

# Web-технологии и web-дизайн

## Сервлеты и JSP



Беркунский Е.Ю., кафедра ИУСТ, НУК  
eugeny.berkunsky@gmail.com  
<http://www.berkut.mk.ua>

# Что такое Servlet?

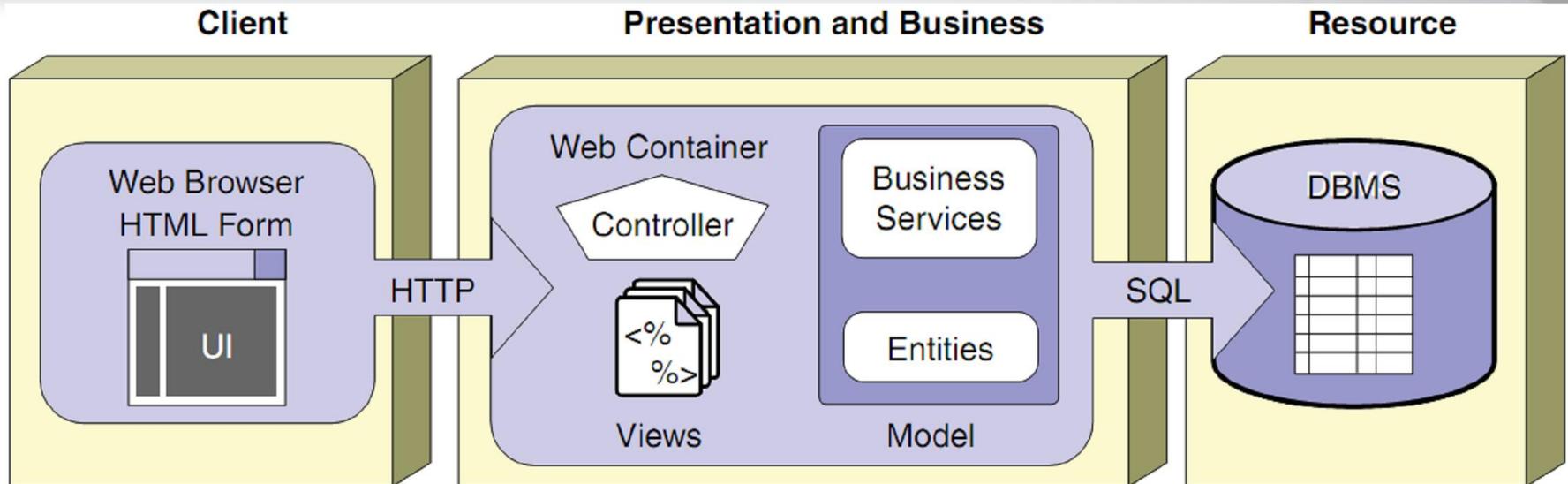
- Сервлет является классом Java, который используется для расширения возможностей серверов, предназначенных для размещения приложений. Сервлеты могут отвечать на запросы и генерировать отклики.
- Базовым классом для всех сервлетов является `javax.servlet.GenericServlet`. Этот класс определяет обобщенный, независимый от протокола сервлет.
- Наиболее распространенный тип сервлета – HTTP-сервлет. Этот тип сервлета используется в обработке HTTP-запросов и генерировании HTTP-откликов. HTTP-сервлет представляет собой класс, который расширяет класс `javax.servlet.http.HttpServlet`, являющийся подклассом базового класса `javax.servlet.GenericServlet`.

# Контейнер сервлетов Servlet Container

- Контейнер сервлетов – это часть Web сервера.
- Контейнер сервлетов обеспечивает сетевые сервисы, посредством которых отправляются запросы (requests) и ответы (responses), декодирует запросы основанные на MIME, и форматирует основанные на MIME ответы.
- Контейнер сервлетов также управляет сервлетами на протяжении их жизненного цикла.



# Контейнер сервлетов Web Container



# HTTP запити

- Сервлет повинен реалізовувати один або більше методів для відповідей на певні HTTP-запити.
- Ці переопреділявані методи визначені в батьківському класі `HttpServlet`.
- Ці методи названі таким чином, щоб можна було інтуїтивно зрозуміти, який метод використовувати в тому чи іншому випадку:

HTTP-запит	Метод HTTP-сервлета
GET	<code>doGet(HttpServletRequest request, HttpServletResponse response)</code>
POST	<code>doPost(HttpServletRequest request, HttpServletResponse response)</code>
PUT	<code>doPut(HttpServletRequest request, HttpServletResponse response)</code>
DELETE	<code>doDelete(HttpServletRequest request, HttpServletResponse response)</code>

*Розробники програм завжди не викликають ці методи безпосередньо. Їх автоматично викликає сервер програм кожного разу, коли отримує відповідний HTTP-запит.*

# HTTP запити

- HTTP-запит GET генерується завжди раз, коли користувач вводить URL сервлета в адресній строці об'єкту перегляду, або клікає по посиланню, що вказує на URL сервлета, або надсилає HTML-форму з використанням методу GET, в якій атрибут action вказує на URL сервлета.
- В будь-якому з цих випадків код сервлета всередині методу doGet() виконується.
- HTTP-запит POST зазвичай генерується, коли користувач надсилає HTML-форму з використанням методу POST і його атрибут action вказує на URL сервлета. В цьому випадку код сервлета всередині методу doPost() виконується.



# Простой сервлет

```
public class SimpleServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) {  
        try {  
            response.setContentType("text/html");  
            PrintWriter printWriter = response.getWriter();  
            printWriter.println("<h2>");  
            printWriter.println(  
                "Если вы читаете это, ваш сервер приложений работает нормально!");  
            printWriter.println("<h2>");  
        } catch (IOException ex) {  
            ex.printStackTrace();  
        }  
    }  
}
```

# Простой сервлет

Дескриптор развертывания – web.xml (v 3.0)

```
<?xml version="1.0" encoding="UTF8"?>
<webapp xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/webapp_3_0.xsd" version="3.0">
  <servlet>
    <servletname>SimpleServlet</servletname>
    <servletclass>server.SimpleServlet</servletclass>
  </servlet>
  <servletmapping>
    <servletname>SimpleServlet</servletname>
    <urlpattern>/simple</urlpattern>
  </servletmapping>
</webapp>
```

# Простой сервлет

Сервлет 3.0 делает дескриптор развертывания приложения, файл `web.xml`, необязательным.

Сервлеты могут быть сконфигурированы с помощью аннотаций вместо использования XML.

*Если веб-приложение конфигурируется и с помощью аннотаций и с помощью дескриптора развертывания `web.xml`, то настройки, указанные в файле `web.xml`, имеют приоритет.*

```
@WebServlet(name="simpleservlet", urlPatterns = {"/simple"})
```

# Простая JSP

- JSP – это страница, содержащая и статическую HTML-разметку, и динамический контент.
- Динамический контент может быть сгенерирован путем использования фрагментов кода Java, называемых скрипглетами (scriptlets), или путем использования стандартных или пользовательских JSP-тегов.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<%@ page import="java.util.Date" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Дата и время сервера</title>
  </head>
  <body>
    <h1>Дата и время сервера: <% out.print(new Date()); %></h1>
  </body>
</html>
```

- 1. Translation**
- 2. Compilation**
- 3. Class Loading**
- 4. Instantiation**
- 5. Initialization**
- 6. Request Processing**
- 7. Destroy**

# Жизненный цикл JSP

- 1. Translation** – JSP контейнер проверяет код JSP страницы, парсит ее для создания кода сервлета.
- 2. Compilation** – JSP контейнер компилирует исходный код `jsp` класса и создает класс на этой фазе.
- 3. Class Loading** – контейнер загружает классы в память на этой фазе.
- 4. Instantiation** – внедрение конструкторов без параметров созданных классов для инициализации в памяти классов.

**5. Initialization** – в контейнере вызывается `init` метод объекта JSP класса и инициализируется конфигурация сервлета с `init` параметрами, которые указаны в дескрипторе развертывания (**`web.xml`**). После этой фазы JSP способен обрабатывать запросы клиентов.

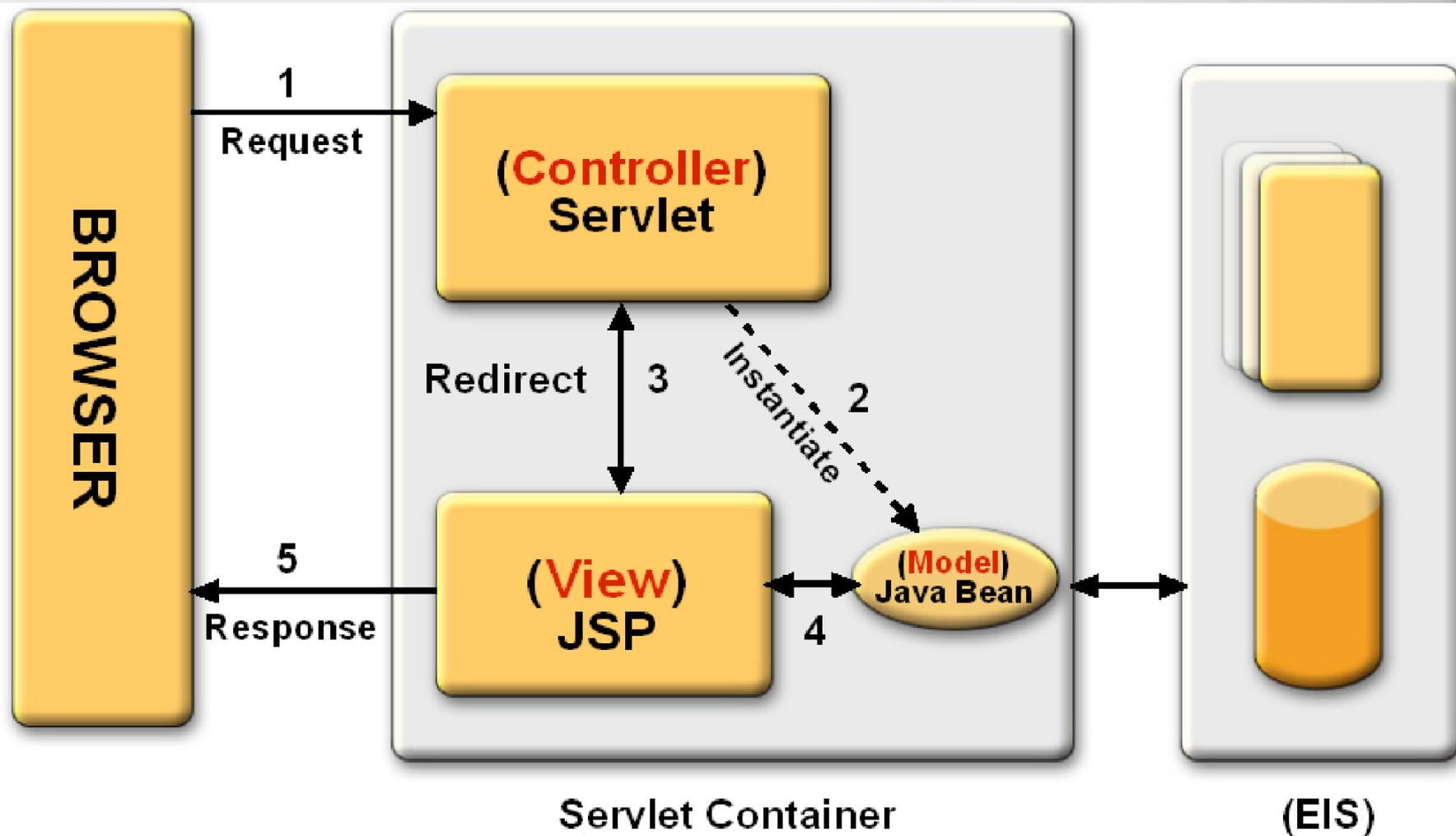
Обычно фазы **1-5** происходят после первого запроса клиента (т.е. ленивая загрузка), но можно настроить загрузку и инициализацию JSP на старте приложения по аналогии с сервлетами.

**6. Request Processing** – длительный этап жизненного цикла - обработка запросов клиента JSP страницей.

Обработка является многопоточной и аналогична сервлетам — для каждого запроса создается новый поток, создаются объекты `ServletRequest` и `ServletResponse` и происходит внедрение сервис методов JSP.

**7. Destroy** – последняя фаза жизненного цикла JSP на которой JSP класс удаляется из памяти. Обычно это происходит при выключении сервера или андеплое приложения.

# Взаємодія сервлета і JSP в рамках шаблону MVC



# Реализация MVC при помощи RequestDispatcher

1. Разработать JavaBeans-объекты, которые будут представлять данные
2. Использовать сервлет для обработки запросов
  - Сервлет считывает параметры запроса, проверяет входные данные и т.д.
3. Поместить данные в JavaBeans-объекты
  - Сервлет тем или иным способом вызывает код бизнес-логики приложения. Результаты помещаются в JavaBeans-объекты
4. Сохранить JavaBean в объекте request, session, или servletContext
  - Сервлет вызывает метод setAttribute объекта request, session, или servletContext, чтобы сохранить ссылку на бин

# Реализация MVC при помощи RequestDispatcher

5. Перенаправить (**forward**) запрос к JSP-странице
  - Сервлет определяет, какая JSP –страница нужна в данной ситуации и использует метод **forward()** объекта **RequestDispatcher** для передачи управления этой странице.
6. Извлечь данные из JavaBeans.
  - JSP страница извлекает данные из бинов при помощи директивы **jsp:useBean** с соответствующим параметром **scope**. Затем страница использует **jsp:getProperty** или выражения Expression Language, чтобы вывести свойства бина
  - JSP не создает и не изменяет бин; она просто извлекает и отображает данные, созданные сервлетом



# Сервлеты + JSP



# Вопросы?



# Web-технологии и web-дизайн

## Сервлеты и JSP



Беркунский Е.Ю., кафедра ИУСТ, НУК  
eugeny.berkunsky@gmail.com  
<http://www.berkut.mk.ua>