ИСПОЛЬЗОВАНИЕ JAVA BEANS В ДРУГИХ СРЕДАХ. СОЗДАНИЕ СЕРВЛЕТОВ, JSP-СТРАНИЦЫ И ПРОСТОГО БРАУЗЕРА

Класс Java Bean должен соответствовать ряду ограничений:

- иметь конструктор, который не принимает никаких параметров
- определять для всех свойств, которые используются в jsp, методы геттеры и сеттеры
- названия геттеров и сеттеров должны соответствовать условностям: перед именем переменной добавляется get (для геттера) и set (для сеттера), а название переменной включается с большой буквы. Например, если переменная называется firstName, то функции геттера и сеттера должны называться соответственно getFirstName и setFirstName.

  Однако для переменных типа boolean для функции геттера используется вместо get приставка is. Например, переменная enabled и геттер isEnabled.

- реализовать интерфейс Serializable или Externalizable

Рассмотрим, как использовать классы JavaBean. Допустим, у нас есть следующая структура:

В папке *Java Resources/src* расположен класс User со следующим кодом:

```
import java.io.Serializable;

public class User implements Serializable {

   private static final long serialVersionUID = 20412755122192399 90L;

   private String name;
   private int age;

   public User() {
      this.name = "";
      this.age = 0;
   }
   public User(String name, int age) {

      this.name = name;
      this.age = age;
   }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

Данный класс представляет пользователя и является классом Java Bean: он реализует интерфейс Serializable, имеет конструктор без параметров, а его методы - геттеры и сеттеры, которые предоставляют доступ к переменным name и age, соответствуют условностям.

В папке *WebContent* определена страница **user.jsp**. Определим в ней следующий код:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>User Java Bean Page</title>
</head>
<body>
<div>
<p>Name: ${user.name}</p>
<p>Age: ${user.age}</p>
</div>
</body>
</html>
```

Данная страница jsp получает извне объект user и с помощью синтаксиса EL выводит значения его свойств. Стоит обратить внимание, что здесь идет обращение к переменным name и age, хотя они являются приватными.

В папке *Java Resources/src* в файле **HelloServlet.java** определен сервлет HelloServlet:

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/hello")
public class HelloServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        User tom = new User("Tom", 25);
        request.setAttribute("user", tom);
        getServletContext()
            .getRequestDispatcher("/user.jsp")
            .forward(request, response);
    }
}
```

Сервлет создает объект User. Для передачи его на страницу user.jsp устанавливается атрибут "user" через вызов **request.setAttribute("user", tom). Далее происходит перенаправление на страницу user.jsp. И, таким образом, страница получит данные из сервлета.**

Java Server Pages

Java Server Pages представляет технологию, которая позволяет создавать динамические веб-страницы. Изначально JSP (вместе с сервлетами) на заре развития Java EE являлись доминирующим подходом к веб-разработке на языке Java. И хотя в настоящее время они уступило свое место другой технологии - JSF, тем не менее JSP продолжают широко использоваться.

По сути Java Server Page или JSP представляет собой html-код с вкраплениями кода Java. В то же время станицы jsp - это не стандартные html-страницы. Когда приходит запрос к определенной странице JSP, то сервер обрабатывает ее, генерирует из нее код html и отправляет его клиенту. В итоге пользователь после обращения к странице JSP видит в своем браузере обычную html-страницу.

Как и обычные статические веб-страницы, файлы JSP необходимо размещать на веб-сервере, к которому обычные пользователи могут

обращаться по протоколу http, например, набирая в адресной строке браузера нужный адрес. Однако чтобы сервер мог обрабатывать файлы JSP, он должен использовать движок JSP (**JSP engine**), который также называют JSP-контейнером. Есть множество движков JSP, и все они реализуют одну и ту же спецификацию и в целом работают одинаково. Однако тем не менее при переносе кода с одного веб-сервера на другой могут потребоваться небольшие изменения.

В данном случае для работы с JSP мы будем использовать Apache Tomcat, который одновременно является и веб-сервером и контейнером сервлетов и JSP.

Создадим простейшую страницу JSP. Для этого где-нибудь на жестком диске определим файл **index.jsp**. Все станицы JSP имеют расширение *jsp*. Откроем этот файл в любом текстовом редакторе и определим в нем следующий код:

```
<%
String header = "Apache Tomcat";
%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>First JSP App</title>
    </head>
    <body>
        <h2><%= header %></h2>
        <p>Today <%= new java.util.Date() %></p>
    </body>
</html>
```

С помощью тегов `<% ... %>` мы можем определить код Java на странице JSP. В данном случае мы просто определяем переменную типа String, которая называется header.

Затем идет стандартный код страницы html. Чтобы внедрить код java внутрь html-страницы применяются теги `<%= %>` - после знака равно указывается выражение Java, результат которого будет выводиться вместо этих тегов. В данном случае, используются две таких вставки. Первая вставка - значение переменной header, которая была определена выше. Вторая вставка - выражение `new java.util.Date()`, которое возвращает текущую дату.

Для тех, кто знаком с веб-разработкой на PHP, это может напоминать оформление файлов php, которые также содержать код html и код php.

Теперь поместим данный файл на сервер - в данном случае в контейнер Tomcat. Перейдем в Apache Tomcat к папке *webapps\ROOT*. Удалим из нее все содержимое и поместим нашу страницу **index.jsp**, которая была создана выше.

Запустим Apache Tomcat (если он не запущен), и обратимся к приложению по адресу *http://localhost:xxxx/index.jsp*, где xxxx - номер порта, по которому запущен Tomcat:

В итоге Tomcat получит запрос к странице index.jsp, обработает код на java, сгенерирует html-страницу и отправит ее пользователю.

По умолчанию Apache Tomcat настроен таким образом, что все запросы к корню приложения по умолчанию обрабатываются страницей index.jsp, поэтому мы также можем обращаться к ней по адресу *http://localhost:xxxx*