JavaServer Pages

JSP-страница

- Текстовый документ, который может возвращать как статическое, так и динамическое содержимое
- Статическое и динамическое содержимое можно чередовать
- Статическое содержимое
  - HTML, XML, Text
- Динамическое содержимое
  - Java-код
  - Отображение свойств JavaBean-ов
  - Вызов бизнес-логики с помощью действий, определяемых программистом

Достоинства JSP

- Разделение содержимого и логики отображения
- Упрощение разработки веб-приложений и отдельных веб-страниц
- Поддержка повторного использования компонентов (JavaBean-ы, действия, определенные программистом)
- Автоматическая установка
  - Автоматическая перекомпиляция при изменении JSP-страницы
- Независимость от платформы

Разделение обработки запроса и представления

Один сервлет

```
Public class   OrderServlet  …{
   public void   doGet (…){
      if( isOrderValid  (req )){
         saveOrder  (req );

         out.  println  (“<html>”);
         out. println  (“<body>”);
                  ……
   private void   isOrderValid  (….){
   ……
   }

   private void   saveOrder  (….){
   ……
   }
}
```

Обработка запроса

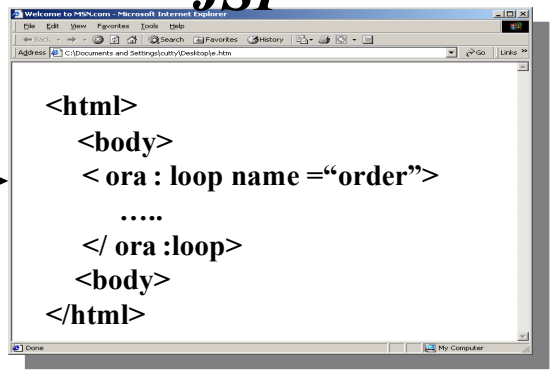
Сервлет

```
Public class   OrderServlet  …{
   public void   doGet (…){
      ……
      if(bean.  isOrderValid  (..)){
         bean.  saveOrder  (….);

         forward(“conf.   jsp ”);
   }
}
```

Представление

JSP



```
<html>
   <body>
    < ora : loop name =“order”>
        …..
    </ ora :loop>
   <body>
</html>
```

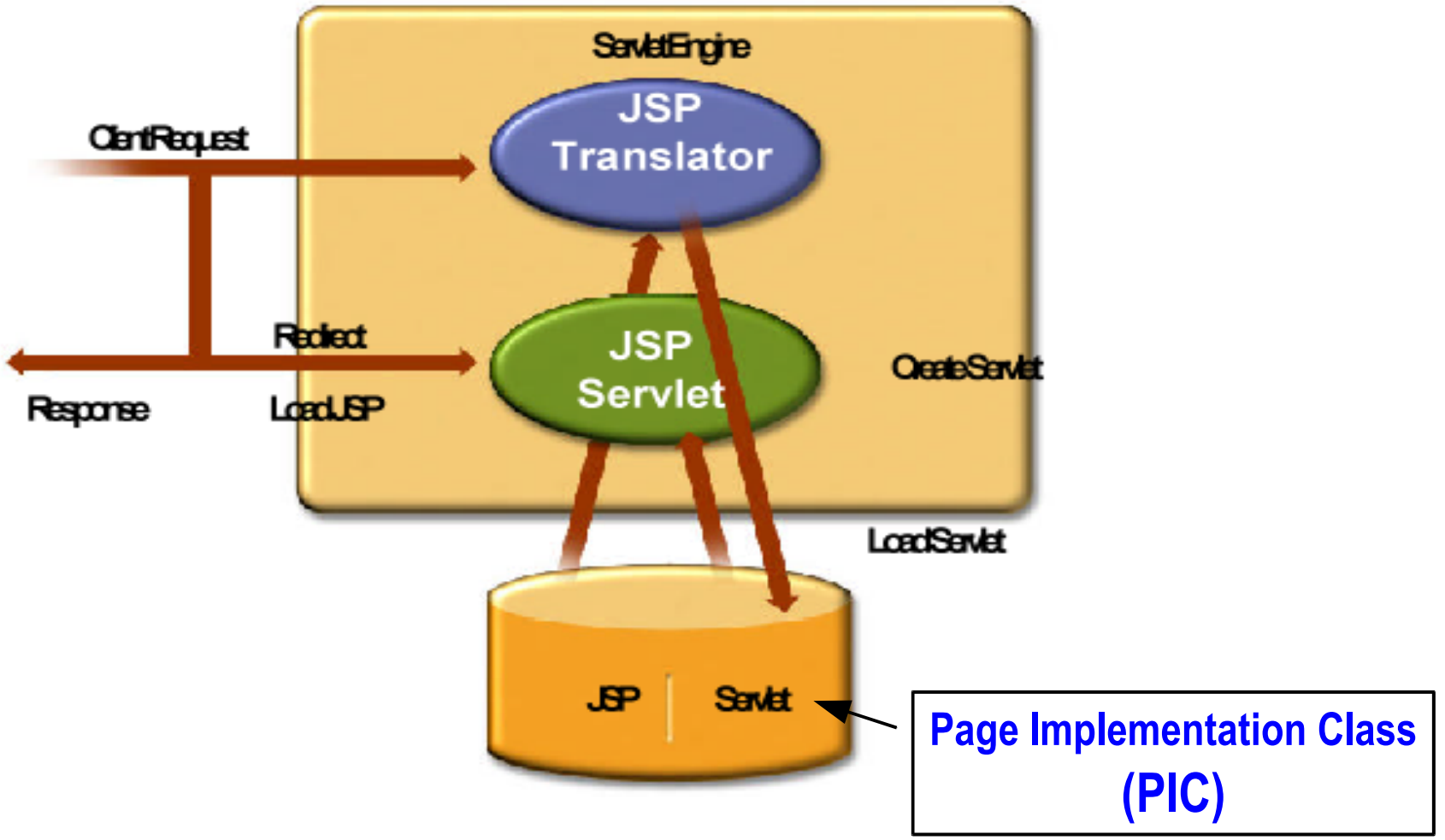
Бизнес-логика

JavaBean-ы

isOrderValid ( )

saveOrder ( )

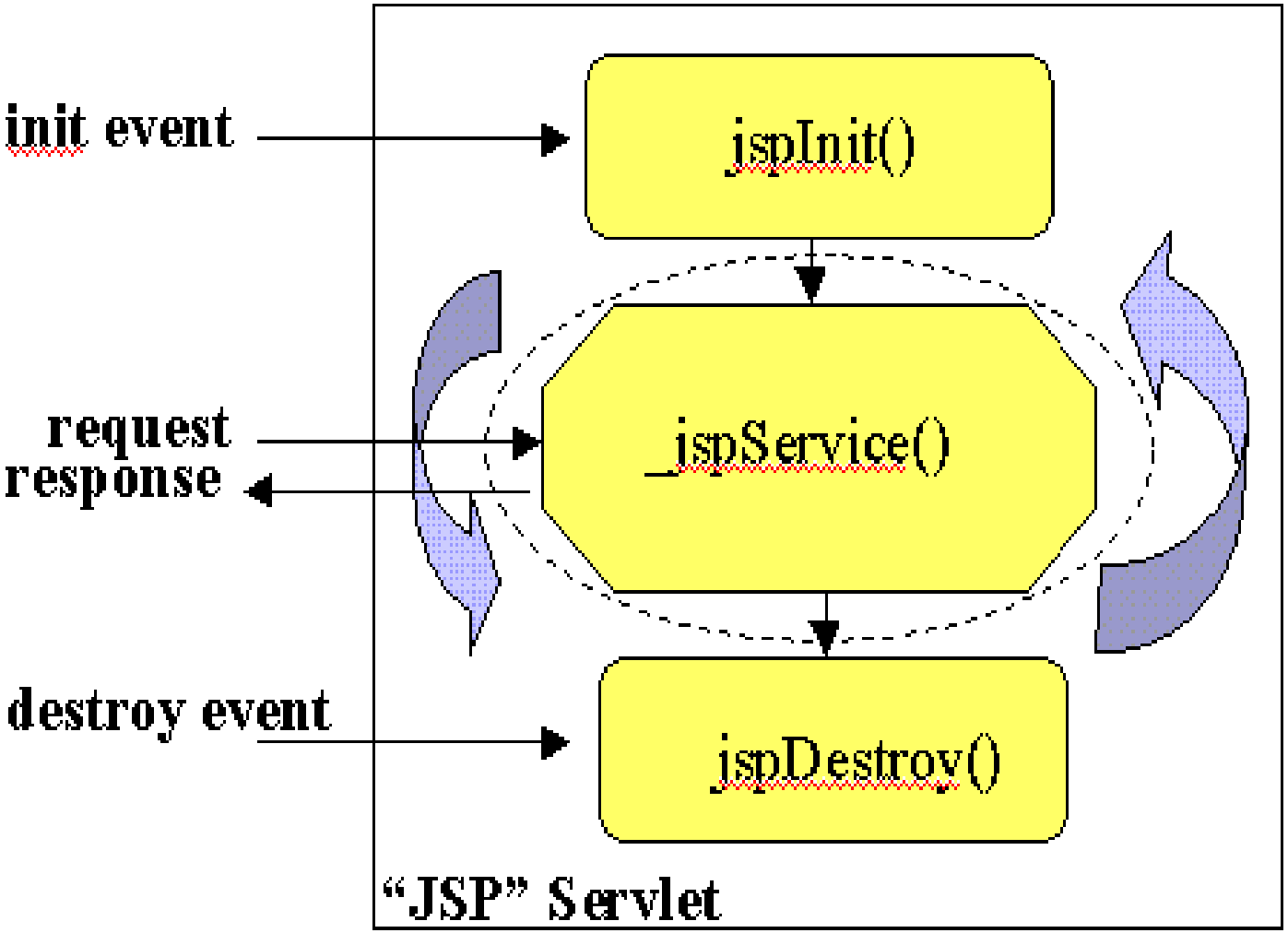
Архитектура JSP



Этапы жизненного цикла JSP-страницы

- Трансляция
  - JSP-страница преобразуется в исходный файл PIC
  - Выполняется контейнером автоматически
  - Правила зафиксированы в спецификации
  - Полученный код можно найти в сервере приложений
- Компиляция
  - Исходный файл PIC компилируется стандартным компилятором Java
  - Для работы веб-контейнера необходим JDK
- Выполнение

Методы ЖЦ на этапе выполнения



Способы динамической генерации содержимого

a) Встраивание Java-кода в JSP-страницу

b) Вызов Java-кода из JSP-страницы

c) Использование **JavaBean**-ов в JSP-страницах

d) Разработка собственных действий, определяемых программистом (**custom actions**)

e) Использование сторонних действий, определяемых программистом, в т.ч. JSTL (JSP Standard Tag Library)

f) Применение шаблона проектирования MVC

g) Использование **каркаса веб-приложения**

Элементы стандартного синтаксиса JSP

- 1) шаблон (template)
- 2) директива (directive)
- 3) скриптовый элемент (scripting element)
- 4) действие (action)
- 5) EL-элемент (EL-element)
- 6) комментарий

Скриптовые элементы

- Содержат Java-код, в точности включаемый в исходный код PIC
- **Не рекомендуются**
  - Можно запретить их использование
- Три вида:
  - Выражения: <%= Expressions %>
  - Скриптлеты: <% Code %>
  - Объявления: <%! Declarations %>

Выражения

- На этапе трансляции
  - out.println(expression)
- На этапе выполнения
  - Выражение вычисляется и преобразуется в строку
  - Строка напрямую записывается в поток вывода сервлета
  - В выражении можно использовать встроенные объекты
- Формат
  - <%= Expression %> или
  - <jsp:expression>Expression</jsp:expression>
  - Нельзя использовать точку с запятой

Примеры выражений

- Вывод случайного числа с помощью класса Math
  - Random number: <%= Math.random() %>
- Использование встроенных объектов
  - Имя Вашего компьютера: <%= request.getRemoteHost() %>
  - Ваш параметр: <%= request.getParameter(“yourParameter”) %>
  - Сервер: <%= application.getServerInfo() %>
  - ID сессии: <%= session.getId() %>

Скриптлеты

- На этапе трансляции
  - Содержат произвольный Java-код, вставляемый в метод jspService() сервлета
- На этапе выполнения
  - Операции, которые невозможны/затруднительны в выражениях
    - Установка заголовков ответа и кода статуса
    - Изменение базы данных
    - Циклы, условные операторы
  - Могут использовать встроенные объекты
- Формат:
  - <% Java-код %> или
  - <jsp:scriptlet>Java-код</jsp:scriptlet>

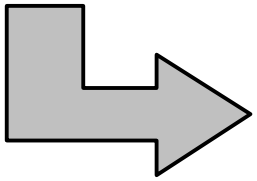
Пример скриптлета с циклом

```
<%
  Iterator i = cart.getItems().iterator();
  while (i.hasNext()) {
    ShoppingCartItem item =
      (ShoppingCartItem)i.next();
    BookDetails bd = (BookDetails)item.getItem();
%>

    <tr>
    <td align="right" bgcolor="#ffffff">
    <%=item.getQuantity()%>
    </td>
    <td bgcolor="#ffffaa">
    <strong><a href="
    <%=request.getContextPath()%>/bookdetails?bookId=
    <%=bd.getBookId()%>"><%=bd.getTitle()%></a></strong>
    </td>
    ...
<%
  // End of while
  }
%>
```

Пример трансляции фрагмента JSP

```
<H2>My HTML</H2>
<%= myExpression() %>
<% myScriptletCode(); %>
```



```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException {
    response.setContentType("text/html");
    HttpSession session = request.getSession(true);
    JSPWriter out = response.getWriter();

    // Шаблон выводится "как есть"
    out.println("<H2>My HTML</H2>");

    // Выражение преобразуется в строку и выводится
    out.println(myExpression());

    // Скриптлет вставляется "как есть" в _jspService()
    myScriptletCode();
    ...
}
```

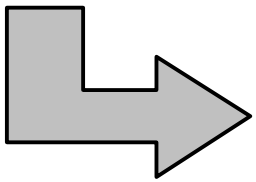
Объявления

- Определяют поля и методы в PIC
  - Встроенные объекты не доступны
- Для инициализации и очистки в JSP-страницах с помощью объявлений переопределяются методы jspInit() и jspDestroy()
- Формат:
  - <%! объявление поля или метода %>
  - <jsp:declaration>объявление поля или метода </jsp:declaration>

Пример трансляции фрагмента JSP

```
<H1>Some heading</H1>
    <%!
      private String randomHeading() {
         return("<H2>" + Math.random() + "</H2>");
      }%>

<%= randomHeading() %>
```



```
public class xxxx implements HttpJSPPage {
  private String randomHeading() {
     return("<H2>" + Math.random() + "</H2>");
  }

  public void _jspService(HttpServletRequest request,
                          HttpServletResponse response)
                          throws ServletException, IOException {
      response.setContentType("text/html");
      HttpSession session = request.getSession(true);
      JSPWriter out = response.getWriter();
      out.println("<H1>Some heading</H1>");
      out.println(randomHeading());
      ...
  }
  ...
}
```

Директивы

- Обрабатываются контейнером на этапе трансляции и влияют на нее
- Не генерируют вывода
- По завершении трансляции директив не существует
- Формат:
  - <%@ имя_директивы атрибут1="значение1" атрибут2="значение2" ... %>

Стандартные директивы

- page: определяет общие свойства страницы, которые влияют на ее трансляцию
  - <%@ page import="java.util.* %>
- include: включение содержимого некоторого файла в состав исходного текста JSP на этапе трансляции
  - <%@ include file="header.html" %>
- taglib: подключение библиотеки действий, определенных программистом
  - <%@ taglib uri="mytags" prefix="codecamp" %>

Директива page

- Определяет
  - Список импортируемых классов
    - <%@ page import="java.util.*,ru.ivgratchev.sample.*" %>
  - MIME-тип генерируемого ответа
    - <%@ page contentType="text/plain" %>
  - Участие страницы в сессии
    - <%@ page session="true" %> <%!--По умолчанию --%>
  - Страницу обработки ошибок
    - <%@ page errorPage="errorpage.jsp" %>

Действия

- Основной механизм для декларативного использования некоторой функциональности
  - **Замена** скриптовых элементов
- Стандартные / определяемые программистом:
  - Группируются в библиотеки действий (TLD)
  - Реализуются обработчиками
    - На языке Java: классические и простые
    - На языке JSP: тэг-файлы
- Формат — в соответствии с синтаксисом XML:

```
<префикс:имя_действия атрибут1="значение1" атрибут2="значение2" ... >
    <!—тело действия -->
    ...
</префикс:имя_действия>
```

EL-элементы

- Expression Language (EL) — простой язык для записи выражений в компактной форме
  - **Замена** скриптовых выражений
  - Используются в шаблоне и в атрибутах действий, вычисляемых во время выполнения
- EL-вычислитель интерпретирует EL-элементы на этапе выполнения
- Формат:

```
${текст_EL-выражения} и #{текст_EL-выражения}
```

- Примеры:

```
${requestScope.username}
${username}, ${product.model}
#{client.name}
```

Комментарии

- **<!-- комментарий -->**
  - часть шаблона, выводится в генерируемый ответ
  - вложенные элементы JSP обрабатываются
- **<%-- комментарий --%>**
  - для отключения части функциональности JSP
  - вложенные JSP-элементы **не** обрабатываются
- комментарии, специфичные для языка скриптов, в теле скриптовых элементов

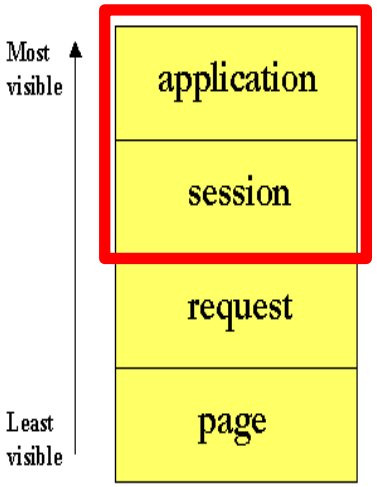
```
<% // комментарий
    if (…) { … } %>
```

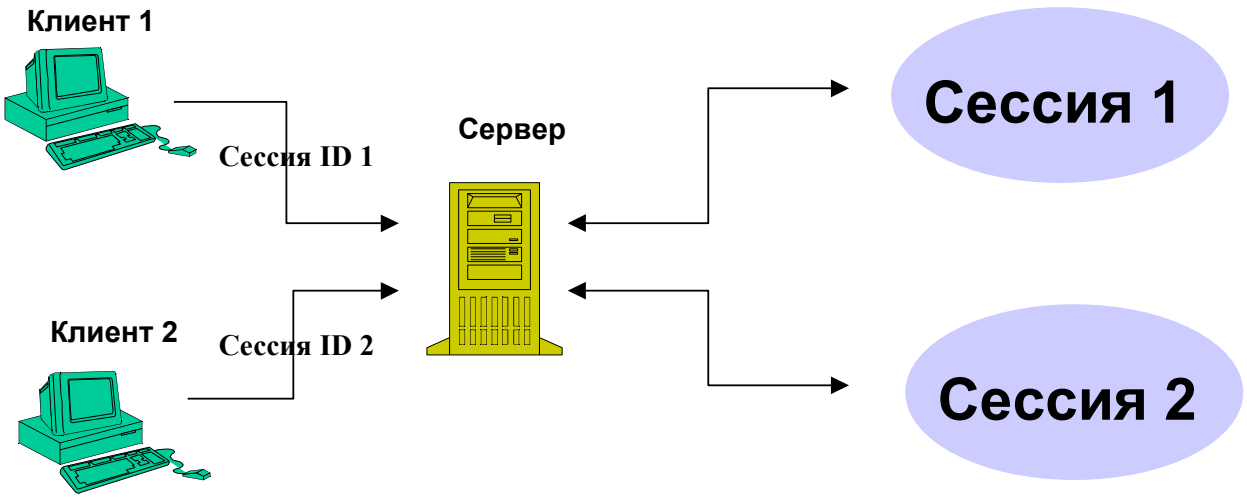
Встроенные объекты

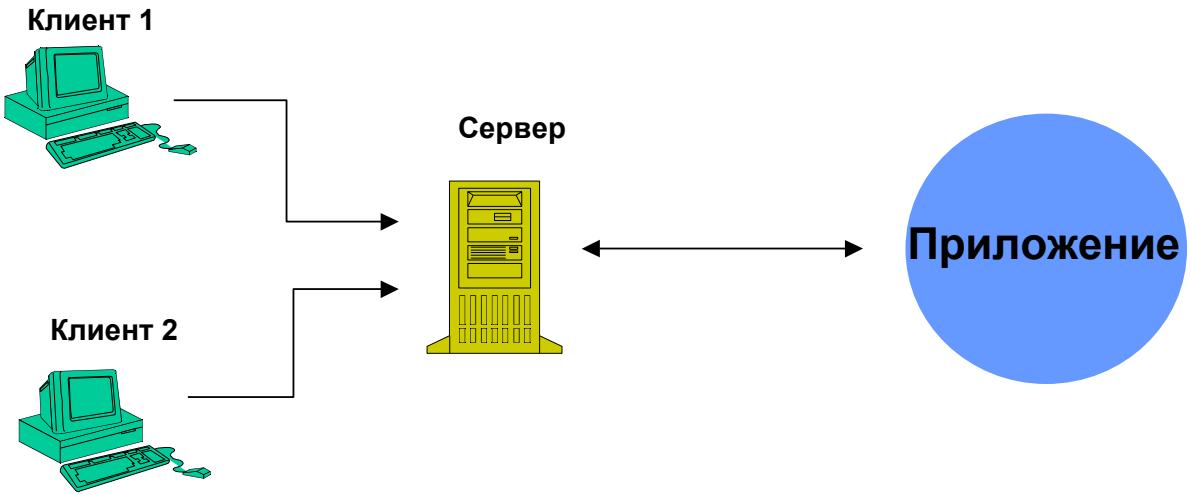
- Доступны на JSP-странице без объявления
- Создаются контейнером автоматически

Название	Тип	Область видимости
request	`javax.servlet.ServletRequest`	request
response	`javax.servlet.ServletResponse`	page
session	`javax.servlet.http.HttpSession`	session
application	`javax.servlet.ServletContext`	application
config	`javax.servlet.ServletConfig`	page
pageContext	`javax.servlet.jsp.PageContext`	page
out	`javax.servlet.jsp.JspWriter`	page
page	`java.lang.Object`	page
exception	`java.lang.Throwable`	page

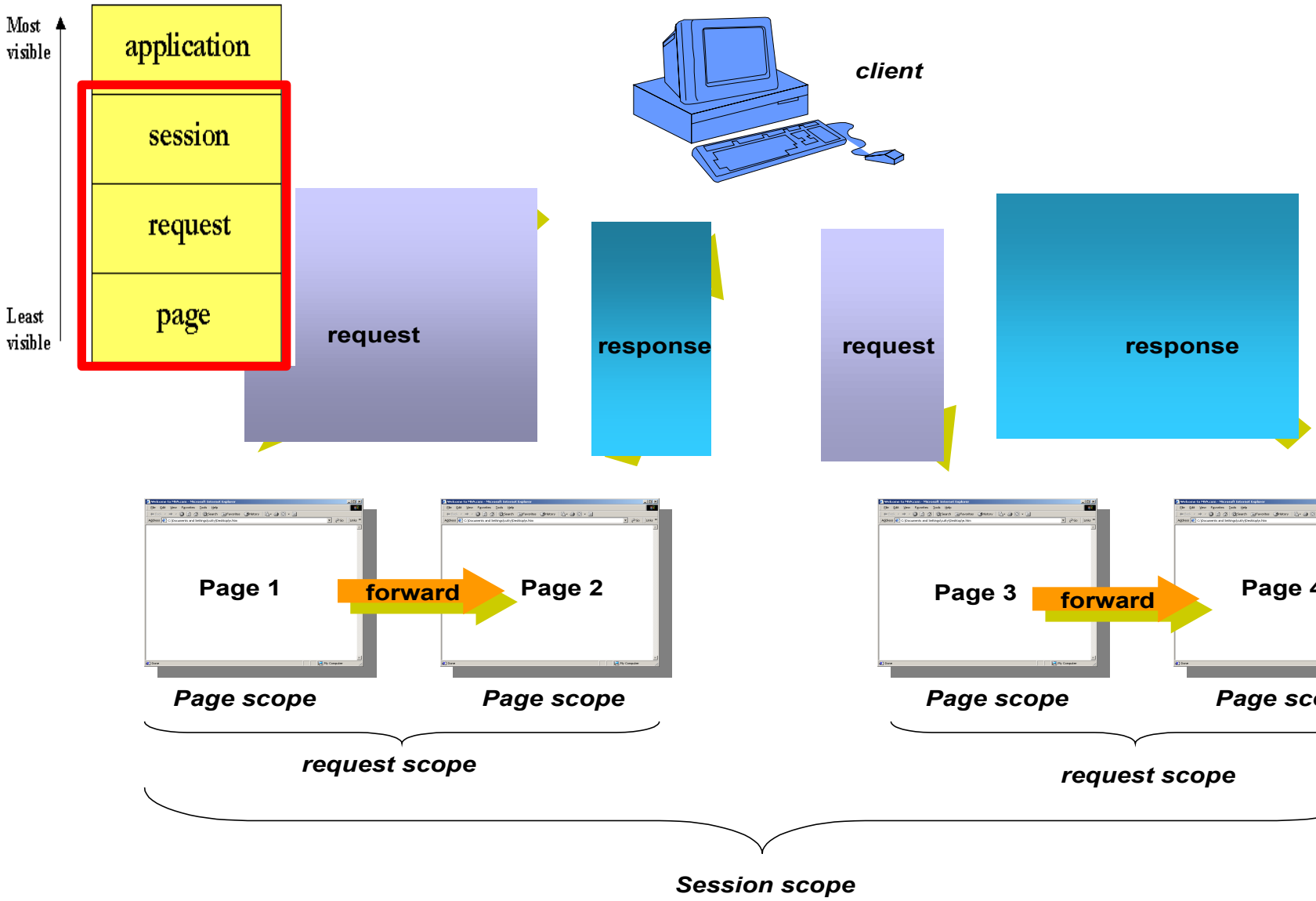
Области видимости (контексты)







Области видимости (контексты)



JavaBean-ы

- Java-классы, которые можно повторно использовать и формировать из них приложение
- JavaBean — обычный Java-класс, отвечающий определенным соглашениям по проектированию
  - JavaBean обладает свойствами
    - Чтение/запись, только для чтения, только для записи
    - Простые и индексированные
  - Для свойств определены get- и/или set-методы
  - У JavaBean должен быть конструктор по умолчанию
- В JSP можно создавать и инициализировать бины, получать и устанавливать значения их свойств
  - Скриптовые элементы или стандартные действия

Достоинства использования JavaBean-ов в JSP-страницах

- Дизайнеру не нужно изучать Java
- Разделение содержимого и представления
- Улучшенное повторное использование кода
- Простое использование контекстов для обмена данными
- Удобное сопоставление параметров запроса и свойств объектов

Создание JavaBean-ов

- Стандартное действие <jsp:useBean>
  - Объявляется локальная переменная с именем cart
  - В сессии ищется объект с именем cart класса cart.ShoppingCart
  - Если такого объекта не существует, он создается и сохраняется в сессии

```
<jsp:useBean id="cart" class="cart.ShoppingCart" scope="session"/>
```

- Скриптлет

```
<%
  ShoppingCart cart = (ShoppingCart)session.getAttribute("cart");
  if (cart == null) {
     cart = new ShoppingCart();
     session.setAttribute("cart", cart);
  }%>
```

Установка свойств JavaBean-ов

- Стандартное действие <jsp:setProperty>
  - Установка строкового значения свойства
    - <jsp:setProperty name="beanName" property="propName" value="string constant"/>
  - Установка свойства по параметру запроса
    - <jsp:setProperty name="beanName" property="propName" param="paramName"/>
    - <jsp:setProperty name="beanName" property="propName"/>
  - Установка всех свойств по параметрам запроса
    - <jsp:setProperty name="beanName" property="*"/>
  - Присвоение свойству значения выражения
    - <jsp:setProperty name="beanName" property="propName" value="<%= expression %>"/>

Вывод свойств JavaBean-ов

- С помощью скриптлета
  - <%= beanName.getPropName() %>
- С помощью стандартного действия <jsp:setProperty>
  - <jsp:getProperty name="beanName" property="propName"/>

Перенаправление запросов

Включение веб-ресурса в ответ

- Статический ресурс
  - Добавляется в ответ «включающего» сервлета
- Динамический веб-компонент (сервлет или JSP)
  - Запрос посылается «включаемому» веб-компоненту
  - Выполняется «включаемый» веб-компонент
  - Результат выполнения «включаемого» добавляется в ответ «включающего» веб-компонента
- Ограничения на работу «включаемого» веб-компонента с ответом
  - Нельзя непосредственно или косвенно устанавливать заголовки ответа

Включение веб-ресурса в ответ сервлета

- Получить диспетчер запросов из контекста сервлета или запроса
  - По относительному URL веб-компонента
    - `RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/banner");`
    - `RequestDispatcher dispatcher = request.getRequestDispatcher("/banner");`
  - По имени веб-компонента
    - `RequestDispatcher dispatcher = getServletContext().getNamedDispatcher("BannerServlet");`
- Вызвать метод include() диспетчера запросов, передав через параметры запрос и ответ
  - `dispatcher.include(request, response);`

Включение веб-ресурса в JSP-страницу

- Директива include
  - Текст, содержащийся в другом файле, напрямую вставляется в JSP-страницу перед трансляцией
  - Синтаксис и пример
    - <%@ include file="путь_к_файлу" %>
    - <%@ include file="banner.jsp" %>
- Стандартное действие <jsp:include>
  - Обрабатывается на этапе выполнения
  - Поддерживается статическое и динамическое содержимое
  - Синтаксис и пример
    - <jsp:include page="относительный_URL" />
    - <jsp:include page="date.jsp"/>

Передача обработки другому веб-компоненту

- Варианты использования
  - Разделение веб-компонентов на Front End и Presentation
  - Обработка запроса по цепочке
- Ответ пользователю полностью формируется тем компонентом, на который передается обработка
  - Если в текущем компоненте уже выполнялся вывод в ServletOutputStream или PrintWriter, то выбрасывается исключение IllegalStateException

Передача обработки из сервлета

- Получить диспетчер запросов из контекста сервлета или запроса
- Вызвать метод forward() диспетчера запросов, передав через параметры запрос и ответ

```
public class Dispatcher extends HttpServlet {
  public void doGet(HttpServletRequest request,
    HttpServletResponse response) {
    request.setAttribute("selectedScreen",
      request.getServletPath());
    RequestDispatcher dispatcher = request.
      getRequestDispatcher("/template.jsp");
    if (dispatcher != null)
      dispatcher.forward(request, response);
  }
  public void doPost(HttpServletRequest request,
  ...
}
```

Передача обработки из JSP-страницы

- Стандартное действие <jsp:forward>
  - <jsp:forward page="относительный_URL" />
- Установка дополнительных параметров запроса с помощью действия <jsp:parameter>

```
<jsp:forward page="..." >
  <jsp:param name="param1" value="value1"/>
</jsp:forward>
```

Перенаправление запроса

- Коды статуса 3xx
- Способ 1:

```
res.setStatus(res.SC_MOVED_PERMANTLY);
res.setHeader("Location", "http://...");
```

- Способ 2:

```
public void sendRedirect(String url)
```