

Основные паттерны J2EE

Front Controller

Ситуация

Механизм обработки запросов яруса презентации должен управлять и согласовывать среди множества запросов обработку каждого пользователя. Такие механизмы контроля могут управляться централизованным или нецентрализованным образом.

Задача

Для обработки запросов яруса презентации системе необходимо централизованное место доступа. Это позволяет поддерживать интеграцию системных служб, извлечение контента, управление видом и навигацию. Если пользователь получает доступ к виду напрямую без прохождения через централизованный механизм, то возникают следующие проблемы:

- Необходимо, чтобы каждый вид обеспечивал собственные системные службы, что часто приводит к дублированию кода.
- Навигация вида оставлена для видов. Это может привести к смешанному содержанию вида и навигации вида.

Кроме того, распределенное управление сложнее поддерживать, так как изменения часто приходится делать в разных местах.

Требования

- Обработка общих системных служб выполняется при каждом запросе. Например, служба безопасности выполняет проверки аутентификации и авторизации.
- Логика, которая наилучшим образом обрабатывается в одном центральном размещении, дублируется вместо этого во множестве видов.
- Существование точек принятия решений зависит от извлечения и управления данными.
- Для того чтобы отвечать похожим бизнес запросам, используется множество видов.
- Использование централизованной точки контакта для обработки запроса может оказаться полезным, например, для управления и ведения журнала движения пользователя по сайту.
- Системные службы и логика управления видом являются достаточно сложными.

Решение

Для обработки запроса используйте контроллер в качестве начальной точки контакта. Контроллер управляет обработкой запросов, включая вызов служб безопасности (аутентификация или авторизация), делегирование бизнес обработки, выбор подходящего вида, обработку ошибок, а также выделение стратегий создания контента.

Контроллер обеспечивает централизованную точку входа, которая управляет обработкой Web-запроса. Путем централизации точек принятия решений и механизмов управления контроллер позволяет также уменьшить количество Java кода (*скриплетов*), внедренного в JSP-страницы.

Централизация управления в контроллере и уменьшение бизнес-логики в виде позволяет повторно использовать один код во всех запросах. Внедрение кода в

многочисленные виды менее предпочтительно, так как может привести к большой опасности возникновения ошибок.

Обычно, контроллер согласовывает работу при помощи компонента-диспетчера. Такие диспетчеры отвечают за управление видом и навигацию. Таким образом, диспетчер выбирает следующий вид для пользователя и направляет контроль к ресурсу. Диспетчеры могут быть инкапсулированы в контроллере напрямую или извлечены в отдельный компонент.

Паттерн Front Controller предлагает централизацию обработки всех запросов и, в отличие от Singleton, не ограничивает количество обработчиков в системе. Приложение может использовать в системе множество контроллеров, причем каждый из них отображает на набор отдельных служб.

Структура

На рисунке 7.7 представлена классовая диаграмма паттерна Front Controller.

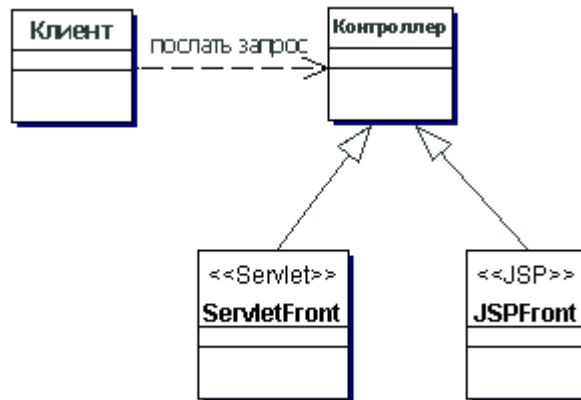


Рисунок 7.7 Классовая диаграмма Front Controller

Участники и обязательства

На рисунке 7.8 представлена циклограмма паттерна Front Controller. На ней схематично отображен процесс обработки запроса контроллером.

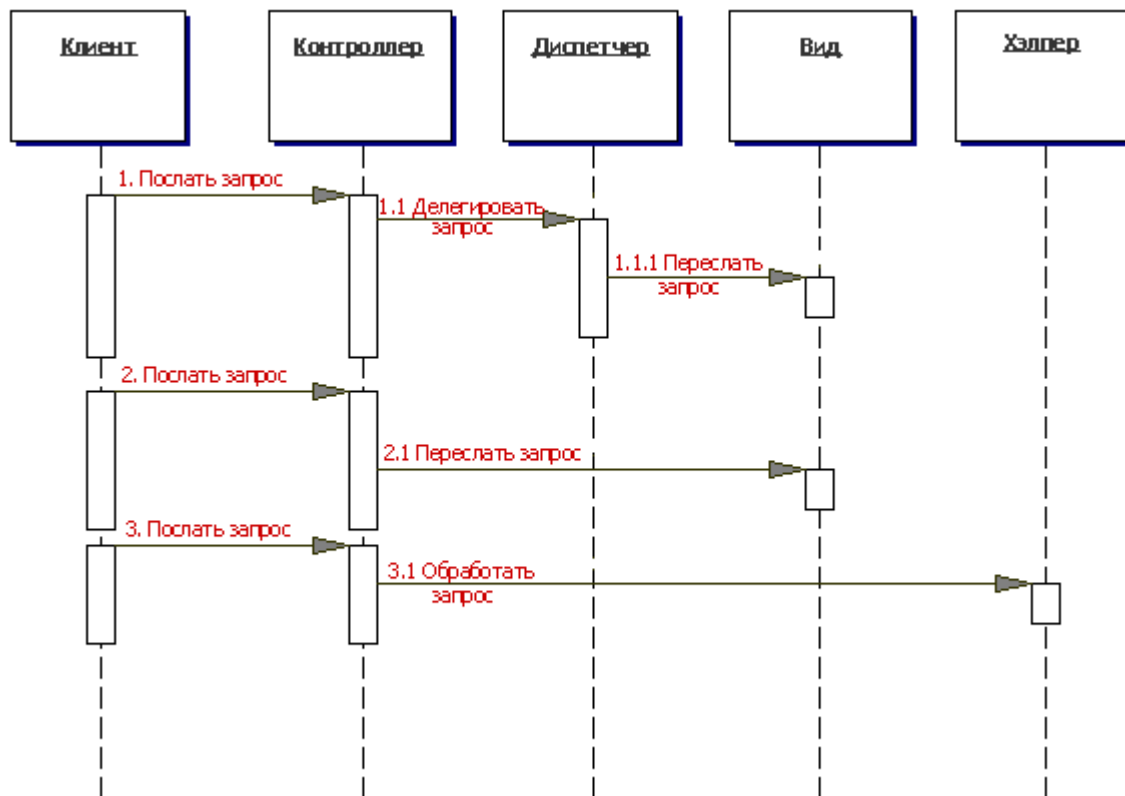


Рисунок 7.8 Циклограмма паттерна Front Controller

Контроллер

Контроллер является начальной точкой контакта для обработки запросов в системе. Для выполнения аутентификации и авторизации пользователя или для инициирования извлечения контакта контроллер может передавать полномочия хелперу.

Диспетчер

Диспетчер отвечает за управление видом и навигацию, выбор следующего вида, который необходимо отобразить пользователю, и обеспечение механизма для направления контроля к данному ресурсу.

Диспетчер может быть инкапсулирован в контроллере или являться отдельным согласованным компонентом. Он обеспечивает либо статическую координацию для вида, либо более сложный динамический механизм координирования.

Диспетчер использует объект RequestDispatcher (поддерживаемый спецификацией сервлета) и инкапсулирует некоторую дополнительную обработку.

Хелпер (helper)

Хелпер отвечает за содействие в выполнении обработки вида или контроллера. Таким образом, хелперы выполняют несколько функций, куда входит сбор данных, необходимых для вида и сохранение промежуточной модели. В этом случае на хелпер иногда ссылаются, как на компонент-значение. Кроме этого, хелперы могут адаптировать модель данных для использования ее в виде. Хелперы могут обслуживать запросы данных из вида путем обычного предоставления доступа к необработанным данным или путем форматирования данных как Web-содержимого.

Вид может взаимодействовать с любым количеством хелперов, которые обычно реализованы как JavaBean-компоненты (JSP 1.0+) и заказные тэги (JSP 1.1+). Кроме того, хелпер может отображать объект Command, delegate (см. главу «Business

Delegate») или XSL преобразователь. Последний для приведения модели к подходящему виду используются в комбинации с таблицей стилей.

Вид

Вид формулирует и отображает клиенту информацию, извлекаемую из модели. Хелперы поддерживают виды путем инкапсуляции и адаптации модели для использования ее в отображении.

Стратегии

Для реализации контроллера существует несколько стратегий.

Стратегия Servlet Front

Данная стратегия предлагает реализовать контроллер как сервлет. Не смотря на семантическую эквивалентность, такая стратегия более предпочтительна, чем стратегия JSP Front. Контроллер управляет теми аспектами обработки запроса, которые связаны с обработкой деловой информации и управляющей логикой. Данные обязательства имеют отношение к форматированию отображения, однако логически от него не зависят. Кроме того, более правильно обязательства инкапсулированы в сервлете, чем в JSP-странице.

У стратегии Servlet Front есть несколько потенциальных недостатков. В частности, она не использует все возможности некоторых утилит рабочей среды JSP, например, автоматического перенос параметров запроса в свойства хелпера. К счастью, данный недостаток минимален, так как создать или получить похожие утилиты для общего использования относительно легко. Кроме этого существует вероятность того, что функциональные возможности некоторых JSP-утилит смогут полностью быть реализованы в будущей версии спецификации сервлета. В примере 7.14 представлена стратегия Servlet Front.

Пример 7.14 Пример кода стратегии Servlet Front

```
public class EmployeeController extends HttpServlet {
    // инициализация сервлета
    public void init(ServletConfig config) throws
        ServletException {
        super.init(config);
    }

    // разрушение сервлета
    public void destroy() {
    }

    /** Обработка запросов для обоих HTTP
     * методов <code>GET</code> и <code>POST</code>.
     * @param запрос, ответ сервлета
     * @param ответ, ответ сервлета
     */
    protected void processRequest (HttpServletRequest
        request, HttpServletResponse response)
        throws ServletException, java.io.IOException {
        String page;

        /** ApplicationResources обеспечивает простой API
         * для извлечения констант и других
         * предустановленных значений**/
        ApplicationResources resource =
            ApplicationResources.getInstance();
        try {

            // Используйте объект helper для
            // сбора информации о параметре
            RequestHelper helper = new
                RequestHelper (request);

            Command cmdHelper= helper.getCommand();
```

```

        // Укажите хелперу выполнить заказную операцию
        page = cmdHelper.execute(request, response);

    }
    catch (Exception e) {
        LogManager.logMessage(
            "EmployeeController:exception : " +
            e.getMessage());
        request.setAttribute(resource.getMessageAttr(),
            "Exception occurred : " + e.getMessage());
        page = resource.getErrorPage(e);
    }
    // передайте управление виду
    dispatch(request, response, page);
}

/** Обрабатывает HTTP метод <code>GET</code>.
 * @param запрос, запрос сервлета
 * @param ответ, ответ сервлета
 */
protected void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, java.io.IOException {
    processRequest(request, response);
}

/** Обрабатывает HTTP метод <code>POST</code>.
 * @param запрос, запрос сервлета
 * @param ответ, ответ сервлета
 */
protected void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, java.io.IOException {
    processRequest(request, response);
}

/** Возвращает короткое описание сервлета */
public String getServletInfo() {
    return "Front Controller Pattern" +
        " Servlet Front Strategy Example";
}

protected void dispatch(HttpServletRequest request,
    HttpServletResponse response,
    String page)
    throws javax.servlet.ServletException,
    java.io.IOException {
    RequestDispatcher dispatcher =
        getServletContext().getRequestDispatcher(page);
    dispatcher.forward(request, response);
}
}

```

Стратегия JSP Front

В данной стратегии предложено реализовать контроллер как JSP-страницу. Не смотря на их семантическую эквивалентность, стратегия Servlet Front все же предпочтительней, чем стратегия JSP Front. В виду того, что контроллер управляет обработкой, не связанной конкретно с форматированием отображения, реализовывать данный компонент как JSP-страницу не рекомендуется.

Существует также и другая причина, из которой очевидно, почему лучше не реализовывать контроллер как JSP-страницу. Такая реализация требует участия разработчика программного обеспечения для работы со страницей разметки с целью изменения логики обработки запроса. Таким образом, наиболее вероятно, что разработчики программного обеспечения после написания кода, его компиляции, тестирования и отладки посчитают стратегию JSP Front более громоздкой. В примере 7.15 рассмотрен программный код с использованием стратегии JSP Front.

Пример 7.15 Пример кода стратегии Front Strategy

```

<%@page contentType="text/html"%>
<%@ page import="corepatterns.util.*" %>
<html>
<head><title>JSP Front Controller</title></head>
<body>

<h3><center> Employee Profile </h3>

<%
/** Здесь должна быть управляющая логика...
В определенном месте данного блока кода мы извлекаем
информацию о сотруднике, инкапсулируем ее в объекте-
значение и помещаем данный компонент в область действия
запроса с ключом "employee". Этот код был опущен.

В данной точке мы либо направляем к другой JSP-странице, либо
просто позволяем выполняться оставшейся части кода скриплетта. **/
%>
<jsp:useBean id="employee" scope="request"
class="corepatterns.util.EmployeeTO"/>
<FORM method=POST >
<table width="60%">
<tr>
<td> First Name : </td>
<td> <input type="text"
name="<%=Constants.FLD_FIRSTNAME%>"
value="<jsp:getProperty name="employee"
property="firstName"/>"> </td>
</tr>
<tr>
<td> Last Name : </td>
<td> <input type="text"
name="<%=Constants.FLD_LASTNAME%>"
value="<jsp:getProperty name="employee"
property="lastName"/>"></td>
</tr>
<tr>
<td> Employee ID : </td>
<td> <input type="text"
name="<%=Constants.FLD_EMPID%>"
value="<jsp:getProperty name="employee"
property="id"/>"> </td>
</tr>
<tr>
<td> <input type="submit"
name="employee_profile"> </td>
<td> </td>
</tr>
</table>
</FORM>

</body>
</html>

```

Стратегия Command and Controller

Стратегия Command and Controller базируется на паттерне Command [GoF] и предлагает обеспечить общий интерфейс для компонентов хелпера. Контроллер может передать им определенные обязанности, минимизируя при этом связи между данными компонентами. Для получения дополнительной информации по вспомогательным компонентам обратитесь к главе «View Helper». Добавление или изменение работы, которую требуется выполнить при помощи этих хелперов, не требует каких-либо изменений в интерфейсе между контроллером и хелперами. Здесь, скорее всего, потребуется изменить тип и/или содержимое команд. Эти действия обеспечивают гибкий и легко расширяемый механизм, позволяющий разработчикам добавлять режимы обработки запросов.

И, наконец, из-за того, что обработка команды не связана с ее вызовом, механизм обработки может быть повторно использован для различных типов клиентов (не только для Web-браузеров). Данная стратегия облегчает также создание составных команд (смотрите паттерн Composite). Смотрите код в примере 7.16 и циклограмму на рисунке 7.9.

Пример 7.16 Пример кода стратегии Command and Controller

```
/** Метод processRequest вызывается из методов
 * сервлета doGet и doPost**/
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, java.io.IOException {

String resultPage;
try {
RequestHelper helper = new RequestHelper(request);

/** Метод getCommand() использует фэктори внутренним образом
для извлечения командных объектов:
Command command = CommandFactory.create(
request.getParameter("op"));
**/
Command command = helper.getCommand();

// передать объект хелперу командного объекта
resultPage = command.execute(request, response);
}
catch (Exception e) {
LogManager.logMessage("EmployeeController",
e.getMessage() );
resultPage = ApplicationResources.getInstance().
getErrorPage(e);
}

dispatch(request, response, resultPage);
}
```

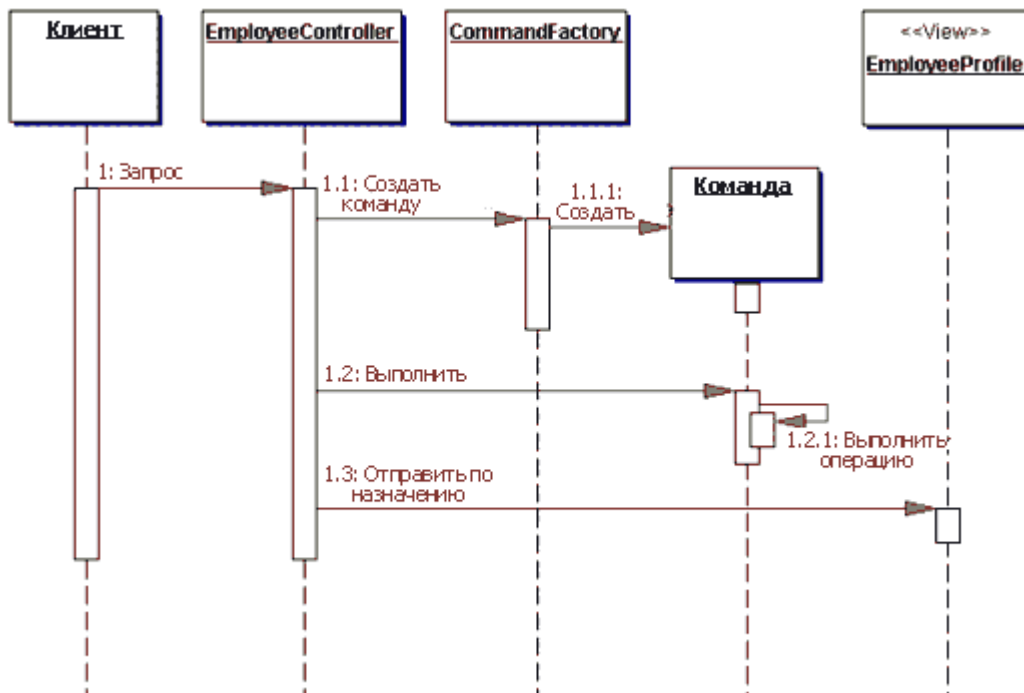


Рисунок 7.9 Циклограмма стратегии Command and Controller

Стратегия Physical Resource Mapping

Все запросы осуществляются не к логическим именам, а к конкретным именам физических ресурсов. Возьмем, к примеру, следующий URL-адрес: <http://some.server.com/resource1.jsp>. В случае использования контроллера примером URL может служить `http://some.server.com/ servlet/Controller`. Стратегия Logical Resource Mapping обычно имеет большие преимущества, так как она обеспечивает гораздо большую гибкость. Стратегия Logical Resource Mapping позволяет

модифицировать отображения ресурсов через декларации в файле конфигурации. Это дает гораздо большую гибкость, чем при использовании стратегии Physical Resource Mapping, требующей от вас изменять при необходимости каждый ресурс.

Стратегия Logical Resource Mapping

Запросы осуществляются не к конкретным именам физических ресурсов, а к логическим именам. Физические ресурсы, на которые ссылаются данные логические имена, могут изменяться через декларации.

Например, URL `http://some.server.com/process` может быть отображен следующим образом:

```
process=resource1.jsp
OR
process=resource2.jsp
OR
process=servletController
```

Стратегия Multiplexed Resource Mapping

Фактически она является подстратегией Logical Resource Naming. Данная стратегия отображает на физический ресурс не просто отдельное логическое имя, а целый набор логических имен. Например, использование группового символа позволяет отображать все запросы, оканчивающиеся на `ctrl`, на конкретный обработчик.

В таблице 7-1 показано, как могут выглядеть запрос и отображение.

Таблица 7-1

Запрос	Отображение
<code>http://some.server.com/action.ctrl</code>	<code>*.ctrl = servletController</code>

Данную стратегию используют движки JSP-страницы для гарантии того, что запросы ресурсов JSP-страниц (то есть ресурсы, чьи имена оканчиваются на `.jsp`) обрабатываются конкретным обработчиком.

К запросу может быть добавлена дополнительная информация, обеспечивающая детали для выполнения логического отображения. Смотрите таблицу 7-2.

Таблица 7-2

Запрос	Отображение
<code>http://some.server.com/profile.ctrl?usecase= create</code>	<code>*.ctrl = servletController</code>

Ключевая выгода от использования данной стратегии состоит в том, что такая стратегия обеспечивает очень большую гибкость при создании компонентов, обрабатывающих запрос. В сочетании с такими стратегиями, как Command and Controller, можно создавать мощный каркас обработки запросов.

Рассмотрим контроллер, обрабатывающий все запросы, оканчивающиеся на `ctrl`, как было описано ранее. Будем считать левую часть имени ресурса, отделенную точкой (в вышеприведенном примере это `profile`) частью имени `use case`. Теперь соединим это имя со значением параметра запроса (в нашем случае `create`). Тем самым мы сигнализируем обработчику запросов о том, что требуется обработать `use case` с названием `create profile`. Сложное отображение ресурса отправляет запрос `servletController`, который является частью отображения из таблицы 7-2. Как описано в стратегии Command and Controller, контроллер создает соответствующий командный объект. Как контроллер узнает, какому командному объекту он должен передать полномочия? Используя дополнительную информацию в URI запроса, контроллер передает полномочия командному объекту, отвечающему за создание профайла.

Объектом, который обслуживает запросы для создания и изменения Profile, может быть ProfileCommand или более специальный объект ProfileCreationCommand.

Стратегия Dispatcher in Controller

Если выполняемые функции диспетчера минимальны, то он может быть свернут в контроллере, как показано на рисунке 7.10.

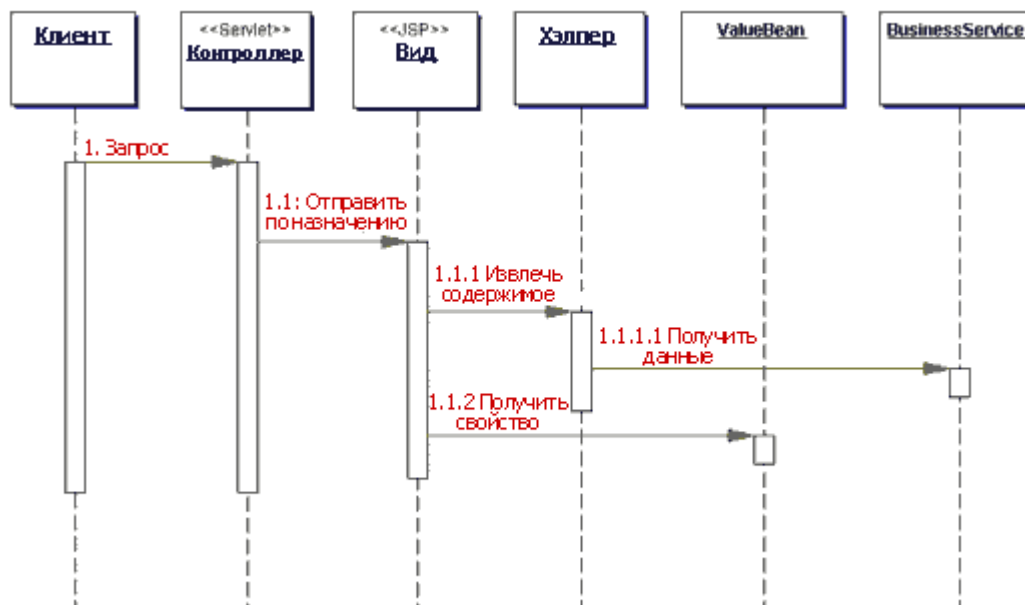


Рисунок 7.10 Циклограмма Dispatcher in the Controller

Стратегия Base Front

В сочетании с Servlet Front данная стратегия предлагает реализовать базовый класс контроллера, реализацию которого могут расширять и другие контроллеры. Base Front может содержать общие реализации и реализации, установленные по умолчанию, а каждый подкласс может переопределять данные реализации. Недостатком данной стратегии является то, что любой общий суперкласс при поддержке повторного использования и распределения способствует созданию слабой иерархии. В такой иерархии изменения, необходимые для одного подкласса, оказывают воздействие на все подклассы.

Стратегия Filter Controller

Для централизованного управления обработкой запросов (смотрите паттерн Intercepting Filter) фильтры обеспечивают похожую поддержку. Таким образом, некоторые аспекты контроллера могут быть в разумных пределах реализованы как фильтр. В то же время фильтры, прежде всего, фокусируются на перехвате и оформлении запроса, а не на его обработке и генерировании ответа. При надлежащем использовании компоненты могут дополнять друг друга, так как у них существуют общие обязанности (например, управление входом в систему или отладка).

Результаты

- **Централизация управления**
Для управления системными службами и бизнес логикой в многочисленных запросах контроллер обеспечивает централизованное пространство. Он

управляет обработкой бизнес логики и запросов. Централизованный доступ к приложению подразумевает, что запросы легко отслеживаются, и для них ведется журнал записей. Помните, что при использовании централизованного управления можно ввести единую точку ошибки. На практике это редко встречается, так как составные контроллеры обычно размещают в отдельном сервере или в кластере.

- **Усовершенствование управления безопасностью**

Контроллер централизует управление путем обеспечения точки заглушки для незаконных попыток получения доступа к Web-приложению. Кроме того, проверка входа в приложение требует меньших затрат, чем проверка безопасности на всех страницах.

- **Улучшение возможности повторного использования**

Контроллер способствует более чистому разделению программ и поддержке повторного использования. Общий для компонентов код помещается в контроллер или управляется им.

Родственные паттерны

- **View Helper**

В сочетании с View Helper паттерн Front Controller описывает вынос бизнес логики из вида и обеспечение центральной точки управления и отправки. Поток логики вносится в контроллер, а код обработки данных возвращается *обратно* к хелперам.

- **Intercepting Filter**

Паттерны Intercepting Filter и Front Controller описывают способы централизации управления конкретными типами обработки запросов, предлагая для этого различные подходы.

- **Dispatcher View and Service to Worker**

Паттерны Dispatcher View и Service to Worker являются другим названием комбинации паттерна View Helper с диспетчером и паттерном Front Controller. Не смотря на то, что Dispatcher View и Service to Worker имеют одинаковую структуру, они описывают различное разделение труда среди компонентов.