Осипов Е. В.

КОНЦЕПЦИЯ РАЗРАБОТКИ ГИБКИХ WEB-ПРИЛОЖЕНИЙ

Создать *HTML*-документ нетрудно. Создать *Web*-приложение, объединяющее средства *HTML*, *JSP* и *Java*, обеспечивающее взаимодействие с базами данных и существующими приложениями, которое отличается гибкостью и удобно в сопровождении, значительно сложнее. В данной статье будут рассмотрены общие подходы к созданию *Web*-приложений на основе технологий *Java*. В качестве примера будет представлен электронный каталог учебно-методических пособий учебно-методического объединения (УМО), разработанный на кафедре «Конструирование радиоэлектронных и микропроцессорных систем».

Основные термины, используемые в статье:

JSP (*Java Server Pages*) – серверная технология, позволяющая встраивать и использовать *Java*-код (скриптлеты) в статических *Web*-страницах.

Servlet (сервлет) – представляет собой *Web*-компонент, эквивалентный активному ресурсу, генерирующий динамическое содержимое по принципу «*HTML* в программном коде». Сервлет, как и многие другие компоненты *Java*-технологий, является *Java*-классом, скомпилированным в платформо-независимый байт-код.

Существует несколько способов создания *Web*-приложений на базе *JSP*:

- Создание документа, как набора *HTML*-дескрипторов и *JSP*-скриптлетов;
- Делегирование функций компонентам *bean*;
- Использование сервлетов, *JSP*-документов и компонентов *Java-bean* для реализации архитектуры «модель-просмотр-контроллер» (*MVC – Model-View-Controller*).

Первый подход – включение больших объемов *Java*-кода в *JSP*-документы – приводит к созданию приложений, которые трудно сопровождать и модифицировать.

При делегировании функций компонентам *bean,* код перемещается в компоненты, поэтому приложения, созданные по такому принципу, становятся более жизнеспособными. Этот подход известен как архитектура *Model 1*.

Последний подход – объединение сервлетов, *JSP* и *bean* в рамках архитектуры *MVC* – позволяет создавать гибкие и расширяемые программы, удобные для сопровождения. При этом инкапсулируются функции и уменьшаются побочные эффекты при внесении изменений. Этот подход носит название архитектуры *Model 2*.

Архитектура *Model 1*, условно показанная на рисунке *1*, включает JSP-документы, компоненты *bean* и бизнес-объекты.

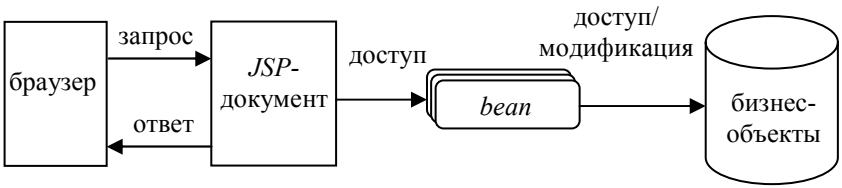

Рис. 1. Архитектура Model 1

Согласно данной архитектуре, запросы передаются *JSP*-документам, которые взаимодействуют с бизнес-объектами посредством *bean*. Такое косвенное обращение уменьшает зависимость *JSP*-документов от изменений, вносимых в бизнес-объекты.

Бизнес-объекты и компоненты *bean* реализуются разработчиками программного обеспечения, а авторы *Web*-страниц отвечают за создание *JSP*-документов. В идеале такое разделение обязанностей должно было бы обеспечить возможность параллельной работы сотрудников различных специальностей, участвующих в выполнении проекта. Однако на практике реализовать одновременное выполнение работ достаточно сложно, поскольку *JSP*-документы не только представляют содержимое, но и генерируют его.

В больших проектах чаще всего для реализации приложения выбирается архитектура *Model 2*.

Подобно *Model 1*, *Model 2* позволяет разделить бизнес-объекты и *JSP*-документы, что очень важно для большинства проектов, где бизнес-объекты претерпевают постоянные изменения. Кроме того, архитектура *Model 2* (рисунок *2*) отделяет генерацию содержимого от его представления.

Согласно данной архитектуре, запросы передаются сервлету, который обращается к бизнес-объектам и создает содержимое. Это содержимое сохраняется в компоненте *bean*, к которому имеет доступ *JSP*-документ. Документ представляет содержимое, применяя для этого, как правило, средства *HTML*.

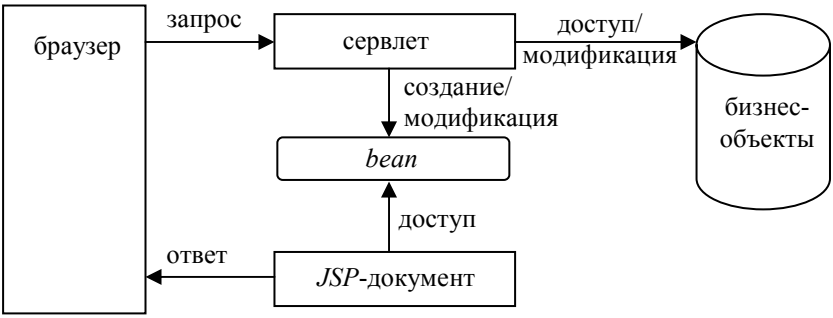

Рис. 2. Архитектура Model 2

Инкапсуляция *Java*-кода позволяет разработчикам программного обеспечения сосредоточить внимание на сервлетах и бизнес-объектах, в то время как авторы *Web*-страниц занимаются созданием *JSP*-документов. Все это позволяет создавать гибкие и расширяемые *Web*-приложения.

Однако можно пойти дальше и инкапсулировать часто используемый код в базовом наборе классов, что еще больше упростит работу над приложениями.

Рассмотрим простой набор классов, соответствующий *Model 2* и использующий те же базовые понятия, что и в *Apache Struts* (*http://www.apache.org*).

В данном наборе классов определены четыре типа объектов:

Action (действие) – интерфейс, реализуемый действиями, специфическими для приложения.

ActionFactory (фабрика действий) – создает экземпляры действий.

ActionServlet (сервлет действия) – отображает запросы в действия.

ActionRouter (маршрутизатор действий) – перенаправляет запросы *JSP*-документам.

Сервлет действия вызывается из *JSP*-документа или сервлета. Этот сервлет обращается к фабрике действий и в зависимости от запроса получает соответствующий тип действия.

После того как сервлет действий получает действие, он вызывает метод *perform* этого действия. Данный метод реализует функциональные возможности, специфические для конкретного приложения. Метод *Action.perform* возвращает маршрутизатор действий, который содержит *URI* для перенаправления запроса. Получив маршрутизатор, сервлет действий вызывает его метод *route*, который перенаправляет запрос указанному *Web*-компоненту (*JSP*-документ, *HTML*-страница). [*1, 2*]

Этот набор классов был применен в электронном каталоге учебно-методических пособий УМО, разработанном на кафедре «Конструирование радиоэлектронных и микропроцессорных систем».

Рассмотрим диаграмму взаимодействия классов при создании новой учетной записи пользователя (рисунок *3*).

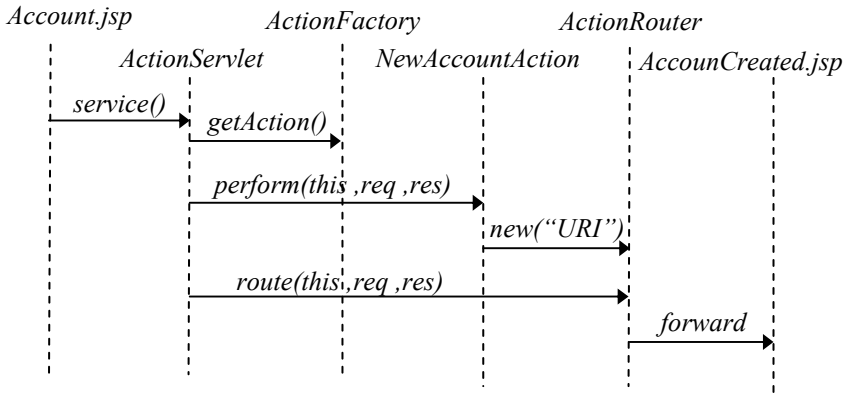

Рис. 3. Диаграмма взаимодействия классов при создании учетной записи

Account.jsp – страница, предназначенная для ввода данных о новом пользователе.

AccountCreated.jsp – страница, сообщающая о том, что новая запись успешно создана.

NewAccountAction – действие, которое проверяет введенную информацию и вносит ее в базу данных.

Кроме того в данном приложении широко используются пользовательские дескрипторы, что в совокупности с базовым набором классов делает его гибким, расширяемым и легким в сопровождении.

Список литературы:

1. Девид М. Гери *Java Server Pages*. Библиотека профессионала.: Пер. с англ. — М.: Издательский дом «Вильямс», *2002*. — *448*с.

2. Мухамедзянов Р.Р. *Java*. Серверные приложения. – М.: СОЛОН-Р, *2003*. – *336*с.

Работа выполнена под руководством к.т.н., доц. кафедры «Конструирование радиоэлектронных и микропроцессорных систем» Артемовой С. В.