен.
- set_schema – (по умолчанию = None) атрибуты, определенные схемой, могут быть установлены или модифицированы.
- schema – GlobalInterface (по умолчанию = None), схема для создания формы ввода данных.
- template – (по умолчанию = None) путь для шаблона формы.

- title – TextLine (по умолчанию = None), название для объекта.
- trusted – Bool (по умолчанию = False), определяет надежность адаптера. Надежные адаптеры открывают доступ к объектам, которые они адаптируют. Если запрошено адаптировать security-proxied объекты, тогда доступен security-proxied адаптер объекта.
- type – GlobalInterface (по умолчанию = None), тип запроса.
- unique – Bool (по умолчанию = Ложь), уникальность. Определяет уникальность средства для менеджера сайта.
- view – TextLine (по умолчанию = None), имя вида для добавления.

Внимание! Символом * далее помечены обязательные атрибуты директив. В скобках для краткости после названия директивы приведено пространство имен, где она определена.

8.2.1 configure

(Все пространства имен). Директива конфигурации configure является группирующей директивой. Она сама не выполняет каких либо действий, но содержит имя пакета, влияющего на интерпретацию относительных точечных имен и файловых путей. Она также задает область i18n, которую нужно использовать. Указанная информация используется вложенными директивами конфигурации. Аргументы директивы:

- xmlns – определяет пространство имен.
- i18n_domain – BytesLine область интернационализации (по умолчанию = None). Это имя программного проекта должно быть правильным системным файловым именем, так как будет использовано для создания и поиска папки, содержащей переводы строк.
- package – GlobalObject (по умолчанию = None). Пакет, относительно которого нужно искать импортируемые файлы.

Пример.

```
<configure
   xmlns="http://namespaces.zope.org/zope"
   i18n_domain="zope"
   >
```

8.2.2 include

(Все пространства имен). Эта директива позволяет включать другой файл ZCML в конфигурацию, что позволяет записывать конфигурационные файлы в отдельных пакетах, а затем связывать их вместе. Аргументы директивы:

- file – BytesLine (по умолчанию = None) относительное имя одного файла конфигурации, который нужно включить.
- files – BytesLine (по умолчанию = None) Относительные имена или шаблоны нескольких файлов конфигурации, которые нужно включить. Шаблоны могут содержать символы:

  * – соответствует 0 или больше символов

? – соответствует единственному символу

[<seq>] – соответствует любому символу из seq

[!<seq>] – соответствует любому символу **не** из seq

Файловые имена включаются в указанном порядке.

- package – GlobalObject (по умолчанию = None), Включить поименованный файл (или configure.zcml) из папки указанного пакета.

Пример.

```
<include package="zope.app.component" />
<include file="browser.zcml" />
<include files="package-includes/*-meta.zcml" />
```

8.2.3 includeOverrides

(Все пространства имен). Эта директива позволяет включить другой файл ZCML в конфигурацию в режиме перекрытия ранее заданных директив. Аргументы директивы:

- file – BytesLine (по умолчанию = None) относительное имя одного файла конфигурации, который нужно включить.
- files – BytesLine (по умолчанию = None) Относительные имена или шаблоны нескольких файлов конфигурации, которые нужно включить. Шаблоны могут содержать символы:

* – соответствует 0 или больше символов

? – соответствует единственному символу

[<seq>] – соответствует любому символу из seq

[!<seq>] – соответствует любому символу **не** из seq

Файловые имена включаются в указанном порядке.

- package – GlobalObject (по умолчанию = None), Включить поименованный файл (или configure.zcml) из папки этого пакета.

Пример.

```
<includeOverrides file="overrides_ftesting.zcml" />
```

8.3 Директивы группы zope

Директивы группы Zope определяют общие свойства объектов содержимого сайта и средства их поддержки.

8.3.1 adapter

(http://namespaces.zope.org/zope). Зарегистрировать адаптер, описываемый аргументами: factory*, for, locate, name, permission, provides, trusted. Пример.

```
  <adapter
    for="buddydemo.interfaces.IBuddy"   <!— Входной интерфейс -->
    provides=".interfaces.IPostalInfo"  <!— Выходной интерфейс -->
    factory=".browser.BuddyCityState"   <!— Класс фабрики -->
    permission="zope.Public"            <!— Разрешение -->
  />
```

8.3.2 allow

(http://namespaces.zope.org/zope). Разрешает доступ к выбранным атрибутам модуля. Доступ разрешен к любым именам, указанным в атрибуте attributes или к именам, определенным в указанных в атрибуте interface интерфейсах. Аргументы директивы: attributes, interface.

8.3.3 authenticatedGroup

(http://namespaces.zope.org/zope). Определяет группу для аутентификации. Аргументы директивы: id*, title*, description. Пример.

```
<authenticatedGroup
  id="zope.Authenticated"
  title="Authenticated Users"
  />
```

8.3.4 class

(http://namespaces.zope.org/zope). Дополнительные утверждения о классе, заданным аргументом class *, которые нельзя задать в коде класса.

Поддирективы директивы class

- **implements** – объявляет, что класс реализует интерфейсы, заданные аргументом interface*.
- **require** – указывает список имен, которые требуют заданное разрешение для доступа. Аргументы поддирективы: attributes, interface, like_class, permission, set_attributes, set_schema.
- **allow** – определяет, что часть класса будет доступна для просмотра всем (требует разрешение zope.Public). Только один из двух атрибутов может быть использован: attributes или interface.
- **factory** – определяет фабрику для создания объекта содержимого. Аргументы поддирективы: description, id (по умолчанию устанавливается как атрибут 'class' директивы content), title.

Пример.

```
<class class=".wikipage.MailSubscriptions">
  <require
     permission="zwiki.EditWikiPage"
     attributes="getSubscriptions"
  />
</class>
```

8.3.5 content

(http://namespaces.zope.org/zope). Задает атрибутом class* класс контент-объектов,.

Поддирективы директивы content:

- **implements** – объявляет, что данный класс контента реализует интерфейсы, заданные аргументом interface*.

- **require** – объявляет список имен, которые требуют данное разрешение для доступа. Аргументы поддирективы: attributes, interface, like_class, permission, set_attributes, set_schema.
- **allow** – объявляет, что часть класса будет доступна для просмотра всем посетителям (требует разрешение уровня zope.Public). Только один из двух атрибутов может быть использован: attributes или interface.
- **factory** – определяет фабрику для создания контент-объекта. Обычно это имя того же класса, что и в директиве content, используемое как конструктор. Аргументы поддирективы: description, id, title.

Пример.

```
<content class=".buddy.Buddy">
    <implements
      interface="zope.app.annotation.IAttributeAnnotatable"
    />
    <require permission="zope.View"
        interface=".interfaces.IBuddy" />
    <require permission="zope.ManageContent"
        set_schema=".interfaces.IBuddy" />
  </content>
```

8.3.6 defaultLayer

(http://namespaces.zope.org/zope), Связать слой по умолчанию с типом. Аргументы директивы: layer*, type*. Пример.

```
<defaultLayer
  type="zope.publisher.interfaces.browser.IBrowserRequest"
  layer="zope.publisher.interfaces.browser.IDefaultBrowserLayer"
/>
```

8.3.7 defaultView

(http://namespaces.zope.org/zope). Задание имени вида по умолчанию (если никакое имя вида не задано явно). Аргументы директивы: name*, type*, for, provides. Пример.

```
<defaultView
    for="zope.security.interfaces.IUnauthorized"
    type="zope.publisher.interfaces.http.IHTTPRequest"
    name="index.html"
    />
```

8.3.8 everybodyGroup

(http://namespaces.zope.org/zope), Определяет группу пользователей. Аргументы директивы: id , title*, description. Пример.

```
<everybodyGroup
  id="zope.Everybody"
  title="All Users" />
```

8.3.9 factory

(http://namespaces.zope.org/zope). Определяет фабрику. Аргументы директивы: component*, description, id, title. Используется в том случае, когда

фабрика определяется не конструктором класса, а через компонент утилиту, который реализует интерфейс IFactory.. Пример.

```
<factory
     component=".plaintext.PlainTextSourceFactory"
     id="zope.source.plaintext"
     title="Plain Text"
     description="Plain Text Source" />
```

8.3.10 grant

(http://namespaces.zope.org/zope) Определяет разрешения для роли и/или принципала. Аргументы директивы: permission, principal, role. Должны быть заданы любые два аргумента, определяющих операцию. Пример.

```
<grant permission="zope.View"
       role="zope.Anonymous" />
```

8.3.11 grantAll

(http://namespaces.zope.org/zope) Определяет все разрешения для роли или принципала. Аргументы директивы: principal, role. Должен быть задан один любой аргумент, определяющий операцию. Пример.

```
<grantAll role="zope.Manager" />
```

8.3.12 interface

(http://namespaces.zope.org/zope), Определяет назначение интерфейса внутри приложения. Аргументы директивы: interface*, type. Например:

```
<interface
  interface=".interfaces.IBuddy"
  type="zope.app.content.interfaces.IContentType"/>
```

Директива нужна в дополнение к объявлению интерфейса с именем IBuddy для указания на то, что объекты с IBuddy еще реализуют и интерфейс IContentType, который в свою очередь определяет: если его обеспечивает объект, то это контент. Можно рассматривать просто как признак контента для объектов с интерфейсом IBuddy. Позволяет реализовывать своеобразное позднее связывание при конфигурировании (а не при написании кода на Питоне) классов объектов и их интерфейсов. Заметим, что интерфейс IContentType не имеет ни атрибутов, ни методов и выступает в роли признака контента при регистрации интерфейсов.

8.3.13 localService

(http://namespaces.zope.org/zope) Утверждения о классе для локального сервиса. Аргументы директивы: class*.

Поддирективы директивы localService

- **implements** – класс реализует данный интерфейс. Аргументы поддирективы: interface*
- **require** – указывает список имен или имена данного интерфейса, требующие данное разрешение для доступа. Аргументы поддирективы: attributes, interface, like_class, permission, set_attributes, set_schema.

- **allow** – часть класса будет доступна для просмотра всем (требует разрешение zope.Public). Только один из следующего двух атрибутов может быть использован: attributes или interface.
- **factory** – определяет фабрику для создания объекта. Аргументы поддирективы: description, id, title.

8.3.14 localUtility

(http://namespaces.zope.org/zope). Утверждения о классе для локальной утилиты. Аргументы директивы: class*.

Поддирективы директивы localUtility

- **implements** – класс реализует данный интерфейс. Аргументы поддирективы: interface*
- **require** – указывает список имен или имена данного интерфейса, требующие данное разрешение для доступа. Аргументы поддирективы: attributes, interface, like_class, permission, set_attributes, set_schema.
- **allow** – часть класса будет доступна для просмотра всем (требует разрешение zope.Public). Только один из следующего двух атрибутов может быть использован: attributes или interface.
- **factory** – определяет фабрику для создания объекта содержимого. Аргументы поддирективы: description, id, title.

Пример.

```
<localUtility class=".error.ErrorReportingUtility">
   <factory id="zope.app.ErrorLogging" />
   <require
       permission="zope.Public"
       interface=".interfaces.IErrorReportingUtility"
       />
 </localUtility>
```

8.3.15 module

(http://namespaces.zope.org/zope). Группа деклараций для модуля. Аргументы директивы: module*. Пример.

```
<module module=".interfaces">
  <allow attributes="ISite" />
</module>
```

8.3.16 modulealias

(http://namespaces.zope.org/zope). Определить новый псевдоним модуля. Аргументы директивы: alias*, module*. Пример.

```
<modulealias
  module="zope.app.error"
  alias="zope.app.errorservice" />
```

8.3.17 permission

(http://namespaces.zope.org/zope). Определить новое разрешение. Аргументы директивы: id*, title*, description. Пример.

```
<permission id="schoolbell.view" title="View" />
```

8.3.18 preferenceGroup

(http://namespaces.zope.org/zope). Регистрировать группу предпочтения. Аргументы директивы: title*, category, description, id, schema. Пример.

```
<preferenceGroup
    id="apidoc"
    title="API Doc Tool"
    description="
       These are all the preferences related to viewing the API
       documentation."
    category="True"
    />
```

8.3.19 principal

(http://namespaces.zope.org/zope), Определить нового принципала. Аргументы директивы: id*, login*, password*, title*, description. Пример.

```
<principal
  id="zope.manager"
  title="Manager"
  login="admin"
  password="ddd"
  />
```

8.3.20 require

(http://namespaces.zope.org/zope). Определить требуемое разрешение для доступа к выбранным атрибутам модуля. Данное разрешение действует непосредственно к именам, указанным в атрибуте attributes или определенных в списке интерфейсов, перечисленных в атрибуте interface. Аргументы директивы: permission*, attributes, interface. Пример.

```
<zope:require
    permission="zope.Security"
    attributes="roles rolesInfo id title description" />
```

8.3.21 resource

(http://namespaces.zope.org/zope), Регистрировать ресурс. Аргументы директивы: name*, type*, allowed_attributes, allowed_interface, component, factory, layer, permission, provides. Пример.

```
<browser:resource
    name="buddy.css"
    file="buddy.css"
    layer="buddy" />
```

8.3.22 role

(http://namespaces.zope.org/zope), Определяет новую роль. Аргументы директивы: id*, title*, description. Пример.

```
<role
  id="zwiki.User"
  title="Wiki User"
```

```
  description="Wiki visitors" />
```

8.3.23 securityPolicy

(http://namespaces.zope.org/zope). Определяет полис безопасности, который будет использован Zope. Аргументы директивы: component*. Пример.

```
<securityPolicy
component="zope.app.securitypolicy.zopepolicy.ZopeSecurityPolicy"
/>
```

8.3.24 subscriber

(http://namespaces.zope.org/zope), Регистрировать подписчика. Аргументы директивы: factory, for, handler, locate, permission, provides, trusted. Пример.

```
<subscriber
  factory=".wikipage.mailer"
  for="zope.app.container.interfaces.IObjectRemovedEvent"
/>
```

8.3.25 unauthenticatedGroup

(http://namespaces.zope.org/zope), Определить нерегистрируемую группу. Аргументы директивы: id*, title*, description. Пример.

```
<unauthenticatedGroup
  id="zope.Anybody"
  title="Unauthenticated Users"
  />
```

8.3.26 unauthenticatedPrincipal

(http://namespaces.zope.org/zope), Определить нового нерегистрируемого принципала. Аргументы директивы: id*,title*, description. Пример.

```
<unauthenticatedPrincipal
  id="zope.anybody"
  title="Unauthenticated User" />
```

8.3.27 utility

(http://namespaces.zope.org/zope), Регистрировать утилиту. Аргументы директивы: component, factory, name, permission, provides. Нельзя одновременно использовать атрибуты component и factory. Пример.

```
<utility
  factory=".stubpostal.Lookup"
  provides=".interfaces.IPostalLookup"
  permission="zope.Public"
/>
```

8.3.28 view

(http://namespaces.zope.org/zope). Регистрировать адаптер вида для компонента, реализующего указанный атрибутом for интерфейс. Аргументы директивы: for*, factory*, type*, allowed_attributes, allowed_interface, class, layer, name, permission, provides. Пример.

```
<zope:view
    for="schoolbell.app.interfaces.ISchoolBellApplication"
    type="zope.publisher.interfaces.browser.IBrowserRequest"
    provides="zope.publisher.interfaces.browser.IBrowserPublisher"
    factory=".app.SchoolBellApplicationTraverser"
    permission="zope.Public"
    />
```

В приведенном примере атрибут for указывает на главный объект приложения, для которого будут использоваться декларированные свойства просмотра, ссылкой на реализуемый им интерфейс. Атрибут type указывает на тип запросов, которые будут обслуживаться этим просмотром. В данном примере это общий запрос, направляемый браузером клиента. Атрибут provides указывает на реализуемый просмотром интерфейс, используемый при поиске просмотра в реестре. Атрибут factory указывает на класс с заданным программистом способом заимствования объектов при прослеживании для публикации в шаблонах страниц.

8.3.29 vocabulary

(http://namespaces.zope.org/zope) Определение поименованного словаря. Связывает имя словаря с фабрикой в глобальном реестре словарей. Каждое имя может быть определено только один раз. Дополнительные ключевые аргументы могут быть переданы фабрике добавлением атрибутов за указанными здесь. Это может быть полезно при использовании словарей, осуществляющих различную фильтрацию.

Аргументы директивы: factory*, name*. Пример:

```
<vocabulary
    name="garys-favorite-path-references"
    factory="zope.app.gary.paths.Favorites" />
```

8.4 Директивы группы browser

Директивы этой группы задают особенности отображения информации на экране браузера клиента.

8.4.1 addMenuItem

(http://namespaces.zope.org/browser). Определить пункт меню «Добавить». Аргументы директивы: title*, class, description, factory, filter, for, icon, menu, order, permission, view. Пример.

```
<browser:addMenuItem
  class=".buddy.Buddy"
  title="Buddy"
  permission="zope.ManageContent"
  view="AddBuddy.html" />
```

8.4.2 addform

(http://namespaces.zope.org/browser) Определение автоматически генерируемой формы «Добавить». Директива addform создает и регистрирует вид для добавления объекта на основе схемы. Добавление объекта сложнее, чем его редактирование, поскольку объекта еще нет, поэтому используется схема при отображении формы. Аргументы директивы: name*, permission*, schema*, arguments,

class, content_factory, content_factory_id, description, fields, for (интерфейс, который поддерживает вид), keyword_arguments, label, layer, menu, set_after_add, set_before_add, template, title.

Поддирективы директивы addform:

- **widget** – регистрация прикладных виджетов для формы. Эта директива позволяет быстро генерировать виджеты для формы. Кроме двух необходимых аргументов, области и класса, можно определить любой набор ключевых аргументов, например, style='background-color:#fefefe;'. Ключевые слова будут загружены как атрибуты в экземпляр виджета. Чтобы увидеть значение ключевых слов, нужно просмотреть код виджета соответствующего класса. Аргументы поддирективы: field*, class.

Пример.

```
<browser:addform
  schema=".interfaces.IBuddy"
  label="Add Buddy information"
  content_factory=".buddy.Buddy"
  arguments="first last email address postal_code"
  name="AddBuddy.html"
  permission="zope.ManageContent" />
```

8.4.3 addview

(http://namespaces.zope.org/browser). Директива определяет вид, который имеют подстраницы. Подстраницы, предусмотренные определяемым видом, доступны для просмотра по имени вида и страничному имени. Аргументы директивы: permission*, allowed_attributes, allowed_interface, class, for, layer, menu, name, provides, title.

Поддирективы:

- **page** – определение страницы. Аргументы директивы: name*, attribute, template.

- **defaultPage** – определение страницы по умолчанию. Аргументы директивы: name.

8.4.4 addwizard

(http://namespaces.zope.org/browser). Определение помощника для автоматической генерации формы добавления (многостраничная форма). Директива addwizard создает и регистрирует вид для дополнения объекта на основе схемы. Аргументы директивы: name*, permission*, schema*, arguments, class, content_factory, content_factory_id, description, for, keyword_arguments, layer, menu, set_after_add, set_before_add, template, title, use_session.

Поддирективы:

- **pane** – определение панелей (страниц) помощника. Аргументы директивы: fields*, label.

8.4.5 containerViews

(http://namespaces.zope.org/browser). Определить контейнерные виды для реализации IContainer. Аргументы директивы: for*, add, contents, index (разрешения, необходимые для добавления страниц, их контента и индексации), layer. Пример.

```
<containerViews
     for="..groupfolder.IGroupFolder"
     contents="zope.ManageServices"
     index="zope.ManageServices"
     add="zope.ManageServices"  />
```

8.4.6 defaultSkin

(http://namespaces.zope.org/browser). Устанавливает по умолчанию имя обличья окна просмотра. Аргументы директивы: name*. Пример.

```
<browser:defaultSkin name="zope.app.rotterdam.Rotterdam" />
```

8.4.7 defaultView

(http://namespaces.zope.org/browser). Имя вида по умолчанию. Аргументы директивы: name*, for, layer. Пример.

```
<defaultView
    for="zope.interface.common.interfaces.IException"
    type="zope.publisher.interfaces.http.IHTTPRequest"
    name="GET"
    />
```

8.4.8 editform

(http://namespaces.zope.org/browser). Определение автоматически сгенерированной формы для редактирования. Директива editform создает и регистрирует вид для редактирования объекта, основанный на схеме. Аргументы директивы: name*, permission*, schema*, class, fields, for, label, layer, menu, template, title.

Поддирективы:

- **widget** – регистрирует виджеты для формы. Директива позволяет быстро генерировать заказные виджеты для формы. Кроме двух необходимых аргументов, области и класса, Можно определить любую последовательность ключевых аргументов, например, style='background-color:#fefefe;'. Ключевые слова будут загружены как атрибуты в экземпляр виджета. Аргументы директивы: field*, class.

Пример.

```
<browser:editform
    schema="zwiki.interfaces.IWikiPage"
    for="zwiki.interfaces.IWikiPage"
    label="Change Wiki Page"
    name="edit.html"
    permission="zwiki.EditWikiPage"
    fields="source type"
    class=".wikipage.EditWikiPage"
    menu="zmi_views" title="Edit" />
```

8.4.9 editwizard

(http://namespaces.zope.org/browser). Определение помощника для автоматической генерации формы редактирования (многостраничная форма). Директива addwizard создает и регистрирует вид для дополнения объекта на основе схемы. Аргументы директивы: name*, permission*, schema*, class, description, for, layer, menu, template, title, use_session.

Поддирективы:

- **pane** – определение панелей (страниц) помощника. Аргументы директивы: fields*, label.

8.4.10 form

(http://namespaces.zope.org/browser). Определение автоматически сгенерированной формы. Директива формы не требует данные, которые нужно загружать в контекст, но допускает хранимую процедуру для метода to. Аргументы директивы: class*, name*, permission*, schema*, fields, for, label, layer, menu, template, title.

Поддирективы

- **widget**

Регистрирует виджеты для формы. Директива позволяет быстро генерировать заказные директивы виджетов для формы. Кроме двух необходимых аргументов, области и класса, можно определить любой набор ключевых аргументов, например, style='background-color:#fefefe;'. Ключевые слова будут загружены как атрибуты в экземпляре виджета. Чтобы видеть, какие ключевые слова значимы, можно посмотреть код виджета определенного класса. Аргументы директивы: field*, class*. Пример.

```
<editform
    for="schoolbell.app.interfaces.IGroup"
    name="edit.html"
    label="Edit Group Information"
    schema="schoolbell.app.interfaces.IGroup"
    fields="title description"
    permission="schoolbell.edit"
    template="templates/simple_edit.pt"
    class=".app.GroupEditView"
    menu="schoolbell_actions"
    title="Edit Info"
    />
```

8.4.11 i18n-resource

(http://namespaces.zope.org/browser). Определяет ресурс i18n, используемый для автоматического перевода сообщений. Аргументы директивы: name*, defaultLanguage, layer, permission.

Поддирективы

- **translation**

Поддиректива для II18nResourceDirective. Аргументы директивы: language*, file, image, layer, permission.

8.4.12 icon

(http://namespaces.zope.org/browser). Определить иконку для интерфейса. Аргументы директивы: for*, name*, file, layer, resource, title. Пример.

```
<icon
    name="zmi_icon"
    for="schoolbell.app.interfaces.ISchoolBellApplication"
    file="resources/icon.png"
    />
```

8.4.13 layer

(http://namespaces.zope.org/browser). Определяет слой окна просмотра при разработке нового обличья. Задаются имя слоя или интерфейс. Если Вы определяете имя, тогда интерфейс слоя будет создан на основе имени и базового интерфейса. Если Вы определяете имя и интерфейс, тогда слой будет зарегистрирован дважды. Последнее используется для обратной совместимыми. Слой доступен через свое точечное имя. Если Вы не определяете базу, тогда IBrowserRequest используется по умолчанию. Нельзя определять одновременно атрибуты интерфейса и базы. Аргументы директивы: base, interface, name. Пример.

```
<browser:layer name="buddy" />
```

8.4.14 menu

(http://namespaces.zope.org/browser). Определить меню окна просмотра. Аргументы директивы: class, description, id, interface, title. Пример.

```
<browser:menu id="help_actions" />
```

8.4.15 menuItem

(http://namespaces.zope.org/browser). Определить один пункт меню. Аргументы директивы: action*, for*, menu*, title*, description, filter, icon, layer, order, permission,.

8.4.16 menuItems

(http://namespaces.zope.org/browser). Определение группы пунктов меню окна просмотра. Эта директива полезна, когда несколько пунктов меню определено для того же интерфейса и меню. Аргументы директивы: for*, menu*, layer.

Поддирективы

- **menuItem**

Определение пункта в пределах группы пунктов меню. Аргументы директивы: action*, title*, description, filter, icon, order, permission.

- **subMenuItem**

Определение пункта меню, который представляет подменю. Аргументы директивы: submenu*, title*, action, description, filter, icon, order, permission. Пример.

```
<browser:menuItems
    menu="save"
    for="zope.interface.Interface">

  <browser:menuItem
      action="javascript:alert('Save All')"
```

```
       title="Save All"
       permission="zope.Public"
       icon="/@@/filesave.png"
       />

   <browser:menuItem
       action="javascript:alert('Save As...')"
       title="Save As ..."
       permission="zope.Public"
       icon="/@@/filesaveas.png"
       />

</browser:menuItems>
```

8.4.17 page

(http://namespaces.zope.org/browser). Страничная директива создает виды, которые обеспечивают url или страницу. Страничная директива создает новый класс вида из заданного шаблона и/или класса и регистрирует его. Аргументы директивы: name*, permission*, allowed_attributes, allowed_interface, attribute, class, for, layer, menu, template, title. Пример.

```
<browser:page
  for="buddydemo.interfaces.IBuddy"
  name="index.html"
  template="info.pt"
  permission="zope.View"
  class=".browser.BuddyInfo"
/>
```

8.4.18 pages

(http://namespaces.zope.org/browser). Определение многих страниц, не повторяя атрибуты for, permission, class, layer, allowed_attributes и allowed_interface. Аргументы директивы: permission*, allowed_attributes, allowed_interface, class, for, layer.

Поддирективы

- **page**

Поддиректива для IPagesDirective. Аргументы директивы: name*, attribute, menu, template, title.

Пример.

```
<browser:pages
     for="zwiki.interfaces.IWiki"
     class=".wikipage.MailSubscriptions"
     permission="zwiki.EditWikiPage">
     <browser:page name="subscriptions.html"
template="subscriptions.pt"
         menu="zmi_views" title="Subscriptions" />
     <browser:page name="changeSubscriptions.html"
attribute="change" />
 </browser:pages>
```

8.4.19 resource

(http://namespaces.zope.org/browser), Определяет ресурс окна просмотра. Аргументы директивы: name*, factory, file, image, layer, permission, template. Пример.

```
<browser:resource
    name="buddy.css" file="buddy.css" layer="buddy" />
```

8.4.20 resourceDirectory

(http://namespaces.zope.org/browser), Определяет директорий, содержащий ресурсы окна просмотра. Аргументы директивы: directory*, name*, layer, permission. Пример.

```
<browser:resourceDirectory
    name="tree_images"
    directory="images" />
```

8.4.21 schemadisplay

(http://namespaces.zope.org/browser). Определяет автоматически сгенерированную дисплейную форму. Директива schemadisplay создает и регистрирует вид для отображения объекта, основанного на схеме. Аргументы директивы: name*, permission*, schema*, class, fields, for, label, layer, menu, template, title.

Поддирективы

- **widget**

Регистрирует виджеты для формы. Директива позволяет быстро генерировать заказные директивы виджетов для формы. Кроме двух необходимых аргументов, области и класса, можно определить любую сумму ключевых аргументов, например, style='background-color:#fefefe;'. Ключевые слова будут загружены как атрибуты в экземпляре виджета. Чтобы увидеть, какие ключевые слова значимы, можно посмотреть код виджета определенного класса. Аргументы директивы: field*, class. Пример.

```
<schemadisplay
    schema="..principalfolder.IInternalPrincipalContainer"
    label="Principal Folder Prefix"
    name="prefix.html"
    fields="prefix"
    permission="zope.ManageServices"
    menu="zmi_views" title="Prefix" />
```

8.4.22 skin

(http://namespaces.zope.org/browser). Определяет обличье для окна просмотра. Если Вы не определяете интерфейс, тогда он автоматически будет создан для Вас на основе имени использованием слоев как базовых интерфейсов. В случае если Вы определяете интерфейс и имя, обличье будет доступно через точечное имя интерфейса и имя. Вы не можете определить как интерфейс, так и атрибуты слоев. Аргументы директивы: interface, layers, name. Пример.

```
<browser:skin name="buddy" layers="buddy rotterdam default" />
```

8.4.23 subMenuItem

(http://namespaces.zope.org/browser). Определить один пункт меню. Аргументы директивы: for*, menu*, submenu*, title*, action, description, filter, icon, layer, order, permission. Пример.

```
<browser:subMenuItem
    submenu="openrecent"
    title="Open Recent"
    permission="zope.Public"
    icon="/@@/fileopen.png"
    />
```

8.4.24 subeditform

(http://namespaces.zope.org/browser), Определять форму subedit. Аргументы директивы: name*, permission*, schema*, class, for, fulledit_label, fulledit_path, label, layer, template.

Поддирективы

- **widget**

Регистрирует виджеты для формы. Директива позволяет Вам быстро генерировать заказные директивы виджетов для формы. Кроме двух необходимых аргументов, области и класса, Вы можете определить любую сумму ключевых аргументов, например, style='background-color:#fefefe;'. Ключевые слова будут загружены как атрибуты в экземпляре виджета. Чтобы видеть какие ключевые слова значимы, Вы должны посмотреть код виджета определенного класса. Аргументы директивы: field*, class.

8.4.25 tool

(http://namespaces.zope.org/browser). Директива для создания новых утилит инструментальных средств. Аргументы директивы: interface*, description, folder, title, unique. Пример.

```
<tool
    interface="zope.app.cache.interfaces.ICache"
    title="Caches"
    description="Caches can be used to make your site run faster."
    />
```

8.4.26 view

(http://namespaces.zope.org/browser). Директива вида определяет вид, который имеет подстраницы. Страницы, предусмотренные определяемым видом, доступны сначала для просмотра по имени вида и затем по страничному имени. Аргументы директивы: permission*, allowed_attributes, allowed_interface, class, for, layer, menu, name, provides, title.

Поддирективы

- **page.** Аргументы директивы: name*, attribute, template.
- **defaultPage**. Аргументы директивы: name*.

Пример.

```
<browser:view
    name="login"
    for=".interfaces.ILoginPasswordPrincipalSource"
    class="zope.app.pluggableauth.PrincipalAuthenticationView"
    permission="zope.Public" />
```

8.5 Директивы группы dav

8.5.1 provideInterface

(http://namespaces.zope.org/dav). Директива назначает новый интерфейс для компонента. Этот интерфейс будет доступен через WebDAV для этого компонента. Аргументы директивы: for*, interface*. Пример.

```
<dav:provideInterface
    for="http://purl.org/dc/1.1"
    interface="zope.app.dublincore.interfaces.IZopeDublinCore" />
```

8.6 Директивы групп help

8.6.1 register

(http://namespaces.zope.org/help). Зарегистрировать тему online. Дополнительно Вы можете зарегистрировать тему для компонента и вида. Аргументы директивы: doc_path*, id*, title*, class, for, parent, resources, view. Пример.

```
<help:register
    id="boston"
    title="Boston Skin (experimental)"
    doc_path="README.txt"
    class="zope.app.onlinehelp.onlinehelptopic.
          RESTOnlineHelpTopic"
    />
```

8.7 Директивы группы i18n

8.7.1 registerTranslations

(http://namespaces.zope.org/i18n) Регистрация директория с переводами сообщений . Аргументы директивы: directory*. Пример.

```
<i18n:registerTranslations directory="locales" />
```

8.8 Директивы группы мета

8.8.1 complexDirective

(http://namespaces.zope.org/meta). Информация, которую должны иметь все директивы верхнего уровня (не subdirectives), Аргументы директивы: handler*, name*, schema*, usedIn*. Пример.

```
<meta:complexDirective
```

```
    name="class"
    schema=".metadirectives.IClassDirective"
    handler=".contentdirective.ContentDirective"
    >
```

8.8.2 directive

(http://namespaces.zope.org/meta). Информация, которую должны иметь все директивы верхнего уровня (не subdirectives). Аргументы директивы: handler*, name*, schema*, usedIn*. Пример.

```
<meta:directive
     name="adapter"
     schema=".metadirectives.IAdapterDirective"
     handler="zope.app.component.metaconfigure.adapter"
     />
```

8.8.3 directives

(http://namespaces.zope.org/meta). Схемы для директивы директив. Аргументы директивы: namespace*. Пример.

```
<meta:directives namespace="http://namespaces.zope.org/zope">
```

8.8.4 groupingDirective

(http://namespaces.zope.org/meta). Информация для всех директив верхнего уровня (не subdirectives). Аргументы директивы: handler*, name*, schema*, usedIn*. Пример.

```
<meta:groupingDirective
     name="module"
     namespace="http://namespaces.zope.org/zope"
     schema=".metadirectives.IModule"
     handler="zope.configuration.config.GroupingContextDecorator"
           />
```

8.8.5 provides

(http://namespaces.zope.org/meta). Информация для директивы <meta:provides> . Аргументы директивы: feature*. Пример.

```
<meta:provides feature="apidoc" />
```

8.8.6 redefinePermission

(http://namespaces.zope.org/meta). Определить разрешение заменой на другое разрешение. Аргументы директивы: from*, to*. Пример.

```
<meta:redefinePermission
    from="zope.View"
    to="zope.Security"
    />
```

8.8.7 subdirective

(http://namespaces.zope.org/meta). Общая для всех определений директив информация. Аргументы директивы: name*, schema*. Пример.

```
<meta:subdirective
```

```
    name="implements"
    schema=".metadirectives.IImplementsSubdirective"
    />
```

8.9 Директива renderer

(http://namespaces.zope.org/renderer). Регистрация генератора для выходного интерфейса, например, IBrowserView. Аргументы директивы: factory*, for*, sourceType*.

8.10 Директива tales

8.10.1 expressiontype

(http://namespaces.zope.org/tales). Регистрировать тип новых TALES выражений. Аргументы директивы: handler*, name*. Пример.

```
<tales:expressiontype
    name="pagelets"
    handler=".tales.TALESPageletsExpression"
    />
```

8.11 Директивы xmlrpc

8.11.1 view

(http://namespaces.zope.org/xmlrpc) Директива вида для методов XML-RPC. Аргументы директивы: for*, class, interface, methods, name, permission. Пример.

```
<xmlrpc:view
     for="zope.i18n.interfaces.ITranslationDomain"
     permission="zope.ManageContent"
     methods="getAllLanguages getMessagesFor"
     class=".methods.Methods"
     />
```

9. Метаданные

Любая продвинутая система имеет необходимость определять метаданные для различных нужд, особенно для объектов, которые составляют содержимое сайта. Для публикующей среды подобно Zope3 важно иметь стандартный комплект полей метаданных для всех контент-объектов. В продукте CMF Zope2, для этого был использован стандарт Dublin Core.

Данные и метаданные принципиально отличаются следующим. Данные – внутренняя информация объекта, передаваемая по наследству. Они представляют состояние или позволяют идентифицировать объект. Метаданные, с другой стороны, – внешняя информация об объекте и его содержимом. Нет необходимости для объекта содержать внутри себя функции, поставляющие такую информацию (методы объекта не должны зависеть от метаданных), но метаданные могут быть важными для работы с объектом в окружающей среде, обеспечивая дополнительную информацию для идентификации, каталогизации, индексирования, интеграции с другими системами и т.п.

Один из стандартных наборов метаданных определен в спецификации "Dublin Core" (смотри dublincore.org). Dublin Core обеспечивает дополнительную информацию о контент-объектах, как например, название, описание (реферат) и авторы. В Dublin Core (DC) все элементы являются списками и могут иметь много значений. Элементы DC полезны, поскольку они покрывают многие наиболее общие схемы метаданных. Хороший пример – участник сайта. Это редактор, переводчик или соавтор контента. Эта информация помогает, если нужно найти человека, последний раз модифицировавшего объект или публикацию. Все элементы DC и как они реализованы, хорошо описаны в интерфейсах модуля ZOPE3/src/zope/app/interfaces/dublincore.py.

Элементы Dublin Core

Следующий список элементов Dublin Core заимствован из ресурса http://dublincore.org/documents/2003/02/04/dces/ и снабжен некоторыми комментариями их реализации в среде Zope3.

Title – удобочитаемое имя ресурса. В Zope3 имя ресурса является уникальной строкой в пределах своего контейнера. Тем не менее, имена объектов часто не понятны конечному пользователю. Элемент Title используется для идентификации объекта для конечного пользователя.

Creator – сущность, создавшая ресурс. Создатель в Zope - специальный экземпляр принципала, который может использовать много вариантов для задания элемента. Обычно это конструктор приложения. Система сообщает его имя в поле формы ввода или изменения объекта.

Subject – тема содержимого ресурса. Обычно Subject задается некоторыми ключевыми словами, фразами принятой классификации, которые описывают тему ресурса. Наилучшая рекомендуемая практика – выбирать значение из пополняемого словаря или формальной системы классификации, что хорошо подходит для последующей каталогизации ресурсов.

Description – описание содержимого ресурса. Описания включают реферат, оглавление, ссылки на графическое представление содержимого или текстовый обзор содержимого. В Zope3 обычно используется описание некоторых деталей семантики объекта/ресурса, чтобы потребитель получил более полное понимание.

Publisher – сущность, ответственная за получение доступного ресурса. Издатель задается значением name/id принципала.

Contributor – сущность, ответственная за пополнение содержимого ресурса, например, человек, организация или сервис. Обычно для указания сущности используется имя участника.

Date – дата события в жизненном цикле ресурса. Обычно дата связана с созданием или опубликованием ресурса. Рекомендуемая практика для кодирования даты определена в профиле ISO 8601 [W3CDTF] и включает дату в форме YYYY-MM-DD.

Type – природа или жанр содержимого ресурса. Тип включает понятия, описывающие общие категории, функции, жанры или уровни агрегации для содержимого. Рекомендуется выбор из некоторого словаря. Для описания физического или цифрового проявления ресурса, рекомендуется элемент "Format", для контент-объектов – "Content Type". Для других объектов тип может быть просто именем зарегистрированного компонента. В Zope3 этот элемент не используется.

Format – физическое или кодированное внутреннее представление ресурса. Обычно формат может включать типы носителя или размерность ресурса. Формат используется для идентификации программного обеспечения, аппаратных средств, или другого оборудования для отображения или обслуживания ресурса. Рекомендуется выбирать значение из задаваемого словаря (например, список Internet Media Types [MIME] или определяя компьютерные форматы носителя). В Zope3 этот элемент не используется.

Identifier – однозначная ссылка на ресурс в пределах данного контекста. Рекомендуется идентифицировать ресурс строкой или числом, соответствующим выбранной формальной системе идентификации. Формальные системы идентификации – Универсальный Однородный Идентификатор Ресурса (Uniform Resource Identifier – URI), включая Универсальный Локатор Ресурса (Uniform Resource Locator – URL), Цифровой Объектный Идентификатор (Digital Object Identifier – DOI) и Международный Стандартный Книжный Номер (International Standard Book Number – ISBN). В Zope3 им может быть путь объекта или уникальный идентификатор.

Source – ссылка на источник, из которого поступил данный ресурс. Настоящий ресурс может быть производным от другого ресурса целиком или частично. Zope3 не использует этот элемент.

Language – язык содержимого ресурса. Рекомендуемая практика – использование RFC 3066, который в связи с ISO639 определяет первичные языковые теги с дополнительными подтегами. Например, "en" или "eng" для Английского, "akk" для Akkadian, и "en-GB" для Английского, использованного в Объединенном Королевстве. Обратите внимание на систему описания locales во введении в интернационализацию и локализацию.

Relation – ссылка на связанный ресурс. Рекомендуется идентифицировать ссылочный ресурс строкой или числом в соответствии с формальной системой идентификации.

Coverage – область распространения содержимого ресурса. Обычно область определяет пространственную диспозицию (географическое название или географические координаты), временной период (имя периода, даты, или диапазон дат) или юрисдикцию (например, поименованное административное образование).

Рекомендуется выбирать значение из словаря (например, Тезаурус Географических Имен [Thesaurus of Geographic Names TGN]).

Rights – информация о правах хранения и манипулирования ресурсом. Обычно права определят полномочия на управление ресурсом или ссылку на сервис, обеспечивающий такую информацию. Правовая информация включает параметры интеллектуальных прав (Intellectual Property Rights IPR), авторского права и другие права. Если элемент прав отсутствует, то никакие другие предположения не могут быть сделаны о правах на получение или манипулирование ресурсом.

10 Защита компонент

Zope3 поставляется с гибким механизмом обеспечения безопасности на основе двух фундаментальных понятий – разрешений и принципалов. Разрешения соответствуют ключам, которые открываются доступ в конкретные функции приложения. Например, нужно иметь разрешение zope.View, чтобы просматривать экран сообщений. Принципалы, с другой стороны, – пользователи системы, которые выполняют некоторые действия. Предоставление разрешения принципалам, это функция другой подсистемы – политики безопасности (security policy).

Система Zope3 не навязывает разработчикам какой либо фиксированной политики безопасности. Задача администратора тщательно выбрать полис безопасности и использовать тот, который удовлетворяет потребности принципалов и приложения наилучшим образом. Встроенный в дистрибутив Zope3 полис безопасности (zope.app.securitypolicy), поддерживает понятие ролей и разрешений. Каждый потребитель может иметь несколько ролей, но применяя их только поочередно. Например, это роли редактора и администратора. Встроенная политика безопасности поддерживает задание разрешений для принципалов через их роли.

Первая важная задача – определить установку разрешений и задать директивы для использования разрешений. Это скучное, но важное занятие, и делать это надо тщательно, так как качество безопасности приложения зависит от решения этой задачи. Большое преимущество использования Zope3 по сравнению с другими системами (например, Apache и PHP) в том, что защита объекта не требует модификации существующего кода Питона, поскольку все параметры доступа конфигурируются с использованием языка ZCML, оставляя код Питона нетронутым.

10.1 Объявление ролей и разрешений

Декларация ролей характерна для полиса безопасности по умолчанию в Zope3. Другие полисы безопасности могут не использовать понятия ролей. Для нашего пакета нам действительно нужно только две роли “User” и “Editor”, которые объявляются следующим образом:

Директивы определения безопасности должны располагаться в самом начале файла конфигурации, поскольку их определения используются в остальных директивах.

```
<role
   id="buddydemo.User"
   title="Buddy demo User"
   description="Users that actually use the Buddy demo."/>

<role
   id="buddydemo.Editor"
   title="Buddy Editor"
   description="The Editor can edit and delete Buddy."/>
```

Директива role создает и регистрирует новую роль в глобальном реестре ролей. Кроме того, атрибут id должен быть уникальным идентификатором, который используется для всего процесса конфигурации.

Для большинства приложений достаточно определить следующее основные разрешения:

- View - позволяет иметь доступ к данным. Каждый Пользователь должен иметь это разрешение.
- Add - позволяет создавать (то есть передавать) объекты контента.
- Edit - позволяет редактировать содержимое (после его создания).
- Delete - позволяет удалять объекты контента.

Определения разрешений нужно расположить вслед за определениями ролей, чтобы они были определены, когда выполняются другие директивы, использующие разрешения. Для нашего примера можно добавить четыре директивы к основному файлу configure.zcml, определяющие уровни доступа к ресурсам приложения: View, Add, Edit, Delete.

```
<permission
   id="buddydemo.View"
   title="View Buddy"
   description="View the Byddy and all its content."
   />
<permission
   id="buddydemo.Add"
   title="Add Buddy"
   description="Add Buddy."
   />
<permission
   id="buddydemo.Edit"
   title="Edit Buddy"
   description="Edit Buddy."
   />
<permission
   id="buddydemo.Delete"
   title="Delete Buddy"
   description="Delete Buddy."
   />
```

Директива permission определяет и создает новое разрешение в глобальном реестре разрешений. Атрибут id должно быть уникальным для разрешения, что можно обеспечить использованием точечных префиксов, например,buddydemo.Add. Атрибут id должно быть правильным URI или точечным именем, если нет точки в точечной версии, то будет поднято исключение ValidationError. Атрибут id используется как идентификатор в следующих шагах конфигурации.

Далее новым ролям предоставляются разрешения с использованием оператора grant, создающим связи «разрешение – роль».

```
<grant
   permission="buddydemo.View"
   role="buddydemo.User"
```

```
    />
  <grant
    permission="buddydemo.Add"
    role="buddydemo.Editor"
    />
  <grant
    permission="buddydemo.Edit"
    role="buddydemo.Editor"
    />
  <grant
    permission="buddydemo.Delete"
    role="buddydemo.Editor"
    />
```

После определения уровней разрешений их можно использовать в основном файле конфигурации buddydemo/configure.zcml для приписывания их объектам приложения. Для этого достаточно заменить имя принципала в атрибутах permission="zope.ManageContent" на имя "buddydemo.Edit", а строках permission="zope.View" на имя "buddydemo.View". Для вступления в силу изменений необходимо перезапустить Zope3.

10.2 Декларация принципалов

Для сохранения ранее созданных файлов конфигурации, зададим полисы безопасности в отдельном файле с именем buddydemo/security.zcml. Чтобы создать пакет в полном объеме, нужно обеспечить нужные для работы роли некоторым принципалам. Для этого создадим два новых принципала с именами buddyuser и buddyeditor. Директива principal создает и регистрирует в системе нового принципала с заданными атрибутами login и password для входа в систему. Подобно всем директивам безопасности для принципала требуются название id и атрибуты. Можно было бы предоставить роль пользователя принципалу anybody, чтобы все могли просматривать сообщения. Принципал anybody является неавторизованным принципалом, который определен в директиве unauthenticatedPrincipal, который имеет те же три основных атрибута директивы principal, но не имеет атрибутов login и password.

```
<configure
  xmlns="http://namespaces.zope.org/zope"
  >
  <principal
      id="buddydemo.buddyuser"
      title="Buddy demo User"
      login="buddyuser" password="buddy"
      />
  <grant
      role="buddydemo.User"
      principal="buddydemo.buddyuser"
      />

  <principal
      id="buddydemo.buddyeditor"
```

```
        title="Buddy demo Editor"
        login="buddyeditor" password="buddy"
        />
    <grant
        role="buddydemo.User"
        principal="buddydemo.buddyeditor"
        />
    <grant
        role="buddydemo.Editor"
        principal="buddydemo.buddyeditor"
        />
</configure>
```

Наконец, необходимо включить файл security.zcml в конфигурацию Zope3. Это можно сделать дополнением следующей директивы включения в файле etc/principals.zcml
<include package="buddydemo" file="security.zcml" />

Для окончательной настройки службы безопасности компонент приложения buddydemo необходимо в ZMI для папки с информацией о приятелях определить права доступа пользователей. Находясь в этой папке, выберите закладку «Права», в окне искать задайте имя «Buddy demo Editor» и в следующем окне установите в столбце «Разрешить» радио-кнопки для добавления, удаления и редактирования информации о приятелях. Не забудьте перезагрузить Zope3.

11 Интернационализация пакетов

Одним из замечательных черт системы Zope3 является встроенная поддержка перевода посылаемых клиенту сообщений в зависимости от настроек браузера, обычно определяемых страной пребывания или языком клиента.

Интернационализация (I18n) – создание инструментов для перевода сообщений, форматирования даты, времени относительно среды и места проживания клиента. Локализация (L10n) – создание файла переводов сообщений для конкретного языка. Часто перевод делается не программистом, а переводчиком (поставщиком локализации).

Локали – объекты, которые содержат информацию о конкретной физической или абстрактной области в мире, определяющей язык, диалект, денежное обращение, форматы даты, времени, чисел и так далее. Пример места действия может быть «de_DE_PREEURO» (язык, страна/область, диалект), который описывает Германию до введения Евро. Ссылка de задает правильное место действия, отсылающее на Германию – говорящий на немецком языке регион. Вы можете представить, что есть иерархия мест действия. de_DE_PREEURO – более конкретный указатель места, чем de_DE , который в свою очередь более конкретный, чем de. После того как установлено место действия - de_DE_PREEURO при поиске шаблона формата дат, система ищет его в de_DE_PREEURO , de_DE и, наконец, в de , пока он не будет найден.

11.1 Интернационализация кода Питона

Большинство переводимых строк для видов обычно помещаются в шаблонах страниц. При написании кода Питона есть несколько точек, где необходима интернационализация. Одной из них являются схемы, где определяются удобочитаемые названия, описания и значение по умолчанию для полей. Zope использует id сообщений, чтобы находить переводимые строки. Переводимые строки должны всегда принадлежать домену, выбранной области перевода.

Для создания id сообщения используется фабрика, определяемая в коде программы нашего приложения следующим образом:

```
1  from zope.i18n import MessageIDFactory
2  _ = MessageIDFactory('buddydemo')
```

Строка 2 определяет новое имя, состоящее из одного символа подчеркивания, чтобы ссылаться на объект-функцию перевода. В нашем случае, она используется как конструктор/фабрика переведенного сообщения. Аргументом конструктора MessageIDFactory является задание домена переводов строк, в нашем случае – 'buddydemo'. Конструктор создает вызываемый (callable) объект – переводчик в указанном домене.

Схемы записываются в файлах определения интерфейсов и имеют переводимые строки сообщений. Для интернационализации каждого такого сообщения нужно обратиться к функции **_(”text”)**. Например, описание поля first в схеме IBuddy может выглядеть так

```
first = TextLine(title=_("First name"))
```

То же самое преобразование необходимо заказать для всех полей всех схем в модуле интерфейсов. Название и описание полей требуют строки уникода, но так

как id сообщения использует уникод как базовый класс, то мы можем передать фабрике сообщений регулярную строку.

11.2 Интернационализация шаблонов страниц

Интернационализация шаблонов - более интересна во многих отношениях. Необходимо беспокоиться об обнаружении правильных тегов для интернационализации с учетом глубокой вложенности. Чтобы достичь интернационализации в Zope3 шаблонов, было разработано новое пространство имен i18n, определяющее домен для переводчика.

Приведем фрагмент файла info.pt, демонстрирующий основные особенности интернационализации шаблонов.

```
1. <html metal:use-macro="context/@@standard_macros/page"
   i18n:domain=.buddydemo.>
2. <body>
3. <div metal:fill-slot="body">
4. <table>
5. <caption i18n:translate="">Buddy information</caption>
6. <tr>
7. <td i18n:translate="">Name:</td>
8. <td>
   <span tal:replace="context/first">First</span>
   <span tal:replace="context/last">Last</span>
9. </td>
10.     </tr>
11.     <tr>
12.     <td i18n:translate="">Email:</td>
13.     <td tal:content="context/email">foo@bar.com</td>
14.     </tr>
```

- Строка 1 определяет спецификацию домена в теге html.
- Строки 5, 7 и 12 содержат атрибут i18n:translate="", задающий необходимость переводить строки, вложенные в соответствующие теги.

Отметьте, что нет необходимости использовать параметр i18n:attributes. Тем не менее, когда мы имеем дело с кнопками, то используем эту инструкцию довольно часто. Например:

```
1  <input type="submit" value="Add"   i18n:attributes="value add-button" />
```

В примере i18n:attributes атрибут value будет заменен переводом «add-button» или останется встроенной строкой (Add), если перевод не будет обнаружен.

11.3 Интернационализация ZCML

Интернационализация ZCML - простой шаг. Все что здесь нужно сделать, добавить i18n_domain="имя" в теге configure основного файла configure.zcml. Это - ответственность автора директивы определить, какие значения атрибутов должны быть преобразованы в сообщение id. Установка этого атрибута влияет на все предупреждения, которые Вы получаете при запуске Zope3.

11.4 Интернационализация объектов

Некоторые объекты Zope3, например, рисунки и файлы, нельзя автоматически интернационализировать с использованием системы перевода. Поэтому нужен механизм, для создания и использования версий объектов для различных языковых доменов. Для решения этой задачи в состав среды Zope3 входят поставляемые компоненты I18n Image (многоязыковый рисунок) и I18n File (многоязыковый файл). При добалении их в контейнер менеджер сайта имеет возможность задания различных соответствующих версий объектов для разных языков. Остальное берет на себя среда при публикации объекта в зависимости от языкового домена клиента.

12 Контейнеры и их содержимое

12.1 Связывание компонент по включению

Интерпретатор Питона не допускает использования имени где-либо, пока не закончено его определение. Например, при определении интерфейса некоторого объекта, являющимся контейнером для объектов того же класса есть желание включить в текст объявления интерфейса фрагмент вида:

```
class IMyObj(IContainer):
  """ ... """
  def __setitem__(name, object):
    """Add a IMyObj object."""
  __setitem__.precondition = ItemTypePrecondition(IMyObj)
```

Это будет диагностировано как ошибка, поскольку объявление интерфейса IMyObj содержит рекурсивную ссылку на само себя в ItemTypePrecondition(IMyObj). Для преодоления этих трудностей используется следующий прием. В начале создается описание вспомогательного интерфейса, например, с именем IMyObjContainer вида:

```
class IMyObjContainer(IContainer):
  """ . . . """

  def __setitem__(name, object):
        """Add a IMyObj object."""

  __setitem__.precondition = ItemTypePrecondition(IMyObj, . . .)
```

Далее для объявления возможности иметь в своем содержимом экземпляры класса MyObj создается описание вспомогательного интерфейса, например, с именем IMyObjContained вида:

```
class IMyObjContained(IContained):
  """ . . . """
  __parent__ = Field(
     constraint = ContainerTypesConstraint(IMyObj))
```

Наконец при объявлении класса MyObj включают утверждение о реализации им интерфейса IMyObjContainer, например:

```
class MyObj(Contained):
    """ . . .  """
    implements(IMyObj, IMyObjContained, IMyObjContainer)
```

Контейнер может обеспечивать задание ограничений на то, что может быть содержимым контейнера. Контейнер задает ограничения через атрибут precondition (предусловие) своего метода с именем «__setitem__» при описании интерфейса (смотри пример выше). Предусловия могут быть либо перечнем имен интерфейсов добавляемых объектов, либо вызываемым объектом, например, функцией. Если ограничение не удовлетворяется, то функция должна поднимать исключение «zope.interface.Invalid». Пример:

```
def preNoZ(container, name, ob):
  "Silly precondition example"
```

```
  if name.startswith("Z"):
    raise zope.interface.Invalid("Names can not start with Z")

class I1(zope.interface.Interface):
  def __setitem__(name, on):
    "Add an item"
  __setitem__.precondition = preNoZ

from zope.app.container.interfaces import IContainer
class C1(object):
  zope.interface.implements(I1, IContainer)
  def __repr__(self):
    return 'C1'
```

Имея данное предусловие, мы можем проверить, может ли объект быть добавленным в контейнер:

```
c1 = C1()
checkObject(c1, "bob", None)
checkObject(c1, "Zbob", None)
Traceback (most recent call last):
   ...
Invalid: Names can not start with Z
```

Можно задать ограничение на контейнер, в который только и может быть добавлен объект. Это выполняется с помощью установки атрибута «__parent__» при объявлении интерфейса объекта как поля с заданным ключевым параметром «constraint», Например, интерфейс IMyObjContained будет поддерживаться объектами класса MyObj, которые, в свою очередь, могут быть элементами контейнера с интерфейсом IMyContainer. Тогда объявление интерфейса «IMyObjContained» будет иметь вид:

```
class IMyObjContained(IContained):
  """ . . . """
  __parent__ = Field(
      constraint = ContainerTypesConstraint(IMyContainer))
```

Важную роль для понимания внутренних процессов связывания объектов играет метод «__setitem__(name, object)», который добавляет данный объект в контейнер под данным именем. При этом поднимается исключение «TypeError», если имя задано строкой не в коде уникод или ascii, и «ValueError», если имя пустая строка. Контейнер может добавить другой объект, чем объект, переданный в этот метод. Если объект не осуществляет «IContained», тогда должно быть сделано одно из двух:

1) Если объект реализует интерфейс «ILocation», тогда для объекта должен быть объявлен интерфейс IContained;

2) В противном случае для объекта создается и сохраняется объект «ContainedProxy».

Атрибуты добавляемого объекта «__parent__» и «__name__» устанавливаются соответственно в контейнер и заданное имя.

Если старый родитель добавляемого объекта отсутствовал (был None), тогда генерируется событие «IObjectAddedEvent», в противном случае событие

«IObjectMovedEvent». Для контейнера генерируется событие «IObjectModifiedEvent». Если генерируется событие «IObjectAddedEvent» и объект адаптирует интерфейс `IObjectAddedEvent`, тогда вызывается адаптерный метод «addNotify». Если объект адаптирует `IObjectMovedEvent`, тогда вызывается адаптерный метод «moveNotify».

Если объект заменяет другой объект, тогда старый объект удаляется прежде, чем новый объект будет добавлен, если контейнер не запретит замену, поднимая исключение.

Если атрибуты «__parent__» и «__name__» уже были ранее установлены, тогда не возбуждаются никакие события и их обработка. Это позволяет продвинутым разработчикам перекрывать генерацию событий.

12.2 События и подписчики

События являются мощным средством программирования и являются первичным понятием в Zope3. Все современные технологии, так или иначе, используют метафору событийного программирования. В основе этого лежат два основных понятия: событие и его обработка. Источником событий в Zope3 могут быть изменения в базе данных, генерируемые системой, методы объектов, создаваемые программистами. Базовые средства обработки событий в Zope3 реализуются пакетом zope/event. Этот пакет обеспечивает базовую систему событий, на основе которой могут быть построены прикладные системы событий. Прикладной код может генерировать события, не заботясь об их обработчиках. Обработчики события, находясь за пределами точки возникновения события, позволяют существенно расширить функциональность системы.

Пакет имеет встроенный атрибут «subscribers» – список подписчиков. Прикладной код может управлять подпиской на события, управляя содержимым этого списка. Для демонстрации примеров сохраним текущее содержание списка существующих подписчиков и сделаем его пустым. Мы восстановим первоначальное содержимое списка подписчиков после окончания работы примеров.

```
import zope.event
old_subscribers = zope.event.subscribers[:]
del zope.event.subscribers[:]
```

Пакет обеспечивает функцию «notify», которая используется для информирования подписчиков о наступлении некоторого события:

```
class MyEvent:
    pass

event = MyEvent()
zope.event.notify(event)
```

Функция «notify» вызывается с одним параметром – объектом-событием. Подписчики являются простыми функциями, выполняющими некоторые операции по обработке событий, например:

```
def f(event):
    print 'got:', event
```

Функция помещается в список подписки оператором:

```
zope.event.subscribers.append(f)
```

и при вызове zope.event.notify(42) произведет печать строки «got: 42».

Для отказа от подписчика он просто удаляется из списка оператором:

```
zope.event.subscribers.remove(f)
```

Обычно, прикладные программы обеспечивают более гибкие механизмы подписки, которые строятся на основе описанного механизма. Приложения обычно устанавливают подписчиков, которые потом управляют другими подписчиками на основе типов событий или их данных.

Не забудем восстановить старый список подписки оператором
`zope.event.subscribers[:] = old_subscribers`

Таким образом, по мнению Jim Fulton [4] базовая система событий Zope3 имеет следующие недостатки:

- требуется, чтобы все объекты-события реализовывали интерфейс IEvent (или интерфейс, который наследует IEvent);
- требуется, чтобы все подписчики реализовывали интерфейс ISubscriber. Что делает невозможным использовать непосредственно функции как подписчики (требуется вызываемый объект соответствующего класса);
- имеется локальный и глобальный сервисы публикации событий и подписки на события. Услуги подписки на события обеспечивают некоторое функциональное назначение, которое обычно не используется (фильтрация), но не обеспечивает нужное функциональное назначение (диспетчирование как событий, так и объектов при их наступлении);
- имея дело с подпиской, базирующейся на концепции путей, система создает сложности в локальном сервисе подписки.

Между прочим, сложность делает трудным использование системы событий вне Zope. Им предлагается использовать адаптеры для моделирования подписчиков событий. Чтобы поддерживать эту функциональность, добавлен новый тип адаптера – подписной адаптер. Подписные адаптеры работают подобно регулярным адаптерам за исключением того, что они возвращают совокупность объектов. При поиске подписного адаптера, результатом является не наилучший адаптер, который соответствует адаптируемым объектам, а все адаптеры, соответствующие заданным объектам. Модель подписки, поддерживаемая подписными сервисами имеющегося события – действительно модель подписных адаптеров. Она позволяет отыскивать все адаптеры для данного события, на основе типа события.

Предлагается [4] существенно упростить архитектуру событий за счет следующих допущений.

1) Допустить любому объекту быть событием.

2) Заменять существующие глобальную и локальную публикацию событий и сервис подписки простым модулем уведомления о событиях. Этот модуль реализует базовый интерфейс:

```
class IBaseNotification(Interface):
  subscribers = Attribute("List of subscribers")
  def notify(event):
    """Notify all subscribers of an event. """
```

Это позволяет приложениям задавать подписчиков, манипулируя списком zope.event.subscribers. Zope устанавливает единственного подписчика:

```
def dispatch(*event):
  for ignored in subscribers(event, None):
    pass
```

Этот подписчик просто получает все адаптеры "обработчиков" для события. Обработчик является подписным адаптером, который зарегистрирован для обработки `None` и, который делает всю эту работу в своей фабрике. Обработчики являются обычными функциями Питона. Метод «`subscribers`» должен быть предусмотрен адаптерным сервисом. Индивидуальные адаптеры могут затем получать и вызывать другие адаптеры, как и требуется.

Наиболее важная характеристика базовой системы уведомления - то, что она не диктует какую либо политику. Вызываются все базовые подписчики. Если требуется специфическая политика (например, использование фильтрации), то приложения могут добавить политику, регистрируя нового базового подписчика или обеспечивая специализированный адаптер.

Поскольку мы ищем события как адаптеры, то нет необходимости строго определять интерфейсы для событий, поскольку адаптеры могут быть зарегистрированы для классов, а также и для интерфейсов.

Становятся не нужными локальные подписные сервисы. Всегда найдутся подходящие локальные адаптерные сервисы, способные обработать локальную подписку. Это заслуживает некоторого пояснения на следующем примере. Пусть есть узел, который подписывается на определенные события. Он в свою очередь может иметь своих подписчиков, регистрируемых в каталоге или в списке. Когда объект модифицирован, событие поступает в локальный сервис, который передает их в объектный узел. Узел превращает событие в событие узла и уведомляет своих подписчиков.

Переназначение будет реализовано следующим образом. У нас будет подписчик, который адаптирует события модификации объекта. Этот подписчик найдет все каталоги и индексы (утилиты) и вызовет для них метод reindex. Каждый каталог или индекс опросит id для своего узла (может быть несколько). Если он получает его, тогда индекс скорректирует свою информацию для объекта.

Возможно, что некоторые модели подписчиков нуждаются в учете времени. Описанная модель достаточно простая, и дополнительные модели могут быть созданы на ее основе без затруднений.

Примером организации событий и подписки может быть следующая задача [4] электронной почтовой рассылки уведомлений о модификации контента на сервере для некоторого обобщенного приложения. Необходимо также продемонстрировать добавление аннотаций к объекту. Для решения этой задачи нужны два основных компонента. Первый – подписной адаптер для контента, управляющий почтовой рассылкой электронной почты. Второй компонент – подписчик события, который слушает все поступающие события и начинает процесс почтовой рассылки, если наступило подходящее событие. Продемонстрированная ниже схема (рисунок 8)

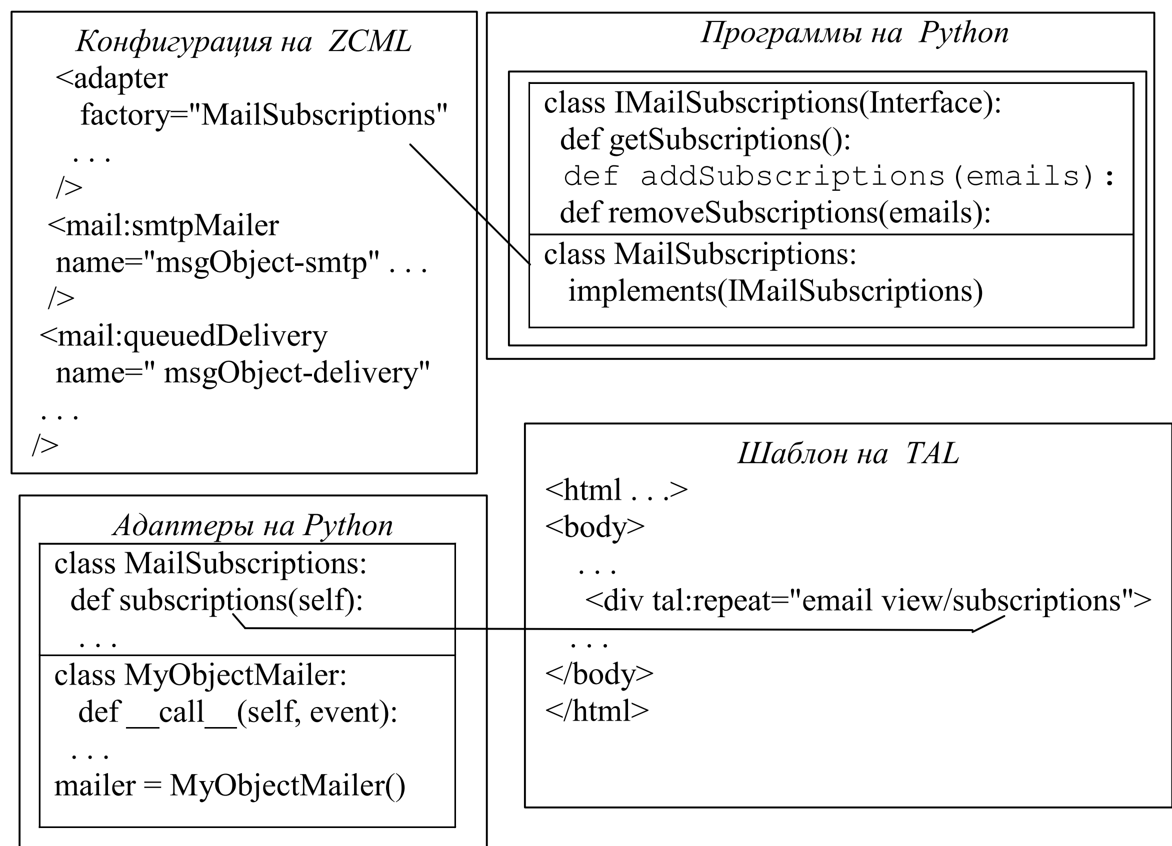

Рисунок 8 – Структура подписки в Zope 3

достаточно универсальна и позволяет встраивать ее в произвольное приложение. Шаги, связанные с тестированием, могут не включаться в конкретную разработку в зависимости от обстоятельств.

Шаг I: Интерфейс почтовой подписки

Нам нужно создать интерфейс для управления почтовой подпиской с целью добавления, удаления и получения адреса электронной почты. Добавим следующий интерфейс к модулю интерфейсов приложения:

```
class IMailSubscriptions(Interface):
   """This interface allows you to retrieve a list of
      E-mails for mailings. In our context these are MyObject.
   """
   def getSubscriptions():
       """Return a list of E-mails."""
   def addSubscriptions(emails):
       """Add a bunch of subscriptions;
```

```
        one would be okay too."""
    def removeSubscriptions(emails):
        """Remove a set of subscriptions."""
```

Шаг II: Написание адаптера почтовой подписки

Реализация адаптера достаточно простая. Подписка реализована в виде простого кортежа, который доступен через адаптер аннотаций. Добавьте следующий код в файл объявления классов приложения:

```
1  from zope.app.annotation.interfaces import IAnnotations
2  from <приложение>.interfaces import IMailSubscriptions
3
4  SubscriberKey='http://www.zope.org/myobj#1.0/MailSubscriptions/emails'
5
6
7  class MailSubscriptions:
8     """ Mail Subscriptions."""
9
10    implements(IMailSubscriptions)
11    __used_for__ = IMyObject
12
13    def __init__(self, context):
14       self.context = self.__parent__ = context
15       self._annotations = IAnnotations(context)
16       if not self._annotations.get(SubscriberKey):
17          self._annotations[SubscriberKey] = ()
18
19    def getSubscriptions(self):
20       "See .interfaces.IMailSubscriptions"
21       return self._annotations[SubscriberKey]
22
23    def addSubscriptions(self, emails):
24       "See .interfaces.IMailSubscriptions"
25       subscribers = list(self._annotations[SubscriberKey])
26       for email in emails:
27          if email not in subscribers:
28             subscribers.append(email.strip())
29       self._annotations[SubscriberKey] = tuple(subscribers)
30
31    def removeSubscriptions(self, emails):
32       "See  .interfaces.IMailSubscriptions"
33       subscribers = list(self._annotations[SubscriberKey])
34       for email in emails:
35          if email in subscribers:
36             subscribers.remove(email)
37       self._annotations[SubscriberKey] = tuple(subscribers)
```

В строке 2 и далее ключ <приложение> ссылается на папку, где расположен пакет, для которого организуется подписка.

Строка 4. Уникальный ключ подписчика, однозначно идентифицирующий данные об объекте приложения.

Строка 11: Декларация реализации адаптера для объектов IMyObject.

Строка 14: Поскольку этот адаптер использует аннотации, он будет «надежным» адаптером, что означает его обработку объектом «proxied». Все proxied объекты должны обеспечить информацию о месторасположении (по крайней мере, через атрибут __parent__), чтобы могли быть найдены его персональные декларации разрешения. В противном случае будут доступны только глобальные параметры разрешения.

Строка 15: Получаем адаптер аннотаций, обеспечивающий отображение объекта, в который мы загрузим комментарии. Отметьте, что это утверждение не сообщает ничего о типе комментария, который мы собираемся получить.

Строки 16 –17: Убеждаемся, что хранилище для подписчиков с нашим ключом существует. Если нет, создаем пустой кортеж.

Строки 19 – 37: Объявления методов для работы адаптера. Единственным стоящим является факт использования немутируемых кортежей вместо списков. Кортежи автоматически сохраняются в ZODB, если атрибут объекта – экземпляр класса AttributeAnnotations, а в нем то и реализован интерфейс IAnnotations.

После выполнения работ на этом шаге почтовой подписки можно зарегистрировать новый компонент через ZCML, используя адаптерную директиву:

```
1  <adapter
2      factory=".MyObject.MailSubscriptions"
3      provides=".interfaces.IMailSubscriptions"
4      for=".interfaces.IMyObject"
5      permission="<приложение>.Add"
6      trusted="true" />
```

Строки 2 – 4: Определяют необходимую регистрационную информацию для адаптера.

Строка 6: Если адаптер объявлен надежным, тогда контекст (объект, передаваемый в конструктор адаптера), не будет проверяться системой безопасности. Необходимо обеспечить, чтобы адаптер комментариев мог использовать атрибут __annotations__ для хранения комментария. Если адаптер не надежен и контекст проверяется службой безопасности, тогда всякий раз будет поднята ошибка ForbiddenAttribute, когда делается попытка получить доступ к комментариям.

Строка 5: Как только адаптер будет проверен, сам адаптер становится проверяющим безопасность. Следовательно, нам нужно определять разрешение, которое требуется для адаптера.

Шаг III: Тестирование адаптера

Для проверки правильности кодирования можно организовать тестирование кода рассылки. Тесты такие же простые, как и реализация. В строке документации для класса MailSubscriptions нужно добавить следующий тестирующий код.

```
1  Verify the interface implementation
2
3  >>> from zope.interface.verify import verifyClass
4  >>> verifyClass(IMailSubscriptions, MailSubscriptions)
5  True
```

```
6
7  Create a subscription instance of a message
8
9  >>> msg = MyObject()
10  >>> sub = MailSubscriptions(msg)
11
12  Verify that we have initially no subscriptions and then add some.
13
14  >>> sub.getSubscriptions()
15  ()
16  >>> sub.addSubscriptions(('foo@bar.com',))
17  >>> sub.getSubscriptions()
18  ('foo@bar.com',)
19  >>> sub.addSubscriptions(('blah@bar.com',))
20  >>> sub.getSubscriptions()
21  ('foo@bar.com', 'blah@bar.com')
22  >>> sub.addSubscriptions(('doh@bar.com',))
23  >>> sub.getSubscriptions()
24  ('foo@bar.com', 'blah@bar.com', 'doh@bar.com')
25
26  Now let's also check that we can remove entries.
27
28  >>> sub.removeSubscriptions(('foo@bar.com',))
29  >>> sub.getSubscriptions()
30  ('blah@bar.com', 'doh@bar.com')
31
32  When we construct a new mail subscription adapter instance, the values
33  should still be there.
34
35  >>> sub1 = MailSubscriptions(msg)
36  >>> sub1.getSubscriptions()
37  ('blah@bar.com', 'doh@bar.com')
```

Строки 3 – 5: Делает подробный анализ описания для проверки того, что класс MailSubscriptions осуществляет интерфейс IMailSubscriptions.

Строки 7 – 10: В док-тестах очень полезно описывать шаги теста. Здесь объявлено о явном создании отдельной секции теста.

Строки 12 – 24: Проверка возможности извлечения списка подписчиков и добавления новых.

Строки 26 – 30: Проверка работоспособности удаления подписки.

Строки 32 – 37: Когда создается новый адаптер, используя то же сообщение, подписка должна быть все еще доступна. Здесь проверяется, что данные не потеряны, когда адаптер уничтожен. Отметьте, что не проверяется вариант отсутствия комментария. Это является следствием того, что конструктор MailSubscriptions сам должен убедиться, что комментарий есть, даже если это пустая строка, так что эта возможность учтена в реализации и не нуждается в тестировании.

Так как адаптер использует аннотации, он требует установку компонентной архитектуры для запуска теста. Для этого необходимо зарегистрировать адаптер,

чтобы обеспечивать комментарии. Следовательно, нужно написать и использовать метод setUp(). Код в tests/test_MyObject.py изменяется на:

```
1  from zope.interface import classImplements
2
3  from zope.app.annotation.attribute import AttributeAnnotations
4  from zope.app.interfaces.annotation import IAnnotations
5  from zope.app.interfaces.annotation import IAttributeAnnotatable
6  from zope.app.tests import placelesssetup
7  from zope.app.tests import ztapi
8
9  def setUp(test):
10     placelesssetup.setUp()
11     classImplements(MyObject, IAttributeAnnotatable)
12     ztapi.provideAdapter(IAttributeAnnotatable, IAnnotations,
13                          AttributeAnnotations)
14
15  def test_suite():
16     return unittest.TestSuite((
17        DocTestSuite('приложение.MyObject',
18                     setUp=setUp, tearDown=placelesssetup.tearDown),
19        unittest.makeSuite(Test),
20        ))
```

Строка 7: Модуль ztapi содержит очень полезные функции, чтобы устанавливать компоненты для теста, например, вид и адаптерную регистрацию.

Строка 9: Метод setUp() ожидает аргумент тест – экземпляр класса DocTest. Вы можете использовать этот объект для обеспечения глобальных переменных теста.

Строка 11: ZCML обычно используется для объявления того, что объект реализует интерфейс IAttributeAnnotatable. Поскольку ZCML не выполняется для автономных тестов, мы здесь должны сделать это вручную.

Строки 12 – 13: Установка адаптера поиска комментариев для любого объекта, реализовующего интерфейс IAttributeAnnotatable.

Шаг IV: Подготовка шаблона для вида

Последнее, что необходимо обеспечить – это страницу для управления подпиской через вэб интерфейс. Страничный шаблон для этих целей (subscriptions.pt) мог бы выглядеть так:

```
1  <html metal:use-macro="views/standard_macros/view">
2    <body>
3      <div metal:fill-slot="body" i18n:domain="приложение">
4
5        <form action="changeSubscriptions.html" method="post">
6
7          <div class="row">
8            <div class="label"
```

```
9                i18n:translate="">Current Subscriptions</div>
10             <div class="field">
11            <div tal:repeat="email view/subscriptions">
12                 <input type="checkbox" name="remails:list"
13                        value="" tal:attributes="value email">
14                 <div tal:replace="email">zope3@zope3.org</div>
15               </div>
16               <input type="submit" name="REMOVE" value="Remove"
17                    i18n:attributes="value remove-button">
18             </div>
19         </div>
20
21         <div class="row">
22             <div class="label" i18n:translate="">
23               Enter new Users (separate by 'Return')
24             </div>
25             <div class="field">
26            <textarea name="emails" cols="40" rows="10"></textarea>
27             </div>
28         </div>
29
30              <div class="row">
31                <div class="controls">
32                  <input type="submit" value="Refresh"
33                 i18n:attributes="value refresh-button" />
34                  <input type="submit" name="ADD" value="Add"
35                      i18n:attributes="value add-button" />
36                </div>
37              </div>
38
39       </form>
40
41     </div>
42   </body>
43  </html>
```

Строки 7 – 19: Первая часть включает существующие подписки и позволяет выбирать их для удаления.

Строки 20 – 38: Вторая часть обеспечивает элемент «textarea» для новых добавляемых подпискчиков. Каждый адрес электронной почты должен заканчиваться концом строки (один адрес в строке).

Для поддержки класса вида нужно реализовать метод subscriptions() (смотри строку 11 выше) и обработку формы. Установите следующий код в browser/MyObject.py:

```
1  from  <приложение> .interfaces import IMailSubscriptions
2
3  class MySubscriptions:
4
5     def subscriptions(self):
```

```
6         return IMailSubscriptions(self.context).getSubscriptions()
7
8     def change(self):
9         if 'ADD' in self.request:
10            emails = self.request['emails'].split('\n')
11            IMailSubscriptions(self.context).addSubscriptions(emails)
12        elif 'REMOVE' in self.request:
13            emails = self.request['remails']
14            if isinstance(emails, (str, unicode)):
15                emails = [emails]
16            IMailSubscriptions(self.context).removeSubscriptions(emails)
17
18        self.request.response.redirect('./@@subscriptions.html')
```

Строки 9 и 12: Мы просто используем имя передающей кнопки, чтобы решать какое действие выбрал пользователь.

Остальная часть кода понятна без дополнительных разъяснений. Вид можно зарегистрировать следующим образом:

```
1  <pages
2     for="<приложение>.interfaces.IMyObject"
3     class=".MyObject.MySubscriptions"
4     permission="<приложение>.Edit"
5     >
6   <page
7       name="subscriptions.html"
8       template="subscriptions.pt"
9       menu="zmi_views" title="Subscriptions"
10       />
11   <page
12      name="changeSubscriptions.html"
13      attribute="change"
14      />
15  </pages>
```

Строка 1: Директива browser:pages позволяет нам регистрировать несколько страниц для интерфейса, используя класс вида и разрешения. Это особенно полезно для видов, которые обеспечивают различное функциональное назначение.

Строки 6 – 0: Эта страница использует шаблон для создания HTML.

Строки 11 – 14: Вид использует атрибут класса. Обычно методы в классе вида не возвращают HTML, но переназначают окно просмотра на другую страницу.

Строка 9: Убедитесь, что в виде подписки есть закладка для объекта MyObject.

Шаг V: Доставка сообщений

До сих пор мы даже не упоминали о событиях. Напомним общую систему событий: есть список подписчиков и функция notify(). Подписчики могут быть добавлены в список. Чтобы отменить подписку, подписчик должен быть удален из списка. Подписчики не должны быть какими-либо специальными типами объектов, они просто должны быть вызываемыми. Функция notify() получает аргумент объект-

событие и затем повторяет для всего списка вызов каждого подписчика, передавая ему как параметр событие.

Поэтому необходимо реализовать метод __call__() для нашего доставщика сообщений для того, чтобы он сам мог стать подписчиком. В целом класс MyObjectMailer должен выглядеть так (поместим его в модуль MyObject.py):

```
1  from zope.app import zapi
2  from zope.app.container.interfaces import IObjectAddedEvent
3  from zope.app.container.interfaces import IObjectRemovedEvent
4  from zope.app.event.interfaces import IObjectModifiedEvent
5  from zope.app.mail.interfaces import IMailDelivery
6
7  class MyObjectMailer:
8      """Class to handle all outgoing mail."""
9
10     def __call__(self, event):
11         """Called by the event system."""
12         if IMyObject.providedBy(event.object):
13             if IObjectAddedEvent.providedBy(event):
14                 self.handleAdded(event.object)
15             elif IObjectModifiedEvent.providedBy(event):
16                 self.handleModified(event.object)
17             elif IObjectRemovedEvent.providedBy(event):
18                 self.handleRemoved(event.object)
19
20     def handleAdded(self, object):
21         subject = 'Added: '+zapi.getName(object)
22         emails = self.getAllSubscribers(object)
23         body = object.body
24         self.mail(emails, subject, body)
25
26     def handleModified(self, object):
27         subject = 'Modified: '+zapi.getName(object)
28         emails = self.getAllSubscribers(object)
29         body = object.body
30         self.mail(emails, subject, body)
31
32     def handleRemoved(self, object):
33         subject = 'Removed: '+zapi.getName(object)
34         emails = self.getAllSubscribers(object)
35         body = subject
36         self.mail(emails, subject, body)
37
38     def getAllSubscribers(self, object):
39         """Retrieves all email subscribers."""
40         emails = ()
41         msg = object
42         while IMyObject.providedBy(msg):
43             emails += tuple(IMailSubscriptions(msg).getSubscriptions())
44             msg = zapi.getParent(msg)
45         return emails
```

```
46
47     def mail(self, toaddrs, subject, body):
48         """Mail out the MyObject changed message."""
49         if not toaddrs:
50             return
51         msg = 'Subject: %s\n\n\n%s' %(subject, body)
52         mail_utility = zapi.getUtility(IMailDelivery, 'msgboard-delivery')
53         mail_utility.send('mailer@<приложение>.org' , toaddrs, msg)
54
55  mailer = MyObjectMailer()
```

Строки 2 – 4: Мы хотим для нашего подписчика производить обработку событий добавления, редактирования и удаления. Мы импортируем интерфейсы этих событий, чтобы могли их различать.

Строки 10 – 18: Это главная часть подписчика. Когда происходит событие, то вызывается метод __call__(). Сначала нам нужно проверить было ли вызвано событие изменением объекта IMyObject; если это так, то проверяем, какое событие было инициировано. На основе произошедшего события вызывается соответствующий метод обработки.

Строки 20 – 36: Это три метода обработки, которые обрабатывают различные события. Отметьте, что обработка события модификации находит отличия, вместо повторной посылки всего объекта.

Строки 38 – 45: Этот метод извлекает всю подписку текущего объекта и всех его предков. Этот для тех, кто подписался на сам объект и получает e-mail о всех его комментариях.

Строки 47 – 53: Этот метод является быстрым введением в утилиту доставки почты. Метод send() утилиты доставки почты – часть API для smtplib. Политика и конфигурация посылки почты сконфигурированы полностью через ZCML. Смотри последующую часть конфигурации ниже.

Строка 55: Мы можем подписывать только экземпляры вызываемых объектов в системе событий, поэтому нам нужно иметь экземпляр компонента класса MyObjectMailer.

Наконец, нам нужно регистрировать мэйлер в утилите события и установить правильно почтовую утилиту. Зайдите в ваш файл конфигурации и зарегистрируйте следующей namespace в элементе configure:

```
xmlns:mail="http://namespaces.zope.org/mail"
```

Затем мы устанавливаем утилиту почты:

```
1  <mail:smtpMailer name="msgObject-smtp" hostname="localhost" port="25" />
2
3  <mail:queuedDelivery
4     name=" msgObject-delivery"
5     permission="zope.SendMail"
6     queuePath="./mail-queue"
7     mailer=" msgObject-smtp" />
```

Строка 1: Здесь мы решили посылать почту через сервер SMTP из localhost через стандартный порт 25. Мы могли бы посылать почту через командную строку sendmail.

Строки 3 – 7: Утилита доставки почты не посылает почту непосредственно, а планирует ее для отправки независимым процессом. Это имеет огромные преимущества, так как запрос не должен ожидать, пока почта не будет послана. Тем не менее, эта версия утилиты почты требует папку в резиденции приложения для временного хранения почтовых отправлений, пока они не будут посланы. Мы определяем такой директорий «mail-queue» в пакете приложения. Использованное ниже имя MyObjectMailer позволяет извлекать утилиту доставки поставленной в очередь почты. Другие утилиты позволяют прямую доставку, которая блокирует запрос, пока почта не будет послана.

Теперь мы регистрируем нашего доставщика для событий, которые мы хотим обрабатывать:

```
1  <subscriber
2      factory=".MyObject.mailer"
3      for="zope.app.event.interfaces.IObjectModifiedEvent" />
4
5  <subscriber
6      factory=".MyObject.mailer"
7      for="zope.app.container.interfaces.IObjectAddedEvent" />
8
9  <subscriber
10     factory=".MyObject.mailer"
11     for="zope.app.container.interfaces.IObjectRemovedEvent" />
```

Директива добавляет нового подписчика (определенного через атрибут factory) в список подписчиков. Атрибут for определяет интерфейс, который событие должно реализовать для этого подписчика. В прикладном сервере Zope подписка реализована через адаптеры, и мы зарегистрировали непосредственно адаптер с нужными интерфейсами.

Шаг VI: Тестирование доставки

Функция setUp() модуля test_message.py() нужна для регистрации позиции адаптеров и адаптера подписчика почтового сообщения. Это должно выглядеть так:

```
1  from zope.app.location.traversing import LocationPhysicallyLocatable
2  from zope.app.location.interfaces import ILocation
3  from zope.app.traversing.interfaces import IPhysicallyLocatable
4
5  from приложение.interfaces import IMailSubscriptions
6  from приложение.interfaces import IMyObject
7  from приложение.MyObject import MailSubscriptions
8
9  def setUp():
10     ...
11     ztapi.provideAdapter(ILocation, IPhysicallyLocatable,
```

```
12                      LocationPhysicallyLocatable)
13     ztapi.provideAdapter(IMyObject, IMailSubscriptions, MailSubscriptions)
```

Строки 1 – 3 и 11 – 12: Адаптер позволяет использовать API для доступа к родителям объектов или к пути до объекта.

Строки 5 – 7 и 13: Регистрация подписного адаптера почты, чтобы мэйлер мог найти подписчиков.

Строка 10: Троеточие представляет существующее содержимое функции.

Теперь сделана вся подготовка и мы можем начать писать док-тест. Взглянем на тест метода getAllSubscribers(). Мы хотим создавать объект и добавлять ответы к нему. Оба объекта будет иметь подписчика. Когда вызван метод getAllSubscribers() для ответа, должны быть возвращены подписчик для оригинального объекта и ответа. Код теста, который устанавливается в docstring getAllSubscribers():

```
1    Here a small demonstration of retrieving all subscribers.
2
3  >>> from zope.interface import directlyProvides
4  >>> from zope.app.traversing.interfaces import IContainmentRoot
5
6  Create a parent object as it would be located.
7 Also add a subscriber to the object.
8
9  >>> msg1 = MyObject()
10  >>> directlyProvides(msg1, IContainmentRoot)
11  >>> msg1.__name__ = 'msg1'
12  >>> msg1.__parent__ = None
13  >>> msg1_sub = MailSubscriptions(msg1)
14  >>> msg1_sub.context.__annotations__[SubscriberKey] = ('foo@bar.com',)
15
16  Create a reply to the first object and also give it a subscriber.
17
18  >>> msg2 = MyObject()
19  >>> msg2_sub = MailSubscriptions(msg2)
20  >>> msg2_sub.context.__annotations__[SubscriberKey] = ('blah@bar.com',)
21  >>> msg1['msg2'] = msg2
22
23  When asking for all subscriptions of object 2, we should get the
24  subscriber from  object  1 as well.
25
26  >>> mailer.getAllSubscribers(msg2)
27  ('blah@bar.com', 'foo@bar.com')
```

Строки 3 – 4: Импортируют некоторые общие функции и интерфейсы, которые использованы в тесте.

Строки 6 – 14: Создан первый объект. Примечание: объект должен реализовывать IContainmentRoot (строка 10), что останавливает поиск заимствованием, как только этот объект будет обнаружен. Используя подписной адаптер почты (строка 13-14), мы регистрируем подписчика для объекта.

Строки 16 – 21: Создаем ответ на первый объект. Родитель и имя второго объекта автоматически будут добавлены в вызов __setitem__.

Строки 23 – 27: Мэйлер должен извлечь оба подписчика. Если тест проходит, то он это делает.

Наконец, мы непосредственно тестируем метод __call__ () – единственный общий метод. Для уведомления, чтобы работать правильно, мы должны создать и регистрировать утилиту с интерфейсом IMailDelivery с именем “'myobject-delivery”. Так как мы не хотим действительно посылать почту во время тестирования, целесообразно написать заглушку реализации утилиты. Для этого запустите тесты для метода notify(), добавив реализацию метода поставки почты к строке теста:

```
1  >>> mail_result = []
2
3  >>> from zope.interface import implements
4  >>> from zope.app.mail.interfaces import IMailDelivery
5
6  >>> class MailDeliveryStub(object):
7  ...     implements(IMailDelivery)
8  ...
9  ...     def send(self, fromaddr, toaddrs, message):
10  ...         mail_result.append((fromaddr, toaddrs, message))
11
12  >>> from zope.app.tests import ztapi
13  >>> ztapi.provideUtility(IMailDelivery, MailDeliveryStub(),
14  ...                  name='myobject-delivery')
```

Строка 1: Запросы почты загружаются в эту глобальную переменную, чтобы мы могли проверить добавление посланной почты.

Строки 6 – 10: Утилита почты требует только метод send(), который нужно реализовать для сохранения данных.

Строки 12 – 14: Использование модуля ztapi позволяет быстро зарегистрировать утилиту. Проследите за тем, чтобы задать правильное имя, в противном случае тест не будет работать.

Далее подобно предшествующему тесту, мы должны создать объект и добавить подписчика.

```
1  Create a object.
2
3  >>> from zope.interface import directlyProvides
4  >>> from zope.app.traversing.interfaces import IContainmentRoot
5
6  >>> mobj = MyObject ()
7  >>> directlyProvides(msg, IContainmentRoot)
8  >>> mobj.__name__ = 'obj'
9  >>> mobj.__parent__ = None
10  >>> mobj.title = 'Hello'
11  >>> mobj.body = 'Hello World!'
12
13  Add a subscription to message.
```

```
14
15  >>> mobj_sub = MailSubscriptions(msg)
16  >>> mobj _sub.context.__annotations__[SubscriberKey] = ('foo@bar.com',)
```

Наконец, мы создаем событие модификации, используя сообщение, и посылаем его методу notify(). Для проверки правильного функционирования метода имеется глобальная переменная mail_result.

```
1  Now, create an event and send it to the message mailer object.
2
3  >>> from zope.app.event.objectevent import ObjectModifiedEvent
4  >>> event = ObjectModifiedEvent(msg)
5  >>> mailer(event)
6
7  >>> from pprint import pprint
8  >>> pprint(mail_result)
9  [('mailer@messageboard.org',
10    ('foo@bar.com',),
11    'Subject: Modified: mobj\n\n\nHello World!')]
```

Строки 3 – 4: В этом тесте используется событие модификации объекта. Другое IObjectEvent может быть создано посылкой неверного объекта как аргумента конструктора события.

Строка 5: Здесь мы уведомляем мэйлер о модификации объекта. Отметьте, что мэйлер является экземпляром класса MyObjectMailer и создан в конце модуля для объектов.

Строки 7 – 11: Модуль печати (pprint) очень удобен для вывода сложных структур данных.

Необходимо запустить тесты, чтобы проверить реализацию почтовой рассылки.

Шаг VII: Использование почтовой подписки

Прежде всего, мы должны перезапустить Zope и убедиться в правильной загрузке. Обратите внимание на новую закладку Subscriptions. На странице подписки есть текстовая область, в которой можно ввести адреса электронной почты подписчиков, которые получат почтовые сообщения, когда объект или любые его потомки изменяются.

В заключении обсуждения событий и подписчиков рекомендуется внимательно рассмотреть пример организации подписки в демонстрационном приложении ZWiki [10]. В файле интерфейсов обнаруживаем объявление классов IWikiPageEditEvent и IMailSubscriptions с аналогичным описанным ранее кодом. В файле WikiPage.py наблюдаем фрагмент описания классов WikiPageEditEvent, MailSubscriptions и WikiMailer, реализующие соответствующие интерфейсы. В файле конфигураций есть секции «Mail Subscriptions support» и «browser:pages», содержащие утверждения языка ZCML для регистрации подписки. Файл subscriptions.pt содержит шаблон документов для генерации страниц просмотра и редактирования подписчиков. Все фрагменты соответствуют описанной выше технологии работы с событиями и подписчиками. Встроенные в документацию на модули и классы тесты отсутствуют.

12.3 Обеспечение встроенной помощи

Предоставление помощи потребителю в любой точке графического интерфейса – важная характеристика для любых приложений и не только с вэб интерфейсом. Для этого необходимо решить две задачи. Сначала нужно написать кадры помощи (это может быть сплошной, неструктурированный, структурированный текст или документ HTML). Далее необходимо их зарегистрировать. Так как помощь будет нужна для просмотра видов, рекомендуется устанавливать файлы помощи в папке help в директории пакета.

Сначала создайте несколько файлов с кадрами помощи, например, файл intro.rst:

```
==========================
 Buddy Demo Package
==========================
This package demos various features of the Zope3 Framework. If y
ou have questions or concerns, please let me know.
```

Затем файл review.rst:

```
This view lists all buddies in the collection for publication.
```

Затем файл edit.rst:

```
This screen allows you to edit the data (of the Buddy object).
```

Все что осталось сделать – зарегистрировать новые кадры помощи. Темы помощи могут быть организованы в иерархическом порядке. Для того, чтобы сохранить весь пакет экранов вместе в одном поддереве, мы сделаем тему info.rst родителем для всех других кадров помощи. Откройте ваш файл конфигурации (buddydemo/configure.zcml). Затем нам нужно добавить namespace помощи в элемент zope:configure, используя следующую декларацию:

xmlns:help=http://namespaces.zope.org/help

Далее можно добавить следующие директивы:

```
1  <help:register
2      id="buddydemo"
3      title=" Buddy demo help"
4      parent="ui"
5      for=" buddydemo.interfaces.IBuddy"
6      doc_path="./help/intro.rst"/>
7
8  <help:register
9      id="buddy.review"
10      title="Publication Review"
11      parent="ui/buddydemo"
12      for=" buddydemo.interfaces.IBuddy"
13      view="review.html"
14      doc_path="./help/review.rst"/>
```

```
15
16  <help:register
17      id=" buddy.edit"
18      title="Change Buddy"
19      parent="ui/buddydemo"
20      for=" buddydemo.interfaces.IBuddy"
21      view="edit.html"
22      doc_path="./help/edit.rst"/>
```

Строка 2: Это - id кадра помощи доступного в URL.
Строка 3: Название темы помощи, отображаемое выше содержимого темы.
Строка 4: Путь родительской темы помощи – всегда имя «ui», встроенное в Zope3.
Строка 5: Регистрирует тему как встроенную контекстную помощь для объектов приятелей. Это - дополнительный атрибут.
Строка 6: Относительный путь в файл помощи. Возможные расширения файлов включают txt, rst, html и stx.
Строки 12 – 13: Регистрация темы для вида review.html.
Строки 11 и 19: Определение родительского кадра помощи.

Теперь нужно перезапустить Zope и перейти к просмотру приятелей.

12.4 Разработка нового обличья

При разработке проблемно-ориентированных приложений естественным образом возникает необходимость проектирования пользовательских интерфейсов, удовлетворяющих требованиям функциональности и дизайна. Что касается простых презентаций, то эти требования легко выполняются дизайном шаблонов страниц, отправляемых клиенту сайта. Для приложений, использующих новые компоненты, это простое решение не всегда удовлетворительно. Для решения таких задач в Zope используется понятие Skins – обличья, имеющие более сложное устройство, чем простые презентационные страницы с использованием стандартного HTML и CSS.

Обличье является стеком слоев. Каждый слой может содержать любой набор видов и ресурсов, допускающее перекрытие конкретных видов и ресурсов. Например, CSS может быть определена во встроенном слое, но она не устраивает нашим потребностям. Мы можем создавать новый слой обличья и установить в нем другой CSS.

Шаг I: Подготовка

Прежде, чем создавать новую форму, нужно сделать некоторую подготовку. Создадим папку-пакет skin обычно в директории browser пакета и в нем пустой файл __init__.py и файл configure.zcml:

```
<configure
   xmlns="http://namespaces.zope.org/browser">

</configure>
```

Далее добавьте в числе первых директив ссылку на этот файл в конфигурации configure.zcml пакета browser, используя директиву включения:

```
<include package=".skin" />
```

Шаг II: Создание новой формы

Создание новой формы выполняется директивами ZCML в конфигурации обличья, где ранее атрибутом xmlns задано использование пространства имен browser для директив семейства обличий. Добавим следующие директивы в файл конфигурации пакета skin:

<layer name="buddy"/>

<skin name=" buddy" layers=" buddy rotterdam default" />

Первая директива создает новый слой с новыми шаблонами, который сделает форму уникальной. Вторая директива создает форму с именем buddy, которая состоит из трех элементов стека слоев. Самый низкий слой default является слоем по умолчанию, который перекрыт слоем rotterdam , перекрытый слоем buddy. Каждая директива представления окна просмотра поддерживает атрибут слой, который определяет слой обличья, в котором установлен вид или ресурс. Если не был определен никакой слой, то компонент представления устанавливается во встроенном слое обличья.

Слой Rotterdam, как Basic, Debug и StaticTree, является одним из встроенных слоев системы Zope и содержит много полезных определений.

Шаг III: Модификация базового шаблона

Первой задачей всегда является перекрытие макросов skin_macros и dialog_macros. Обычно макрос skin определен в файле с именем template.pt, а макрос dialog в dialog_macros.pt. Новый template.pt файл мог бы выглядеть приблизительно так:

```
1  <metal:block define-macro="page">
2    <metal:block define-slot="doctype">
3      <!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN"
4          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
5    </metal:block>
6
7  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="
en">
8
9    <head>
10      <title metal:define-slot="title"> Buddy demo
for Zope3</title>
11
12      <style type="text/css" media="all"
13         tal:content=
14         "string: @import url(${context/++resource++buddy.css});
">
15        @import url(buddy.css);
16      </style>
17
```

```
18      <meta http-equiv="Content-Type"
19          content="text/html;charset=utf-8" />
20
21      <link rel="icon" type="image/png"
22        tal:attributes="href context/++resource++favicon.png" />
23    </head>
24
25    <body>
26
27      <div id="buddy_header" i18n:domain=" buddydemo" tal:on-
error="nothing">
28        <img id="buddy_logo"
29            tal:attributes="src context/++resource++logo.png" />
30        <div id="buddy_greeting"> 
31           <span i18n:translate="">Zope3 Buddy demo </span>
32        </div>
33      </div>
34
35      <div id="workspace">
36
37        <div metal:define-slot="buddy" id="buddy"></div>
38
39        <div id="content">
40          <metal:block define-slot="body">
41         This is the content.
42          </metal:block>
43        </div>
44
45      </div>
46
47      <div id="footer">
48
49        <div id="actions">
50          <metal:block define-slot="actions" />
51        </div>
52        <div id="credits" i18n:domain="buddydemo">
53          Powered by Zope3.<br>
54          Stephan Richter in 2003
55        </div>
56      </div>
57
58    </body>
59
60  </html>
61
62  </metal:block>
```

Строки 12-16: Вместо стандартного файла стилей zope3.css мы хотим использовать свой buddy.css.

Строки 21-22: Ресурс favicon предусмотрен обличьем rotterdam.

Строки 27-33: Простой заголовок, состоящий из пары стилей, логотипа и простого названия.

Строки 47-56: Нижний колонтитул состоит из метки-заполнителя, где мы позже можем заменить слот actions и секцию credits для размещения авторской информации.

Это пока еще не соответствует полному шаблону. Заметим насколько это проще, чем эквивалентный пример rotterdam, который находится в папке src/zope/app/rotterdam. Аналогично, страничный шаблон dialog_macros.pt может быть следующим.

```
<metal:block define-macro="dialog">
  <metal:block define-slot="doctype">
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
  </metal:block>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">
  <head>
    <title metal:define-slot="title">Buddy demo for Zope3</title>

    <style type="text/css" media="all"
       tal:content="string: @import
url(${context/++resource++buddy.css});">
      @import url(buddy.css);
    </style>
    <meta http-equiv="Content-Type"
content="text/html;charset=utf-8" />
    <link rel="icon" type="image/png"
          tal:attributes="href context/++resource++favicon.png" />
  </head>
  <body>
    <div id="workspace">
      <div metal:define-slot="buddy" id="buddy"></div>
      <div id="content">
        <metal:block define-slot="body">
       This is the content.
        </metal:block>
      </div>
    </div>
  </body>
</html>
</metal:block>
```

В вышеуказанных шаблонах мы ссылались на два новых ресурса, logo.png и buddy.css. Оба конфигурируются следующим образом:

```
1  <resource
2     name=" buddy.css" file="buddy.css" layer=" buddy" />
3
4  <resource
5     name="logo.png" file="logo.png" layer=" buddy" />
```

Файл CSS (buddy.css) содержит определения стилей для различных разделов формируемого документа. Пример такого определения можно найти в файле

zope/app/rotterdam/zope3.css. Познакомиться с использованием каскадных таблиц стилей можно, например, в [11].

Шаблоны для слоя регистрируются следующим образом:

```
1  <page
2     for="*"
3     name="skin_macros"
4     permission="zope.View"
5     layer="buddy"
6     template="template.pt" />
7
8  <page
9     for="*"
10     name="dialog_macros"
11     permission="zope.View"
12     layer="buddy"
13     template="dialog_macros.pt" />
```

Строки 2 и 9: Символ «*» означает, что эта страница доступна для всех объектов.

Строки 5 и 12: Определение слоя обличья.

13 Проектирование приложений

Проектирование приложений начинается с разработки требований к функциям и интерфейсам с конечным пользователем. Далее начинается решение частных вопросов архитектуры приложения, дизайна страниц, средств управления функциями, управления контентом и других. Выбор используемых средств системы Zope 3 во многом определяет эффективность будущего приложения. Далее обсуждаются некоторые варианты решения перечисленных задач. Рассмотрение принципов создания приложений проиллюстрируем фрагментами текстов проекта Schoolbell, распространяемого под лицензией GPL, являющегося частью проекта SchoolTool, финансируемого Shuttleworth Foundation (http://www.schooltool.org/schoolbell).

13.1 Объект приложения

Zope 3 предоставляет разные возможности организации работы с будущими пользователями разрабатываемого приложения. Некоторые варианты организации приложений без использования файлов на Питоне и других рассматривались в предыдущих разделах. При разработке же функционально полных приложений, обеспечивающих самостоятельно, без участия ZMI все необходимые сервисы, определяется класс **Application** в нашем примере со следующими интерфейсами.

```
from schoolbell.calendar.interfaces import IEditCalendar
from zope.app.location.interfaces import ILocation
class ISchoolBellCalendar(IEditCalendar, ILocation):
    """ SchoolBell календарь для хранения внутри себя всех
        назначенных событий ISchoolBellCalendarEvent.
```

```
    """

    title = TextLine(title=u"Title",
        description=u"Title of the calendar.")

class ICalendarOwner(Interface):
    """An object that has a calendar."""

    calendar = Object(
        title=u"The object's calendar.",
        schema=ISchoolBellCalendar)

from zope.app.container.interfaces import IReadContainer
class ISchoolBellApplication(IReadContainer, ICalendarOwner):
    """
       Главный объект SchoolBell. Приложение является контейнером
       только для чтения со следующими разделами:

        'persons'   - IPersonContainer
        'groups'    - IGroupContainer
        'resources' - IResourceContainer

       Этот объект может быть добавлен как обычный контент в папку,
       или может быть использован как корневой объект приложения.
    """

from zope.app.annotation.interfaces import IAttributeAnnotatable
from persistent.dict import PersistentDict
from zope.app.container.sample import SampleContainer
from zope.app.site.servicecontainer import SiteManagerContainer

class SchoolBellApplication(Persistent, SampleContainer,
                            SiteManagerContainer):
    """ Главный объект приложения SchoolBell. """

    implements(ISchoolBellApplication, IAttributeAnnotatable)

    def __init__(self):
        SampleContainer.__init__(self)
        # XXX Do we want to localize the container names?
        self['persons']   = PersonContainer()
        self['groups']    = GroupContainer()
        self['resources'] = ResourceContainer()
        self.calendar = Calendar(self)

    def _newContainerData(self):
        return PersistentDict()

    def _title(self):
        """ Это необходимо при работе для просмотра календаря. """
        return IApplicationPreferences(self).title
```

```
    title = property(_title)
```

По требованиям Zope 3 все потомки класса SampleContainer должны перекрывать метод _newContainerData для создания хранилища данных своего экземпляра. В данном случае таким хранилищем является словарь – экземпляр класса PersistentDict, находящийся в ZODB. При создании экземпляра приложения конструктором класса в словарь будут добавлены экземпляры контейнеров для персон, групп персон и ресурсов. Атрибут calendar будет содержать ссылку на принадлежащий приложению собственный экземпляр календаря.

При работе с объектами, принадлежащими приложению, публикатору Zope требуется вести поиск этих объектов в базах данных приложения. Публикатор Zope вызывает средства поиска, когда встречает в тексте шаблона выражения пути, например, item/title. Так как о том, как хранится и где находится тот или иной объект знает только приложение, в его состав должен входить адаптер, реализующий функции поиска. Таким адаптером в SchoolBell является экземпляр класса SchoolBellApplicationTraverser, который указывается в операторе view языка ZCML файла конфигурации приложения.

```
<zope:view
  for="schoolbell.app.interfaces.ISchoolBellApplication"
  type="zope.publisher.interfaces.browser.IBrowserRequest"
provides="zope.publisher.interfaces.browser.IBrowserPublisher"
  factory=".app.SchoolBellApplicationTraverser"
  permission="zope.Public"
/>
```

Объявление самого класса адаптера имеет вид:

```
from schoolbell.app.browser.cal import CalendarOwnerTraverser

class SchoolBellApplicationTraverser(CalendarOwnerTraverser):
    """Поиск (traverser) объектов приложения
        SchoolBellApplication.
    """
    adapts(ISchoolBellApplication)

    def publishTraverse(self, request, name):
        if name in ('persons', 'resources', 'groups'):
            return self.context[name]

        return CalendarOwnerTraverser.publishTraverse(self,
               request, name)
```

Здесь в соответствии с номенклатурой данных приложения и правилами хранения объекты извлекаются либо из словаря приложения (для persons, resources, groups), либо для остальных объектов из хранилища календарей или других мест, о которых знает метод publishTraverse адаптера родительского класса CalendarOwnerTraverser.

```
class CalendarOwnerTraverser(object):
```

```
    """A traverser that allows to traverse to a calendar owner's
       calendar."""

    adapts(ICalendarOwner)
    implements(IBrowserPublisher)

    def __init__(self, context, request):
        self.context = context
        self.request = request

    def publishTraverse(self, request, name):
        if name == 'calendar':
            return self.context.calendar
        elif name in ('calendar.ics', 'calendar.vfb'):
            calendar = self.context.calendar
            view = queryMultiAdapter((calendar, request),
                        name=name)
            if view is not None:
                return view

        view = queryMultiAdapter((self.context, request),
                        name=name)
        if view is not None:
            return view

        raise NotFound(self.context, name, request)

    def browserDefault(self, request):
        return self.context, ('index.html', )
```

Здесь поиск для календарей ведется в хранилищах календарей, а для объектов других классов вызовом запроса (если он есть) зарегистрированного в файлах конфигурации адаптера для пары «вход-выход». Например, для «IPerson – IPersonDetails» регистрация адаптера производится следующим оператором ZCML

```
  <adapter
      for=".interfaces.IPerson"
      provides=".interfaces.IPersonDetails"
      factory=".app.getPersonDetails"
      trusted="true"
      />
```

где функция

```
def getPersonDetails(person):
    """Adapt an IPerson object to IPersonDetails."""
    annotations = IAnnotations(person)
    key = 'schoolbell.app.PersonDetails'
    try:
        return annotations[key]
    except KeyError:
```

```
        annotations[key] = PersonDetails()
        annotations[key].__parent__ = person
        return annotations[key]
```

возвращает объект из словаря аннотаций с ключом 'schoolbell.app.PersonDetails', а если его нет, то создает новый объект класса PersonDetails, сохраняет в словаре, и возвращает его как результат поиска.

Аналогично в файлах конфигурации должны быть указаны средства поиска для всех объектов, явно или неявно фигурирующих в запросах пользователей.

Для включения главного объекта приложения в состав доступных объектов ZMI в файле конфигурации помещается оператор

```
  <content class=".app.SchoolBellApplication">
    <allow interface=".interfaces.ISchoolBellApplication" />
    <allow attributes="getSiteManager title" />
  </content>
```

Появление в меню добавляемых объектов ZMI главного приложения обеспечивается оператором

```
  <addMenuItem
      title="SchoolBell"
      class="schoolbell.app.app.SchoolBellApplication"
      permission="zope.ManageContent"
      />
```

Задание главной страницы для просмотра приложения посетителями производится оператором

```
  <page
      name="index.html"
      for="schoolbell.app.interfaces.ISchoolBellApplication"
      class="schoolbell.app.browser.app.SchoolBellApplicationView"
      template="templates/index.pt"
      permission="zope.Public"
      />
```

Фабрика адаптеров выглядит следующим образом

```
from zope.app.publisher.browser import BrowserView
class SchoolToolApplicationView(BrowserView):
    """Вид для главного приложения."""

    def update(self):
        prefs = IApplicationPreferences(
                    getSchoolToolApplication())
        if prefs.frontPageCalendar:
```

```
            url = zapi.absoluteURL(self.context.calendar,
                     self.request)
            self.request.response.redirect(url)
```

Когда пользователем запрашивается объект приложения, то системой будет найден адаптер по имени "index.html" для интерфейса ISchoolBellApplication, создаваемый как экземпляр класса SchoolBellApplicationView. При создании адаптера ему будут переданы параметрами контекст и запрос (смотри класс BrowserView), несущие информацию о сеансе пользователя, обслуживаемого в данный момент сервером.

13.2 Управление пользователями

Система управления пользователями (принципалами) Zope 3 предполагает предварительную их регистрацию в конфигурационных файлах операторами principal, role, grant прописанными администратором системы, как правило, в папке etc резиденции Zope. Если создается приложение, у которого должны быть свои собственные пользователи и их группы, то необходимы средства их оперативной регистрации и редактирования, совместимые по контролю полномочий с системой безопасности Zope 3. Для решения таких задач разработчику приложения предоставляются средства программной регистрации принципалов, расположенные в пакете zope/security.

В рассматриваемом примере приложения SchoolBell использованы следующие средства работы с пользователями. Регистрация разрешений производится группой операторов вида

```
  <permission id="schoolbell.view" title="View" />
```

Для автономной регистрации своих пользователей имеется класс Person

```
class Person(Persistent, Contained):
    """Персона."""

    implements(IPersonContained, IHaveNotes, IHavePreferences,
               IAttributeAnnotatable)

    photo = None
    username = None
    _hashed_password = None

    groups = RelationshipProperty(URIMembership, URIMember,
                   URIGroup)
    overlaid_calendars = OverlaidCalendarsProperty()

    def __init__(self, username=None, title=None):
        self.title = title
        self.username = username
        self.calendar = Calendar(self)

    def setPassword(self, password):
```

```
        self._hashed_password = hash_password(password)

    def checkPassword(self, password):
        return (self._hashed_password is not None
                and hash_password(password) ==
                    self._hashed_password)

    def hasPassword(self):
        return self._hashed_password is not None

    def __conform__(self, protocol):
        if protocol is ISchoolBellApplication:
            return self.__parent__.__parent__
```

Задание параметров нового пользователя производится с использованием схемы IPersonAddForm экземпляром адаптера класса PersonAddView

```
class IPersonAddForm(Interface):
    """Schema for person adding form."""

    title = TextLine(
        title=_("Full name"),
        description=_("Name that should be displayed"))

    username = TextLine(
        title=_("Username"),
        description=_("Username"))

    password = Password(
        title=_("Password"),
        required=False)

    verify_password = Password(
        title=_("Verify password"),
        required=False)

    photo = Bytes(
        title=_("Photo"),
        required=False,
        description=_("""Photo (in JPEG format)"""))
```

Класс PersonAddView будучи производным от системного класса AddView добавляет нового пользователя в контейнер персон приложения, получив данные о персоне из формы, сгенерированной на основе схемы и шаблона person_add.pt, указанных при регистрации адаптера оператором

```
  <page
      name="add.html"
      for="schoolbell.app.interfaces.IPersonContainer"
      class="schoolbell.app.browser.app.PersonAddView"
      template="templates/person_add.pt"
      permission="schoolbell.create"
```

```
    menu="schoolbell_actions" title="New Person" />
```

Когда менеджер приложения производит добавление нового пользователя, вызывается обработчик события, зарегистрированный оператором

```
<subscriber
    for="zope.app.container.interfaces.IObjectAddedEvent"
    handler=".security.authSetUpSubscriber"
    />
```

Процедура authSetUpSubscriber производит регистрацию нового пользователя вызовом метода setUpLocalAuth и присваивает ему полномочия 'schoolbell.view' вызовом grantPermissionToPrincipal.

```
def authSetUpSubscriber(event):
    """Set up local authentication for newly added SchoolBell
                apps.

    This is a handler for IObjectAddedEvent.
    """
    if IObjectAddedEvent.providedBy(event):
        if ISchoolBellApplication.providedBy(event.object):
            setUpLocalAuth(event.object)

            # Grant schoolbell.view to all authenticated users
            allusers = zapi.queryUtility(IAuthenticatedGroup)
            if allusers is not None:
                perms = IPrincipalPermissionManager(event.object)
                perms.grantPermissionToPrincipal(
                     'schoolbell.view', allusers.id)
```

13.3 Управление просмотрами

Общий дизайн приложения обычно определяется средствами макросов, описанных в главе 3. Макросы содержат слоты, заполняемые в шаблонах страниц (например, index.pt), для формирования документов, передаваемых пользователю в ответ на его запросы. Для задания различных вариантов оформления частей приложения используются слои (layer) и обличья (skin), регистрируемые в конфигурационных файлах приложения. В рассматриваемом примере это выглядит следующим образом.

```
<layer
    name="SchoolBell"
    interface=".skin.ISchoolBellLayer" />

<skin
    name="SchoolBell"
    interface=".skin.ISchoolBellSkin" />
```

В целях динамического связывания обличья с объектами приложения в проекте применен следующий прием. Зарегистрирован подписчик на событие IBeforeTraverseEvent оператором

```
<zope:subscriber
    handler=".skin.schoolBellTraverseSubscriber"
    for="zope.app.publication.interfaces.IBeforeTraverseEvent"
    />
```

Обработчик события имеет следующий код:

```
def schoolBellTraverseSubscriber(event):
    """ Подписчик для BeforeTraverseEvent устанавливает обличье
        ISchoolBellSkin, если разыскиваемый объект является
        экземпляром приложения SchoolBell.
    """
    if (ISchoolBellApplication.providedBy(event.object) and
        IBrowserRequest.providedBy(event.request)):
        applySkin(event.request, ISchoolBellSkin)
```

Обработчик события для всех запрашиваемых пользователем объектов, производных от приложения, методом applySkin переопределяет обличье на нужное. Следует напомнить, что при регистрации адаптеров и их поиске используются поддерживаемые ими интерфейсы, которые становятся индикаторами соответствующих объектов. В нашем случае речь идет об идентификации обличья интерфейсом с именем ISchoolBellSkin. Доступ по именам считается устаревшим (deprecated).

Что касается работы с внутренними объектами приложения, то необходимо обеспечить регистрацию их как элементов контента, способ их поиска, страницу для просмотра и, если нужно, для редактирования. Например, для персон в приложении SchoolBell регистрация контента производится оператором

```
<content class=".app.Person">
    <require permission="schoolbell.view"
             interface="schoolbell.app.interfaces.IReadPerson"
             attributes="__cmp__"/>
    <require permission="schoolbell.edit"
             interface="schoolbell.app.interfaces.IWritePerson"
             set_schema="schoolbell.app.interfaces.IPerson" />
    <allow interface="schoolbell.app.interfaces.ICalendarOwner" />
</content>
```

Регистрация адаптера для поиска персон производится оператором

```
<view
      for="schoolbell.app.interfaces.IPerson"
      type="zope.publisher.interfaces.browser.IHTTPRequest"
      provides=".IRestTraverser"
      factory=".app.PersonHTTPTraverser"
      permission="zope.Public"
      name="photo"
      />
```

Регистрация адаптера для просмотра персон производится оператором

```
<page
      name="index.html"
      for="schoolbell.app.interfaces.IPerson"
      class="schoolbell.app.browser.app.PersonView"
      template="templates/person.pt"
      permission="zope.View"
      menu="zmi_views"
      title="View"
      />
```

Используемый конструктор адаптеров определяется объявлением класса

```
class PersonView(BrowserView):
    """A Person info view."""

    __used_for__ = IPersonContained

    def __init__(self, context, request):
        BrowserView.__init__(self, context, request)
        self.details = IPersonDetails(self.context)
```

13.4 Отладчик Zope 3

При отладке приложений большую помощь оказывет встроенный в систему отладчик (debugger). Запуск отладчика производится командным файлом debugzope.bat из папки <Zope instance>\bin. После старта выдается приглашение интерпретатора языка Python. Последующие манипуляции выполняются в командной строке интерпретатора. В режиме отладки доступны объекты среды Zope3 как для просмотра, так и для редактирования и все операторы языка Питон. Примерами наиболее интересных запросов и манипуляций являются:

dir() – запрос доступных атрибутов сеанса отладки;

```
['CONFIG_FILE', 'INSTANCE_HOME', 'SOFTWARE_HOME', ...,
'app', 'debugger', 'os', 'root', 'sys']
```

dir(app) – запрос доступных атрибутов приложения отладчика;

```
[ ... , '_request', 'db', 'debug', 'fromDatabase',
'publish', 'root', 'run']
```

dir(root) – запрос доступных атрибутов корневого объекта ZODB;

```
 [..., 'data', 'get', 'getSiteManager', 'items', 'keys',
'setSiteManager', 'values']
```

app.root() – корневая папка приложения;

```
<zope.app.folder.folder.Folder object at 0x00AD75F0>
```

list(root) – список имен содержимого корневой папки;

```
[u'st']
```

root.data.get('имя') или root[‘имя’] – запрос объекта с заданным именем в корневой папке ZODB;

Пример сценария корректировки базы данных в диалоге с отладчиком:

1) Запрос ветви-контейнера с именем 'groups' и ее свойств у объекта st

```
>>> st = root['st']
>>> gr = st['groups']
>>> gr
<schooltool.app.GroupContainer object at 0x02DE8F70>
>>> list(gr)
[u'asd', u'pss', u'pss-2']
>>> type(gr)
<class 'schooltool.app.GroupContainer'>
```

2) Удаление из контейнера gr объекта с именем 'pss-2'

```
>>> gr.__delitem__('pss-2')
```

3) Проверка нового содержимого ветви

```
>>> gr = st['groups']
>>> list(gr)
[u'asd', u'pss']
```

Есть несколько методов, которые можно вызывать для прикладного объекта с целью его тестирования. Все методы отладчика имеют необязательные аргументы:

path = url – отлаживаемого объекта;

basic = user:password – имя и пароль для доступа к объекту.

Метод publish исполняет запрос и возвращает заголовок и тело ответа. Оператор вызова метода имеет вид:

```
>>> debugger.publish(path='/mainfolder/myobject')
```

Например, запрос вида

```
>>> debugger.publish(path='/st')
```

возвращает текст ответа, направляемого браузеру на запрос, заданный параметром path. Фрагменты текста приведены ниже.

```
Status 302 Moved Temporarily
Content-Length: 3203
Content-Type: text/html;charset=utf-8
Location: http://127.0.0.1/st/calendar
Set-Cookie: zope3_cs_834aaed=nD0OOQEYWC8nEm-
1O1Xm2Ijn3MUoIu8DptYy3zPrgb0guJLXlUX
3gg; Path=/;
X-Powered-By: Zope (www.zope.org), Python (www.python.org)

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
<base href="http://127.0.0.1/st/index.html" />

    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
    <link rel="icon" type="image/png"
          href="http://127.0.0.1/st/@@/favicon.ico" />
. . .
```

```
    <title>SchoolTool</title>
  </head>
  <body>
    <div id="header">
      <div class="logo">
        <a href="http://127.0.0.1/st">
          <img src="http://127.0.0.1/st/@@/logo.png"
               alt="SchoolTool" />
        </a>
      </div>

     . . . . . . . .

    </div>
  </body>
</html>
(1.6029999256134033, 1.6008998350348997, 302)
```

Метод run выполняет запрос аналогично предыдущему, но выводит лишь «посмертную» обратную трассировку вызовов методов, если произошла ошибка при обработке запроса. Например:

```
>>> debugger.run(path='/sqt')

Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "C:\Python24\Lib\site-packages\zope\app\debug\debug.py", line 115, in run
. . .
zope.publisher.interfaces.NotFound: Object:
<zope.app.folder.folder.Folder objec
t at 0x0301BF70>, name: u'sqt'
```

Для получения подробной информации о проблемах можно вызвать информатор:

```
>>> import pdb
>>> pdb.pm()
> c:\python24\lib\site-
packages\zope\app\container\traversal.py(78)publishTraver
se() -> raise NotFound(self.context, name, request)
```

и дополнительными запросами получить нужные сведения, например:

```
(Pdb) dir()
['name', 'request', 'self', 'view']
(Pdb) name
u'sqt'
(Pdb) request
<zope.publisher.browser.TestRequest instance
URL=http://127.0.0.1/sqt>
(Pdb) self
```

```
<zope.app.container.traversal.ItemTraverser object at 0x03279FB0>
(Pdb) view
(Pdb) a
self = <zope.app.container.traversal.ItemTraverser object at
0x03279FB0>
request = CONTENT_LENGTH:       0
GATEWAY_INTERFACE:      TestFooInterface/1.0
HTTP_HOST:      127.0.0.1
PATH_INFO:      /sqt
QUERY_STRING:
SERVER_URL:     http://127.0.0.1
name = sqt
```

Глоссарий

Adapter

Этот основной компонент для обеспечения дополнительной функциональности объекта. Он использует некоторый интерфейс, реализованный объектом, и обеспечивает (адаптирует, преобразует) новый интерфейс. Например, объект с интерфейсом IMessage может иметь адаптер, который обеспечивает интерфейс IMailSubscription для этого же объекта. Адаптер выполняет вычисления новых данных с использованием дополнительных методов. Является альтернативой наследованию классов в ООП, поскольку не затрагивает кода имеющихся классов.

Адаптер может рассматриваться как некоторая обложка, которая придает объекту новую или дополнительную функциональность. Конструктор адаптерного объекта всегда получает аргумент – контекст.

В дополнение адаптеры допускают множественную адаптацию для нескольких исходных интерфейсов в некоторый один интерфейс. В этом случае адаптер требует несколько контекстов

Смотри: Component Architecture

Annotation

Аннотации являются необязательными частями данных об объекте. Они включают различные мета данные, например, Dublin Core.

Комментарии обеспечивают пространство имен данных объекта и определяются своими уникальными ключами. Безопасный способ – использовать аналог деклараций XML-namespace или точечные имена.

Часто аннотации используются для хранения дополнительной информации об объекте, например, личные данные, адреса и др. Наиболее общая форма аннотации - хранение комментариев в специальном атрибуте объекта. Объект просто должен осуществлять маркерный интерфейс IAttributeAnnotable, чтобы иметь комментарии

Ссылка: zope.app.annotation.interfaces.IAnnotations,
zope.app.annotation.interfaces.AttributeAnnotatable

Cache

Кеш позволяет хранить результаты сложных операций с использованием ключей. Например, иногда нет необходимости заново генерировать страничный шаблон, когда сделан запрос, так как выход не мог измениться за короткое время. Наиболее общая реализация кеша является RAM (память только для чтения), которая хранит данные в памяти быстрого доступа. Правильное использование кеша может увеличить быстродействие вашего приложения.

Ссылка: zope.app.cache.interfaces.ICache

Checker

Контролеры описывают ограничения безопасности для конкретного типа (наследуемого класса). Получив объект и имя атрибута, контролер может сообщить: имеет ли текущий пользователь право на чтение или изменение атрибута. Контролер может также создать защищенный прокси (proxy) для данного объекта.

Общий контроль основывается на данных, где хранятся разрешения для каждого атрибута, заданных при конфигурации.

Ссылка: zope.security.interfaces.IChecker

Смотри: Security, Proxy

Component

Компонент может быть любым объектом языка Питон и является фундаментальным понятием среды Zope 3. Компоненты обычно задаются своими классами, но могут быть любыми произвольными объектами Питона. Основные компоненты - Service, Adapter, Utility, Presentation, Content и Factory.

Смотри: Component Architecture, Service, Adapter, Utility, Presentation, Factory, Interface

Component Architecture

Компонентная архитектура является набором из пяти фундаментальных реестров сервисов, в которых регистрируются специфические компоненты. Сервисами являются Service Manager (известный как Service Service), Adapter Service, Utility Service, Presentation Service (известный как View Service) и Factory Service.

Компоненты могут взаимодействовать через интерфейсы, которые они поддерживают. Используя интерфейсы и вышеупомянутые сервисы, можно строить любые основанные на компонентах системы. Преимущество компонент перед смесью базовых классов и соглашением о выборе имен в том, что компоненты позволяют значительно лучше декларировать функциональное назначение, ясные контракты взаимодействия и возможность повторного использования.

Ссылка: zope.component.interface.IComponentArchitecture

Смотри: Component, Service

Container

Контейнер является объектом, который может содержать другие объекты, различаемые по их имени. Контейнер имеет словарь (mapping), для доступа к которому может быть использован синтаксис языка Питон. Любой контент объект, который содержит дочерние, должен поддерживать интерфейс контейнера.

Контейнер может ограничить типы компонентов, которые он может содержать. Это выполняется установкой предусловий в методе __setitem__() интерфейса контейнера.

Ссылка: zope.app.container.interfaces.IContainer

Смотри: Folder, Content

Content

Этот основной компонент, хранимый в объектной базе данных или в других местах. Основной интерфейс компонент содержимого – IContentType. Компоненты содержимого обычно создаются фабриками, которые определены в конфигурации. Например, компоненты содержимого включают File, Image, TemplatedPage и SQLScript.

Ссылка: zope.app.content.interfaces.IContentType

Смотри: Component

Context

Контекст обычно является местом в иерархии объектов, где выполняется код. Имея компоненты, связанные контекстом, есть возможность обрабатывать зависящие от позиции экземпляры конкретного компонента. Это позволяет перекрывать, дополнять или обогащать функциональное назначение компонента при движении вниз в дереве объектов.

Концепция контекста использована также для адаптеров и видов. Там они описывают компонент, который обернут адаптером. Для вида, контекст является просто компонентом, для которого создан вид.

Смотри: Global Components, Local Components, Adapter, View

Doctests

Doctests обеспечивают написание блок тестов в строках документации или в простых текстовых файлах. Тест подобен экранному выводу диалогового сеанса в оболочке Питона.

Doctests позволяет использовать сами тесты как документацию в форме примеров. Они позволяют значительно легче показать шаги прохождения теста. Doctests стал главным способом написания блоков теста в Zope 3.

Ссылка: zope.testing.doctestunit

Смотри: Tests, Unit Tests, Functional Tests

Document Template Markup Language (DTML)

Язык DTML является производным от языка HTML для создания документов. DTML остался в Zope 3 как контент компонент и для написания динамических SQL скриптов. Страница DTML, в отличии от ZPT, не является правильным документом HTML или XML.

Ссылка: zope.documenttemplate

Смотри: Zope Page Template

Dublin Core

Dublin Core определяет конечный набор атрибутов мета-данных, например, "name " и “title". Для получения полного списка атрибутов, включая подробные описания, Вы можете просмотреть соответствующую главу в книге, интерфейсы, или официальный Web сайт Dublin Core http://www.dublincore.org.

Ссылка: zope.app.dublincore.interfaces.IZopeDublinCore

Смотри: Annotation, Meta-data

Event

События в Zope 3 – это объект, который представляет извещение о событии или действии системы. События может чем угодно, реализующим интерфейс IEvent. Несколько общих примеров включают IObjectCreatedEvent, IObjectModifiedEvent и IObjectCopiedEvent. Все они принадлежат группе объектных событий и всегда обработчик события получает объект.

Ссылка: zope.app.event.interfaces.IEvent

Смотри: Event Channel, Event Subscriber

Event Channel

Источник события (или в общих чертах объект ISubscribable), может посылать события в список подписчиков событий. Когда делается подписка, передаются объект подписчика, тип события и фильтр. Подписчик извещается только тогда, когда наступило событие заданного типа и оно удовлетворяется требованиям фильтра.

Можно понимать источник события в терминах менеджера почтового списка, подобно почтальону. Люди могут подписаться указанием адреса электронной почты (подписчик события) в конкретном списке корреспондентов (тип события). Когда кто-нибудь посылает почту для списка корреспондентов, менеджер списка определяет, для какого списка корреспондентов пришла почта (событие) и посылает ее (уведомляет) всем подписчикам.

Смотри: Event, Event Subscriber

Event Subscriber

Подписчик события может подписаться на любой источник событий. Когда возникает соответствующее событие, в котором заинтересован подписчик, он извещается о случившемся.

Смотри: Event, Event Channel

Factory

Фабрика, создающая новые экземпляры объектов. К тому же она может проверить, какие интерфейсы будут реализованы создаваемым компонентом. Фабрики также имеют название и описание, которые могут быть использованы в интерфейсах пользователя.

Заметьте, что фабрики не должны требовать разрешение для создания объекта, так как создание не означает необходимости иметь доступ к чему-либо на объекте.

Ссылка: zope.component.interfaces.IFactory

Смотри: Component, Component Architecture

Field

Области или поля являются расширенными описаниями атрибутов в интерфейсах класса объектов, содержат мета-данные, используемые для автоматической генерации форм ввода при создании и корректировке объекта по требованию пользователя. Примерами доступных полей являются: Text, TextLine, Int, Float, Bool, Choice, Tuple, List, Set, Dict, Date, Time, Datetime, Bytes, BytesLine и Password.

Ссылка: zope.schema.interfaces

Смотри: Schema, Form, Widget

Folder

Папка используется в пространстве контента как первичный контейнерный объект. Она не устанавливает никаких ограничения на типы объектов, которые может содержать. Кроме того, любая папка может быть преобразована в сайт. Папки реализуют интерфейс IContainer.

Ссылка: zope.app.folder.interfaces.IFolder

Смотри: Container, Content, Site

Form

Форма это вид всей схемы или частей схемы. Для браузера, форма способна произвести полную форму HTML, чей ввод данных преобразуется в объект языка Питон, проверяет и загружает экземпляр компонента, для которого прописана схема. Формы в сочетании со схемами компонент являются оригинальными решениями Zope для генерации окон ввода и отображения их пользователю.

Смотри: Schema, Field, Widget, View

Functional Tests

Этот класс тестов исполняется под управлением полностью функциональной системы. Часто используются для тестирования правильности работы сгенерированных видов и обработки данных при взаимодействии с компонентами низкого уровня. Функциональные тесты обычно зависят от конкретного типа представления, как например, окно просмотра Web или FTP.

Ссылка: zope.app.tests.functional

Смотри: Tests

Global Components

Некоторый компонент, созданный вне контекста, считается глобальным и всегда доступен. Обычно, глобальные компоненты создаются в период запуска системы Zope, главным образом через директивы ZCML.

Глобальные компоненты не могут сохранять свое состояние между запусками Zope. Всякий раз, когда Zope завершает работу все компоненты уничтожаются. Следовательно, только директивы ZCML полностью описывают состояние таких компонентов и никакой другой механизм не может модифицировать их состояние.

Смотри: Component, Local Component, Zope Configuration Markup Language

Interaction

Взаимодействие предполагает, что все независимые участники (действующие принципалы) имеют необходимое разрешение на доступ к атрибуту объекта.

Взаимодействие является основой системы безопасности, поскольку оно применяет эти правила системы в конкретных операциях.

Ссылка: zope.security.interfaces.IInteraction

Смотри: Security, Security Policy, Participation, Principal

Interface

Интерфейс в Zope, подобно многим языкам программирования и средам, формально описывает функциональное назначение объекта. Он определяет все методы и атрибуты объекта, как часть общедоступного API. Интерфейсы используются также как первичная документация API и как маркеры.

Ссылка: zope.interface.interfaces.IInterface

Смотри: Component, Component Architecture

Internationalization

Интернационализация, обычно обозначаемая как I18n, – процесс получения программного обеспечения для перевода на другие языки и регионы, включая согласование форматов чисел, даты, времени, поддержку уникода, кодирования, декодирования текстов и так далее.

Смотри: Localization, Locale

Local Components

Эти компоненты доступны только в относительном контексте или месте. Они определены на объектах сайта (специальные папки) и могут быть доступны на сайте и его потомках. Создание и конфигурация обычно выполненяются через графический интерфейс пользователя GUI или ZMI. Локальные компоненты хранятся в ZODB и могут сохранить свое состояние между запусками Zope.

Смотри: Component, Global Component, Site, Context

Locale

Локаль является объектом, который содержит специфическую информации о регионе проживания пользователя и языке, например, форматы чисел, даты, времени, имена месяцев и так далее. Zope 3 использует LDML файлы XML для получения данных о более чем 200 локалей. Подробности можно узнать в http://www.openi18n.org

Смотри: Internationalization, Localization

Localization

Локализация обычно обозначается как L10n, – фактический процесс получения программного обеспечения для конкретного региона и языка. Так как информация для региона обычно доступна через локаль, локализация в Zope 3 состоит главным образом из перевода строк из стандартного каталога сообщений.

Смотри: Internationalization, Locale, Message Catalog

Location

Позиция внутри дерева объектов, хранимых в ZODB. Объекты, которые имеют атрибут позиции, содержат информацию о родителе и своем имени. Примером компонентов, которые обычно имеют позицию, являются контент компоненты. Очевидно, что не все компоненты должны иметь позицию, например, все глобальные компоненты, создаваемые в ZCML.

Ссылка: zope.app.location.interfaces.ILocation

Смотри: Local Components, Global Components, Content

Message Catalog

Каталоги сообщений являются наборами переводов для конкретного языка и региона. Для файловых систем, формат каталога сообщений определяется стандартным методом Gettext (известны как файлы PO) и для хранения используется структура папок, тогда как локальная версия использует структуры языка Питон.

Ссылка: zope.i18n.interfaces.IMessageCatalog

Смотри: Localization, Domain

Meta-data

Мета-данные в Zope 3 – дополнительные данные об объекте, не хранимые в атрибутах самого объекта. Используются для лучшей интеграции объектов в среду. Примеры мета-данных включают “title”, “author”, “size” и “modification data”.

Смотри: Annotation, Dublin Core

Namespace

В Zope 3 этот термин, использован в двух смыслах. При работе с XML (ZCML или ZPT), термин namespace ссылается на XML namespaces, который используется в директивах ZCML.

Другое использование namespace – в прослеживании имен в URL. Всякий раз, когда сегмент пути начинается и заканчивается символами "++", доступно новое namespace для прохождения при поиске. Namespaces при прохождении первоначально использовались для выделения текстов из программного обеспечения при формировании документации, для включения новых параметров используемого обличья или для виртуальной обработки URL. Доступные namespaces включают: etc, view, resource, attribute, item, acquire, skin, help, vh и apidoc..

Ссылка: zope.app.traversing.namespace

Смотри: Zope Configuration Markup Language, Traversal

Pair Programming

Когда два программиста сидят за одним компьютером и разрабатывают вместе часть программного обеспечения. Идея в том, что оператор (который вводит текст), постоянно проверяется вторым человеком на наличие опечаток, дефектов и проектных ошибок. Парное программирование также ускоряет поиск проектных решений и есть оперативный обмен идеями.

Смотри: Sprint

Participation

Участие, которое лучше было бы называть участник, – одно из лиц (принципал) при взаимодействии. Участие основано на взаимодействии пары лиц.

Ссылка: zope.security.interfaces.IParticipation

Смотри: Security, Principal, Interaction, Security Policy

Permission

Разрешения используются для допуска или не допуска потребителя к атрибуту объекта. Они являются минимальной частью данных в списке управления доступом. Разрешения являются обычными строками, за исключением разрешения zope.security.checker.CheckerPublic, которое делает атрибут доступным для всех (публичный).

Смотри: Security, Checker, Role

Persistent

Объект считается "устойчивыми", если он может храниться в ZODB и изменения его атрибутов регистрируются и сохраняются автоматически. Устойчивые объекты должно быть потомками класса persistent.Persistent или обеспечивать другую реализацию интерфейса IPersistent.

Ссылка: persistent.interfaces.IPersistent

Смотри: Zope Object Database

Presentation

Компоненты представления обеспечивают интерфейс между внутрисистемными компонентами и пользователем или другим протоколом связи. Они включают Browser, WebDAV, XML-RPC и FTP. Тем не менее, выход может быть некоторым изображением. В этом смысле, компоненты представления похожи на адаптеры, за исключением того, что они обычно преобразовывают данные для внешних интерфейсов, а не для внутренних элементов Python/Zope.

Если компонент представления не создает выходного представления (фрагмент документа на языке HTML), тогда он может обеспечить формальный интерфейс. Этот тип представления затем используется другими компонентами представления. Ярким примером таких посредников являются виджеты, которые обеспечивают виды полями.

Ссылка: zope.component.interfaces.IPresentation

Смотри: Component, Component Architecture, View, Resource

Principal

В общем случае, принципал является агентом, использующим систему. Система может связать разрешения с принципалом и, следовательно, предоставить ему доступ к объектам, которые требуют разрешения. Принципалами могут быть сертификаты безопасности, группы, и отдельные пользователи.

Ссылка: zope.app.security.interfaces.IPrincipal

Смотри: Security, Permission, User

Proxy

Прокси (уполномоченный) это специальные оболочки, содержащие объекты, чтобы защитить их или придать им дополнительную функциональность. В Zope основной класс полномочий определен в пакете безопасности и отвечает за обеспечение защитной оболочки вокруг объекта, чтобы только принципалы с достаточными разрешениями могли иметь доступ к атрибутам объекта.

Ссылка: zope.security.proxy.Proxy

Смотри: Security, Checker

Publisher

Публикатор Zope, ответственный за обработку запроса в приложении Zope. Этим самым ему доверяется информация запроса, поиск объектов, формирование выхода и данные о принципале, который послал запрос. Наиболее часто процесс издания делегируется другим компонентам.

Ссылка: zope.publisher.interfaces.IPublisher

Смотри: Request, Principal, Component

Relational Database Adapter

Адаптер реляционной базы данных является посредником между Zope и конкретной СУБД. Для каждой СУБД есть свои собственные адаптеры. Адаптеры позволяют получить доступ на чтение и запись к таблицам БД.

Ссылка: zope.app.rdb.interfaces.IZopeDatabaseAdapter

Python Developer

Лицо, выполняющее разработку кода пакетов на языке Питон и продуктов для Zope 3. Разработчики Питона являются наиболее продвинутой группой разработчиков. Ожидается, что они очень хорошо знают язык Питон и знакомы с общим программированием и объектно-ориентированной разработкой.

Request

Запрос содержит всю информацию, которую система извлекает из обращения пользователя к издателю. Информация в запросе включают путь к доступному объекту, пользователя, регион и язык пользователя, выходной формат возвращаемых данных, возможные переменные среды и входные данные.

Ссылка: zope.publisher.interfaces.IRequest

Смотри: Publisher, User

Resource

Ресурс является компонентом представления и не зависит от других компонент. Обычно поставляет контекстно-независимые данные. Наиболее часто использовано ресурсы для окна просмотра для обеспечения их CSS, Javascript и образами при формировании HTML-страниц.

Ссылка: zope.component.interfaces.IResource

Смотри: Component, Presentation, View

Role

Роли являются коллекцией разрешений, которые могут быть предоставлены принципалу. Роли предусмотрены стандартным полисом безопасности Zope, который ответственный за управление ролями.

Отметьте, что роль является понятием, которое не обязано быть предусмотрено всеми полисами безопасности и, следовательно, прикладной код пакета не должен зависеть от них.

Ссылка: zope.products.securitypolicy.interfaces.IRole

Смотри: Security, Permission, Principal

Schema

Схема является интерфейсом, который содержит поля вместо методов и атрибутов. Схемы используются для обеспечения дополнительных мета-данных полей, которые они поддерживают. Эти дополнительные данные помогает системе проверять значения и автоматически генерировать интерфейсы пользователя с использованием виджетов в формах HTML.

Смотри: Field, Form, Widget

Scripter

Автор текстов скриптов в Zope3, имеющая в своем распоряжении классический язык HTML, CSS и Javascript. Они используют Zope для быстрой по возможности разработки динамических страниц. Они не знакомы с программированием и формальной разработкой. Zope 3 обеспечивает для этой группы людей различные средства, включая разнообразные шаблоны и высокоуровневую TTW разработку компонентов.

Security

Zope имеет хорошо проработанную модель безопасности, позволяющую защищать свои компоненты в окружении, которому не следует полностью доверять. Всякий раз, когда компонент запрашивается непроверенным кодом, он вкладывается в защитную оболочку, прокси. Когда такой код просит атрибут, то объектный контроль независимо проверяет полномочия зарегистрированного клиента на доступ к атрибуту. В результате этот процесс принимает решение, имеет ли принципал необходимое разрешение для атрибута объекта.

Смотри: Checker, Permission, Principal, Proxy, Role, Security Policy, Interaction, Participation

Security Policy

Полис безопасности является основным при взаимодействии. Только ответственность пользователя должна обеспечивать взаимодействие. Хотя роли не обязательны для функционирования системы безопасности, встроенный в Zope полис безопасности обеспечивает расширенные средства управления и использования ролей в процессе принятия решений.

Ссылка: zope.security.interfaces.ISecurityPolicy

Смотри: Security, Interaction, Participation, Principal, Permission, Role

Service

Сервисы обеспечивают фундаментальное функциональное назначение и необходимы для правильного функционирования системы. Сервисы, в отличие от многих других компонентов, не удаляются сами или не создаются заново каждый раз, когда они вызываются. Следовательно, можно считать, что они имеют некоторый фиксированный статус. Глобальные сервисы всегда полностью встроены и получают все данные из процесса конфигурации, в отличии от них локальные реализации сервисов могут сохраняться в ZODB и загружаться любое число раз в период исполнения.

Смотри: Component, Component Architecture

Session

Сессии позволяют хранить состояние пользователя обычно в браузере клиента. Особенно важно то, что если связь с потребителем после некоторого запроса разорвана, то состояние должно быть потеряно. Стандартный протокол HTTP закрывает связь после обработки каждого запроса клиента и не поддерживает длительных взаимодействий.

Ссылка: zope.app.session.interfaces.ISession

Site

Сайт является папкой, которая может содержать программное обеспечение и конфигурацию. Он обеспечивает связь пространства содержимого и локального программного пространства, допускает разработку TTW (through-the-web) компонент, например, динамических страниц. Из другой точки зрения, сайт просто обеспечивает локальный сервис менеджера. Папки могут быть всегда преобразованы в сайт.

Ссылка: zope.app.site.interfaces.ISite

Смотри: Component, Component Architecture, Service, Folder

Sprint

Спринт в общих чертах это двух на трех дневные сессии интенсивной разработки программ в технологии экстремального программирования. Включает этапы кодирования, тестирования и документирования, выполняемые парой программистов для выполнения технического задания на разработку. Спринт был использован в период разработки Zope 3, чтобы ускорить разработку и предоставлять программное обеспечение заинтересованным участникам.

Смотри: Pair Programming

Template Attribute Language (TAL)

Это расширение языка HTML позволяет нам создавать скрипты на стороне сервера в шаблонах страниц, не нарушая стандарты HTML. Шаблон является правильным кодом языка HTML, что позволяет в отличии от DTML использовать для разработки инструментальные средства класса WYSIWIG подобные Dreamweaver для редактирования документов.

Ссылка: zope.tal

Смотри: TALES, Zope Page Template

Template Attribute Language Expression Syntax (TALES)

TALES являются выражениями, вычисляющими значения и возвращающими результаты при формировании документа. Используются различные типы выражений.

Ссылка: zope.tales.engine

Смотри: TAL, Zope Page Template

Tests

Тесты проверяют программное обеспечение на соответствие функциональному назначению и обнаруживают возможные ошибки. Эта техника программирования первоначально использовалась в экстремальном программировании, которое было использовано при разработке Zope 3. Есть несколько уровней тестирования: блоки, регрессии и функциональные тесты.

Смотри: Unit Tests, Functional Tests

Through-the-Web Development

Понятие TTW имеет отношение к процессу разработки программного обеспечения через интерфейс менеджера Zope 3 (ZMI). TTW разработка часто проще, чем разработка продукта на языке Питон и обеспечивает авторов средствами миграции в формальную компонентно-ориентированную разработку. Разработанные TTW компоненты обычно называют локальными компонентами, так как они применимы только для сайта, где они были разработаны.

Смотри: Zope Management Interface, Site, Local Component

Transaction

Транзакция является набором операций в базе данных, в нашем случае ZODB. Транзакции Zope аналогичны таковым в реляционных базах данных, они могут быть начаты, завершены или прерваны. При подтверждении, если набор операций не вызывает проблем, то они выполняются. Если происходят ошибки, то создается исключительная ситуация и незаконченный список операций может быть прерван.

Процесс сначала независимо проверяет возможность успешного выполнения операций и затем производит их фактическое выполнение, известное как двухфазная фиксация транзакции. Двухфазное подтверждение важно для целостности и непротиворечивости данных.

Ссылка: ZODB.interfaces.ITransaction

Смотри: Publisher, ZODB

Translation Domain

Определяет место поиска переводов фраз при интернационализации приложений.

Ссылка: zope.i18n.interfaces.ITranslationDomain

Смотри: Utility, Domain, Message Catalog, Internationalization, Localization

Traversal

Прохождение (поиск) является процессом преобразования пути в фактический объект относительно базового объекта (начальная точка). Прохождение является центральным понятием в Zope 3 и его поведение может разным в зависимости от цели использования.

Например, если Вы просматриваете URL, поступивший из окна браузера клиента, то механизм прохождения должен быть способным просматривать пространства имен, виды и другие особенности и не может быть простым поиском объекта. Возможно изменение правил прохождения для данного объекта за счет регистрации пользовательского компонента прохождения.

Ссылка: zope.app.traversing.interfaces.ITraverslAPI

Смотри: Component, Namespace, View

Unit Tests

Блочные тесты проверяют правильное функционирование API и деталей реализации единственного компонента и его поведение в конкретной рабочей среде. Блочные тесты являются наиболее общими тестами и должны существовать для

каждого компонента системы. Каждый пакет Zope 3 должен иметь модуль тестов, который содержит блоки тестов.

Ссылка: unittest

Смотри: Tests, Doctests, Functional Tests

User

Потребитель, получающий доступ к системе через любой метод связи. Различают авторизованных и неавторизованных пользователей. Система соотносит различные данные с пользователем, включая имя и пароль, область, язык и может быть даже компьютер, с которого он имеет доступ к Zope 3. Приложения могут присоединить другие данные к пользователям, основанные на своих потребностях.

Ссылка: zope.app.pluggableauth.interfaces.IUserSchemafied

Смотри: Security, Principal

Utility

Это основной компонент, обеспечивающий функциональное назначение системы, не зависящее от состояния. Если конкретная утилита отсутствует, она не должна приводить к краху всю систему. Хорошие примеры утилит – соединения с базами данных, почтовые службы, кеши и языковые переводчики.

Если у вас есть проблема выбора сервиса или утилиты для конкретного функционального назначения, обдумайте следующее: Если потребность просто в регистрации компонентов, которые будут использованы, тогда лучше осуществить это через утилиту; сервис регистрации утилит позволит вам потребовать все утилиты, которые реализуют определенный интерфейс.

Смотри: Component, Component Architecture, Service

View

Виды являются компонентами представления для других компонентов. Для функционирования они всегда требуют компонент (узнанный как контекст) и запрос. Виды могут брать много форм, основанных на типе представления запроса. Для оконных видов, например, они обычно просто оценивают шаблон, чей результат (HTML) возвращается клиенту.

Ссылка: zope.component.interfaces.IView

Смотри: Component, Component Architecture, Presentation, Resource

Virtual Hosting

Виртуальный север – характерный для протокола HTTP режим допускает запуск сервера Zope под управлением другого веб-сервера (например, Apache) правильно перенаправляя ему запросы и возвращая ответы на браузер клиента. Виртуальный URL может быть определен через пространство имен "++vh++".

Смотри: Namespace

Volatile Objects

Летучие объекты появляются в вашем пути прохождения, но не хранятся в ZODB и уничтожаются при окончании транзакции. Они не могут быть использованы для загрузки устойчивых объектов. Наиболее очевидный пример – объект SQL, который извлекает и загружает данные в реляционную базу.

WebDAV

WebDAV – расширение для HTTP, которое определяет дополнительные операции, используемые для лучшего управления доступными ресурсами Сети. Например, это позволяет Вам, чтобы загружать любые мета данные об объекте и блокировать/деблокировать их при редактировании. В Zope 3 WebDAV реализовано частично.

Ссылка: zope.app.dav.interfaces

Widget

Виджет является простым полем просмотра характерным для представления. Для виджетов окна просмотра используется автоматическая генерация элементов HTML для отображения и ввода данных атрибута объекта. Виджеты могут также преобразовать данные, поступившие из окна просмотра клиента, в необходимый объект Питона, который удовлетворяет типу поля.

Ссылка: zope.app.form.interfaces.IWidget

Смотри: Field, Form, Schema, View

Workflow

Управление состоянием, в котором может находиться объект, и процессом изменения состояния. Есть два метода в workflows. Первый следит за состоянием объекта. Состояние может быть изменено с использованием перехода. Если пользователь имеет необходимое разрешение, то он может вызвать переход в другое состояние. Другая модель (разработанное WfMC), использует деятельность, через которую объект может пройти. Пока в Zope3 реализован только первый вариант.

Ссылка: zope.app.interfaces.workflow

ZConfig

Этот пакет допускает запись файлов конфигурации наподобие Apache, которые автоматически преобразуются в объекты конфигурации. ZConfig полезен для конфигураций, которые могут редактироваться администраторами, поскольку синтаксис записи хорошо знаком. Zope 3 использует файлы ZConfig для конфигурирования своих серверов, ZODB и журналов.

Ссылка: ZConfig

Zope Configuration Markup Language (ZCML)

ZCML – специальный диалект XML для задания конфигураций приложений в компонентной архитектуре Zope 3 в период запуска. Все глобальные компоненты регистрируются посредством ZCML, за исключением нескольких компонентов начальной загрузки. Язык ZCML легко может быть расширен реализацией новых пространств имен и новых директив.

Смотри: Component, Component Architecture

Zope Management Interface (ZMI)

ZMI – формальное имя графического веб-интерфейса пользователя Zope 3, который используется для управления экземплярами компонент содержимого и TTW программного обеспечения.

Смотри: Content, Through-the-Web Development

Zope Object Database (ZODB)

ZODB хранит все устойчивые данные в Zope. Это масштабируемая, прочная и хорошо отлаженная объектная база данных, полностью написанная на языке Питон. Используя механизм устойчивых данных, объекты могут быть загружены без дополнительного кода. ZODB также имеет двухфазную фиксацию транзакций и масштабируемость, гарантирующие работу с распределенными данными по нескольким машинам, использующим пакет Zope Enterprise Option (ZEO).

Ссылка: ZODB

Смотри: Persistent, Transaction

Zope Page Template (ZPT)

Страничные шаблоны, объединяющие возможности языков TAL, TALES и METAL. Вы можете писать внутри шаблонов так называемые скрипты, которые выполняются в период исполнения запроса. Шаблоны обеспечивают текущий контекст, данные окружения и базовые имена для использования в выражениях языка TAL. Все это делает шаблоны чрезвычайно полезным инструментом при разработке интерфейсов с конечным пользователем на языке HTML в Zope 3.

Ссылка: zope.pagetemplate.pagetemplate.PageTemplate

Смотри: TAL, TALES

Список использованных источников

1 Сузи Р.А. Python. [Текст] – СПб: БХВ-Петербург, 2002. – 768 с.

2 Краткое введение в Питон. [Электронный ресурс] http://zope.net.ru

3 Programming with the Zope 3. Component Architecture. Tutorial for Python Programmers. [Электронный ресурс] http://dev.zope.org/Zope3/programers_tutorial.pdf

4 Zope3Book. [Электронный ресурс] http://www.zope.org/Wikis/DevSite/Projects/ComponentArchitecture/FrontPage/Zope3Book

5 HyperText Markup Language (HTML) Home Page. [Электронный ресурс] http://www.w3.org/MarkUp/

6 The Zope Book (2.6 Edition).Basic DTML [Электронный ресурс] http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/DTML.stx

7 The Zope Book (2.6 Edition). Advanced DTML [Электронный ресурс] http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/AdvDTML.stx

8 The Zope Book (2.6 Edition). Using Zope Page Templates [Электронный ресурс] http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/ZPT.stx

9 The Zope Book (2.6 Edition). Advanced Page Templates [Электронный ресурс] http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/AdvZPT.stx

10 ZWiki for Zope 3. [Электронный ресурс] http://www.zope.org/Products/Zope3-Packages/zwiki/view

11 Айзекс С. Dynamic HTML. – СПб.: БХВ-Петербург, 2001. – 496 с.

12 Спикльмайр С. и др. Zope. Разработка Web-приложений и управление контентом: Пер. с англ. – М.: ДМК Пресс, 2003. – 464 с.

Приложение А

А1 Основы языка Питон

Программирование на языке Python (следует читать «Пайтон», но допустимо использовать перевод «Питон») становится все более заметным явлением среди программистов, чему способствуют следующие его качества.

- Современный, интенсивно развивающийся язык объектно-ориентированного программирования, адаптированный для обработки объектов со сложной, изменяемой во время исполнения программ структурой. По своей идеологии язык близок к фреймам, широко используемым в технологиях искусственного интеллекта.
- Реальная, а не декларируемая независимость от платформы, на которой исполняется приложение. Имеется поддержка языка на платформах Unix/Linux, MS Windows, Mac и других. Для переноса приложения на другую платформу не требуется какой-либо переработки текста программ.
- Встроенная поддержка потребностей системного программирования (документирование, доступ к информации о программе и среде исполнения, интерпретация кода, модульность, расширяемость).
- Свободное распространение с открытым кодом основных модулей и поддержка разработки, верификации и сопровождения проекта мировым сообществом программистов (по аналогии с ОС Linux).
- Наличие среды Zope для разработки высокоэффективных веб-приложений на языке Python. Zope интегрирует в себе публикатор веб-сервера, средства разработки новых компонент с произвольной бизнес-логикой, а также создания, хранения и управления содержимым, средства авторизации доступа, утилиты и пакеты для взаимодействия с СУБД. Идеология Zope приближена к прогрессивной технологии CORBA для создания интегрированных информационных систем масштаба корпорации и более.

Языку Python посвящено множество учебных и профессиональных материалов [1–2], многие из которых можно найти в Интернете. Для знакомства с языком Python программистам, уже владеющим другими языками объектно-ориентированного программирования, важно иметь компактное справочное руководство с примерами написания типовых программ. Наиболее подходящим для этого является материал «Python 2.4 Quick Reference», размещенный в Интернете по адресу http://rgruet.free.fr/PQR24/PQR2.4.html. Данное руководство содержит выполненный составителем перевод его основных разделов, дополненный примерами написания программ с использованием модулей доступа к базам данных и создания оконных интерфейсов.

Все вопросы получения дистрибутивов и установки программного обеспечения отражены на сайтах http://www.python.org, http://www.python.ru, http://www.zope.org, http://zope.net.ru, http://plone.org.ru, http://itconnection.ru/ и других.

А1.1 Лексические соглашения

Ключевые слова

and	**del**	**for**	**is**	**raise**
assert	**elif**	**from**	**lambda**	**return**
break	**else**	**global**	**not**	**try**
class	**except**	**if**	**or**	**while**
continue	**exec**	**import**	**pass**	**yield**
def	**finally**	**in**	**print**	

- Список ключевых слов имеется в модуле *keyword*.
- Запрещенные символы (допустимы только в строках): $?
- Предложение должно целиком находиться в одной строке или переноситься на следующую с использованием символа "\". Исключением является разрешение переносов внутри скобок (), [], {} и для строк в тройных кавычках.
- В одной строке может появиться более чем одно утверждение, если они разделены символом «точка с запятой».
- Комментарии начинают с символа "#" и продолжаются до конца строки.

Идентификаторы

{*letter* | "_"} [*letter* | *digit* | "_"]…

- Ключевые слова и идентификаторы — регистрочувствительные.
- Специальные формы: _*ident* (не импортируются оператором «from module import *»); __*ident*__ (системное имя); __*ident* (внутреннее имя класса).

Строки

Два варианта: ***str*** (8 бит на символ – простые старые строки) и ***unicode*** (16 бит на символ – строки UCS2).

Примеры:

"строка в двойных кавычках"

'другой вариант строки в одиночных кавычках, допускает символ “ внутри'

'''строки, которые могут внутри содержать символ новой строки и кавычки'''

""" аналогичен предыдущему примеру """

u'*unicod*-строка'

U"Другая Unicode-строка"

r'однострочный литерал, в котором можно использовать символ «\» для записи регулярных выражений и путей к файлам Windows'

R"другой вариант однострочного литерала " — строку нельзя переносить

ur'однострочный литерал в кодировке *unicode'*

UR"другой вариант однострочного литерала в кодировке *unicode* "

- Чтобы продолжить строку на следующей, используйте символ «****» в конце строчки.
- Смежные строки объединяются в одну, например, 'Monty ' 'Python' понимаются как 'Monty Python'.
- Сцепление строк: u'hello' + ' world --> u'hello world' (в уникоде).
- Строки и тъюплы нельзя модифицировать (не мутируемые объекты).

Булевские константы

- True
- False

Значения True и False являются целыми числами 1 и 0 соответственно.

Числа

- **Decimal** integer: 1234, 1234567890546378940**L** (или **l**).
- **Octal** integer: **0**177, **0**177777777777777777L (начинаются с **0**).
- **Hex** integer: **0x**FF, **0X**FFFFffffFFFFFFFFFFL (начинаются с **0x** или **0X**).
- **Long** integer (неограниченная точность): 1234567890123456**L** (оканчиваются символом **L** или **l**) или **long(**1234**)**. Представлены одинаково с **Decimal** integer.
- **Float** (удвоенная точность): 3**.**14**e-10**, .001, 10., 1E3 и др.
- **Complex**: 1**J**, 2**+**3**J**, 4**+**5**j** (оканчиваются символом **J** или **j; +** разделяет действительную и мнимую части *float*).

Последовательности

- **Строки** (типы **str** и **Unicode,** см. п. «Строки», с. 4): '', '1', "12", 'hello\n'.
- **Тъюплы** (тип **tuple**): **( )**, **(** 1**,)**, **(**1,2**)**; запятая обязательна при длине, равной 1.
- **Списки** (type **list**): **[ ]**, **[** 1 **]**, **[** 1,2 **]**.

Индексация начинается с нуля. Отрицательные индексы отсчитывают элементы начиная с конца последовательности.

Вырезка последовательности — **имя[*[начало]* : *[конец] [*: *шаг]*]**.
По умолчанию начало — нуль, конец — **len(имя)**, шаг — 1.

```
a = (0,1,2,3,4,5,6,7)
a[3] == 3
a[-1] == 7
a[2:4] == (2, 3)
```

```
a[1:] == (1, 2, 3, 4, 5, 6, 7)
a[:3] == (0, 1, 2)
a[:] == (0,1,2,3,4,5,6,7)    # создание копии
последовательности
a[::2] == (0, 2, 4, 6)       # только элементы в четных
позициях
a[::-1] = (7, 6, 5, 4, 3 , 2, 1, 0)   # обратный порядок
```

Словари (Mapping)

Словари (type **dict**): **{ }**; **{** 1 : 'first' **}**; **{** 1 : 'first', 'two': 2, ***key*:*value* }**

Ключи ***key*** должны быть хешируемого типа (для которого определена хеш-функция). Значения ***value*** могут быть любого типа.

А1.2 Выражения и порядок их вычисления

Таблица 1.

Таблица 1

Операции

<table>
<tr><th>Оператор</th><th>Комментарий</th></tr>
<tr><td>, [...] {...} `...`</td><td>Создание tuple, list и dict; преобразование в строку</td></tr>
<tr><td>s[I] s[i:j] s.attr f(...)</td><td>Индексирование и вырезка; атрибут; вызов функции</td></tr>
<tr><td>+x, -x, ~x</td><td>Унарные операции</td></tr>
<tr><td>x ** y</td><td>Возведение в степень</td></tr>
<tr><td>x * y x / y x % y</td><td>Умножение, деление, остаток по модулю</td></tr>
<tr><td>x + y x - y</td><td>Сложение, вычитание</td></tr>
<tr><td>x << y x >> y</td><td>Сдвиги битов</td></tr>
<tr><td>x & y</td><td>Побитовое «And»</td></tr>
<tr><td>x ^ y</td><td>Побитовое «Xor»</td></tr>
<tr><td>x | y</td><td>Побитовое «Or»</td></tr>
<tr><td>x < y x <= y x > y
x >= y x == y x != y x
<> y
x is y x is not y
x in s x not in s</td><td>Сравнения: меньше, не больше, больше,
не меньше, равно, не равно,
не равно
Проверка идентичности
Проверка вхождения в последовательность</td></tr>
<tr><td>not x</td><td>Булевское «Нет»</td></tr>
<tr><td>x and y</td><td>Булевское «И»</td></tr>
<tr><td>x or y</td><td>Булевское «Или»</td></tr>
<tr><td>lambda args: expr</td><td>Анонимная функция</td></tr>
</table>

- Старшинство операций определяется их порядком в таблице сверху вниз от высшего к низшему.

- Альтернативные имена определены в модуле **operator** (например, __add__ и add для +).
- Большинство операторов являются перекрываемыми.

Примечания:

1. Проведение сравнения может быть перекрыто для данного класса определением специального метода __cmp__
2. X < Y < Z < W ожидает результат, в отличие от языка Си.
3. Идентификация объекта id(obj) не является значением объекта (внутренний номер объекта).

Значение None

- **None** используется как встроенное возвращаемое значение в функциях. Встроенный простой объект типа **NoneType** может стать ключевым словом в будущем.
- Значение **None** не печатается при работе «Питона» в диалоге.
- **None** — константа; попытка связывать имя **None** со значением — синтаксическая ошибка.
- Значение None, нуль и пустые последовательности рассматриваются в условиях как False, остальные — как True.
- **int(x), long(x), float(x)** преобразовывают аргумент к указанному типу.
- Если **z** — комплексное число, то **z.real** и **z.imag** — соответственно его действительная и мнимая части.

A1.3 Последовательности

Таблица 2

Операции для всех типов последовательностей (list, tuple, string)

Операции	Результат	Примечание
x **in** s	**True** если s содержит x, иначе **False**	(3)
x **not in** s	**False** если s не содержит x, иначе **True**	(3)
s1 **+** s2	Конкатенация s1 и s2	
s * n, n * s	Конкатенация n копий s	
s[I]	i-й элемент s, начиная с 0	(1)
s[I : j] s[I : j : step]	Вырезка из s от i (включительно) до j (не включая)	(1), (2)
len(s)	Длина s	
min(s)	Минимальный элемент s	
max(s)	Максимальный элемент s	
reversed(s)	Возвращает итератор (не последовательность)	

Операции	Результат	Примечание
	с обратным порядком элементов	
sorted(iterable [, cmp] [, cmp=cmpFct] [, key=keyGetter] [, reverse=bool])	Новая копия отсортированной последовательности	

Примечания:

1. Если значения i или j отрицательные, то индекс исчисляется относительно конца строки, заменяется на len(s) + i или len(s) + j. Но заметьте, что -0 — все еще 0.
2. Вырезка из s от i до j определена как последовательность с индексом k, таким что i<= k < j. Если i или j больше, чем len(s), используется len(s). Если i опущен, используется len(s). Если i больше или равен j, то вырезка пустая.
3. Для строк: в более ранних версиях языка **x** должно быть единственной символьной строкой. Начиная с версии 2.3, **x in s** — True, если **x** — подстрока s.

Таблица 3

Операции для списков

Операции	Результат	Примечание
s[I] = x	i-й элемент заменяется значением x	
s[I : j [: step]] = t	Вырезка элементов от i до j заменяются на t	
del s[I : j [: step]]	Эквивалентно s[i:j] = []	
s.**append**(x)	Эквивалентно s[len(s) : len(s)] = [x]	
s.**extend**(x)	Эквивалентно s[len(s):len(s)]= x	(5)
s.**count**(x)	Возвращает число элементов, для которых s[i] == x	
s.**index**(x[, start[, stop]])	Индекс первого элемента со значением s[i]==x в вырезке списка от start до stop	(1)
s.**insert**(i, x)	Эквивалентно s[i:i] = [x] if i>= 0. Если i == -1 вставить перед последним элементом	
s.**remove**(x)	Эквивалентно del s[s.index(x)]	(1)
s.**pop**([I])	Эквивалентно x = s[i]; del s[i]; return x	(4)
s.**reverse**()	Развернуть список на месте	(3)
s.**sort**([cmp]) s.**sort**([cmp=cmpFct] [, key=keyGetter] [, reverse=bool])	Сортировать список на месте	(2), (3)

Примечания:

1. Создает исключение ValueError, когда x не обнаружен в s (или выход из диапазона индексов).
2. Метод sort() берет дополнительный аргумент **cmp**, определяющий функцию сравнения, получающую 2 элемента списка и возвращающую -1, 0 или 1, если 1-й аргумент считается меньшим, равным, или большим 2-го аргумента. Отметьте, что это значительно замедляет процесс сортировки.
   В версии 2.4 аргумент cmp может быть определен как ключевое слово, и добавлены два дополнительных ключевых аргумента:
   - key — функция, которая берет пункт списка и возвращает ключ для использования в сравнении (быстрее, чем cmp);
   - reverse: если равно True, то список разворачивается.

   Начиная с версии «Питон 2.3» сортировка гарантированно «стабильна». Это означает, что два данных с равными ключами возвращаются в той же последовательности, в какой они были введены. Например, можно отсортировать список людей по имени, затем сортировать список по возрасту и закончить списком, отсортированным по возрасту, где имена людей с одинаковым возрастом отсортированы по алфавиту.
3. Методы sort() и reverse() модифицируют список на месте для экономии памяти при сортировке или обращении больших списков. Они не копируют возвращаемый отсортированный или обратный список.
4. Метод pop() не поддерживается другими мутируемыми последовательностями, кроме списков. Дополнительный аргумент i устанавливается по умолчанию в -1, чтобы по умолчанию последний пункт был возвращен и удален.
5. Поднимает исключение TypeError, когда x — не список.

Таблица 4

Операции со словарями

Операции	Результат	Примечание
len(d)	Число элементов в словаре d	
dict() **dict**(**kwargs) **dict**(iterable) **dict**(d)	Создание пустого словаря Создание словаря заполнением его аргументами из kwargs Создание словаря из пар (key, value) в iterable Создание новой копии словаря d	
d.**fromkeys** (iterable, value=None)	Метод класса для создания словаря с ключами из iterable и всеми значениями value	
d[k]	Элемент d с ключом k	(2)
d[k] = x	Заменить d[k] на x	
del d[k]	Удалить d[k] из d	(2)
d.**clear**()	Удалить все элементы из d	
d.**copy**()	Поверхностная копия d	
d.**has_key**(k) k **in** d	Проверка наличия ключа k в словаре d	
d.**items**()	Список пар (key, item)	(3)

Операции	Результат	Примечание
d.**keys**()	Список ключей	(3)
d1.**update**(d2)	for k, v in d2.items(): d1[k] = v В версии 2.4 разрешены update(**kwargs) и update(iterable)	
d.**values**()	Список значений	(4)
d.**get**(k, defaultval)	Элемент d с ключом k	(4)
d.**setdefault**(k [, defaultval])	d[k] если k in d, иначе установить defaultval	(5)
d.**iteritems**()	iterator из пар (key, value)	
d.**iterkeys**()	iterator из ключей	
d.**itervalues**()	iterator из значений	
d.**pop**(k [,default])	Удалить ключ k и вернуть значение. Если ключа нет, то вернуть default. Если значение default не задано, то возбудить KeyError	
d.**popitem**()	Удалить ключ и вернуть (key, value)	

Примечания:

1. Возбуждается TypeError, если ключ неприемлемый.
2. KeyError, если ключа k нет в словаре.
3. Ключи и значения указаны в произвольном порядке.
4. Никогда не создает исключение, если k нет в словаре, взамен возвращает defaultval. Значение defaultval не обязательно. Когда его нет и k отсутствует в словаре, то возвращается None.
5. Никогда не создает исключение, если k нет в словаре, взамен возвращается defaultVal. Добавляет ключ k со значением defaultVal. Значение defaultval не обязательно. Когда его нет и k отсутствует в словаре, то возвращается None, и ключ добавляется в словарь.

A1.4 Строковые данные

Таблица 5

Операции со строками (тип str и unicode)

Операции	РЕЗУЛЬТАТ	Приме-чание
s.**capitalize**()	Копия строки с первыми заглавными буквами	
s.**center**(width)	Копия по центру области шириной width	(1)
s.**count**(sub [,start [,end]])	Число вхождений подстроки в строку	(2)
s.**decode**([encoding [,errors]])	Строка unicode с использованием encoding. Обратный методу encode.	(3)

Операции	РЕЗУЛЬТАТ	Приме-чание
s.**encode**([encoding [,errors]])	Строка в кодировке encoded. Обратный методу decode	(3)
s.**endswith**(suffix [,start [,end]])	Возвращается True, если s заканчивается suffix; в противном случае — False	(2)
s.**expandtabs**([tabsize])	Возвращает копию s, где все символы tab заменены пробелами	(4)
s.**find**(sub [,start [,end]])	Возвращает первый индекс в s, где найдена подстрока или –1, если подстроки нет	(2)
s.**index**(sub [,start [,end]])	подобно find(), но вызывает ValueError, когда подстрока не обнаружена	(2)
s.**isalnum**()	Возвращается True, если все символы в s — буквы и цифры; в противном случае — False	(5)
s.**isalpha**()	Возвращается True, если все символы в s — буквы; в противном случае — False	(5)
s.**isdigit**()	Возвращается True, если все символы в s — цифры; в противном случае — False	(5)
s.**islower**()	Возвращается True, если все символы в s — строчные; в противном случае — False	(6)
s.**isspace**()	Возвращается True, если все символы в s — пробелы; в противном случае — False	(5)
s.**istitle**()	Возвращается True, если все символы в s — заголовочные; в противном случае — False	(7)
s.**isupper**()	Возвращается True, если все символы в s — заглавные; в противном случае — False	(6)

Окончание табл. 5

Операции	РЕЗУЛЬТАТ	Приме-чание
separator.**join**(seq)	Возвращает конкатенацию строк из набора seq, разделенных знаком separator, например: ",".join(['A', 'B', 'C']) -> "A, B, C"	
s.ljust/rjust/center(width[, fillChar=' '])	Выравнивание строки в области шириной width	(1), (8)
s.**lower**()	Копия строки s строчными буквами	
s.**lstrip**([chars])	Копия s с удаленными ведущими символами chars (по умолчанию — пробелы)	
s.**replace**(old, new[, maxCount =-1])	Копия s с заменой первых указанных в maxCount (-1: не ограничено) вхождений подстроки old на new	(9)
s.**rfind**(sub [, start [, end]])	Возвращает самый верхний индекс в s, где обнаружена подстрока; -1, если подстрока не обнаружена	(2)
s.**rindex**(sub [, start [, end]])	Аналогично rfind(), но исключение ValueError, если подстрока не обнаружена	(2)
s.**rstrip**([chars])	Копия s с удаленными конечными символами chars (по умолчанию — пробелы)	
s.**split**([separator[, maxsplit]])	Возвращает список слов в s, используя separator как разделитель слов	(10)
s.**rsplit**([separator[, maxsplit]])	Аналогично split, но разбиение с конца строки	(10)
s.splitlines([keepends])	Возвращает список строчек в s, прерываясь на границах строчек	(11)
s.**startswith**(prefix [, start[, end]])	Возвращает True, если s начинается с префикса, в противном случае False. Отрицательные позиции могут быть использованы для начала и конца	(2)
s.**strip**([chars])	Копия s с удаленными ведущими и конечными символами chars (пробелы по умолчанию)	
s.**swapcase**()	Копия s с заменой регистра	
s.**title**()	Копия s с заменой стиля на titlecased	
s.**translate**(table [, deletechars])	Копия s, переведенная с использованием таблицы table	(12)
s.**upper**()	Копия s с заменой регистра на верхний	
s.**zfill**(width)	Возвращает числовую строку, заполненную слева нулями в области длиной width	

Примечания:

1. Заполнение с использованием пробела или данного символа.

2. Если дополнительный аргумент start задан, то обрабатывается подстрока s[start:]. Если заданы дополнительные аргументы start и end, то обрабатывается подстрока s[start:end].
3. Кодирование по умолчанию — sys.getdefaultencoding(), может быть изменено через sys.setdefaultencoding(). Дополнительный аргумент errors может быть задан для другой схемы обработки ошибки. Встроенный обработчик ошибок **'strict'** для ошибки кодирования создает исключительную ситуацию ValueError. Другие возможные значения — **'ignore'** и **'replace'**. Смотри модуль codecs.
4. Если дополнительный аргумент tabsize не задан, то принят размер табуляции 8 символов.
5. Возвращает False, если строка s не содержит по крайней мере одного символа.
6. Возвращает False, если строка s не содержит по крайней мере одного помещенного символа.
7. Строка titlecased содержит первые символы верхнего регистра и символы нижнего регистра вслед за ними.
8. Возвращает s, если width менее чем len(s).
9. Если задан дополнительный аргумент maxsplit, то заменяются только первые указанные в maxsplit вхождения.
10. Если sep не определен или его значение None, то разделителем является любой пробел. Если maxsplit дан, то делается не более указанного в maxsplit разделений.
11. Прерывания строки не включены в результирующий список, если keepends задан True.
12. table должен быть строкой длины 256. Все символы, входящие в дополнительный аргумент deletechars, удаляются до выполнения перевода.

Форматирование строк операцией %

formatString **%** *args* --> преобразование в строку

- formatString использует printf-коды формата языка Си: %, c, s, i, d, u, o, x, X, e, E, f, g, G, r.
- Чтобы задать фактическую ширину или точность, используется символ «*».
- Понимаются следующие символы-флаги: -, +, пробел, # и 0.
- %s преобразовывает аргумент любого типа в строку (использует str()).
- *args*. может иметь единственный аргумент или кортеж аргументов.

Пример:

'%s has %03d quote types.' **%** ('Python', 2) == 'Python has 002 quote types.'

- Правый аргумент также может быть словарем (функция vars() очень удобна для использования справа).

Пример:

a = '%**(lang)**s has %**(c)**03d quote types.' % {'c':2, 'lang':'Python'}

Таблица 6

Коды формата

Код	Значение
d	Десятичное целое со знаком
i	Десятичное целое со знаком
o	Восьмеричное целое без знака
u	Десятичное целое без знака
x	Шестнадцатеричное целое без знака (нижний регистр)
X	Шестнадцатеричное целое без знака (верхний регистр)
e	Экспоненциальный формат с плавающей точкой (нижний регистр)
E	Экспоненциальный формат с плавающей точкой (верхний регистр)
f	Десятичное число с плавающей точкой
F	Десятичное число с плавающей точкой
g	Аналог "e", если порядок больше -4 или меньше точности, иначе — "f"
G	Аналог "E", если порядок больше -4 или меньше точности, иначе — "F"
c	Одиночный символ (целое или строка с одним символом)
r	Строка (преобразовывает любой объект «Питона», используя repr())
s	Строка (преобразовывает любой объект «Питона», используя str())
%	Никакой аргумент не преобразовывается, символ "%" передается в результат. (Полная спецификация — %%)

Таблица 7

Флаги преобразования

Флаг	Значение
#	Преобразование величины использует «альтернативную форму»
0	Преобразование будет заполнять нулем
-	Преобразованная величина выравнивается влево (аннулируется «-»)
(пробел)	Пробел должен остаться перед положительным числом (или пустая строка)
+	Символ («+» или «-») будет предшествовать числу (аннулирует флаг «пробел»)

Строковые шаблоны

В версии 2.4 модуль string обеспечивает новый механизм для замены переменных в шаблоне строки. Переменные, которые нужно заменять, начинаются с символа $. Фактические значения обеспечиваются через

словарь методами substitute или safe_substitute (substitute вызывает KeyError, если ключ отсутствует, тогда как safe_substitute игнорирует это):

```
t = string.Template('Hello $name, you won $$$amount') # (note $$ to litteralize $)
t.substitute({'name': 'Eric', 'amount': 100000}) # -> u'Hello Eric, you won $100000'
```

Файловые объекты (тип file)

Создаются встроенной функцией open() [предпочтительно] или ее псевдонимом file(). Объект может быть создан также функциями других модулей. Теперь файловые имена поддерживают уникод для всех функций, получающих или возвращающих файловые имена (open, os.listdir, и т. п.).

Таблица 8

Операторы для файловых объектов

Оператор	Результат
f.**close**()	Закрыть файл f
f.**fileno**()	Запрос fileno (fd) для фала f
f.**flush**()	Освободить внутренний буфер файла f
f.**isatty**()	1, если файл f связан с терминалом, иначе — 0
f.**read**([size])	Прочитать не больше чем заданное число байт из файла f и возвратить как строковый объект. Если размер опущен, то прочитать до конца файла
f.**readline**()	Прочитать одну строчку из файла f. Возвращаемая строчка отслеживает \n и EOF
f.**readlines**()	Прочитать файл до EOF при помощи readline() и возвратить список прочитанных строчек
f.**xreadlines**()	Возвращает последовательность для чтения файла «строчка за строчкой», не загружая весь файл в память. В версии 2.2 лучше использовать for line in f
for line **in** f: do something...	Повторять для строчек файла (использование readline)
f.**seek**(offset[, whence=0])	Установить позицию в файле: • whence == 0 — использовать абсолютный индекс • whence == 1 — смещение относительно текущего положения • whence == 2 — смещение относительно конца файла
f.**tell**()	Вернуть текущую позицию в файле f (байтовое смещение)
f.**write**(str)	Запись строки в файл f
f.**writelines**(list)	Запись списка строк в файл f. Признак EOL не добавляется

Файловые исключения

EOFError — при чтении достигнут конец файла (может подниматься много раз, например, если файловый объект — терминал).

IOError — связанная с файлом неудачная операция ввода/вывода.

A1.5 Множества

В версии 2.4 «Питон» имеет два новых встроенных типа с быстрой реализацией на языке Си: set и frozenset (неизменное множество). Множества являются неупорядоченными совокупностями уникальных элементов (без дубликатов). Элементы множеств должны быть доступны с использованием хеширования (hashable). Множество frozensets может быть элементом других множеств, тогда как set — нет. Все множества допускают циклы по своим элементам (iterable).

Начиная с версии 2.3, классы set и ImmutableSet были доступны в модуле sets. Этот модуль остается в версии 2.4 в библиотеках дополнительно к встроенным типам.

A1.6 Дополнительные типы

Для дополнительной информации к перечню приведенных ниже типов смотрите руководства по языку «Питон».

- *Module* objects
- *Class* objects
- *Class instance* objects
- *Type* objects
- *File* objects
- *Slice* objects
- *Ellipsis* object
- *Null* object
- *XRange* objects
- **Callable** types:
  - User-defined (написаны на Python):
    - ➔ User-defined *Function* objects
    - ➔ User-defined *Method* objects
  - Built-in (написаны на Си):
    - ➔ Built-in *Function* objects
    - ➔ Built-in *Method* object
- **Internal** Types:
  - *Code* objects (байт-компилированный исполняемый код Python: *bytecode*)

– *Frame* objects (исполняемые фреймы)
– *Traceback* objects (обратная трассировка при прерываниях)

A2 Операторы

Таблица 9

Общие операторы

Предложение	Результат
pass	Пустой оператор
del name[, name]*	Освободить имя от объекта. Объект будет косвенно (и автоматически) удален, если на него больше нет ссылок
print[>> fileobject,] [s1 [, s2]* [,]	Писать на sys.stdout или fileobject, если указано. Устанавливает пробелы между значениями аргументов и переводит строку, если утверждение не заканчивается запятой
exec x [in globals [, locals]]	Выполняет **x** в предусмотренном пространстве имен. Устанавливается по умолчанию в текущий namespace. **x** может быть строкой, файловым объектом или функциональным объектом. locals может быть любым словарем, а не только dict
callable(value,... [id=value] , [*args], [**kw])	Вызов функции с параметрами. Параметры могут передаваться по имени или быть опущены, если функция определяет значение по умолчанию. Например, если вызываемое определено как "def callable(p1=1, p2=2)" "callable()" <=> "callable(1, 2)" "callable(10)" <=> "callable(10, 2)" "callable(p2=99)" <=> "callable(1, 99)" *args есть тъюпл позиционных аргументов **kw есть словарь ключевых аргументов

A2.1 Присваивание

Таблица 10

Присваивание

Оператор	Результат	Примечание
a **=** b	Основной вариант – связать объект *b* с именем *a*	(1) (2)
a **+=** b	Эквивалентно a = a + b	(3)
a **-=** b	Эквивалентно a = a - b	(3)
a ***=** b	Эквивалентно a = a * b	(3)
a **/=** b	Эквивалентно a = a / b	(3)
a **//=** b	Эквивалентно a = a // b	(3)

Оператор	Результат	Примечание
a **%=** b	Эквивалентно a = a % b	(3)
a ****=** b	Эквивалентно a = a ** b	(3)
a **&=** b	Эквивалентно a = a & b	(3)
a **|=** b	Эквивалентно a = a | b	(3)
a **^=** b	Эквивалентно a = a ^ b	(3)
a **>>=** b	Эквивалентно a = a >> b	(3)
a **<<=** b	Эквивалентно a = a << b	(3)

Примечания:

1. Можно распаковать tuple, list и string:
```
first, second = l[0:2]  # ==> first=l[0]; second=l[1]
[f, s] = range(2)       # ==> f=0; s=1
c1,c2,c3 = 'abc'        # ==> c1='a'; c2='b'; c3='c'
(a, b), c, (d, e, f) = ['ab', 'c', 'def']
           # ==> a='a'; b='b'; c='c'; d='d'; e='e'; f='f'
x,y = y,x                # Перестановка x и y.
```
2. Возможно множественное присваивание:
```
a = b = c = 0

l1 = l2 = [1, 2, 3]     # l1 и l2 ссылаются на тот же список
                        # (l1 is l2)
```
3. Не точно эквивалентно — ***a*** оценивается только раз. Также, где возможно, действие выполняется на месте — ***a*** модифицируется, а не заменяется новой ссылкой.

Для простых типов присваивание выполняется в семантике копирования значения, для объектов и последовательностей — в семантике копирования указателей.

A2.2 Управление выполнением программы

Таблица 11

УПРАВЛЯЮЩИЕ ОПЕРАТОРЫ

Предложение	Результат

if condition:
  suite
[**elif** condition: suite]*
[**else**:
  suite]
	Условный оператор

while condition:
  suite
[**else**:
  suite]
	Оператор цикла. Часть else выполняется, если цикл не завершен оператором *break*

Предложение	Результат
for element in sequence: suite [**else**: suite]	Оператор цикла по элементам последовательности. Используется функция range() или xrange() для генерирования нужных последовательностей. Часть else выполняется, если цикл не завершен оператором *break*
break	Немедленно выходит из цикла for или while
continue	Немедленно делает следующую итерацию для цикла for или while
return [result]	Выход из функции (или метода) и возврат результата (используйте кортеж, чтобы возвращать более чем одно значение). Если значение не задано, то возвращает None
yield expression	Используется только в пределах тела функции — генератора за пределами try..finally. Возвращает оцененное выражение

Таблица 12

Обработка исключений

Предложение	Результат
assert expr[, message]	Вычисляется expr; если False — то создает исключение AssertionError с сообщением. В версиях до 2.3, если __debug__ = 0, то исключения нет
try: suite1 [**except** [exception [, value]: suite2]+ [**else**: suite3]	Выполняются утверждения suite1. Если происходит исключение, просматриваются элементы в except для идентификации исключения. Если найдено соответствие, то выполняется блок этой секции. Если никакое исключение не происходит, блок suite3 в секции **else** выполняется после блока suite1. Если исключение имеет значение, оно помещается в переменную value. Исключение может быть также кортежем исключений, например, except(KeyError, NameError), val: print val
try: suite1 **finally**: suite2	Выполняются утверждения suite1. Если нет исключений, то выполняется suite2 (даже если бы suite1 был завершен утверждением return, break или continue). Если исключение произошло, то немедленно выполняется suite2 и повторно возбуждается исключение
raise exceptionInstance	Создает экземпляр класса, производный от exceptionClass
raise exceptionClass [, value [, traceback]]	Создает исключение данного класса exceptionClass с дополнительной переменной value. Аргумент traceback определяет объект прослеживания для использования при печати следа исключения
raise	Создает повторно последнее исключение в текущей функции

- Исключение является экземпляром класса исключений (в версиях до 2.0 оно может быть также простой строкой).
- Классы исключений должны быть производными от встроенного класса Exception:

```
class TextException(Exception): pass
try:
```

```
    if bad:
        raise TextException()
except Exception:
    print 'Oops'
    # Это будет напечатано, так как TextException
    # является подклассом Exception
```

- Когда печатается сообщение об ошибке для необрабатываемого исключения, печатается имя класса, затем двоеточие и пробел, и, наконец, экземпляр, преобразованный в строку с использованием встроенной функции str().
- Все встроенные исключения происходят от класса StandardError, сам он — производный от `Exception`.

A2.3 Пространство имен

Импортируемые файлы модулей должны быть расположены в директории, указанном в путях «Питона» (sys.path). В версии 2.3 они могут находиться в **zip**-файле (например, sys.path.insert(0, "theZipFile.zip")).

Пакеты (начиная с версии 1.5) являются пространством имен в директории, включающем модули и специальный модуль инициализации __init__.py (возможно, пустой). Пакеты/директории могут быть вложенными. Адресация к символу модуля производится через `[package.[package...].module.symbol.`
В Mac и Windows имена файлов модулей должны соответствовать их использованию в утверждении `import`.

Таблица 13

Определение имен

Предложение	Результат
import module1 [as name1] [, module2]*	Импорт модулей: если задано name1, то module1 переименовывается. Модули должны ссылаться через квалификаторы на пакеты, например: import sys; print sys.argv import package1.subpackage.module package1.subpackage.module.foo()
from module **import** name1 [**as** othername1][, name2]*	Импорт имени из модуля в текущее пространство имен. Вы можете поместить список имен в круглые скобки. from sys import argv; print argv from package1 import module; module.foo() from package1.module import foo; foo()
from module **import** *	Импорт всех имен из модуля в текущее пространство имен за исключением начинающихся с символа «_». Использовать с осторожностью из-за возможного столкновения имен. from sys import *; print argv

Предложение	Результат
	from package.module import *; print x Законен только на верхнем уровне модуля. Если модуль определяет __all__ атрибут, будут импортированы только указанные в __all__ имена. **from** *package* **import** * импортирует только символы, определенные в файле __init__.py пакета, а не в модулях пакета!
global name1 [, name2]	Имена — из глобальной области (обычно для модуля), а не из локального пространства имен (обычно в теле функции). Например, в функции без утверждения global первое появление имени «x» при попытке читать приводит к ошибке NameError; попытка писать создает локальную переменную функции. Если объект «x» не определен в функции, но есть в модуле, то чтение из «x» приводит к получению значения из модуля, а запись в «x» — к созданию локального «x», но «x[0]=3» начинает поиск «x» в глобальных объектах, если нет локального «x». Будьте осторожны с мутируемыми объектами. Поэкспериментируйте!

A2.3.1. Определение функций

Создает объект-функцию и связывает его с именем `func_id`.

```
def func_id ([param_list]):
    suite
```

```
param_list ::= [id [, id]*]
id ::= value | id = value | *id | **id
```

Аргументы передаются значением. Таким образом, только аргументы, представляющие мутируемые объекты, могут быть модифицированы (inout параметры) в теле функции. Используйте кортежи, чтобы возвращать более чем одно значение.

Пример:

```
def test(p1, p2=5+3, *args, **kwargs):
```

- Параметры со знаком «**=**» задают значения по умолчанию (*default value)* (вычисляются при определении функции).
- Если в списке параметров имеется «**args*», то остальные аргументы назначены кортежем неключевых слов, переданных в функцию.
- Если в списке имеется «***kwargs*», то kwargs является словарем остальных дополнительных аргументов, переданных как ключевые слова.
- *args* и *kwargs* — общеупотребительные имена, но могут использоваться также и другие имена.

A2.3.2. Определение классов

```
class className [(super_class1[, super_class2]…)]:
  suite
```

Создает объект класса и назначает ему имя className. `suite` (блок) может содержать определения локальных методов класса и присваивания значений атрибутам класса.

Примеры:

```
class MyClass (class1, class2): ...
```

Создает объект класса, наследующий как class1, так и class2. Назначает новый объект класса имени MyClass.

```
class MyClass: ...
```

Создает базовый объект класса (ничего не наследует). Назначает новый объект класса имени MyClass.

```
class MyClass (object): ...
```

Создает новый стиль/тип класса (наследование от object делает класс новым стилем класса). Назначает новый объект класса имени MyClass.

- Первый аргумент в методе экземпляра класса — всегда целевой объект по соглашению с именем **`self`**.
- Специальный метод __init__() вызывается при создании экземпляра объекта класса.
- Специальный метод __del__() вызывается, когда нет больше ссылок на объект (сборка мусора).
- Создание экземпляра объекта класса производится вызовом имени класса с аргументами ((instance=apply(aClassObject, арг....) создает экземпляр!)
- В версиях до 2.2 не было возможности создания подклассов потомков встроенных классов подобно list, dict. Начиная с версии 2.2, можно создавать такие подклассы непосредственно (см. Types/Classes).

Пример:

```
class c (c_parent):
    def __init__(self, name):
        self.name = name
    def print_name(self):
        print "I'm", self.name
    def call_parent(self):
        c_parent.print_name(self)


instance = c('tom')
print instance.name
'tom'
```

```
instance.print_name()
"I'm tom"
```

Вызов метода родительского суперкласса производится по имени класса при явной непосредственной передаче параметра "**self**" (см. «call_parent» в примере).

Многие другие специальные методы доступны для осуществления арифметических операторов, последовательностей, индексирования и т. п.

Унифицированные типы/классы

Базовые типы `int, float, str, list, tuple, dict` и `file`, начиная с версии 2.2, ведут себя подобно классам, производным от базового класса **object**, и могут образовывать подклассы:

```
x = int(2)   # встроенная конверторная функция — конструктор базового типа
y = 3        # <=> int(3)  (литералы — экземпляры базового типа)
print type(x), type(y) # int, int

assert isinstance(x, int) # заменяет isinstance(x,types.IntType)
assert issubclass(int, object) # базовый тип, производный от базового
                               # класса 'object'
s = "hello" # <=> str("hello")
assert isinstance(s, str)
f = 2.3  # <=> float(2.3)
class MyInt(int): pass    # подкласс базового типа
x,y = MyInt(1), MyInt("2")
print x, y, x + y # => 1,2,3
class MyList(list): pass
l = MyList("hello")
print l # ['h', 'e', 'l', 'l', 'o']
```

Строки документации

Модули, классы и функции могут содержать встроенные строки документации как первое утверждение в блоке. Документация может быть извлечена с использованием атрибута «__doc__» из модуля, класса или функции.

```
class C:
    "A description of C"
        def __init__(self):
            "A description of the constructor"
            # etc.

c.__doc__ == "A description of C".
c.__init__.__doc__ == "A description of the constructor"
```

Итераторы

- Итератор перечисляет элементы набора. Это объект с единственным методом `next()`, возвращающий следующий элемент или вызывающий StopIteration.
- Вы получаете итератор для объекта через новую встроенную функцию **iter(obj),** которая вызывает `obj.__class__.__iter__()`.
- Набор может быть собственным итератором, реализующим как __iter__(), так и next().
- Встроенные наборы (списки, кортежи, строки, словари) реализуют __iter__(); словари перечисляют их ключи; файлы перечисляют их строки.
- Вы можете построить список или кортеж из итератора, например, `list(anIterator)`
- «Питон» использует итераторы для организации циклов:
  - ➔ **for** elt **in** *collection*:
  - ➔ **if** elt **in** *collection*:
  - ➔ при присваивании тъюплов: x,y,z= *collection*

Генераторы

- Генератор является функцией, которая сохраняет свое состояние между двумя вызовами и возвращает новое значение при каждом вызове. Значения возвращают поочередно, с использованием ключевого слова `yield`. Генератор вызывает исключение StopIteration(), чтобы уведомить о конце последовательности.
- Типичное использование — продуцирование IDs, имен или серийных номеров.
- Для того чтобы использовать генератор, вызовите функцию генератора (чтобы получить объект генератора), затем вызовите generator.next(), чтобы получить следующую величину, пока не произойдет исключение StopIteration.

Версия 2.4 вводит выражения генератора аналогично понятию списка, за исключением того, что элементы возвращаются поодиночке, пригодные для обработки длинных последовательностей. Код генератора должен заключаться в круглые скобки, например:

```
linkGenerator = (link for link in get_all_links() if not
           link.followed)
  for link in linkGenerator:
    ...process link...
```

Пример использования генератора:

```
def genID(initialValue=0):
  v = initialValue
  while v < initialValue + 1000:
    yield "ID_%05d" % v
    v += 1
  return    # or: raise StopIteration()

generator = genID() # Create a generator
```

```
for i in range(10): # Generates 10 values
  print generator.next()
```

Доступ к дескрипторам/атрибутам

Дескрипторы являются объектами, осуществляющими по крайней мере первый из трех методов протокола:

- `__get__(self, obj, type=None) --> значение`
- `__set__(self, obj, value)`
- `__delete__(self, obj)`

«Питон» прозрачно использует дескрипторы для описания и доступа к атрибутам и методам классов нового стиля (производных от *object*). Встроенные дескрипторы позволяют определять:

- Статические методы: использование staticmethod(f) делает метод f(x) статическим (несвязанным).
- Методы класса: подобны статическим, но требуют имя класса как 1-й аргумент. Использование f = classmethod(f) делает метод f(theClass, x) методом класса.
- Свойства: *property* является экземпляром нового встроенного типа, который осуществляет протокол дескриптора для атрибутов. Использование

  ```
  propertyName = property(getter=None, setter=None, deleter=None, description=None)
  ```

  определяет свойство внутри или за пределами класса для доступа к нему через propertyName или obj.propertyName
- Слоты. Классы нового стиля могут определить атрибут класса __slots__, чтобы ограничивать список назначаемых имен, чтобы избежать опечаток (которые обычно не обнаруживаются «Питоном» и приводят к созданию новых атрибутов), например, __slots__ = ('x', 'y'): Согласно последним дискуссиям, реальная цель слотов кажется все еще неясной (оптимизация?), и их использование должно быть осторожным.

Декораторы для функций и методов

- Декоратор D обозначается @D в строке, предшествующей функции/методу:

  ```
  @D
  def f(): ...
  ```
  эквивалентно:
  ```
  def f(): ...
  f = D(f)
  ```
- Несколько декораторов могут использоваться каскадом:

  ```
  @A @B @C
  def f(): ...
  ```
  эквивалентно:
  ```
  f = A(B(C(f)))
  ```
- Декоратор — просто функция, берущая декорируемую функцию и возвращающая ту же функцию или тот же вызываемый объект.
- Функции декоратора могут получать аргументы:

  ```
  @A @B @C(args)
  ```

становится:

```
def f(): ...
_deco = C(args)
f = A(B(_deco(f)))
```

- Декораторы @staticmethod и @classmethod заменяют более изящно эквивалентные декларации f = staticmethod(f) и f = classmethod(f).

Анонимные функции

```
lambda [param_list]: returnedExpr
```

Создает анонимную функцию; returnedExpr должно быть записано в одну строку. Используется по большей части для `filter()`, `map()`, `reduce()` и графического интерфейса пользователя.

Создание списка

```
result = [expression for item1 in sequence1 [if condition1]
             [for item2 in sequence2 ... for itemN in sequenceN]
         ]
```

эквивалентно

```
result = []
for item1 in sequence1:
    for item2 in sequence2:
        ...
        for itemN in sequenceN:
            if (condition1) and further conditions:
               result.append(expression)
```

A2.3.3. Встроенные функции

Таблица 14

Встроенные функции

Функция	Результат
__import__(name[, globals[,locals[,from list]]])	Импорт модуля внутрь данного контекста
abs(x)	Возвращает абсолютную величину числа x
apply(f, args[, keywords])	Вызов функции/метода f с аргументами args и дополнительными ключевыми параметрами
buffer(object[, offset[, size]])	Возвращает буфер с вырезкой объекта, который должен поддержать буферный интерфейс вызова (строка, массив, буфер)
callable(x)	True, если x можно вызывать, иначе — False
chr(i)	Односимвольная строка с ASCII кодом = i

Функция	Результат
classmethod(function)	Возвращает метод класса для функции. Метод класса получает класс как подразумевающийся первый аргумент, подобно методам экземпляра, получающим экземпляр. Для тогоы чтобы объявлять метод класса, используйте эту идиому: class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f) Затем вызовите метод для класса **c.f()** или для экземпляра **c().f().** Экземпляр игнорируется, а воспринимается его класс. Если метод класса вызван для производного класса, производный класс передается как неявный первый аргумент. Начиная с версии 2.4, можно, кроме того, использовать нотацию декоратора: class C: @classmethod def f(cls, arg1, arg2, ...): ...
cmp(x,y)	Возвращает -1, 0, 1, если x <, ==, > y соответственно
coerce(x,y)	Возвращает кортеж двух числовых аргументов, преобразованный в общий тип
compile(string, filename, kind[, flags[, dont_inherit]])	Компиляция строки в кодовый объект. filename используется в сообщении об ошибке, может быть любой строкой. Обычно это файл, из которого код был прочитан, или, например, '<string>', если чтение не из файла. kind может быть 'eval', если строка — единственный оператор, или 'single', который печатает результат выражения, если оно не None, или 'exec'. Новые аргументы flags и dont_inherit зарезервированы для будущего
complex(real[, image])	Создает комплексное число
delattr(obj, name)	Удаляет атрибут с именем name у объекта obj <=> del obj.name
dict([mapping-or-sequence])	Конструктор словаря
dir([object])	Без параметров возвращает список имен текущей локальной таблицы символов. С модулем, классом или объектом класса возвращает список имен в его словаре
divmod(a,b)	Возвращает tuple (a/b, a%b)
enumerate(iterable)	Итератор, возвращает пары (index, value) List(enumerate('Py')) -> [(0, 'P'), (1, 'y')]

Продолжение табл. 14

Функция	Результат
eval(s[, globals[, locals]])	Оценивает строку s, представляющую единственное выражение «Питона», в контексте (дополнительно) глобальных и локальных переменных. s не должна быть пустой или содержать newlines. s может также быть кодовым объектом. locals может быть любого словарного типа, а не только регулярным dict «Питона». *Пример:* x = 1; assert eval('x + 1') == 2 (Для выполнения утверждения, а не единственного выражения, используется утверждение exec или функция execfile)
execfile(file[, globals[, locals]])	Выполняет файл, не создавая новый модуль, в отличие от import. locals может быть любым словарным типом
file(filename[, mode[, bufsize]])	Открывает файл и возвращает новый файловый объект. Псевдоним для open
filter(function, sequence)	Создает список из тех элементов последовательности, для которых функция равна true. Функция получает один параметр
float(x)	Преобразовывает число или строку в число с плавающей точкой
getattr(object, name[, default]))	Запрос атрибута name у объекта, getattr(x, 'f') <=> x.f). Если атрибут не обнаружен, то поднимает AttributeError или возвращает значение по умолчанию, если оно определено
globals()	Возвращает словарь из текущих глобальных переменных
hasattr(object, name)	Возвращается true, если объект имеет атрибут name
hash(object)	Возвращает hash для объекта (если он имеет его)
help([object])	Вызов встроенной системы помощи. Если без аргументов, то вызывается диалоговая помощь; если указан объект-строка (имя модуля, функции, класса, метода, ключевого слова или тема документации), то страница помощи печатается на консоли; в противном случае генерируется страница помощи на объекте
hex(x)	Преобразовывает число x в шестнадцатеричную строку
id(object)	Возвращает целое — уникальный идентификатор для объекта

Продолжение табл. 14

Функция	Результат
input([prompt])	Печатает подсказку, если задана. Читает ввод и оценивает его. Использует редактирование строки и историю. Использует строчки, если модуль readline доступен
int(x[, base])	Преобразовывает число или строку в простое целое. Дополнительный параметр base определяет основание системы счисления
intern(aString)	Выводит aString на экран и возвращает ту же строку
isinstance(obj, classInfo)	Возвращается true, если obj — экземпляр класса classInfo или объект типа classInfo (classInfo может также быть кортежем классов или типов). Если issubclass(A,B), тогда isinstance(x,A) => isinstance(x,B)
issubclass(class1, class2)	Возвращается true, если class1 производный от class2 (или если class1 есть class2)
iter(obj[,sentinel])	Возвращает итератор для obj. Если sentinel отсутствует, obj должно быть коллекцией, реализующей метод либо __iter__(), либо __getitem__(). Если sentinel задан, obj будет вызван без параметров; если возвращаемая величина равняется sentinel, StopIteration будет поднят, в противном случае будет возвращено значение.
len(obj)	Возвращает длину (количество пунктов) объекта (последовательность, словарь или экземпляр класса, реализующего метод __len__)
list([seq])	Создает пустой список или список с теми же элементами. seq может быть последовательностью, контейнером, который поддерживает итерации, или объектом итератора. Если seq — список, то возвращает его копию
locals()	Возвращает словарь текущих локальных переменных
long(x[,])	Преобразовывает число или строку в длинное целое. Дополнительный параметр base определяет основание системы счисления. long('12',8) ==> 10L
map(function, list, ...)	Применяет функцию к каждому элементу списка и возвращает список результатов. Если дополнительные аргументы заданы, функция должна получить их при каждом вызове

Продолжение табл. 14

Функция	Результат
max(seq[, args...])	С единственным аргументом seq возвращает самый большой элемент непустой последовательности (например, строка, кортеж или список). Более чем с одним аргументом возвращает самый большой из аргументов
min(seq[, args...])	С единственным аргументом seq возвращает самый маленький элемент непустой последовательности (например, строка, кортеж или список). Более чем с одним аргументом возвращает самый маленький из аргументов
oct(x)	Преобразовывает число в восьмеричную строку
open(filename [, mode='r', [bufsize]])	Возвращает новый файловый объект. См. также псевдоним file(). Использование codecs.open() позволяет открывать закодированный файл и обеспечивать прозрачное кодирование/декодирование. filename является именем файла, mode указывает, как файл должен быть открыт: 'r' — для чтения; 'w' — для записи (обновляет файл); 'a' — для дозаписи; '+' — (дополнительно в любом из предшествующих режимов) открыть файл для обновления (заметим, что 'w+' обновляет файл); 'b' — (дополнительно в любом из предшествующих режимов) открывать файл в двоичном режиме; 'U' (or 'rU') — открыть файл для чтения в универсальном режиме концов строк: все варианты EOL (CR, LF, CR+LF) будут переведены в одиночный LF ('\n); bufsize равен 0 при отсутствии буферизации, 1 для построчной буферизации, отрицательному или опущенному — для системного умолчания, >1 — для буфера заданного размера
ord(c)	Целое ASCII код символа (строка длиной 1). Работает с символами уникода
pow(x, y [, z])	x в степени y [по модулю z]. См. операцию ** (табл. 1)
range(start [,end [, step]])	Список из целых от >= start и < end с шагом step. Если задан только start, то список от 0 до start-1. Шаг по умолчанию равен 1

Функция	Результат
raw_input([prompt])	Печатает подсказку, если она задана, и читает строку из стандартного ввода (нет отслеживания \n). См. также input()
reduce(f, list [, init])	Применяет двоичную функцию f к элементам списка с тем, чтобы сводить список к единственному значению. Если init задано, то оно «добавлено» в список первым. ll = range(5) reduce(lambda x,y: y+x, ll, 10)
reload(module)	Перезагружает и инициализирует уже импортированный модуль. Полезный в диалоговом режиме. Если модуль был синтаксически правильным, но имел ошибку в инициализации, нужно импортировать его еще раз перед вызовом reload()
repr(object)	Возвращает строку, содержащую выводимое и, если возможно, оцененное представление объекта. Эквивалент операции `object`. В классе переопределяется методом (__repr__). См. также str()
round(x, n=0)	Значение с плавающей точкой x, округленное до n цифр после знака десятичной дроби
setattr(object, name, value)	Это аналог getattr(). setattr(o, 'foobar', 3) <=> o.foobar = 3. Создает атрибут, если он не существует!
slice([start,] stop[, step])	Возвращает объект-вырезку, представляющий диапазон

staticmethod(function)	Возвращает статический метод для функции. Статический метод не получает подразумевающийся первый аргумент. Для того чтобы объявлять статический метод, используйте идиому:
	  class C:
	    def f(arg1, arg2, ...): ...
	    f = staticmethod(f)
	Затем вызовите его для класса C.f() или экземпляра C().f(). Экземпляр игнорируется, учитывается только класс. Вы можете, кроме того, использовать нотацию декоратора:
	  class C:
	    @staticmethod
	    def f(cls, arg1, arg2, ...): ...

str(object)	Возвращает строковое представление объекта. В классе переопределяется методом __str__. См. также repr()

Окончание табл. 14

Функция	Результат
sum(iterable[, start=0])	Возвращает сумму последовательности чисел (не строки), добавленную к значению параметра start

super(type[, object-or-type])	Возвращает superclass для типа. Если второй аргумент опущен, то возвращаемый суперобъект несвязанный. Если второй аргумент — объект, то значение isinstance(obj, type) должно быть истинным. Если второй аргумент является типом, то значение issubclass(type2, type) должно быть истинным. Типичное использование:
	class C(B):
	  def meth(self, arg):
	    super(C, self).meth(arg)

tuple([seq])	Создает пустой кортеж или кортеж с элементами из seq. seq может быть последовательностью, контейнером, которые поддерживают итерацию или объект итератора. Если seq — кортеж, то возвращается сам кортеж (не копия)
type(obj)	Возвращает тип объекта. *Пример:* import types; if type(x) == types.StringType: print 'It is a string'. Лучше, чем использовать: if isinstance(x, types.StringType)...
unichr(code)	Возвращает строку уникода из одного символа с данным кодом
unicode(string[, encoding[,error]]])	Создает строку уникода из 8-битовой строки, используя заданное имя кодировки и обработку ошибки ('strict', 'ignore' или 'replace'). Для объектов, которые обеспечивают метод __unicode__(), вызывает этот метод без аргументов
vars([object])	Без аргументов возвращает словарь текущих локальных символов. Для модуля, класса или объекта класса возвращает словарь символов объекта. Используется с оператором форматирования строки «%»
xrange(start [, end [, step]])	Подобно range(), но не хранит весь список сразу. Хорошо использовать в циклах "for", когда нужен большой диапазон при небольшой памяти
zip(seq1[, seq2,...])	Возвращает список кортежей, где каждый кортеж содержит n-й элемент каждой последовательности. В версиях до 2.4 возвращает пустой список, если вызван без аргументов (прежде создавал TypeError)