Построение веб-приложений на базе архитектуры Model-View-Controller

Архитектура Model-View-Controller и веб-приложения

Архитектура Model-View-Controller (MVC) широко используется для разработки интерактивных приложений: как для приложений к графическим интерфейсом пользователя, так и для веб-приложений.

В архитектуре MVC приложение разделяется на три части – модель, представление и контроллер, каждая из которых выполняет свои определенные задачи и обязанности перед другими частями.

Модель представляет данные и бизнес-логику или операции, которые управляют доступом и изменением данных. Модель позволяет представлениям получить доступ к своему состоянию и оповещает их о его изменениях. Также модель позволяет контроллеру выполнять функции приложения, инкапсулированные в модели.

Представление отображает содержимое модели. Оно берет данные из модели и определяет, как они будут представлены, а также выполняет обновление при изменении модели. Представление направляет ввод пользователя на контроллер.

Контроллер определяет поведение приложения. Он обрабатывает запросы пользователя и выбирает необходимые представления. Контроллер интерпретирует ввод пользователя и определяет, какое действие должна выполнить модель. В приложении с графическим интерфейсом пользователя ввод пользователя включает в себя нажатия на кнопки и выбор пунктов из меню. В веб-приложении ввод пользователя представлен HTTP-запросами. Контроллер определяет отображаемое представление по действиям пользователя и результатам операций с моделью. Как правило, для каждой группы взаимосвязанных функций в приложении создается отдельный контроллер.

На рис. 14.1 показаны взаимосвязи между отдельными частями приложения, построенного по архитектуре MVC.

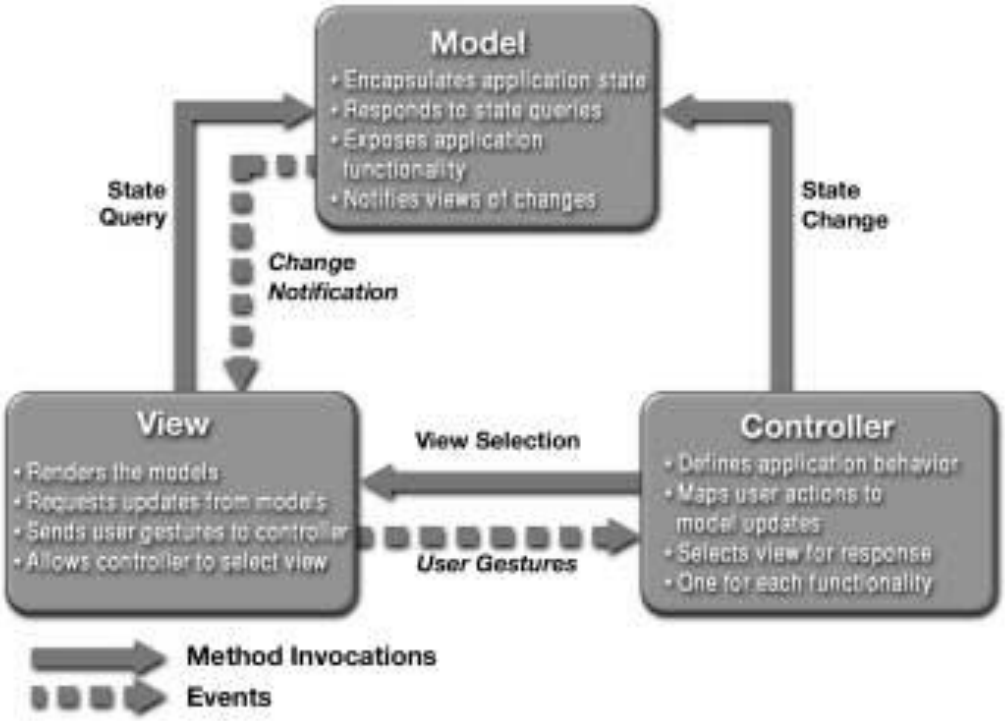


Рис. 14.1. Архитектура MVC

Разделение обязанностей между объектами модели, представления и контроллера (разделение бизнес-логики и логики представления) обладает следующими достоинствами:

- Уменьшение воздействия изменений. Бизнес-правила изменяются только в слое бизнес-логики, что вносит минимум изменений в представление. И наоборот, интерфейс приложения или последовательность страниц могут изменяться, не влияя на бизнес-логику.
- Упрощение сопровождения (в том числе нахождения и исправления ошибок). Как правило, одни и те же фрагменты бизнес-логики используются во многих частях приложения. Если представить их в виде отдельных компонентов, то изменения, сделанные в одном компоненте, приведут к изменению поведения во всех частях, где он используется. Аналогичными достоинствами обладает повторное использование логики представления (включение JSP-страниц, использование действий, определенных программистом).
- Независимость от типа клиента и повторное использование кода. Смешение представления данных и бизнес-логики привязывает бизнес-логику к конкретному типу клиента. Например, бизнес-логика, реализованная в скриптлете, не может быть использована сервлетом или клиентским приложением (application client – стандартный тип клиента для платформы Java EE).
- Специализация труда разработчиков – в проекте могут принимать участие разработчики прикладной логики, функций доступа к БД, определяемых программистом действий.JSP и авторы JSP-страниц.
- Централизация управления навигацией, безопасностью, журналированием и другими инфраструктурными функциями приложения.

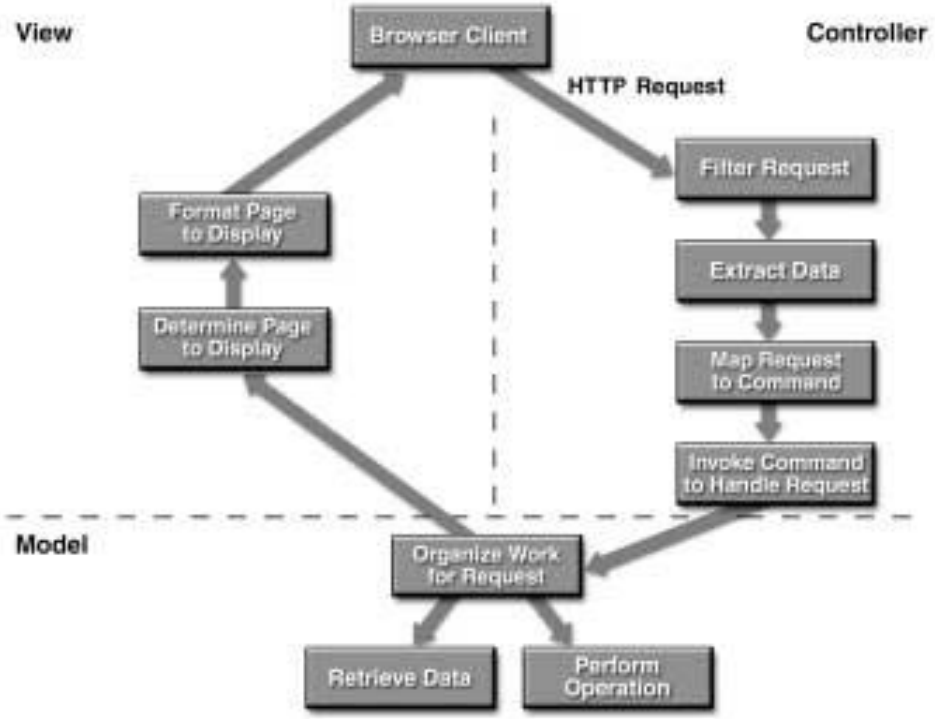


Рис. 14.2. Обработка запроса веб-приложением[1]

В классической архитектуре MVC представления самостоятельно регистрируются в модели для получения извещений об ее изменении. При изменении состояния модели

[1] Рисунок нуждается в доработке: напрямую модель и представление не связаны, так как определение отображаемой страницы выполняет контроллер

представление получает соответствующее извещение и может изменить свой внешний вид. Взаимодействие с клиентом по протоколу HTTP построено на принципе «запрос-ответ», поэтому в веб-приложении выполняемые по запросу изменения в модели не вызывают извещения, а сообщаются как ответ через представление.

На рис. 14.2 приведена последовательность обработки запроса веб-приложением и распределение отдельных действий по частям данного приложения.

Контроллер принимает HTTP-запрос, интерпретирует его, вызывает в модели соответствующую запросу бизнес-логику, по результатам операции и состоянию модели выбирает отображаемое представление, генерирует его и отправляет клиенту в качестве ответа.

Типовые архитектуры веб-приложений на платформе Java EE

Выделяют две типовые архитектуры веб-приложений на платформе Java EE: Model 1 и Model 2. Каркасы веб-приложений (Web application frameworks) в основном реализуют архитектуру Model 2.

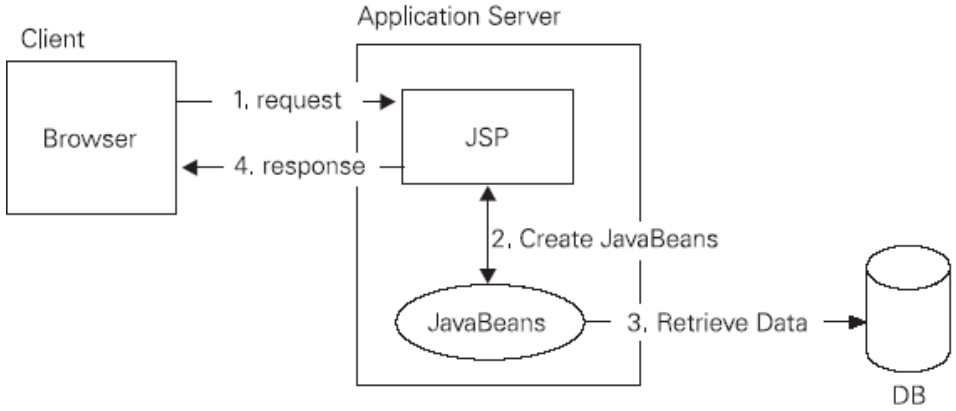


Рис. 14.3. Типовая архитектура Model 1

Архитектура Model 1 (рис. 14.3) состоит из браузера, напрямую обращающегося к JSP-страницам (или сервлетам, генерирующим представление). JSP-страницы обращаются к компонентам JavaBeans, которые представляют модель приложения, а следующее отображаемое представление определяется гиперссылками в исходной странице или параметрами запроса. Управление в приложениях, построенных по Model 1, децентрализовано, так как страница, отображаемая в текущий момент, определяет следующую страницу. Кроме того, JSP-страница самостоятельно обрабатывает параметры запроса.

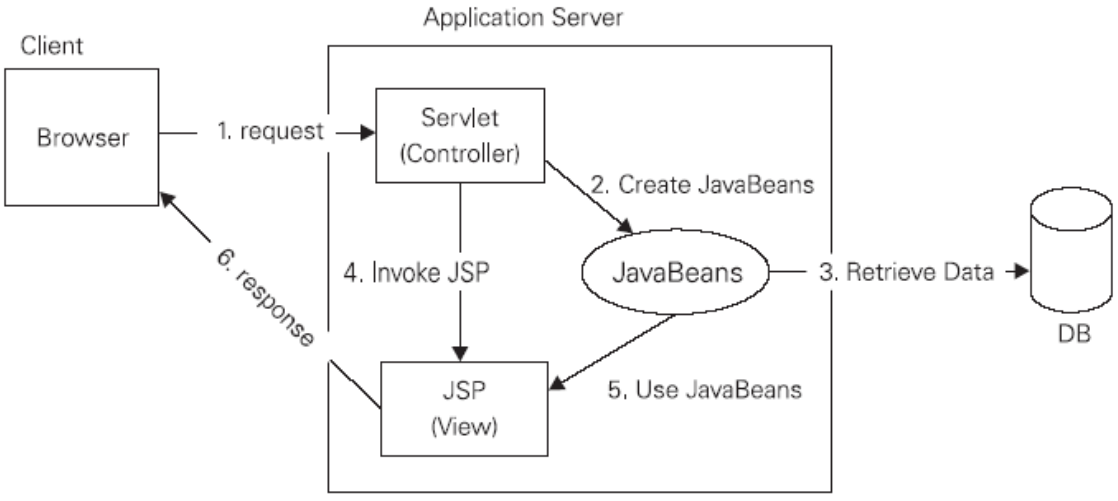


Рис. 14.4. Типовая архитектура Model 2

В архитектуре Model 2 (рис. 14.4) между браузером и JSP-страницами вводится сервлет-контроллер, который направляет запрос на нужное представление на основе URL запроса, входных параметров и состояния приложения. Приложения, построенные согласно Model 2, проще сопровождать и расширять, так как представления не зависят друг от друга напрямую. Сервлет-контроллер обеспечивает единственную точку управления безопасностью и журналированием, и часто преобразует входные данные в форму, используемую моделью MVC-приложения. По этим причинам архитектура Model 2 рекомендуется для большинства интерактивных приложений.

Каркас веб-приложения (Web application framework)

Функции веб-приложения можно разделить на две группы: специфичные для конкретного приложения и используемые во всех веб-приложениях.

У всех веб-приложений есть набор общих базовых требований, которые не выполняются самой платформой Java EE. Данные требования могут быть реализованы в отдельном слое ПО, который называется каркасом веб-приложения (Web application framework) и разделяется между приложениями, построенными на его основе (рис. 14.5).

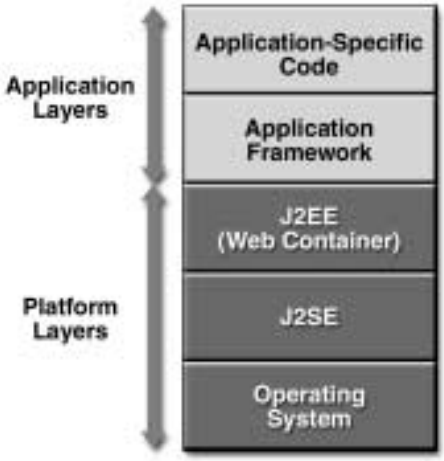


Рис. 14.5. Уровни платформы Java EE и приложения

Каркас веб-приложения обеспечивает функции, общие для всех веб-приложений, такие как определение операции по запросу пользователя, вызов методов модели и выбор отображаемого представления. Разработчики расширяют, используют или реализуют классы и интерфейсы каркаса для выполнения специфичных для приложения функций. Например, каркас может содержать интерфейс, который разработчик может реализовать для выполнения бизнес-логики в ответ на события, возникающие в приложении.

Использование каркаса веб-приложения обладает следующими достоинствами:

- Разделение представления и логики по различным компонентам.
- Специализация труда разработчиков.
- Обеспечение единой точки управления.
- Упрощение тестирования и сопровождения модулей.
- Предоставление широкого набора возможностей, разработанных экспертами в области проектирования веб-приложений.
- Поощрение разработки и использования стандартизованных компонентов – большинство каркасов включает набор определенных программистами действий JSP, упрощающих создание представлений.
- Стабильность базовых классов и интерфейсов каркаса.
- Поддержка сообществом, в том числе исправление ошибок, создание обучающих материалов, консультирование, разработка полезных дополнений.

- Поддержка проверки пользовательского ввода (input validation).

Наибольшую известность и распространение получили следующие каркасы веб-приложений:

- *JavaServer Faces* (JSF) – стандартизованный каркас веб-приложений, его реализация должна иметься в каждом веб-сервере, отвечающего требованиям платформы Java EE 5. Данный каркас обеспечивает обработку запросов JavaBeans-компонентами, относящимися к модели приложения. JSF поддерживает на стороне сервера представление интерфейса пользователя в виде дерева компонентов, запоминающих свое состояние между запросами. Определена библиотека стандартных компонентов интерфейса пользователя и имеется возможность разработки дополнительных компонентов. Данный каркас также обеспечивает проверку пользовательского ввода, поддержку различных типов клиентов и интернационализацию. Представление, как правило, разрабатывается в виде JSP-страниц, использующих дополнительную библиотеку действий JSP для вывода компонентов интерфейса пользователя. Связь между представлением и состоянием приложения обеспечивается с помощью особых выражений языка EL.
- *Apache Struts* – разрабатывается организацией Apache Software Foundation, обладает длинным перечнем возможностей, включая универсальный сервлет-контроллер, классы действий и отображений, вспомогательные классы для работы с XML, автоматическое создание компонентов JavaBeans из параметров запроса, проверку пользовательского ввода и поддержку интернационализации. Также содержит набор действий JSP для создания HTML-форм с доступом к состоянию приложения и выполнения логики представления (условные, циклические действия).
- *Apache Struts 2* (бывший *WebWork 2*) – значительно отличается от Struts в архитектурном плане, обеспечивает упрощение разработки действий для вызова бизнес-логики и большую гибкость конфигурирования процесса обработки запроса. Остальные возможности аналогичны каркасу Struts.
- *Tapestry* – по принятому подходу сильно отличается от остальных каркасов. Разработка веб-приложения в Tapestry напоминает разработку приложения с графическим интерфейсом пользователя – каждая страница представляет собой класс, который содержит объекты-компоненты веб-интерфейса. Помимо класса для компонента (страницы) может быть определен шаблон вывода, использующий одну из следующих технологий: JSP, FreeMarker или Velocity. Как и остальные упомянутые каркасы веб-приложений, поддерживает проверку пользовательского ввода и интернационализацию.

Принципы построения каркаса веб-приложения

Каркас веб-приложения (Web application framework), такой как Apache Struts или JavaServer Faces, может значительно упростить разработку приложения по архитектуре Model 2. Каркас включает конфигурируемый сервлет-контроллер и предоставляет абстрактные классы для обработчиков запросов (вызывающих бизнес-логику).

Важным требованием к контроллеру является возможность и простота его расширения при развитии приложения.

Запрос операции может выполняться различными способами, наиболее общие из которых следующие:

- Операция указывается в скрытом поле формы или в качестве параметра GET-запроса.

- Операция указывается в виде части URL запроса. Сервлету-контроллеру ставятся в соответствие все URL с некоторым расширением или некоторой базой.

Большинство каркасов веб-приложений используют второй подход для направления запросов на свой сервлет-контроллер.

После того, как контроллер определил, какую операцию нужно выполнить, он должен вызвать соответствующий метод модели с параметрами, полученными из запроса. Наиболее просто использовать для этого оператор `if-then-else`, например:

```
if ("zakazAct".equals(action)) {
    int idZakaz = Integer.parseInt(request.getParameter("idZakaz"));
    Zakaz zakaz = dao.getZakazOnId(idZakaz);
    request.setAttribute("zakaz",zakaz);
    pageview="ViewZakazOnId.jsp";
} else if ("zakazchikAct".equals(action)) {
    // и так далее
}
```

Подобный подход увеличивает и усложняет метод `service()` сервлета-контроллера, что ухудшает сопровождаемость приложения.

Использование шаблона Команда (см. врезку) позволяет реализовать универсальный сервлет-контроллер, который обрабатывает любые запросы, даже те, которые не были известны во время создания контроллера. Для каждой операции (действия с моделью) создается свой класс. Контроллер определяет, какой класс операции может обработать запрос пользователя, создает экземпляр этого класса и делегирует ему выполнение запроса. Соответствие между операцией и классом операции определяется конфигурацией контроллера, задаваемой параметрами инициализации сервлета или в конфигурационном файле. В качестве последнего может выступать файл свойств или XML-файл, располагающийся в веб-приложении. Наиболее удобно хранить конфигурационные файлы в папке `WEB-INF`, которая является корневой с точки зрения классов веб-приложения.

После выполнения действия с моделью контроллер должен определить представление, которое будет отображаться клиенту. Данная функция контроллера часто называется навигацией. В качестве представления может выступать JSP-страница, сервлет, статическое содержимое или их комбинация. Как правило, представление выбирается с учетом следующих факторов:

- текущее представление;
- результат операции с моделью;
- состояние веб-приложения (хранящееся в контекстах страницы, запроса, сессии и веб-приложения).

Например, выбор представления после операции входа в систему может определяться следующими факторами:

- текущее представление;
- имя пользователя и пароль, содержащиеся в запросе;
- результат операции входа в систему (успех или неудача);
- максимальное число пользователей (атрибут контекста веб-приложения);
- URL, по которому обращался пользователь (свойство запроса).

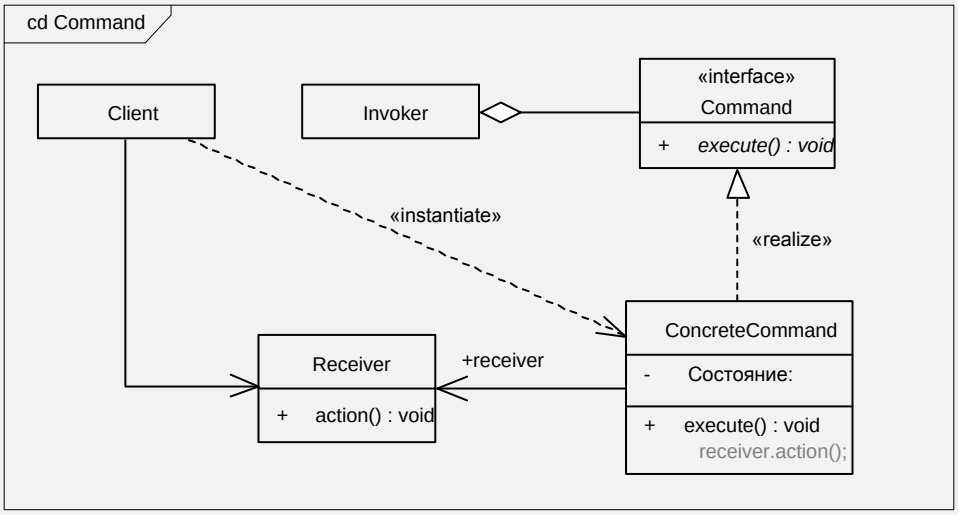
Для отображения представления контроллер перенаправляет запрос на JSP-страницу, сервлет или другой компонент, генерирующий ответ в нужном формате (например, в виде HTML-страницы).

Для конфигурирования выбора представления можно использовать те же средства, что и для описания операций веб-приложения (см. выше), более того, их можно комбинировать в одном конфигурационном файле.

Шаблон Команда (Command) инкапсулирует запрос как объект, позволяя тем самым задавать параметры клиентов для обработки соответствующих запросов, ставить запросы в очередь или протоколировать их, а также поддерживать отмену операций.

Иногда необходимо посылать объектам запросы, не зная о том, выполнение какой операции запрошено и кто является получателем запроса. Например, в библиотеке для построения графического интерфейса пользователя имеются такие элементы, как кнопки и меню, которые посылают запрос в ответ на действие пользователя. Но сама библиотека не способна обработать этот запрос, это может сделать только приложение, использующее библиотеку, так как оно располагает информацией о получателе запроса и об операциях, которые нужно выполнить. Аналогичным примером является каркас веб-приложения, универсальный сервлет-контроллер которого должен направлять запросы на обработку, не владея информацией о выполняемых операциях.

Шаблон Команда позволяет библиотечным объектам отправлять запросы неизвестным объектам приложения, преобразовав сам запрос в объект, который можно хранить и передавать так же, как любой другой объект. В основе шаблона лежит интерфейс `Command` для выполнения операций, который в простейшей своей форме состоит из одного метода `execute()`. Классы, реализующие интерфейс `Command`, определяют пару «получатель-действие», сохраняя получателя (`Receiver`) в переменной экземпляра и посылая к нему запрос в реализации метода `execute()`. У получателя есть информация, необходимая для выполнения запроса.



В описанном виде шаблон Команда применяется в основном при реализации библиотек для построения графического интерфейса пользователя (например, Java Swing API). В каркасах веб-приложений состав участников шаблона и связи между ними могут изменяться, но главный принцип представления запроса как объекта остается.

Apache Struts 2 – пример архитектуры каркаса веб-приложения

В качестве примера каркаса веб-приложения рассмотрим подробнее Apache Struts 2, основные элементы которого и порядок обработки ими запроса представлены на рис. 14.6.

Сначала HTTP-запрос поступает в веб-сервер и проходит через цепочку фильтров, наиболее важным из которых с точки зрения каркаса Struts 2 является `FilterDispatcher`. Данный фильтр обращается к объекту `ActionMapper`, который определяет, приводит ли запрос к вызову действия (операции с моделью). Если это так, то фильтр `FilterDispatcher` передает управление объекту `ActionProxy`, который создает объект

`ActionInvocation` для вызова действия с учетом конфигурации веб-приложения. Объект `ActionInvocation` отвечает за реализацию шаблона Команда.

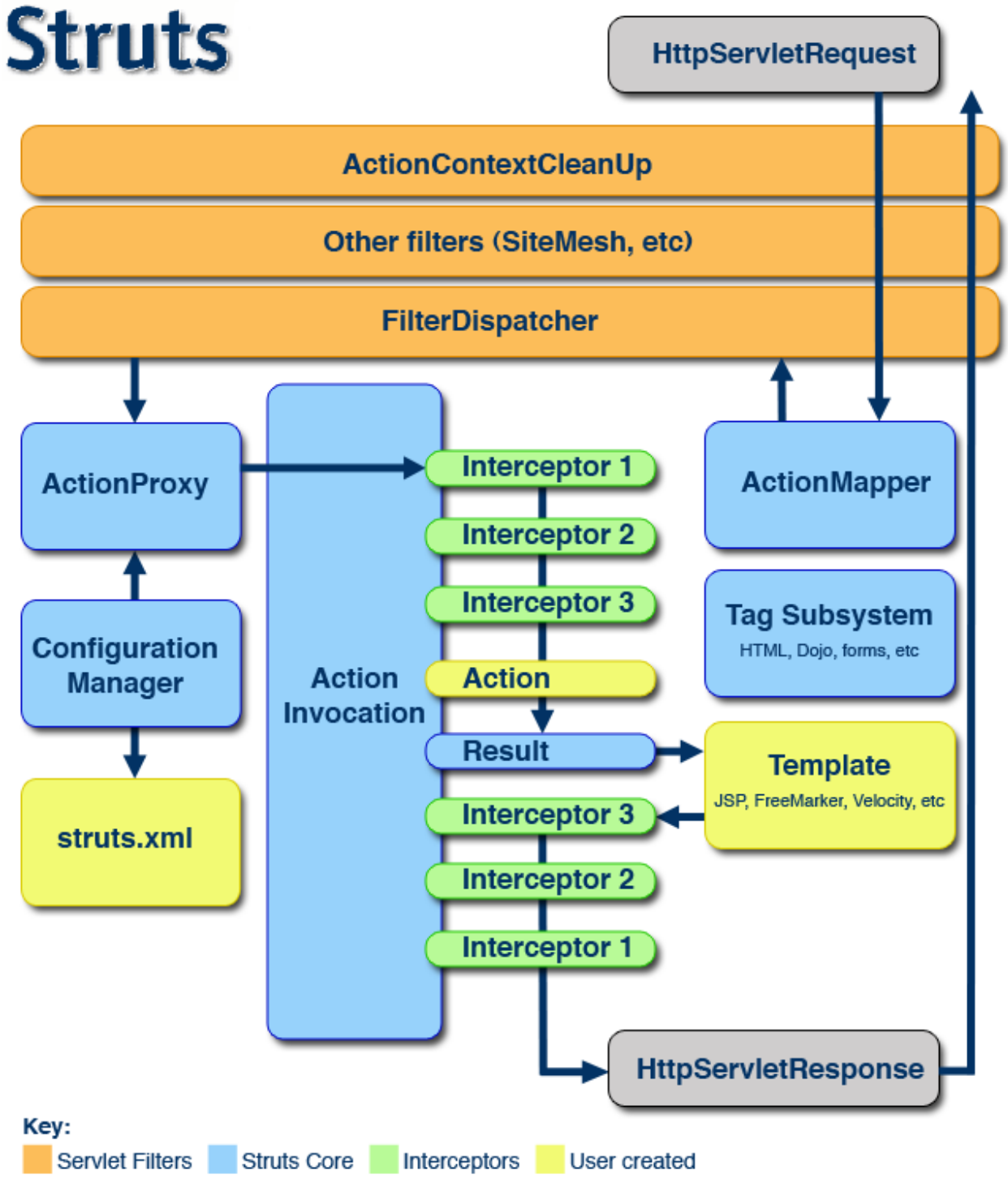


Рис. 14.6. Обработка запроса в каркасе Apache Struts 2

Перед вызовом самого действия объект `ActionInvocation` вызывает сконфигурированные для действия перехватчики (interceptor). В Struts 2 имеется множество предопределенных перехватчиков, которые выполняют предобработку запроса – например, устанавливают параметры действия из параметров запроса и т.п. Перехватчики выстраиваются в цепочки (по аналогии с цепочками фильтров), и в каркасе определена стандартная цепочка, подходящая для обработки большинства запросов.

Класс, выполняющий действие, должен реализовать интерфейс `Action`, в котором определен единственный метод `execute()`, возвращающий строку с кодом результата. Все

необходимые действию параметры устанавливаются перехватчиками, которые определяют параметры автоматически, по наличию set-методов.

С каждым выполнением действия связан контекст действия (объект типа `ActionContext`), который позволяет получить доступ к переменным контекста сессии и приложения, а также параметрам действия. Контекст действия связан с потоком, в котором выполняется действие. Контекст действия не передается объекту действия в качестве параметра, он связан с потоком, в котором выполняется действие, и может быть получен при обработке действия следующим образом: `ActionContext.getContext()`.

После завершения вызова действия (возврата из метода `execute()` объекта действия) объект `ActionInvocation` находит и выполняет объект результата `Result`, соответствующий коду результата согласно конфигурации веб-приложения. Выполнение результата может заключаться, например, в генерации ответа JSP-страницей. В последнем случае можно использовать действия из библиотеки Struts.

После генерации результата он проходит перехватчики в обратном порядке. После этого ответ возвращается клиенту, проходя обратно через цепочку фильтров.

Конфигурация приложения, используемая различными компонентами каркаса Struts 2, хранится в файле `struts.xml`. Рассмотрим пример конфигурирования действия входа в систему:

```
<action name="login" class="simple.LoginAction" >
    <result name="success" type="dispatcher">/pages/welcome.jsp</result>
    <result name="error" type="redirect">/login.jsp</result>
</action>
```

Здесь определено действие с названием `login`, реализуемое классом действия `simple.LoginAction`. Если метод `execute()` объекта данного класса вернет значение `error`, то клиент получит ответ о перенаправлении запроса на страницу `/login.jsp`, предлагающую ввести имя пользователя и пароль. Если метод `execute()` объекта данного класса вернет значение `success`, то для генерации ответа будет использована JSP-страница `/pages/welcome.jsp`.

Само действие входа в систему реализуется следующим классом:

```
public class LoginAction implements Action {

    private String userId;
    private String passwd;

    public String execute() throws Exception {
        if ("admin".equals(userId) && "password".equals(passwd)) {
            Map session = ActionContext.getContext().getSession();
            session.put("loggedin","true");
            return SUCCESS;
        }
        return ERROR;
    }

    public String getPasswd() { return passwd; }

    public void setPasswd(String passwd) { this.passwd = passwd; }

    public String getUserId() { return userId; }

    public void setUserId(String userId) { this.userId = userId; }
}
```

Данное действие имеет два параметра – имя пользователя (свойство `userId`) и пароль (свойство `passwd`). Они устанавливаются автоматически перехватчиком, достаточно в исходной странице (`login.jsp`) определить поля формы с такими же названиями:

```
<form action="login.action" method="post">
User id<input type="text" name="userId" /> <br/>
Password <input type="password" name="passwd" /> <br />
```

```
<input type="submit" value="Login"/>
</form>
```

Каркас Struts 2 обеспечивает только функции контроллера веб-приложения. При необходимости автоматически создавать JavaBeans-компоненты и размещать их в атрибутах различных контекстов (запроса, сессии, веб-приложения) можно воспользоваться каркасом Spring.