# Основные паттерны J2EE
# Service to Worker

## Ситуация

Система контролирует поток выполнения и доступ к бизнес-данным из которых создается содержимое презентации.

**Примечание**

Паттерн Service to Worker, как и паттерн Dispatcher View, описывает общую комбинацию других паттернов каталога. Оба этих макро-паттерна описывают комбинацию контроллера и диспетчера с видами и хелперами. Описывая общую структуру, они придают особое значение паттернам, которые связаны, однако имеют разное применение.

## Задача

Задача является комбинацией других задач, решаемых с помощью паттернов Front Controller и View Helper в ярусе презентации. Здесь не существует централизованного компонента для управления контролем доступа, извлечением содержимого или управления видом, однако управляющий код повторяется в различных видах. Кроме этого, бизнес-логика и логика форматирования презентации объединяются внутри видов, что делает систему менее гибкой и менее подходящей для повторного использования.

Совмещение бизнес-логики с обработкой видов также снижает модульность и обеспечивает плохое разделение ролей между командами Web-разработчиков и разработчиков программного обеспечения.

## Требования

- Необходимо, чтобы проверки авторизации и аутентификации были выполнены относительно каждого запроса.
- Код скриптлета в схемах представления должен быть минимизирован.
- Бизнес-логика должна быть инкапсулирована в компонентах, отличных от вида.
- Управляющая логика является относительно сложной, она основывается на значениях, полученных из динамического контента.
- Логика управления вида является относительно сложной, с множеством видов, потенциально отображаемых на один и тот же запрос.

## Решение

**Для обработки клиентских запросов и подготовки динамической презентации в качестве ответа, объедините контроллер и диспетчер с видами и хелперами (см. главы «Front Controller» и «View Helper»). Контроллеры передают полномочия извлечения контента хелперам, которые управляют наполнением промежуточной модели для вида. Диспетчер несет ответственность за управление видом и навигацию, он может также быть инкапсулирован в контроллер или отдельный компонент.**

Паттерн Service to Worker описывает комбинацию паттернов Front Controller и View Helper с компонентом dispatcher.

Данный паттерн и паттерн Dispatcher View описывают похожую структуру, однако он оба предлагают разное разделение труда между компонентами. В паттерне Service to Worker у контроллера и диспетчера больше обязанностей.

Паттерны Service to Worker и Dispatcher View представляют общую комбинацию других паттернов из каталога, поэтому каждый из них для поддержки эффективной связи между разработчиками гарантирует использование собственного имени. В отличие от паттерна Service to Worker, паттерн Dispatcher View предлагает задержку в извлечении контента на время обработки вида.

В паттерне Dispatcher View диспетчер обычно играет ограниченную или умеренную роль в управлении видом. В паттерне Service to Worker диспетчер играет в управлении видом умеренную или большую роль.

Ограниченная роль диспетчера проявляется в том случае, когда для выбора вида не используются никакие внешние ресурсы. Информация, инкапсулированная в запросе, является достаточной чтобы определить вид для передачи запроса. К примеру:

```
http://some.server.com/servlet/Controller?next=login.jsp
```

Единственное, за что в данном случае отвечает компонент dispatcher – это отправка к виду `login.jsp`.

Примером того, как диспетчер играет умеренную роль, является случай, когда клиент напрямую предоставляет контроллеру запрос с параметром, описывающим выполняемое действие:

```
http://some.server.com/servlet/Controller?action=login
```

Обязанностью компонента dispatcher в данном случае является приведение логического имени `login` к соответствующему виду имени ресурса (в данном случае `login.jsp`), а также отправка к этому виду. Для выполнения данного преобразования диспетчер может получать доступ к таким ресурсам, как файл конфигурации XML, определяющий вид необходимый для отображения.

С другой стороны, в паттерне Service to Worker диспетчер может быть более сложным. Для определения вида необходимого для отображения он может вызывать бизнес-службу.

Общая структура паттернов Service to Worker и Dispatcher View состоит из контроллера, взаимодействующего с диспетчером, видами и хелперами.

СтрукураНа рисунке 7.20 представлена классовая диаграмма паттерна Service to Worker.

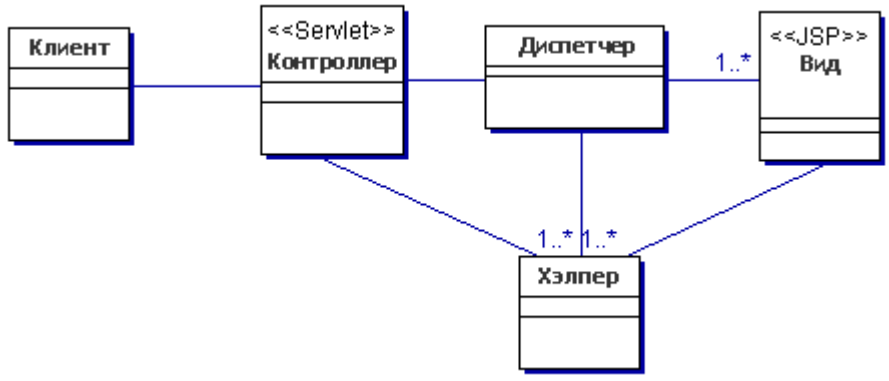


**Рисунок 7.20 Классовая диаграмма паттерна Service to Worker**

## Участники и обязательства

На рисунке 7.21 представлена циклограмма паттерна Service to Worker.

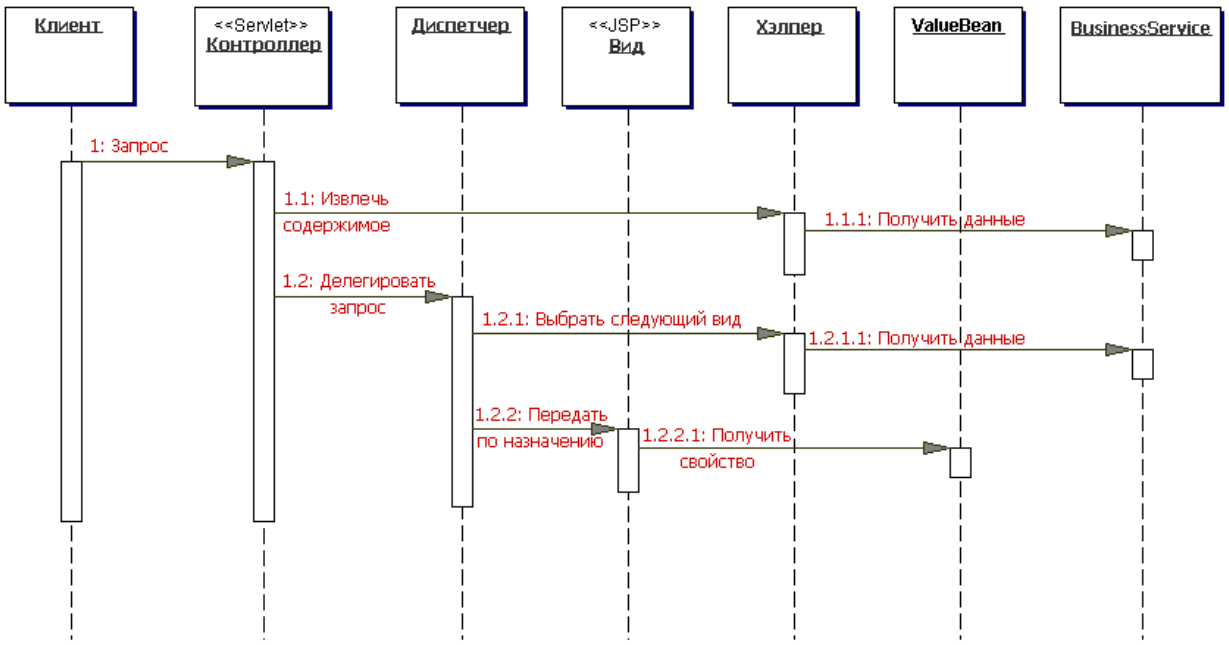


**Рисунок 7.21 Циклограмма паттерна Service to Worker**

Как упоминалось ранее, паттерны Service to Worker и Dispatcher View имеют похожую структуру. Основным их различием является то, что Service to Worker описывает архитектуру, в которой работа контроллера и диспетчера выполняется в начале, тогда как Dispatcher View описывает архитектуру, где эта работа откладывается на время обработки вида. Таким образом, два паттерна предлагают континуум (сплошную среду), в котором поведение либо инкапсулируется ближе к началу, либо переносится в конец потока обработки.

### Контроллер

Контроллер является начальной точкой контакта для обработки запроса. Для управления видом и навигации он взаимодействует с диспетчером. Контроллер управляет аутентификацией, авторизацией, извлечением контента, проверкой и другими аспектами обработки запроса. Для выполнения отдельных частей работы он передает полномочия хелперам.

### 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 (см. главу «Front Controller») или являться отдельным компонентом, согласованно взаимодействующим с контроллером. Он обеспечивает либо статическую координацию для вида, либо более сложный динамический механизм координирования.

Диспетчер использует объект RequestDispatcher (поддерживаемый спецификацией сервлета), однако кроме этого обычно инкапсулирует некую дополнительную обработку. Чем больше функций инкапсулирует данный компонент, тем больше он подходит для паттерна Service to Worker. И наоборот, когда диспетчер играет более ограниченную роль, он более подходит паттерну Dispatcher View.

### Вид (view)

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

### Хелпер

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»).

### ValueBean

ValueBean – это другое название хелпера, отвечающего за удерживание состояния промежуточной модели для использования ее видом. Чаще всего, как показано в циклограмме на рисунке 7.12, бизнес-служба возвращает компонент-значение (value bean) в ответ на запрос. В этом случае ValueBean выполняет роль Transfer Object (см. главу «Transfer Object»).

### BusinessService

BusinessService является ролью, выполняемой службой, к которой клиент стремится получить доступ. Обычно доступ к бизнес-службе (business service) организован через Business Delegate. Ролью business delegate является предоставление управления бизнес-службе и ее защита (см. главу «Business Delegate»).

## Стратегии

### Стратегия Servlet Front

См. главу «Servlet Front Strategy».

### Стратегия JavaServer Pages Front

См. главу «JSP Front Strategy».

### Стратегия JSP View

См. главу «JSP View Strategy».

### Стратегия Servlet View

См. главу «Servlet View Strategy».

### Методика JavaBean Helper

См. главу «JavaBean Helper Strategy».

### Стратегия Custom Tag Helper

См. главу «Custom Tag Helper Strategy».

### Стратегия Dispatcher in Controller

См. главу «Dispatcher in Controller Strategy».

Как уже упоминалось, паттерны Service to Worker и Dispatcher View предлагают континуум, в котором поведение инкапсулируется ближе к началу или переносится дальше к концу потока обработки. На рисунке 7.22 описан сценарий, в котором

контроллер полностью загружен предварительной работой, однако функции, выполняемые диспетчером, являются минимальными.

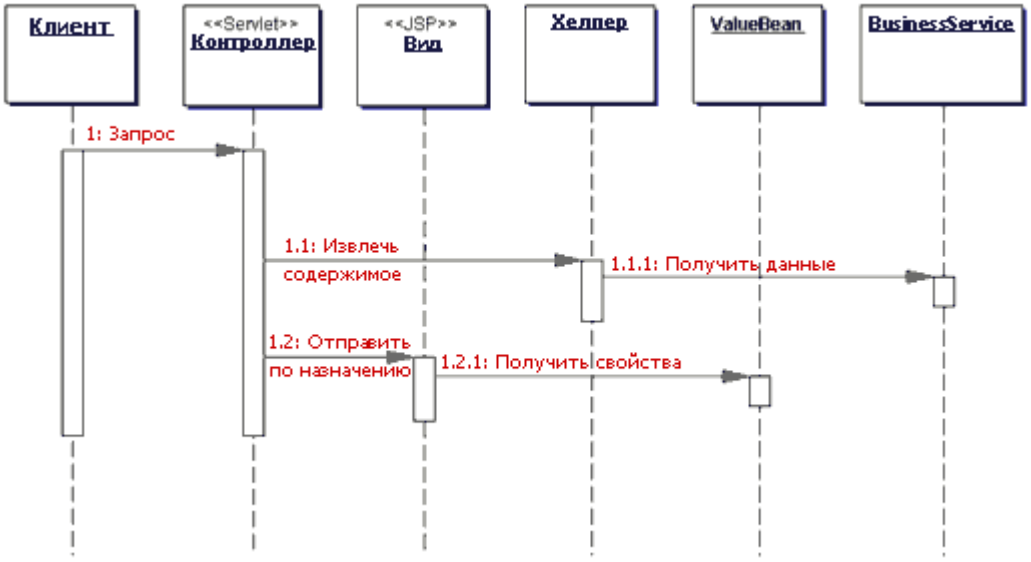


**Рисунок 7.22 Свертывание диспетчера в контроллер**

### Стратегия Transformer Helper

См. главу «Transformer Helper Strategy».

## Результаты

- **Централизация управления, улучшение модульности и возможностей повторного использования**
  Данный паттерн обеспечивает централизованное пространство для обработки системных служб и бизнес-логики во всех запросах. Контроллер управляет обработкой бизнес-логики и запросов. Помните, что при использовании централизованного управления можно ввести единую точку ошибки.

  Паттерн также способствует чистому разделению программ и поддержке повторного использования. Общий код перенесен в контроллер, где используется при обработке каждого запроса, а также перенесен в компоненты хелпера, которым контроллеры и виды передают свои полномочия. Усовершенствованные модульность и возможность повторного использования приводят к меньшему копированию, из-за чего в окружении возникает меньше ошибок.

- **Улучшение разделения программ**
  Использование хелперов приводит к более чистому отделению вида от бизнес-обработки в приложении. Хелперы в форме JavaBean-компонентов (JSP 1.0+) и заказных тэгов (JSP 1.1+) обеспечивают пространство для бизнес-логики, которая будет выноситься за пределы JSP-страницы. Если бизнес-логика содержится в JSP-странице, то в случае больших проектов код скриптлета становится более громоздким.
- **Улучшение разделения ролей**
  Отделение логики форматирования от бизнес-логики приложения уменьшает также зависимости разработчиков, выполняющих разные роли, от одних и тех же ресурсов. К примеру, без подобного разделения разработчик программы должен встраивать собственный код в разметку HTML. При этом члену группы Web-производства необходимо модифицировать разметку страницы и проектировать компоненты, пересекающиеся с бизнес-логикой. Так как ни один разработчик,

выполняющий эти роли, не знаком со спецификой работы другого, увеличивается вероятность случайных изменений, что приводит к ошибкам в системе.

## Пример кода

В следующем примере кода представлена реализация паттерна Service to Worker, использующая сервлет controller, хелпер command, компонент dispatcher, а также вид. Реализация включает в себя стратегии Servlet Front, Command and Controller и JSP View, а также стратегию JavaBean Helper. Кроме этого используется очень простой составной вид. Скриншот отображаемого результата представлен на рисунке 7.23.

В примере 7.29 представлен сервлет controller, который для выполнения обработки управления передает полномочия объекту Command (стратегия Command and Controller). Объект Command извлекается посредством вызова фэктори, который возвращает общий тип Command, интерфейс, представленный в примере 7.30. В коде примера для ведения журнала сообщений используется LogManager. В качестве примера на рисунках 7.23 и 7.28 представлены скриншоты этих сообщений (отображены в нижней части страницы).

**Пример 7.29 Controller Servlet с использованием стратегии Command and Controller**

```
public class Controller extends HttpServlet {
/** Обработка запросов для обоих HTTP-методов
   * <code>GET</code> и <code>POST</code>.
   * @param запрос, запрос сервлета
   * @param ответ, ответ сервлета
   */
  protected void processRequest(HttpServletRequest
    request, HttpServletResponse response)
    throws ServletException, java.io.IOException {
    String next;
    try {
      // Информация из журнала паттерна
      LogManager.recordStrategy(request,
        "Service To Worker",
              " ServletFront Strategy;" +
      " JSPView Strategy; JavaBean helper Strategy");

      LogManager.logMessage(request, getSignature(),
        "Process incoming request. ");

      // Для получения информации о параметре
      // используйте объект helper.
      RequestHelper helper = new
        RequestHelper(request, response);

      LogManager.logMessage(request, getSignature(),
          "Getting command object helper");

     // Получить хелпер объекта command
      Command command = helper.getCommand();
      // передать обработку объекту command,
      // передать туда же объекты запроса и ответа
      next = command.execute(helper);


     /** Если вышеуказанная команда возвращает значение,
        *  будем отталкиваться от контроллера. Между тем,
        * в данном примере command будет использовать
        * отдельный компонент dispatcher для выбора
        * вида и для направления в него. Объект command
        * передает полномочия этому компоненту
        * dispatcher в его методе выполнения, указанном выше, а
        * в этом месте управление вводиться не должно **/
    }
    catch (Exception e) {
      LogManager.logMessage(
        "EmployeeController(CommandStrategy)",
        e.getMessage() );
```

```
     /** ApplicationResources обеспечивает простой API
        * для извлечения констант и других
        * предустановленных значений **/

      next = ApplicationResources.getInstance().
                    getErrorPage(e);
    }

    dispatch(request, response, next);

  }

  /** Обработка HTTP-метода <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);
  }

  /** Обработка HTTP-метода <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);
  }

  /** Возвратить короткое описание сервлета. */
  public String getServletInfo() {
    return getSignature();
  }

  /** метод dispatcher */
  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);
  }

  public void init(ServletConfig config) throws
      ServletException {
    super.init(config);
  }

  public void destroy() { }

  private String getSignature() {
    return "ServiceToWorker-Controller";
  }
}
```

**Пример 7.30 Интерфейс Command**

```
public interface Command {

    public String execute(RequestHelper helper) throws
  javax.servlet.ServletException, java.io.IOException;
}
```

Каждый хелпер Command Object реализует данный общий интерфейс, являющийся примером паттерна GoF Command. Объект Command является экземпляром класса ViewAccountDetails (показано в примере 7.31). Экземпляр интерфейса command передает полномочия AccountingAdapter для вызова бизнес яруса через business

delegate. Класс adapter представлен в примере 7.32. Он использует отдельный компонент dispatcher для определения следующего вида, которому должно передаваться управление и, конкретно, для передачи данному виду.

### Пример 7.30 Интерфейс Command

```
public interface Command {

    public String execute(RequestHelper helper) throws
  javax.servlet.ServletException, java.io.IOException;
}
```

### Пример 7.31 ViewAccountDetailsCommand

```
public class ViewAccountDetailsCommand implements
  Command {
  public ViewAccountDetailsCommand() { }

  // операция view account details
  public String execute(RequestHelper helper)
    throws javax.servlet.ServletException,
  java.io.IOException {
   /** Сообщение пользователю о происшедшей
      * системной ошибке (обычно не отображается).
      * Оно должно сохраняться в файле-источнике**/
    String systemerror =
      "/jspdefaultprocessingerror.jsp";

    LogManager.logMessage(helper.getRequest(),
      "ViewAccountDetailsCommand",
      "Get Account Details from an adapter object");

   /** Для получения данных из бизнес-службы
      * и сохранения их в атрибуте запроса используйте адаптер.
      * Примечание: Создание объекта должно производиться
      * в обход фэктори, однако, в целях примера показано создание
      * экземпляра объекта **/
    AccountingAdapter adapter = new
                          AccountingAdapter();
    adapter.setAccountInfo(helper);

    LogManager.logMessage(helper.getRequest(),
  "ViewAccountDetailsCommand", "processing complete");

   /** Примечание: Создание объекта должно производиться
      * в обход фэктори, однако в целях примера показано создание
      * экземпляра объекта **/
    Dispatcher dispatcher = new Dispatcher();
    dispatcher.dispatch(helper);

   /** При нормальном запуске данного сценария
      * эта строка возврата отправляться не будет, т.к.
      * до прохождения этой точки кода управление
      * передается другому ресурсу. Некоторые команды
      * возвращают String, так что значение return
      * введено здесь для корректности. **/
    return systemerror;
  }
}
```

### Пример 7.32 AccountingAdapter

```
public class AccountingAdapter {
    public void setAccountInfo(
      RequestHelper requestHelper) {
        LogManager.logMessage(
          requestHelper.getRequest(),
          "Retrieving data from business tier");

        // извлечение данных из бизнес яруса посредством
        // delegate. Для краткости блок try/catch опущен.
        AccountDelegate delegate =
                new AccountDelegate();
        AccountTO account =
          delegate.getAccount(
            requestHelper.getCustomerId(),
```

```
            requestHelper.getAccountKey());

        LogManager.logMessage(
          requestHelper.getRequest(),
  "Store account Transfer Object in request attribute");

      // перенести данные используя объект запроса
      requestHelper.getRequest().setAttribute(
         "account", account);
    }
}
```

Результатом вызова бизнес-службы посредством delegate является Account Transfer Object, который адаптер сохраняет в атрибуте запроса для использования его видом. В примере 7.33 показана JSP-страница `accountdetails.jsp`, в которую передается запрос. Transfer Object импортируется посредством стандартного тэга `<jsp:useBean>`, а доступ к его свойствам осуществляется при помощи тэга `<jsp:getProperty>`. Кроме этого, вид использует простую составную стратегию, выполняя включение подвида `trace.jsp` во время трансляции. Этот подвид отвечает за вывод на экран информации, записанной в логе, что сделано исключительно для примера.

### Пример 7.33 Вид – accountdetails.jsp

```
<html>
<head><title>AccountDetails</title></head>
<body>

<jsp:useBean id="account" scope="request"
  class="corepatterns.util.AccountTO" />


<h2><center> Account Detail for <jsp:getProperty
  name="account" property="owner" />
</h2> <br><br>
<table border=3>
<tr>
<td>
Account Number :
</td>
<td>
<jsp:getProperty name "account" property="number" />
</td>
</tr>

<tr>
<td>
Account Type:
</td>
<td>
<jsp:getProperty name="account" property="type" />
</td>
</tr>

<tr>
<td>
Account Balance:
</td>

<td>
<jsp:getProperty name="account" property="balance" />
</td>
</tr>

<tr>
<td>
OverDraft Limit:
</td>
<td>
<jsp:getProperty name="account"
  property="overdraftLimit" />
</td>
</tr>

</table>
```

```
<br>
<br>

</center>
<%@ include file="/jsp/trace.jsp" %>
</body>
</html>
```


**Рисунок 7.23 Скриншот примера Service to Worker**

- Родственные паттерны**Front Controller и View Helper**
  Паттерн Service to Worker является результатом комбинации паттерна View Helper и диспетчера, согласованными с паттерном Front Controller.
- **Dispatcher View**
  Паттерн Service to Worker является другим названием комбинации паттерна Front Controller с диспетчером и паттерном View Helper. Паттерны Service to Worker и Dispatcher View идентичны относительно включенных в них компонентов, однако отличаются разделением труда между ними. Паттерн Dispatcher View предоставляет отсрочку извлечения контента на время обработки вида. Кроме этого, в управлении видом диспетчер выполняет более ограниченную роль, так как выбор вида обычно уже включен в запрос.