Технологии программирования. Компонентный подход

В. В. Кулямин

Лекция 15. Развитие компонентных технологий

Аннотация

Рассказывается о некоторых компонентных средах и технологиях, обрисовывающих направления дальнейшего развития стандартных платформ разработки Web-приложений. Также рассматриваются Web-службы, представляющие собой компонентную технологию другого уровня.

Ключевые слова

Struts, JSF, объектно-реляционный преобразователь, Hibernate, JDO, аспектно-ориентированное программирование, Spring, Web-службы, SOA, WSDL, SOAP, UDDI.

Текст лекции

Программисты, которые долгое время работают с технологиями разработки Web-приложений, представленными в последних двух лекциях, отмечают ряд неудобств, связанных с разработкой отдельных компонентов, построением приложения в целом и настройкой отдельных аспектов его работы. В данной лекции рассказывается о развитии компонентных технологий разработки Web-приложений, нацеленном на повышение их гибкости, удобства их создания и поддержки, а также на снижение трудоемкости внесения изменений в приложения такого рода.

В ряде аспектов разработка отдельных компонентов в рамках .NET несколько проще, тем разработка компонентов с той же функциональностью в рамках J2EE версии 1.4. В то же время разработка приложений в целом в рамках J2EE проще для начинающих разработчиков, поскольку имеющаяся по этой платформе документация четче определяет общую структуру приложений и распределение ответственности между разными типами компонентов в нем.

Большим достоинством J2EE является прозрачность и предсказуемость ее развития, поскольку все его шаги открыты в рамках четко определенного процесса компании Sun для внесения изменений в спецификации платформы и на каждом из этих шагов учитываются интересы множества участников. Развитие платформы J2EE определяется большим количеством открытых проектов отдельных разработчиков и организаций, предлагающих свои решения по построению сред функционирования Web-приложений (Web application frameworks).

Развитие же платформы .NET находится целиком в руках компании Microsoft и пока не является прозрачным для тех, кто не работает в ней или в одной из близких к ней компаний-партнеров. На основании выступлений отдельных представителей компании можно делать выводы, касающиеся лишь общих планов развития платформы, без каких-либо технических деталей. Поэтому в данной лекции рассматриваются, в основном, направления развития технологий J2EE.

Развитие технологий J2EE

Ряд разработчиков выделяет следующие проблемы удобства разработки и поддержки приложений J2EE версии 1.4.

- Громоздкость разработки компонентов EJB и неудобство их использования для описания структуры предметной области.
  Для разработки простейшего такого компонента необходимо определить два интерфейса, класс компонента и написать дескриптор развертывания.
  Полученные классы и интерфейсы достаточно сильно отличаются от обычных классов Java, с помощью которых разработчики описывали бы предметную область в рамках обычного

приложения на платформе J2SE. Поэтому гораздо тяжелее вносить в них изменения, связанные с изменением требований к соответствующим объектам предметной области.

- Отсутствие удобной поддержки для отображения иерархии наследования классов в структуру базы данных приложения.
  Данные класса-предка и класса-наследника могут храниться в одной таблице, в разных и несвязанных таблицах, или общая часть данных может храниться в одной таблице, а специфические данные класс-наследника — в другой. Однако для обеспечения правильной синхронизации данных в каждом из этих случаев достаточно много кода надо написать вручную. Поскольку во многих приложениях объектные модели данных содержат классы, связанные отношением наследования, отсутствие вспомогательных механизмов, автоматически обеспечивающих отображение таких классов на структуру базы данных, приносит много неудобств.
- Невозможность использовать в рамках приложения компоненты EJB, соответствующие данным в базе данных, и временные объекты того же типа, для которых не нужно иметь соответствующих записей в таблицах баз данных.
  Часто такая возможность оказывается удобной при программировании различных методов обработки данных внутри приложений.
- Громоздкость разработки сервлетов для обработки простых (тем более, сложных) запросов пользователя.
  При этом необходимо полностью проанализировать запрос, часто — найти в содержащемся в нем документе HTML поля формы, заполненной пользователем, и указанную им операцию их обработки. Только после этого можно переходить к собственно выполнению этого запроса, что, казалось бы, является основной функцией сервлета. Таким образом, большое количество усилий тратится только на то, чтобы выделить из запроса операцию, которую пользователь хочет произвести, а также ее аргументы.
- Неудобство использования в рамках JSP-страниц специализированных элементов пользовательского интерфейса. Для сравнения: в рамках ASP.NET можно использовать библиотечные и пользовательские элементы управления, которые помещаются на страницу при помощи специального тега, а в параметрах этого тега указываются, в частности, методы для обработки событий, связанных с действиями пользователя.

Для решения этих проблем используются различные библиотеки, инструменты и компонентные среды, созданные в сообществе Java-разработчиков. Некоторые такие библиотеки и техники станут стандартными средствами в рамках платформы J2EE новой версии 5.0 [1].

Jakarta Struts

Среда Jakarta Struts [2,3] создавалась затем, чтобы упростить разработку компонентов Web-приложения, предназначенных для обработки запросов пользователей, и сделать эту обработку более гибкой.

Основные решаемые такими компонентами задачи можно сформулировать следующим образом:

- выделить сам логический запрос и его параметры из HTML документа, содержащегося в HTTP-запросе;
- проверить корректность параметров запроса и сообщить пользователю об обнаруженной некорректности наиболее информативным образом;
- преобразовать корректный логический запрос и его параметры в вызовы соответствующих операций над объектами предметной области;
- передать результаты сделанных вызовов компонентам, ответственным за построение их представления для пользователя.

Как и в рамках базовой платформы J2EE, в Struts основным архитектурным стилем для Web-приложений является образец «данные-представление-обработка». При этом роль представления играют JSP-страницы, а роль обработчиков — сервлеты. Основные отличия Struts от стандартной техники J2EE связаны с большей специализацией сервлетов и некоторой стандартизацией обмена данными между сервлетом, обрабатывающим запросы пользователя, и JSP-страницей, представляющей их результаты.

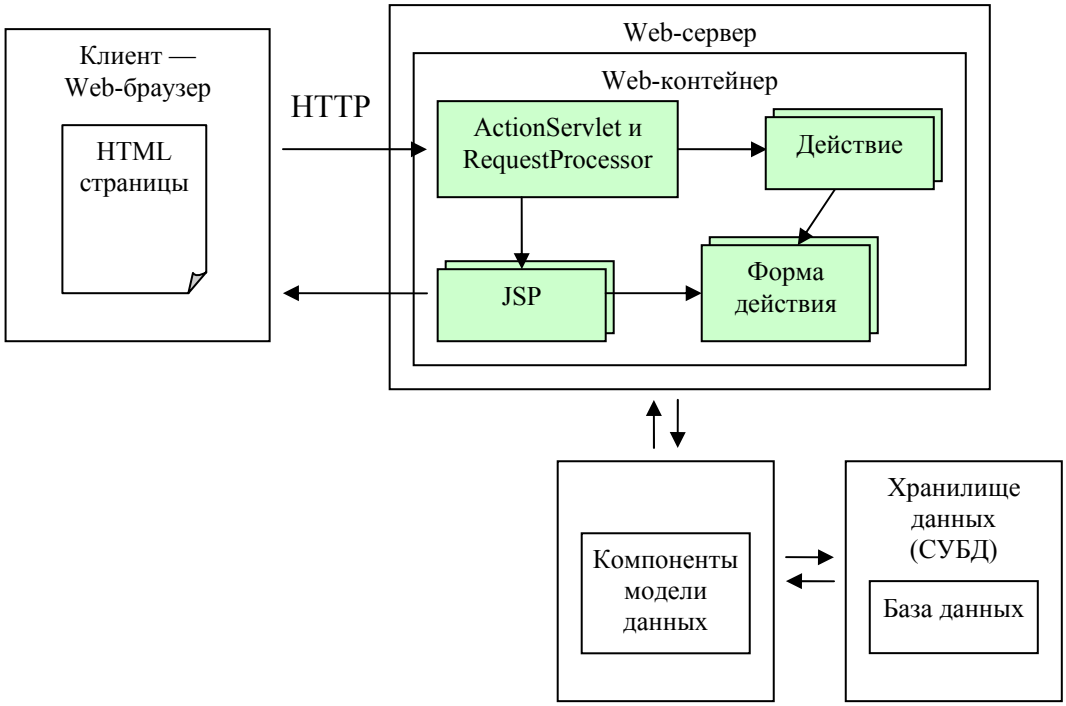

Рисунок 79. Общая схема архитектуры Web-приложений на основе Struts.

В рамках приложения на основе Struts используется ровно один стандартизированный сервлет (ActionServlet), анализирующий запросы пользователя и выделяющий из каждого запроса *действие (action)*, которое пользователь пытается выполнить. Для Интернет-магазина таким действиями, например, могут быть аутентификация (предоставление своего имени и пароля), получение данных о товаре, поиск товара, добавление товара к уже заказанным, изменение заказа, предоставление прав на скидку, выполнение заказа, получение статуса выполнения заказа, отмена заказа и пр. Для каждого действия создается отдельный *класс действия*. Такой класс должен быть наследником класса `org.apache.struts.action.Action` из библиотеки Struts и перегружать метод `ActionForward execute(ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse)` — именно он и вызывается для выполнения этого действия.

Один из параметров метода `execute()` в классе действия имеет тип *формы действия* `org.apache.struts.action.ActionForm`. Для каждого действия определяется свой класс формы действия, наследующий классу `ActionForm`. Объекты этого класса используются для передачи параметров действия — наиболее существенных данных запросов, описывающих данное действие.

В методе `execute()` класса действия обычно строятся или находятся компоненты бизнес-логики приложения, которые реализуют операции, соответствующие данному действию, а затем эти операции выполняются со значениями полей объекта формы действия в качестве аргументов.

Привязка запросов к действиям описывается в дополнительном конфигурационном файле struts-config.xml в формате XML, в теге `action-mappings`. Одно действие описывается с помощью вложенного тега `action`, который имеет следующие атрибуты.

- `path`
  Определяет шаблон URI, обращения к которым будут интерпретироваться как выполнение данного действия.
- `type`
  Определяет имя класса данного действия.
- `name`
  Задает уникальное имя для данного действия.

Привязка действий к определенным для них формам происходит с помощью тегов `from-bean`, вложенных в тег `form-beans`. Каждый тег `form-bean` имеет атрибут `name`, указывающий имя действия для данной формы, и `type`, указывающий имя класса формы действия. Кроме того, такой тег может содержать вложенные теги `form-property`, описывающие свойства формы (в смысле JavaBeans) при помощи таких же атрибутов `name` (имя свойства) и `type` (тип свойства).

Помимо описанного механизма декомпозиции обработки запросов, среда Struts включает библиотеки классов Java, в том числе, классов часто встречающихся действий, и библиотеки пользовательских тегов, предназначенных для более удобного описания размещенных на JSP страницах элементов HTML-форм.

Java Server Faces

Java Server Faces (JSF) [4,5] включают библиотеку элементов управления WebUI `javax.faces` и две библиотеки пользовательских тегов, предназначенных для использования этих элементов управления в рамках серверных страниц Java. С помощью тегов библиотеки `jsf/html` элементы управления размещаются на странице, а с помощью тегов из `jsf/core` описывается обработка событий, связанных с этими элементами, и проверка корректности действий пользователя.

В аспекте построения WebUI на основе серверных страниц Java технология Java Server Faces является развитием подхода Struts (Struts включают решения и для других аспектов разработки приложений), предлагая более богатые библиотеки элементов WebUI и более гибкую модель управления ими. Эта модель включает следующие элементы.

- Возможность различного изображения абстрактного элемента управления (например, элемент управления «выбор одного из многих» может быть изображен как группа радио-кнопок, комбо-бокс или список).
- Возможность изменения визуальных стилей элементов управления.
- Возможность привязки изображаемых элементом управления значений к свойствам компонентов модели данных.
- Возможность привязки элементов управления к методам проверки корректности значений, устанавливаемых в них пользователем.

В дополнение к библиотекам элементов WebUI JSF предлагает определять правила навигации между страницами в конфигурационном файле приложения. Каждое правило относится к некоторому множеству страниц и при выполнении определенного действия или наступлении события предписывает переходить на некоторую страницу. Действия и события связываются с действиями пользователя или логическими результатами их обработки (такими результатами могут быть, например, успешная регистрация заказа в системе, попытка входа пользователя в систему с неправильным паролем и пр.).

Технология Java Server Face версии 1.2 войдет в состав будущей версии 5.0 платформы J2EE [6].

Управление данными приложения. Hibernate

Технологии обеспечения синхронизации внутренних данных приложения и его базы данных развиваются в настоящий момент достаточно активно. Технология EJB предоставляет соответствующие механизмы, но за счет значительного снижения удобства разработки и

модификации компонентов. Обеспечение той же функциональности при более простой внутренней организации кода является основным направлением развития в данной области.

Возможным решением этой задачи являются ***объектно-реляционные преобразователи (object-relation mappers, ORM)***, которые обеспечивают автоматическую синхронизацию между данными приложения в виде наборов связанных объектов и данными, хранящимися в системе управления базами данных (СУБД) в реляционном виде, т.е. в форме записей в нескольких таблицах, ссылающихся друг на друга с помощью внешних ключей.

Одним из наиболее широко применяемых и развитых в технологическом плане объектно-реляционных преобразователей является Hibernate [7-9].

Базовая парадигма, лежащая в основе избранного Hibernate подхода, — это использование объектов обычных классов Java (быть может, оформленных в соответствии с требованиями спецификации JavaBeans — с четко выделенными свойствами) в качестве объектного представления данных приложения. Такой подход даже имеет название-акроним POJO (plain old Java objects, простые старые Java-объекты), призванное показать его отличие от сложных техник построения компонентов, похожих на EJB.

Большое достоинство подобного подхода — возможность использовать один раз созданные наборы классов, представляющих понятия предметной области, в качестве модели данных любых приложений на основе Java, независимо от того, являются ли они распределенными или локальными, требуется ли в них синхронизация с базой данных и сохранение данных объектов или нет.

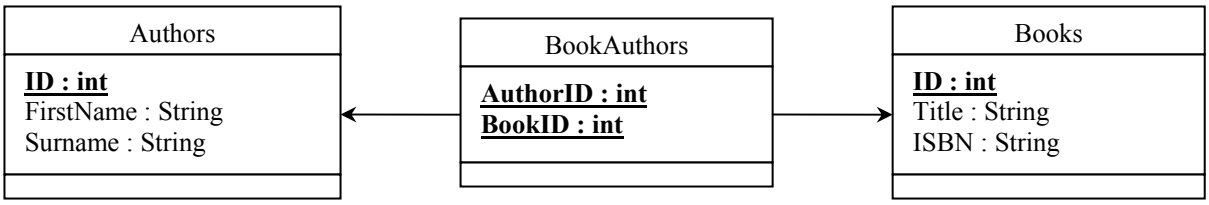

Рисунок 80. Реляционное представление данных о книгах и авторах.

Например, для представления в объектном виде данных о книгах и их авторах, соответствующих показанной на Рис. 80 группе таблиц, могут быть использованы представленные ниже классы.

```
import java.util.Set;
import java.util.HashSet;

public class Author
{
  private int id;

  private String firstName;
  private String surname;

  private Set books = new HashSet();

  public  int  getId ()       { return this.id; }
  private void setId (int id) { this.id = id;   }

  public String getFirstName ()              { return this.firstName;      }
  public void setFirstName (String firstName) { this.firstName = firstName; }

  public String getSurname ()               { return this.surname;    }
  public void   setSurname (String surname) { this.surname = surname; }

  public Set  getBooks ()          { return this.books;  }
  public void setBooks (Set books) { this.books = books; }
}
```

```
public class Book
{
  private int id;

  private String title;
  private String isbn;

  private Set authors = new HashSet();

  public  int  getId ()       { return this.id; }
  private void setId (int id) { this.id = id;   }

  public String getIsbn ()            { return this.isbn; }
  public void   setIsbn (String isbn) { this.isbn = isbn; }

  public String getTitle ()             { return this.title;  }
  public void   setTitle (String title) { this.title = title; }

  public Set  getAuthors ()            { return this.authors;    }
  public void setAuthors (Set authors) { this.authors = authors; }
}
```

Для определения отображения объектов этих классов в записи соответствующих таблиц используются конфигурационные файлы со следующим содержанием. Первый фрагмент представляет собой описание отображения объектов класса `Author` на записи таблицы `Authors`, которое обычно помещается в файл Author.hbm.xml.

```
<hibernate-mapping>
  <class name="Author" table="Authors">
    <id name="id" column="ID">
      <generator class="increment"/>
    </id>
    <property name="firstName" column="FirstName"/>
    <property name="surname"/ column="Surname">
    <set name="books" table="BookAuthors" inverse="true">
      <key column="AuthorID"/>
      <many-to-many column="BookID" class="Book"/>
    </set>
  </class>
</hibernate-mapping>
```

Второй фрагмент представляет собой содержание аналогичного файла Book.hbm.xml, описывающего отображение объектов класса `Book` на записи таблицы `Books`.

```
<hibernate-mapping>
  <class name="Book" table="Books">
    <id name="id" column="ID">
      <generator class="increment"/>
    </id>
    <property name="title" column="Title"/>
    <property name="isbn"/ column="ISBN">
    <set name="authors" table="BookAuthors" inverse="true">
      <key column="BookID"/>
      <many-to-many column="AuthorID" class="Author"/>
    </set>
  </class>
</hibernate-mapping>
```

При использовании объектов указанных классов в рамках приложения на основе Hibernate обеспечивается автоматическая синхронизация данных объектов с данными соответствующих записей, а также автоматическая поддержка описанного отношения типа «многие-ко-многим».

Кроме того, Hibernate поддерживает удобные средства для описания сложных соответствий между объектами и записями таблиц при использовании наследования. Так, легко могут быть поддержаны: отображение данных всех объектов классов-наследников в одну таблицу, отображение объектов разных классов-наследников в разные таблицы, отображение данных полей общего класса-предка в одну таблицу, а данных классов-наследников — в разные таблицы, а также смешанные стратегии подобных отображений.

В приложениях на основе Hibernate объекты одного и того же класса могут быть как хранимыми, т.е. представляющими данные, хранящиеся в базе данных, так и временными, не имеющими соответствующих записей в базе данных. Перевод объекта из одного из этих видов в другой осуществляется при помощи всего лишь одного вызова метода вспомогательного класса среды Hibernate.

С помощью дополнительной службы NHibernate [7] возможности среды Hibernate могут быть использованы и из приложений на базе .NET.

Java Data Objects

Еще более упростить разработку объектно-ориентированных приложений, данные которых могут храниться в базах данных, призвана технология Java Data Objects (JDO) [10,11].

В ее основе тоже лежит использование для работы с хранимыми данными обычных классов на Java, но в качестве хранилища данных может выступать не только реляционная СУБД, но вообще любое хранилище данных, имеющее соответствующий специализированный адаптер (в рамках JDO это должна быть реализация интерфейса `javax.jdo.PersistenceManager`). Основная функция этого адаптера — прозрачная для разработчиков синхронизация хранилища данных и набора хранимых объектов в памяти приложения. Он должен также обеспечивать достаточно высокую производительность приложения, несмотря на наличие нескольких промежуточных слоев между классами самого приложения и хранилищем данных, представляемых ими.

Использование JDO очень похоже на использование Hibernate. Конфигурационные файлы, хранящие информацию о привязке объектов Java классов к записям в определенных таблицах, а также о привязке коллекций ссылок на объекты к ссылкам между записями, также похожи на аналогичные файлы Hibernate. Обычно первые даже несколько проще, поскольку б***о***льшую часть работы по отображению полей объектов в поля записей базы данных берет на себя специализированный адаптер.

В дополнение JDO предоставляет средства для построения запросов с помощью описания свойств объектов, без использования более привычного встроенного SQL.

В целом, подход JDO является обобщением подхода ORM на произвольные хранилища данных, но он требует реализации более сложных специализированных адаптеров для каждого вида таких хранилищ, в то время как один и тот же ORM может использоваться для разных реляционных СУБД, требуя для своей работы только наличия более простого драйвера JDBC.

Enterprise Java Beans 3.0

В рамках следующей, пятой версии платформы J2EE [6] будет использоваться новый набор техник для построения компонентов EJB — стандарт EJB 3.0 [12].

Стандарт EJB версии 3.0 является существенным развитием EJB 2.1 в сторону упрощения разработки и поддержки приложений, использующих технологию EJB. При этом большинство нововведений основывается на тех же идеях, на которых строится работа с хранимыми данными в рамках Hibernate и JDO. Гевин Кинг (Gavin King), создатель первых версий Hibernate, является и одним из разработчиков стандарта EJB 3.0.

В рамках нового стандарта для создания компонента требуется описать только один класс, который не должен наследовать какому-то библиотечному классу или реализовывать какой бы то ни было интерфейс. Объекты этого класса могут быть как хранимыми, так и временными.

Различные описатели свойств компонента и его методов можно оформлять в виде аннотаций, стандартной конструкции для описания метаданных в Java 5. Примерами таких описателей являются: указание вида компонента — компонент данных или сеансовый; отображение свойств класса в поля таблиц и ссылки между ними; отметки нехранимых свойств и полей; отметки специальных методов, вызываемых при переходе между этапами жизненного цикла компонента; транзакционные атрибуты методов и пр. Заметим, что в .NET использование для этого аналогов аннотаций, атрибутов, уже реализовано. Остается и возможность использовать для этих целей XML-дескрипторы, аналогичные используемым в Hibernate конфигурационным файлам. Конфигурационные файлы более удобны для сложных приложений, поскольку сокращают количество мест, в которые нужно вносить модификации при изменениях в структуре базы данных или объектной модели данных приложения. В то же время применение аннотаций позволяет обеспечить более быструю и удобную разработку небольших приложений.

Среда Spring

Среда Spring [9,13-15] представляет собой одну из наиболее технологичных на данный момент сред разработки Web-приложений. В ее рамках получили дальнейшее развитие идеи специализации обработчиков запросов, использованные в Struts. В Spring также используются элементы аспектно-ориентированного подхода к разработке ПО, позволяющие значительно повысить гибкость и удобство модификации построенных на ее основе Web-приложений.

Основная задача, на решение которой нацелена среда Spring, — интеграция разнородных механизмов, используемых при разработке компонентных Web-приложений. В качестве двух основных средств такой интеграции используются идеи *обращения управления (inversion of control)* и *аспектно-ориентированного программирования (aspect-oriented programming, AOP)*.

Обращением управления называют отсутствие обращений из кода компонентов приложения к какому-либо API, предоставляемому средой и ее библиотеками. Вместо этого компоненты приложения реализуют только функции, необходимые для работы в рамках предметной области и решения тех задач, с которыми приложению придется иметь дело. Построение из этих компонентов готового приложения, конфигурация отдельных его элементов и связей между ними — это дело среды, которая сама в определенные моменты обращается к нужным операциям компонентов. Конфигурация компонентов приложения в среде Spring осуществляется при помощи XML-файла springapp-servlet.xml. Этот файл содержит описание набора компонентов, которые настраиваются при помощи указания параметров инициализации соответствующих классов и значений своих свойств в смысле JavaBeans.

Другое название механизма обращения управления — *встраивание зависимостей (dependency injection)* — связано с возможностью указывать в коде приложения только интерфейсы некоторых объектов, а их точный класс, как и способ их получения (создание нового объекта, поиск при помощи службы каталогов, обращение к фабрике объектов и пр.) — описывать только в конфигурационном файле. Такой механизм позволяет разделить использование объектов и их конфигурацию и менять их независимо друг от друга.

Аналогичный механизм предполагается использовать и в рамках EJB 3.0 в следующем виде. При помощи конструкции `@Resource Type object;` некоторый объект может быть объявлен в коде как подлежащий отдельной конфигурации, а в конфигурационном файле для объекта с именем `object` указывается его точный тип и способ инициализации.

Аспектно-ориентированный подход к программированию основыван на выделении *аспектов* — отдельных задач, решаемых приложением — таким образом, чтобы их решение можно было организовать в виде выполнения определенных действий каждый раз, когда выполняется определенный элемент кода в ходе работы программы. При этом действия в рамках данного аспекта не должны зависеть от других аспектов и остальных действий, выполняемых программой. Не все задачи могут быть представлены как аспекты. Поэтому AOP-программа представляет собой композицию из «обычной» программы, описываемой вне рамок AOP, и аспектных действий, выполняемых в определенных точках «обычной» программы. Такие действия называют

указаниями (advice), точки их выполнения в «обычной» программе — *точками вставки (joinpoints)*, а наборы точек вставки, в которых должно выполняться одно и то же действие — *сечениями (pointcuts)*.

Примером указаний могут служить трассировка параметров вызова метода или проверка корректности инициализации полей объекта. В качестве примеров точек вставки можно привести момент перед вызовом определенного метода или сразу после такого вызова, момент после инициализации определенного объекта или перед его уничтожением. Сечения могут описываться условиями типа «перед вызовом в данном объекте метода, чье имя начинается на `"get"`» или «перед уничтожением объекта класса `A` со значением поля `value`, превышающим 0».

Spring дает возможность определять сечения и указания, выполняемые в них, в конфигурационных файлах приложения.

При помощи AOP в Spring реализована поддержка интеграции приложений с хранилищами данных при помощи различных технологий, примерами которых являются JDO и Hibernate. Для того чтобы использовать любую такую технологию в приложении на основе Spring, достаточно иметь специализированный адаптер, который реализует интерфейс, предлагаемый Spring для поддержки синхронизации данных между приложением и хранилищем данных. После указания этого адаптера в конфигурации приложения среда сама позаботится о том, чтобы всякий раз после появления возможных различий между данными приложения и базы данных были вызваны методы этого адаптера, копирующие новые данные в ту или другую сторону.

Похожим образом поддерживается интеграция с различными реализациями служб поддержки транзакций и декларативное управление транзакциями. Методам обычного класса Java в конфигурационном файле (или с помощью аннотаций Java 5) можно приписать определенные транзакционные атрибуты, а также набор типов исключительных ситуаций, вызывающих откат транзакции. Для сравнения — в EJB 2.1 только исключения, чей тип является наследником `java.lang.RuntimeException`, `java.lang.Error` или `javax.ejb.EJBException`, вызывают автоматический откат транзакции. Адаптер конкретной реализации службы транзакций также указывается в конфигурации приложения.

Использование обращения управления позволяет также упростить описание конфигурации сервлетов и *контроллеров* (Spring-аналог действий из Struts) и определение самих контроллеров.

Ajax

Рассказывая о развитии технологий разработки Web-приложений, невозможно обойти вниманием набор техник, известный под названием Ajax [16,17] и используемый для снижения времени реакции Web-интерфейсов на действия пользователя.

Вообще говоря, Web-технологии не очень хорошо приспособлены для построения пользовательского интерфейса интерактивных приложений, т.е. таких, где пользователь достаточно часто выполняет какие-то действия, на которые приложение должно реагировать. Они изначально разрабатывались для предоставления доступа к статической информации, которая меняется редко и представлена в виде набора HTML-страниц. Обычно при обмене данными между Web-клиентом и Web-сервером клиент изредка посылает серверу простые и небольшие по объему запросы, а тот в ответ может присылать достаточно объемные документы.

В интерактивных приложениях обмен данными между интерфейсными элементами приложения и обработчиками запросов несколько иной. Обработчик достаточно часто получает запросы и небольшие наборы их параметров, а изменения, которые происходят в интерфейсе после получения ответа на запрос, обычно тоже невелики. Часто нужно изменить содержание лишь части показываемой браузером страницы, в то время как остальные ее элементы представляют более стабильную информацию, являющуюся элементом дизайна сайта или набором пунктов его меню. Для отражения этих изменений не обязательно пересылать с сервера весь HTML-документ, что предполагается в рамках традиционного обмена информацией с помощью Web. Точно так же, если бы корректность вводимых пользователем данных можно было

бы проверить на стороне клиента, обработка некорректного ввода происходила бы гораздо быстрее и не требовала бы вообще никакого обмена данными с сервером.

Ajax пытается решить эти задачи при помощи комбинации кода на JavaScript, выполняющегося в браузере, и передаваемых время от времени между клиентом и сервером специальных XML-сообщений, содержащих только существенную информацию о запросе или изменениях HTML-страницы, которые должны быть показаны. В рамках браузера в отдельном потоке работает ядро Ajax, которое получает сообщения JavaScript-кода о выполнении пользователем определенных действий, выполняет проверку их корректности, преобразует их в посылку соответствующего запроса на сервер, преобразует ответ сервера в новую страницу или же выдает уже имеющийся в специальном кэше ответ. Запросы на сервер и их обработка осуществляются часто асинхронно с действиями пользователя, позволяя заметно снизить ощущаемое время реакции системы на них.

На настоящий момент Ajax еще не является полноценным элементом компонентных технологий. Используемые в его рамках техники имеют некоторые проблемы, связанные с переносимостью между различными платформами, а также не вполне четко разграничивают ответственность между отдельными подсистемами приложения. Однако, в дальнейшем, аналогичные техники наверняка будут включены в стандартные платформы для разработки Web-приложений — J2EE и .NET.

Web-службы

В настоящее время совместно с компонентными технологиями на базе J2EE и .NET широкое распространение получают *Web-службы* или *Web-сервисы (Web services)* [18-22], представляющие собой компонентную технологию другого рода, реализуемую поверх этих платформ. Основными отличительными признаками ***служб (services)*** любого вида служат следующие.

- Служба чаще всего предоставляет некоторую функцию, которая полезна достаточно широкому кругу пользователей.
- Как и другие компоненты, службы имеют четко описанный интерфейс, выполняющий определенный контракт. Именно контракт фиксирует выполняемую службой функцию.
- Контракт службы не должен зависеть от платформ, операционных систем и языков программирования, с помощью которых эта служба может быть реализована. Интерфейс и реализация службы строго отделяются друг от друга.
- Службы вызываются асинхронно. Ждать или нет результатов вызова службы, решает сам клиент, обратившийся к ней.
- Службы совершенно независимы друг от друга, могут быть вызваны в произвольных комбинациях и всякий раз работают в соответствии со своим контрактом.
  Контракт службы не должен зависеть от истории обращений к ней или к другим службам. Но он может зависеть, например, от состояния каких-либо хранилищ данных.
- Обращение к службе возможно с любой платформы, из любого языка программирования, с любой машины, имеющей какую-либо связь с той, на которой размещается реализация службы. Реализации служб должны предоставлять возможность обратиться к ним и динамически обнаружить их в сети, а также предоставлять необходимую дополнительную информацию, с помощью которой можно понять, как с ними работать.

Архитектура приложений, построенных из компонентов, являющихся такими службами, называется ***архитектурой, основанной на службах (service oriented architecture, SOA)***.

Практически единственным широко используемым видом служб являются Web-службы. ***Web-службами*** называют службы, построенные с помощью ряда определенных стандартных протоколов, т.е. использующие протокол SOAP и инфраструктуру Интернет для передачи данных о вызовах операций и их результатах, язык WSDL для описания интерфейсов служб и реестры UDDI для регистрации служб, имеющихся в сети. Существуют многочисленные дополнительные технологии, протоколы и стандарты, относящиеся к другим аспектам работы Web-служб, однако

они пока применяются гораздо реже. Web-службы могут быть реализованы на основе J2EE, .NET или других технологий.

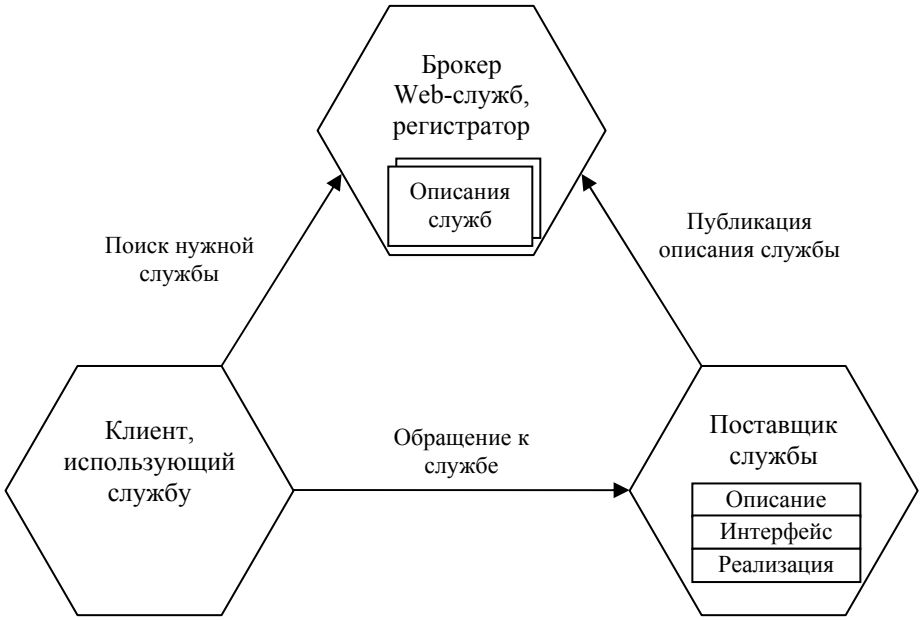

Рисунок 81. Схема архитектуры приложений на основе Web-служб.

Основное назначение Web-служб — предоставление бизнес-услуг организациями отдельным пользователям, а также другим организациями на единой технологической основе. При взаимодействии между различными организациями (business-to-business, B2B) Web-службы делают возможным оказание всех необходимых услуг вообще без вмешательства человека, если, конечно, урегулированы все вопросы, касающиеся стоимости услуг, защищенности данных об оказанных услугах, и доверия организаций друг к другу.

Таким образом, Web-службы представляют собой компонентную технологию построения крупномасштабных распределенных приложений, ориентированных на предоставление бизнес-услуг. В таких приложениях могут участвовать тысячи Web-служб, работающих на совершенно разных платформах, в различных организациях и реализованных с использованием разных технологий. В каждый момент времени такое приложение может не существовать как единое целое: какие-то его компоненты могут не участвовать в текущей работе, другие вообще не работать в связи с отключением соответствующих серверов, третьи — перемещаться с одной машины на другую. Все его компоненты объединяются только едиными стандартами описания интерфейсов и обеспечения взаимодействия между Web-службами.

Схема архитектуры приложений на основе Web-служб, предложенная IBM в конце 1990-х годов, изображена на Рис. 81.

После создания новой службы ее поставщик — организация или частное лицо, которое предоставляет ее, — регистрирует службу у брокера Web-служб (или у нескольких таких брокеров), помещая в его реестр описание службы. Такое описание содержит описание услуги, предоставляемой Web-службой и описание способа доступа к ней — протокол доступа, адрес и порт, к которому надо обращаться.

Клиент, которому понадобилась некоторая услуга, обращается к брокеру Web-служб. Найдя службу, которая, судя по описанию, эту услугу оказывает, он обращается по указанному в ее спецификации адресу и получает описание интерфейса для обращения к этой службе. После этого клиент может обращаться к службе в соответствии с этим интерфейсом.

Использование на всех этапах описанного процесса стандартных описаний в форматах, основанных на XML, позволяет полностью автоматизировать его.

Описание интерфейса Web-служб

Языком описания интерфейса Web-служб служит *WSDL* (читается «виздэл») — *Web Services Description Language, язык описания Web-служб* [23]. Этот язык служит аналогом (и некоторым обобщением) языков описания интерфейсов (IDL), используемых при реализации удаленных вызовов процедур и методов. В настоящее время используется версия WSDL 1.1, но в 2006 году выйдет версия 2.0, в которой достаточно много новых элементов.

Описание интерфейса работы с Web-службой на WSDL состоит из двух частей — абстрактной и конкретной. Абстрактная часть описания содержит определения типов данных, используемых в обращениях к данной службе и определения абстрактных операций, которые можно «вызвать» у службы. Напомним, что все такие «вызовы» являются асинхронными обращениями. Конкретная часть содержит описание привязки операций к определенным адресам, протоколам доступа и портам.

- Типы данных описываются внутри тега `<types>`. Они могут основываться на встроенных XML-типах и использовать XML Schema для описания сложных структур данных.
- С помощью тегов `<message>` описываются типы сообщений, которыми стороны могут обмениваться в ходе работы службы. Для сообщения указывается, является ли оно входящим или исходящим, а его описывается структура в терминах определенных ранее типов данных.
- Далее определяются операции, которые могут включать в себя обмен сообщениями нескольких типов. Для операции указывается используемый в ее рамках шаблон обмена сообщениями. Примерами шаблонов являются: однократное уведомление со стороны службы, запрос со стороны клиента, запрос-ответ. Всего в WSDL 1.1 есть 4 вида шаблонов, в WSDL 2.0 — уже 9 видов.
- Операции группируются в интерфейсы, которые в WSDL 1.1 названы *типами портов (port types)*.
- С помощью элемента `<binding>` определяется привязка интерфейсов к их реализациям. Она задает конкретные форматы сообщений и протоколы их посылки/получения для некоторого интерфейса. Один интерфейс может иметь несколько привязок.
- Элемент `<port>` определяет порт, задающий конкретные адрес и порт некоторой привязки, а также, возможно, транспортный протокол для передачи сообщений на этот адрес.
- Наконец, элемент `<service>` описывает службу целиком, указывая набор портов для доступа к различным ее интерфейсам.

Связь

Связь между Web-службами и их клиентами осуществляется по протоколу *SOAP (Simple Object Access Protocol, простой протокол доступа к объектам)* [24]. Протокол SOAP является протоколом уровня представления по модели OSI, т.е. он определяет формат сообщений, которые пересылаются с помощью некоторого транспортного протокола, в качестве которого обычно используются HTTP, HTTPS, TCP, иногда SMTP.

Формат сообщений SOAP основан на XML. SOAP-сообщение состоит из следующих частей.

- Конверт (envelope) — содержит сообщение целиком.
- Заголовок (header) — содержит значения некоторых дополнительных атрибутов сообщения, используемых при его обработке или переадресации. Заголовок может отсутствовать и используется обычно для передачи информации о координации нескольких сообщений, идентификации сеансов и передачи разного рода сертификатов для защиты информации.

- Тело (body) — основное содержимое сообщения, должно относится к одному из типов сообщений, которыми можно обмениваться с данной службой согласно описанию ее интерфейса. Должно быть в любом сообщении.

Простой пример SOAP-сообщения приведен ниже.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Header>
    <t:Transaction
      xmlns:t="http://company.com/soap-headers/attrs"
      SOAP-ENV:mustUnderstand="1">
      5
    </t:Transaction>
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="http://company.com/web-services/trading">
      <symbol>DEF</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Кроме определения формата сообщений, протокол SOAP определяет процесс их обработки различными посредниками и получателями.

Именование

Роль служб именования и каталогов в приложениях на основе Web-служб играют реестры Web-служб, организованные в соответствии со стандартом UDDI (Universal Description, Discovery and Integration, универсальный стандарт описания, поиска и интеграции) [25].

Существует всего лишь несколько универсальных реестров, регистрирующих любые доступные в Интернет Web-службы. Каждый из них поддерживается одной из крупных компаний, играющих заметную роль в развитии технологий разработки Web-служб. Такие реестры есть у IBM, Microsoft, SAP, NTT. К сожалению, они содержат не очень много записей о работающих Web-службах. Гораздо больше специализированных реестров, предназначенных для использования в рамках одной организации или компанией и ее партнерами.

UDDI описывает структуру реестров Web-служб. Каждая запись в таком реестре является XML-документом. Наиболее важная информация содержится в документах следующих видов.

- `businessEntity`. Такой документ описывает организацию (или лицо), предоставляющую набор Web-служб. В частности, он содержит название (имя), адрес, контакты, профиль, т.е. характеристику области ее (его) деятельности.
- `businessService`. Это список Web-служб, предоставляемых некоторой организацией.
- `bindingTemplate`. Описывает технические аспекты предоставляемых служб, в частности, адреса, к которым нужно обращаться, списки дополнительных описаний (`tModels`).
- `tModel` (technical model). Содержит дополнительную информацию о службе, в частности, предоставляемые ею услуги, условия и ограничения ее использования, предоставляемые гарантии и пр.

Помимо структуры реестра, UDDI определяет интерфейс для работы с ним, позволяющий публиковать или удалять информацию о предоставляемых службах, изменять собственника служб, искать нужные службы по набору характеристик и т.д.

Процессы

Поскольку Web-службы считаются совершенно независимыми от реализации компонентами, а управление процессами и потоками сильно зависит от платформы, в контексте Web-служб они не

рассматриваются. Можно считать, что каждый экземпляр Web-службы работает в своем отдельном процессе, к которому не имеют доступа все остальные экземпляры других Web-служб или той же самой службы.

Синхронизация и целостность

Базовые и общепризнанные стандарты построения Web-служб (WSDL, SOAP и UDDI) не рассматривают вопросы синхронизации работы нескольких Web-служб. В то же время, эти вопросы очень важны при построении одних служб на базе других и разработке приложений из наборов взаимодействующих Web-служб.

Одной из попыток стандартизации протоколов совместной работы Web-служб является технология *WS-Coordination (Web Services Coordination)* [18,26]. Она предлагает набор протоколов, языков и инфраструктуру их использования, совместно позволяющих описывать и осуществлять синхронизацию и координацию нескольких Web-служб, которые работают над одной задачей.

Для обеспечения целостности при совместной работе нескольких служб могут использоваться технологии на основе стандартов *WS-Transactions* и *WS-BusinessActivity* [18,26], построенных на базе WS-Coordination.

Задачи синхронизации могут решаться с помощью средств, помогающих строить приложения на основе композиции Web-служб или при помощи их «оркестровки» (web services orchestration) [18]. Одним из таких подходов является *BPEL (Business Process Execution Language, язык исполнения бизнес-процессов)* [18,27]. Это графический язык, дающий возможность описать достаточно сложные потоки работ, каждая из которых выполняется отдельной службой, и скомпилировать такое описание в реализацию новой Web-службы.

Отказоустойчивость

Возможность обеспечения отказоустойчивости Web-служб заложена в архитектуру приложений на их основе. Ее можно добиться дублированием их реализаций и регистрацией нескольких точек доступа к службам, реализующим один и тот же интерфейс.

Для обеспечения отказоустойчивости при передаче сообщений разрабатывается дополнительный стандарт *WS-Reliability* [28], расширяющий SOAP. Использование WS-Reliability позволяет гарантировать доставку сообщений, используемых в работе Web-служб.

Защита

Наиболее вероятным кандидатом на место широко используемого стандарта защиты информации, передаваемой в сообщениях при работе с Web-службами, является стандарт *WS-Security* [18,29].

Он расширяет SOAP, добавляя в заголовки сообщений этого протокола информацию, с помощью которой можно подтвердить целостность сообщения, подтвердить личность отправителя или затруднить доступ к его содержанию для третьих партий, определив алгоритм шифрования содержимого.

Литература к Лекции 15

[1] Java Platform Enterprise Edition Specifications, version 5. Доступны через http://java.sun.com/j2ee/5.0/index.jsp.
[2] Web-сайт проекта Apache Struts http://struts.apache.org/.
[3] C. Cavaness. Programming Jakarta Struts. O'Reilly, 2002.
[4] Java Server Faces Specification, version 1.2. Доступна на http://java.sun.com/j2ee/javaserverfaces/download.html.
[5] H. Bergsten. JavaServer Faces. O'Reilly, 2004.
[6] Спецификации J2EE 5.0. Доступны через http://java.sun.com/javaee/5/javatech.html.
[7] Web-сайт проекта Hibernate http://www.hibernate.org/.

[8] C. Bauer, G. King. Hibernate in Action. Manning, 2004.
[9] B. A. Tate, J. Gehtland. Better, Faster, Lighter Java. O'Reilly, 2004.
[10] Web-сайт технологии JDO http://jdocentral.com/.
[11] D. Jordan, C. Russell. Java Data Objects. O'Reilly, 2003.
[12] Спецификации Enterprise Java Beans 3.0. Доступны через http://java.sun.com/products/ejb/docs.html.
[13] Web-сайт проекта Spring http://www.springframework.org/.
[14] R. Johnson. Expert One-on-One J2EE Design and Development. Wrox, 2002.
[15] R. Johnson, J. Hoeller, A. Arendsen, T. Risberg, C. Sampaleanu. Professional Java Development with the Spring Framework. Wiley, 2005.
[16] http://developer.mozilla.org/en/docs/Category:AJAX:Articles.
[17] D. Crane, E. Pascarello, D. James. Ajax in Action. Manning, 2005.
[18] G. Alonso, F. Casati, H. Kuno, V. Machiraju. Web Services. Concepts, Architectures and Applications. Springer-Verlag, 2004.
Сайт этой книги http://www.inf.ethz.ch/personal/alonso/WebServicesBook.
[19] Сайт IBM, посвященный Web-службам и SOA. http://www-128.ibm.com/developerworks/webservices/.
[20] Э. Ньюкомер. Веб-сервисы. Для профессионалов. СПб.: Питер, 2003.
[21] Х. Дейтел, П. Дейтел, С. Сантри. Технологии программирования на Java 2. Книга 3: Корпоративные системы, сервлеты, JSP, Web-сервисы. М.: Бином, 2003.
[22] А. Феррара, М. Мак-Дональд. Программирование web-сервисов для .NET. СПб.: Питер-BHV, 2003.
[23] Спецификации WSDL 1.1 http://www.w3.org/TR/wsdl.
[24] Спецификации SOAP 1.2 http://www.w3.org/TR/soap/.
[25] Web-сайт стандарта OASIS UDDI http://www.uddi.org/.
[26] Спецификации WS-Coordination, WS-Transactions и WS-BusinessActivity. Доступны через http://www-128.ibm.com/developerworks/library/specification/ws-tx/.
[27] Спецификации BPEL. Доступны через http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.
[28] Спецификации WS-Reliability. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrm.
[29] Спецификации WS-Security. http://www-128.ibm.com/developerworks/webservices/library/ws-secure/.