Понятие Cookie

- Спецификация Cookie разработана компанией Netscape Communications
- **Cookie** - это текстовый файл, хранящийся на клиентской машине и доступный браузеру.
- С программной точки зрения, **cookie** - это пара «название - значение», которая посылается браузеру сервером при первом обращении, а затем передается браузером тому же серверу при обращении к определенным ресурсам.
- Этот механизм позволяет на протяжении нескольких **HTTP** запросов сохранять для браузера на клиенте ту или иную информацию, полученную от сервера.

Схема обмена Cookie

Cookie

Response Header:

```
Set-Cookie: cname=cvalue;Expires=Tue,
  14-Feb-2006 23:13:26 GMT;Path=/
```

Request Header:

```
Cookie: cname=cvalue
```

Internet Explorer:

...\Documents and Settings\...\Cookies

Класс javax.servlet.http.Cookie

- Cookie(String name, String value)

- get/setName(String)
- get/setValue(String)
- get/setAge(int)
- get/setPath(String)
- is/setSecure(boolean)
- get/setVersion(int)

Пример использования.Cookie

```
Cookie c = new Cookie("cname", "cvalue");
c.setPath("/");
c.setMaxAge(3600);
response.setContentType = "text/html";
response.addCookie(c);
```

```
Cookie cookies[] = request.getCookies();
```

Преимущества Cookie

- Отслеживание сеанса пользователя
- Пользовательские настройки
- Подстановка имени и пароля при повторном заходе на сайт
- Направленная реклама
- Использование cookie не представляет угрозы безопасности с точки зрения атак
- Браузеры принимают только 20 cookies на сайт и 300 всего, каждое Cookie до 4 кбайт – отсутствует проблема засорения жесткого диска

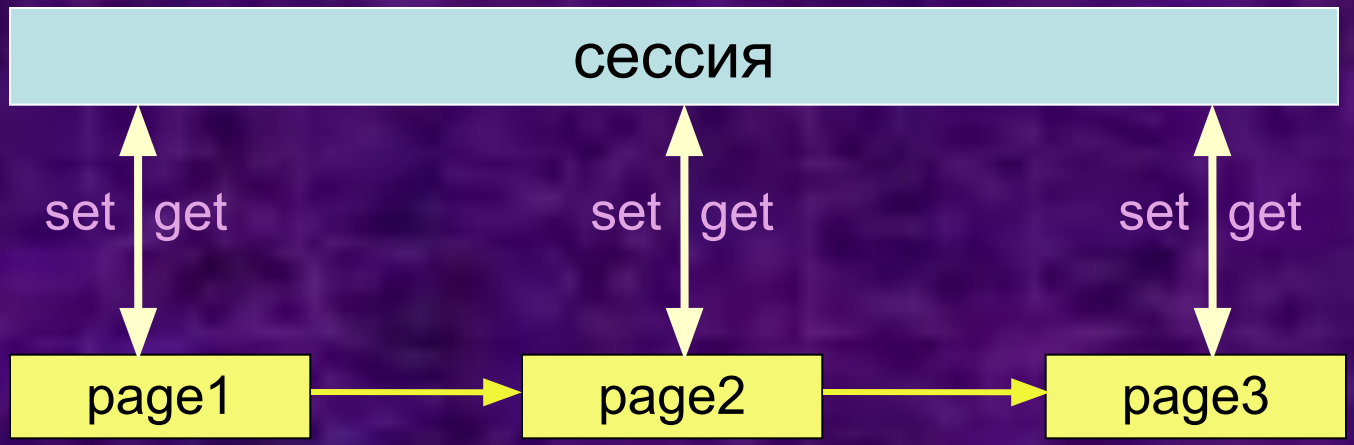
Недостатки Cookies

- Не представляют угрозу *безопасности*, но угрожают *конфиденциальности:*
  - eMail с HTML текстом может загружать Web-ресурсы, передающие cookies
  - Cookie могут подделываться для идентификации пользователя в качестве другого
  - Cookie – открытые текстовые файлы. Поэтому в них нельзя хранить конфиденциальную информацию. Обычно хранится только идентификатор – данные в хеш-таблице или базе данных на сервере

Отслеживание сеанса

- HTTP – stateless протокол
  - каждый запрос – отдельное соединение
  - сервер не имеет данных о предыдущем запросе
- `Connection: keep-alive` не спасает ситуацию

Сессии



Сессии

- Для организации сессий существует три типичных подхода:
  - Cookies

    response.addCookie(“JSESSIONID”, sessionId);
  - URL-rewriting

    http://www.some.com/page.jsp?jsessionid=12345

    http://www.some.com/page.jsp/12345

    http://www.some.com/page.jsp;jsessionid=12345
  - Скрытые поля форм

    <input type=“hidden” name=“JSESSIONID” value=“12345”>
- В Java Web-контейнерами обычно используются по умолчанию Cookies, но если браузер их не поддерживает – автоматически осуществляется переход на URL-Rewriting

Сессии в Java

- Объект **сессии** создается каждый раз при получении запроса от нового клиента и впоследствии идентифицирует его уникальным образом
- Разным пользователям соответствуют различные объекты сессий
- Сессии “живут” на сервере в течение заданного времени, но только для одного клиентского браузера

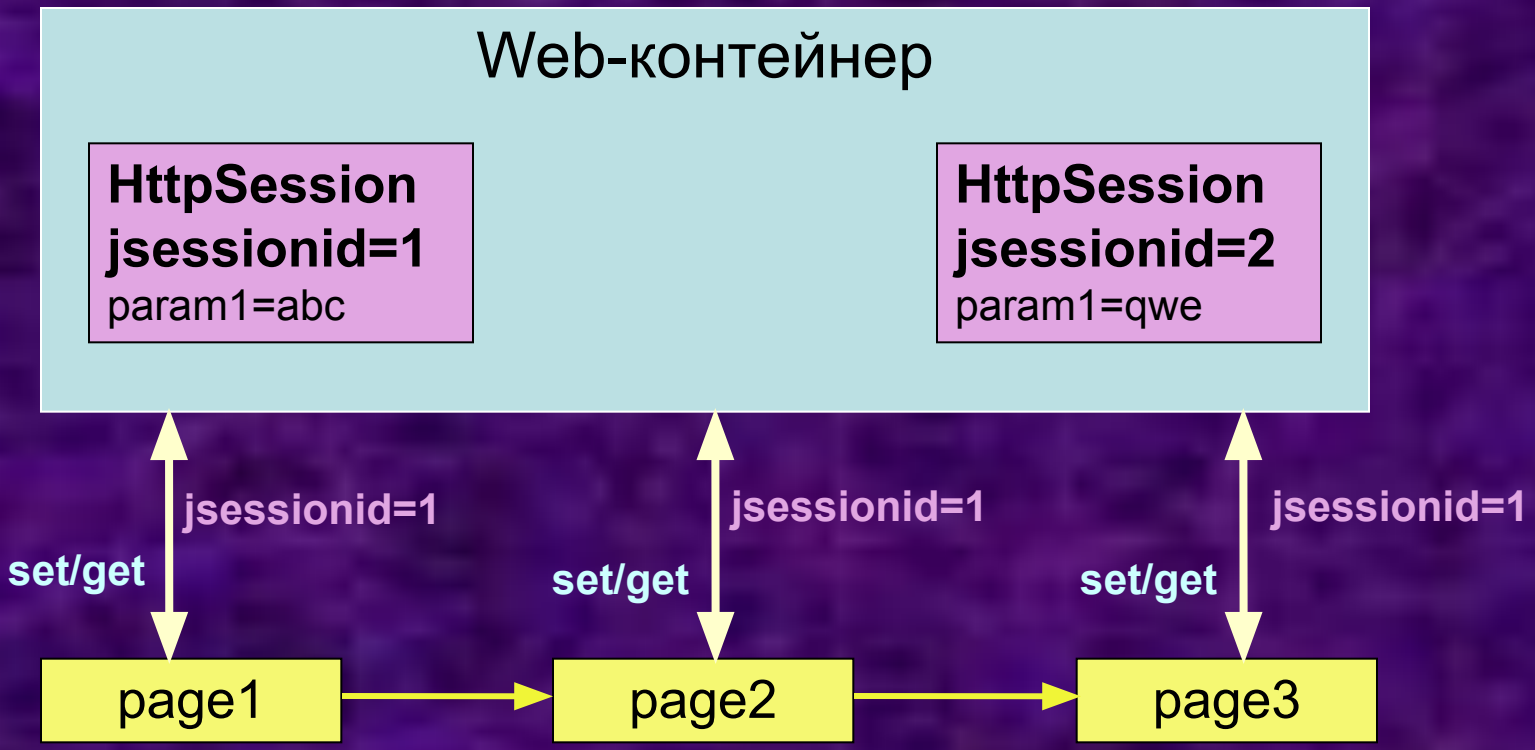
Интерфейс HttpSession

- Высокоуровневый интерфейс для работы с сеансами;
- Автоматически обеспечивает поддержку сеанса при помощи cookies или перезаписи URL
- Позволяет манипулировать данными о сессии, такими, как идентификатор, время создания, время жизни и т.п.
- Позволяет сохранять данные, введенные клиентом в течение нескольких переходов по страницам
- Получить ссылку на объект HttpSession в сервлете можно с помощью метода
  - request.getSession()

Реализация сессий

- В случае использования cookies автоматически формируется Cookie с именем JSESSIONID и значением типа 02395C69B8FB84D3278B49F1B05F3379, которое используется в качестве ключа в хеш-таблице на сервере
- Если cookie отключены, формируется URL вида: http://.../some_path;jsessionid=02395C69B8FB84D3278B49F1B05F3379
- Это делается автоматически только в случае, если URL, к которому обращаются из Web-приложения, закодирован методом
  - HttpServletResponse.encodeURL()
  - или encodeRedirectURL()

Сессии в Java



Пример работы с сессией

- Фрагмент сервлета, проверяющего правильность ввода имени и пароля

```
…
// Установка времени жизни сессии
request.getSession().setMaxInactiveInterval(30*60*1000);
// Берем параметры из запроса
String login = request.getParameter("login");
String password = request.getParameter("password");
// создаем объект User
User user = new User(login, password):
if (userDatabase.contains(user)){       // Если такой есть – помещаем в сессию
    request.getSession().setAttribute("user", user);
}
…
```

- Фрагмент сервлета, выводящего имя зарегистрированного пользователя

```
…
// извлекаем объект User из сессии
User user = (User)request.getSession().getAttribute("user");
// Печатаем имя пользователя
response.getWriter().println("User name: " + user.getName());
…
```

Данные, общие для всего приложения

- Объект ServetContext существует в единственном экземпляре для одного WEB-приложения.
- В нем можно хранить глобальные настройки приложения (с помощью методов get/setAttribute(…))
- Другие методы ServletContext:
  - getRealPath(String path) – возвращает реальный путь к ресурсу файловой системы, находящемся по заданному виртуальному пути
  - getResourceAsStream(String path) – возвращает поток байт реального ресурса файловой системы
- Получить ссылку на ServletContext из сервлета можно, например, так:
  - getServletContext()

  или по объекту request
  - request.getSession().getServletContext()

Установка атрибутов

- В качестве атрибутов выступают объекты
- Атрибуты можно устанавливать на уровне
  - запроса

    request.setAttribute(“myattr”, new Integer(1));
    Integer attr = (Integer)request.getAttribute(“myattr”);
  - сессии

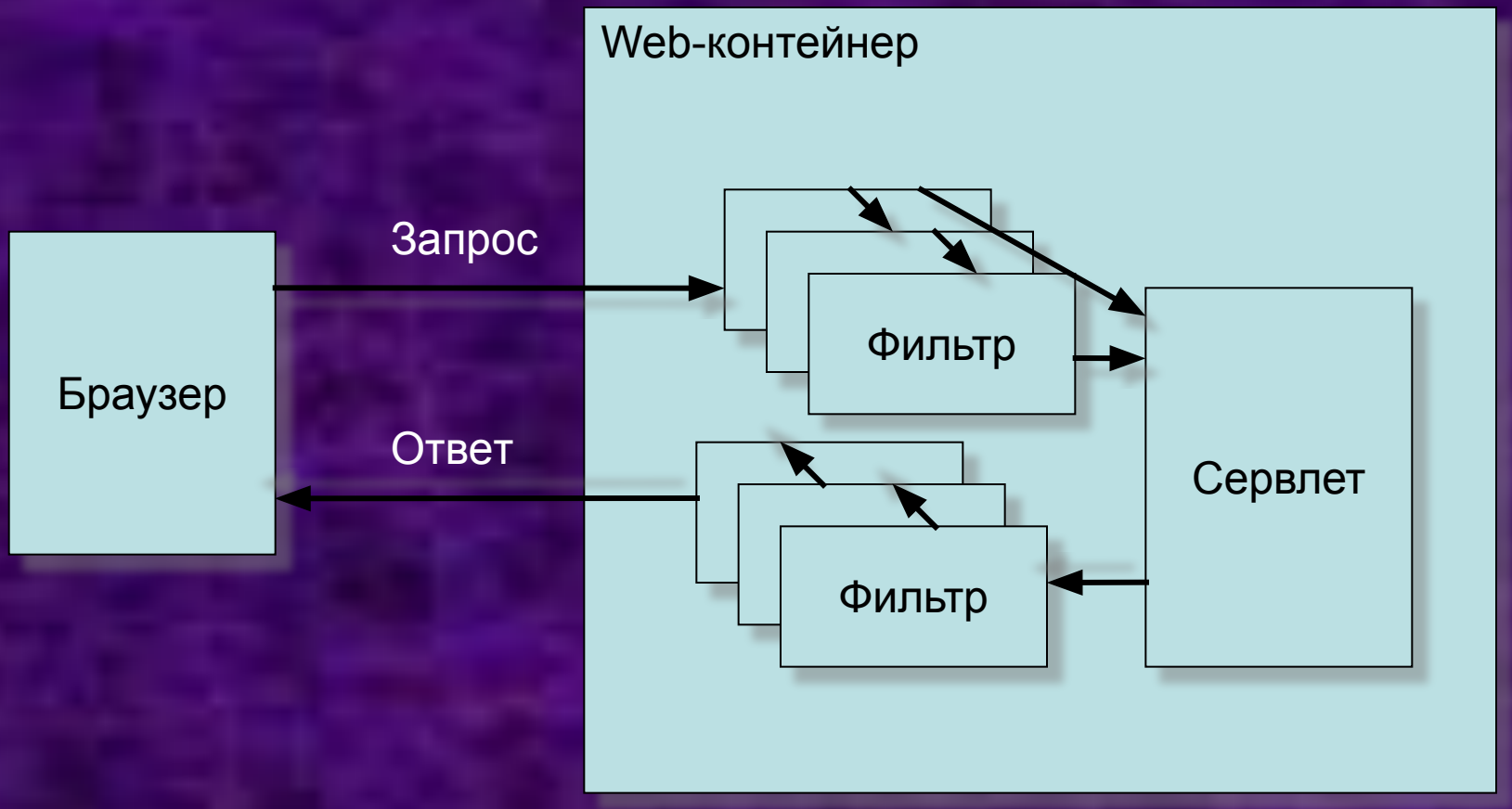
    request.getSession().setAttribute(“myattr”, new Integer(1));
  - приложения

    getServletContext().setAttribute(…)

Фильтры

- Фильтр – это Java-код, пригодный для многократного использования и позволяющий осуществлять операции над содержимым HTTP-запросов, ответов и заголовков.
- определены в спецификации сервлетов начиная с версии 2.3.
- Фильтр способен обрабатывать:
  - запрос до его получения сервлетом;
  - ответ после его обработки сервлетом.

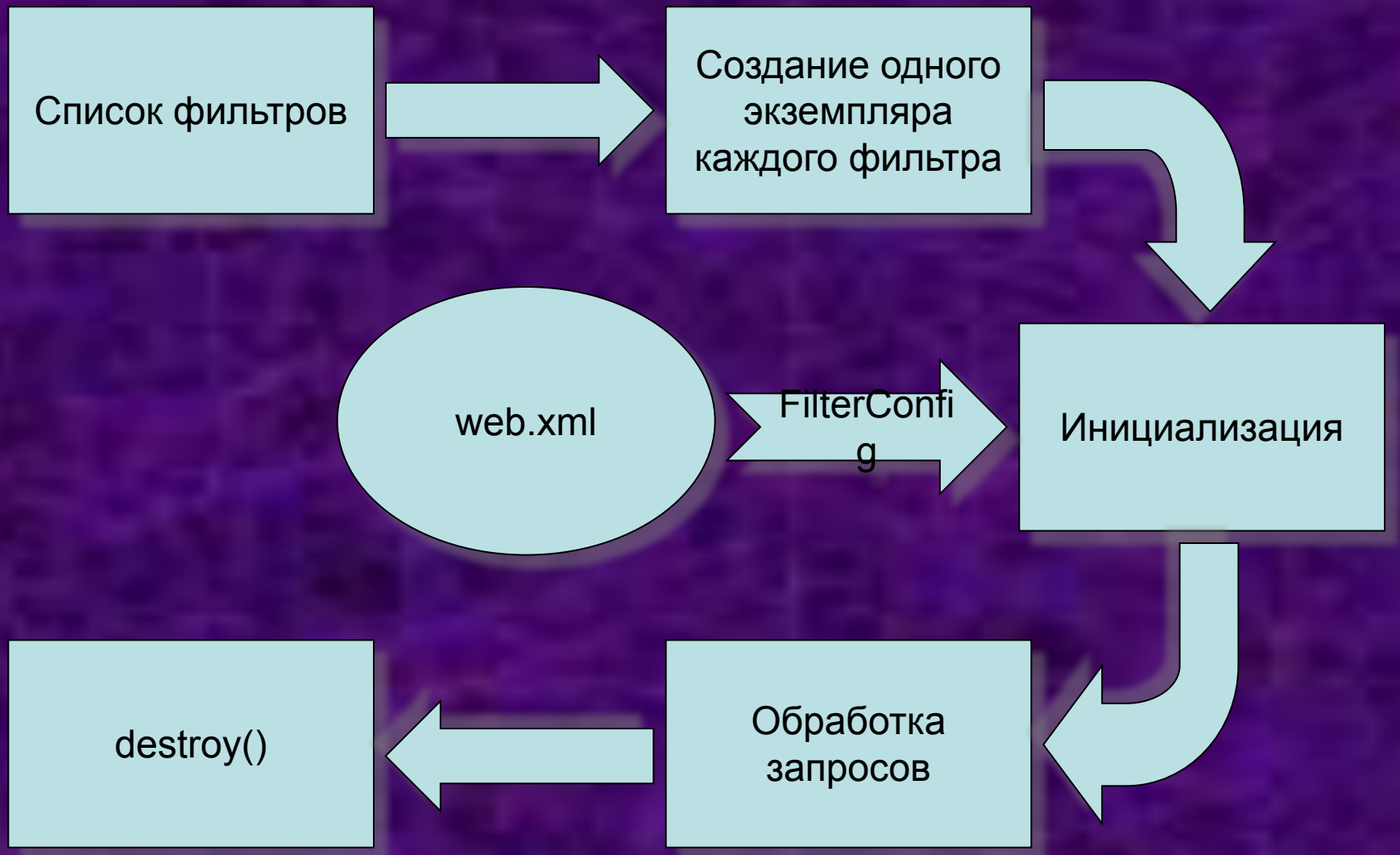
Роль фильтра в обработке запроса



Интерфейс javax.servlet.Filter

- Интерфейс Filter имеет следующие методы:
  - init(FilterConfig);
  - doFilter(ServletRequest, ServletResponse, FilterChain);
  - destroy().

Жизненный цикл фильтра



Цепочка фильтров

- Предназначена для обработки запроса последовательно несколькими фильтрами.
- Представлена интерфейсом javax.servlet.FilterChain.
- Объект создается автоматически контейнером
- Метод:
  - doFilter(ServletRequest, ServletResponse)

Интерфейс javax.servlet.FilterConfig

- Служит для передачи информации о настройках фильтру при его инициализации.
- Имеет следующие методы:
  - String getFilterName()
  - String getInitParameter(String)
  - Enumeration getInitParameterNames()
  - ServletContext getServletContext()

Пример

```
//пакет и импорты

public class LoggerFilter implements Filter {

   private FilterConfig filterConfig;

   public void setFilterConfig(FilterConfig fc) {
         filterConfig = fc;
   }

   public FilterConfig getFilterConfig() { return filterConfig; }

   public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
                                       throws IOException, ServletException {
      if (filterConfig == null) {
          return;
      }
      ServletContext ctx = filterConfig.getServletContext();
      ctx.log ("" + new Date() + " - resource: " +  ((HttpServletRequest)request).getRequestURL()
      +
                  " is requested by: " + request.getRemoteHost()) ;
      chain.doFilter(request, response) ;
   }

   public void init(FilterConfig config) throws ServletException { this.filterConfig = config; }
   public void destroy() { }
}
```

Описание фильтров в web.xml

- В файле web.xml фильтры описываются в элементе filter.
- Он имеет следующие параметры:
  - filter-name
  - filter-class
  - init-params
  - icon (опциональный)
  - description (опциональный)
  - display-name (опциональный)

Описание фильтров в web.xml

- Для каждого определенного в web.xml фильтра создается ровно один экземпляр.
- Разные теги <filter> с одним и тем же классом, но разным именем – разные фильтры

- Пример описания фильтра в web.xml

```
<filter>
    <filter-name>LoggerFilter</filter-name>
    <filter-class>ua.kharkov.kture.ius.wbis.LoggerFilter</filter-class>
 <init-param>
     <param-name>logLevel</param-name>
     <param-value>medium</param-value>
 </init-param>
</filter>
```

Описание фильтров в web.xml

- Описание фильтров и их привязок описывается в web.xml-файле перед определением сервлетов.
- Цепочка фильтров-обработчиков строится исходя из последовательности появления соответствующих привязок в web.xml.

Описание фильтров в web.xml

- Привязка фильтров бывает:
  - к сервлету

```
<filter-mapping>
  <filter-name>Filter1</filter-name>
  <servlet-name>Servlet1</servlet-name>
</filter-mapping
```

  - к ресурсу по маске

```
<filter-mapping>
  <filter-name>SystemAccessFilter</fliter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Пример фильтра

```
package filters;

//импорты

public class SetCharacterEncodingFilter implements Filter {
  protected String encoding = null;
  protected FilterConfig filterConfig = null;

  public void destroy() {
        this.encoding = null;
        this.filterConfig = null;
  }

  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
      IOException,                                                    ServletException {
        String encoding = selectEncoding(request);
        if (encoding != null)
            request.setCharacterEncoding(encoding);

        chain.doFilter(request, response);
  }

  public void init(FilterConfig filterConfig) throws ServletException {
        this.filterConfig = filterConfig;
        this.encoding = filterConfig.getInitParameter("encoding");
  }

  protected String selectEncoding(ServletRequest request) {  return (this.encoding); }
}
```

Пример фильтра

- Фрагмент web.xml:

```
...

<filter>
    <filter-name>Set Character Encoding</filter-name>
    <filter-class>filters.SetCharacterEncodingFilter</filter-class>
    <init-param>
            <param-name>encoding</param-name>
            <param-value>CP1251</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>Set Character Encoding</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>


...
```

Изменение ServletResponse

```
public void doFilter(ServletRequest request, ServletResponse
    response, FilterChain chain) throws IOException,
                ServletException {

      // действия фильтра до передачи обработки запроса

       chain.doFilter(request, response);

    // действия фильтра после обработки запроса

    ((HttpServletResponse)response).setStatus(403);


    ((HttpServletResponse)response).getWriter().println("Bye");
}
```

Изменение содержимого HTTP-ответа

```
 public void doFilter(ServletRequest request, ServletResponse response,
                      FilterChain chain) throws IOException, ServletException {
    ReverseResponse rr = new ReverseResponse((HttpServletResponse) response);
    chain.doFilter(request, rr);
    response.getWriter().println(rr.getContent().reverse());
  }
}

class ReverseResponse extends HttpServletResponseWrapper {
  private StringWriter sw = new StringWriter();
  private PrintWriter pw = new PrintWriter(sw);

  public ReverseResponse(HttpServletResponse response) {
    super(response);
  }
  public PrintWriter getWriter() {
    return pw;
  }
  public StringBuffer getContent() {
    return sw.getBuffer();
  }
}
```

Слушатели событий

- Слушатели событий (Events listeners) - это классы, реализующие один или более интерфейсов слушателей событий сервлета.

- Делятся на:
  - Servlet context listeners
    - События, связанные с контекстом сервлета.
  - HTTP session listeners
    - События, связанные с сеансом HTTP

Servlet context listeners

- Представлены интерфейсами:
- javax.servlet.ServletContextListener:
  - void contextInitialized(ServletContextEvent sce)
  - void contextDestroyed(ServletContextEvent sce)
- javax.servlet.ServletContextAttributeListener:
  - void attributeAdded(ServletContextAttributeEvent e)
  - void attributeRemoved(ServletContextAttributeEvent e)
  - void attributeReplaced(ServletContextAttributeEvent e)

События слушателей контекста

- javax.servlet. ServletContextEvent
  - ServletContext getServletContext()
- javax.servlet.ServletContextAttributeEvent
  - String getName()
  - Object getValue()

HTTP session listeners

javax.servlet.HttpSessionListener:

- void sessionCreated(HttpSessionEvent se)
- void sessionDestroyed(HttpSessionEvent se)

- javax.servlet.HttpSessionAttributeListener:
  - void attributeAdded(HttpSessionBindingEvent e)
  - void attributeRemoved(HttpSessionBindingEvent e)
  - void attributeReplaced(HttpSessionBindingEvent e)

- javax.servlet.HttpSessionBindingListener:
  - void valueBound(HttpSessionBindingEvent event)
  - void valueUnbound(HttpSessionBindingEvent event)

События слушателей сессии

- javax.servlet.http.HttpSessionEvent
  - HttpSession getSession()
- javax.servlet.http.HttpSessionBindingEvent
  - HttpSession getSession()
  - String getName()
  - Object getValue()

Описание в web.xml

- В web.xml слушатели событий прописываются следующим образом:

```
<listener>
    <listener-class>Полное имя класса</listener-class>
</listener>
```

- Слушатели событий описываются после привязок фильтров до определения сервлетов.
- HttpSessionBindingListener в web.xml не прописывается, а реализуется классом, который должен отслеживать свою привязку и удаления из сесии

Пример

- Листинг:

```
public class LifeCycleServletContextListener implements
                ServletContextListener {

    public LifeCycleServletContextListener() {
    }

    public void contextInitialized(ServletContextEvent sce) {
       System.out.println("LifeCycleServletContextListener: contextInitialized");
    }

    public void contextDestroyed(ServletContextEvent sce) {
    System.out.println("LifeCycleServletContextListener:contextDestroyed");
    }
}
```

- Фрагмент web.xml

```
<listener>
    <listener-class>
      testr1.LifeCycleServletContextListener
    </listener-class>
</listener>
```

Пример

- Листинг:

```
public class LifeCycleServletContextListener implements
                ServletContextListener {

    public LifeCycleServletContextListener() {
    }

    public void contextInitialized(ServletContextEvent sce) {
       System.out.println("LifeCycleServletContextListener: contextInitialized");
    }

    public void contextDestroyed(ServletContextEvent sce) {
    System.out.println("LifeCycleServletContextListener:contextDestroyed");
    }
}
```

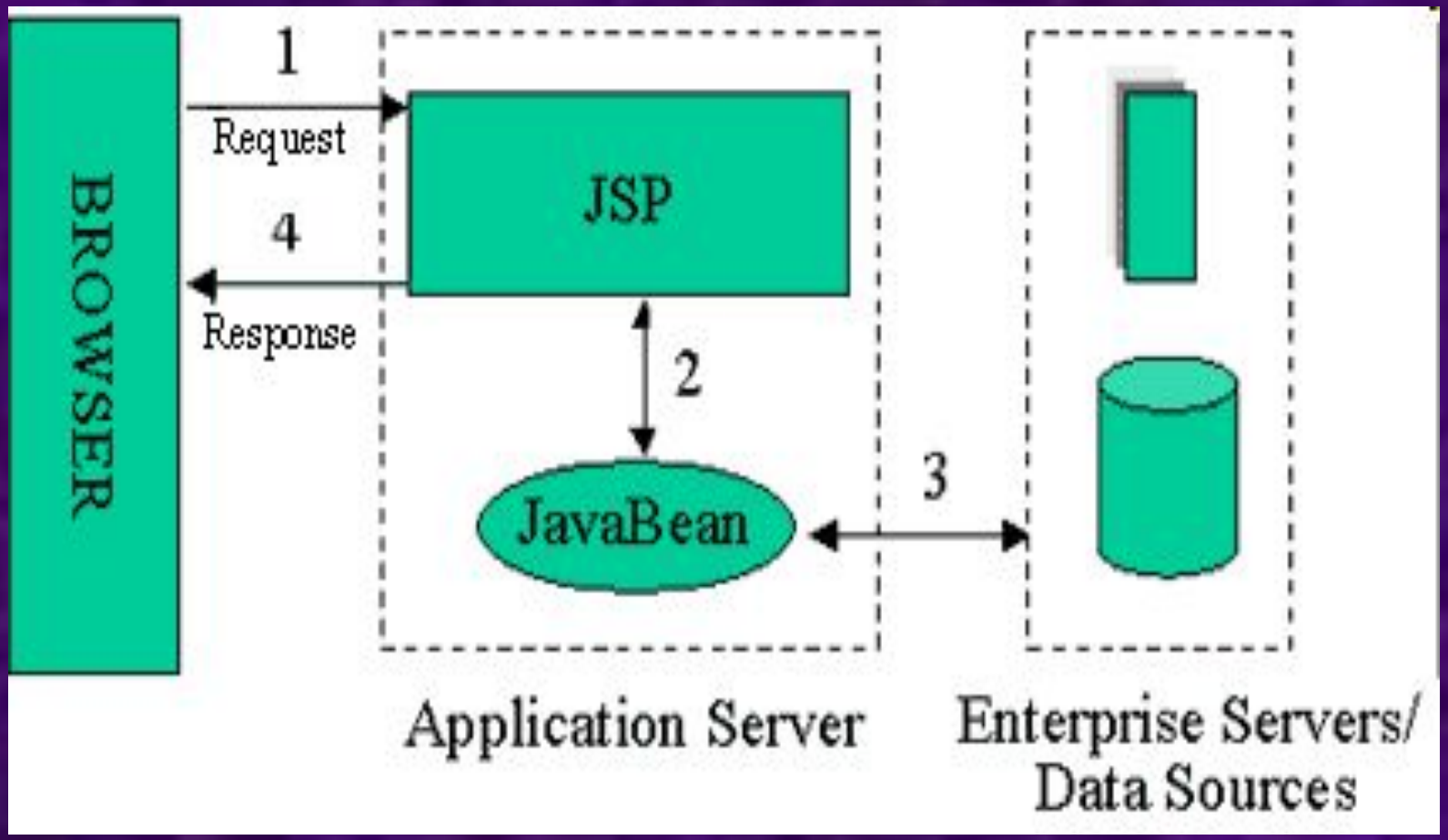
- Фрагмент web.xml

```
<listener>
    <listener-class>
      testr1.LifeCycleServletContextListener
    </listener-class>
</listener>
```

Подходы к созданию Web-базированных ИС

- Совмещение бизнес-логики и дизайна на JSP-страницах.
- Вынесение части логики в bean-компоненты (Model 1).
- Разделение Web-приложения на 3 части: Модель, Представление и Контроллер (Model 2).

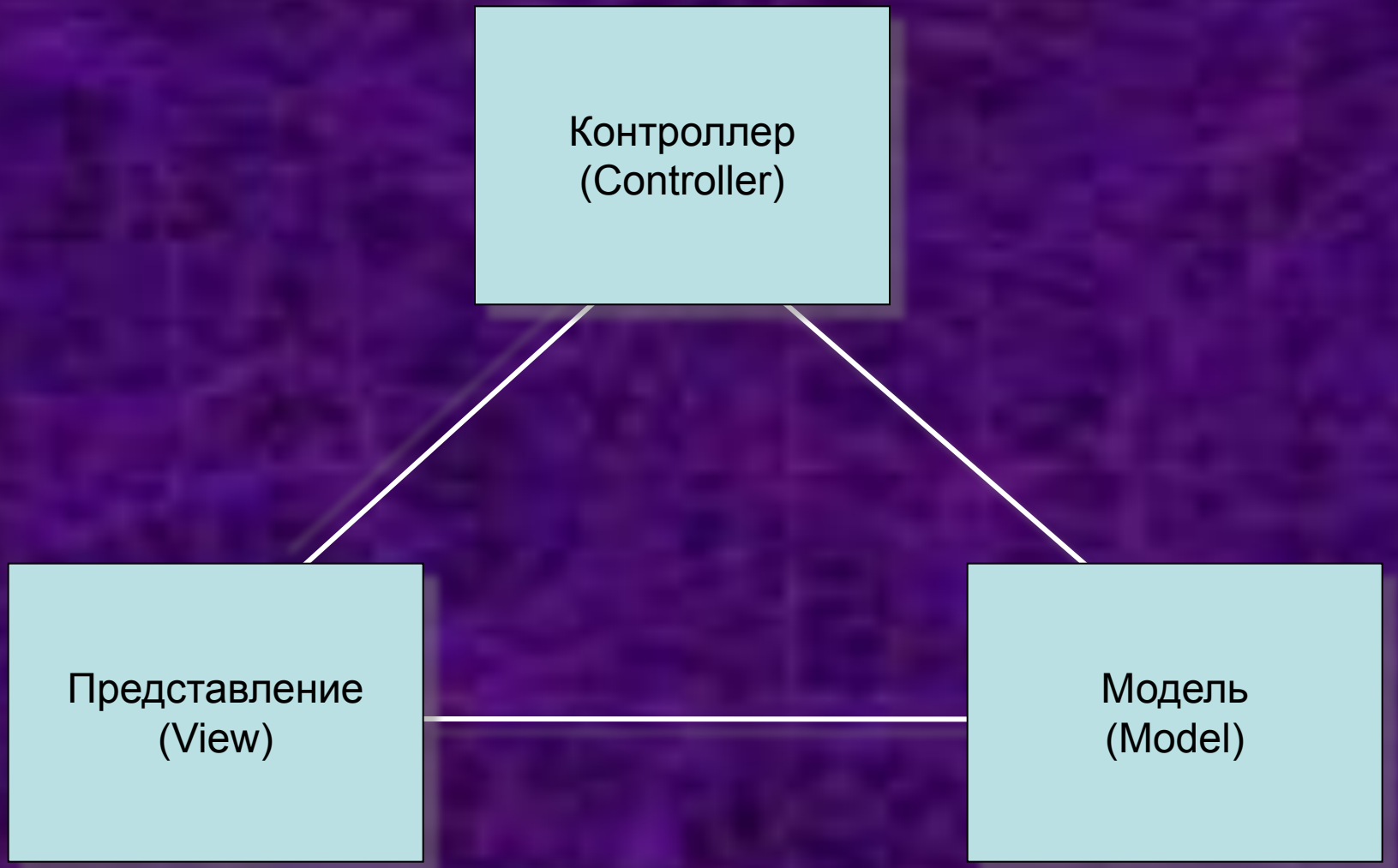
Model 1


BROWSER
1
Request
4
Response
JSP
2
JavaBean
3
Application Server
Enterprise Servers/
Data Sources

Особенности Model 1

- Преимущества:
  - простота разработки;
  - разработка JSP и бизнес-объектов может вестись параллельно благодаря отделению бизнес-логики от представления.
- Недостатки:
  - необходимо вовлечение программистов в процесс создания JSP-документов;
  - оправдано создание только приложений, ориентированных на единый тип клиентов.

Треугольник MVC



Model 2

- Приложение состоит из 3-х частей:
  - Модель:
    - представляет собой состояние бизнес объектов;
    - обрабатывает запросы о состоянии;
    - представляет бизнес-логику приложения;
    - оповещает Представление об изменении состояния.
  - Контроллер:
    - определяет поведение приложения;
    - по запросу пользователя изменяет состояние Модели;
    - выбирает отображения для вывода;
    - единственный для каждой функции.
  - Представление:
    - отображает Модель;
    - получает от Модели информацию об изменении состояния;
    - передает ввод пользователя контроллеру;
    - позволяет контроллеру выбрать Представление.

Общая схема Model 2

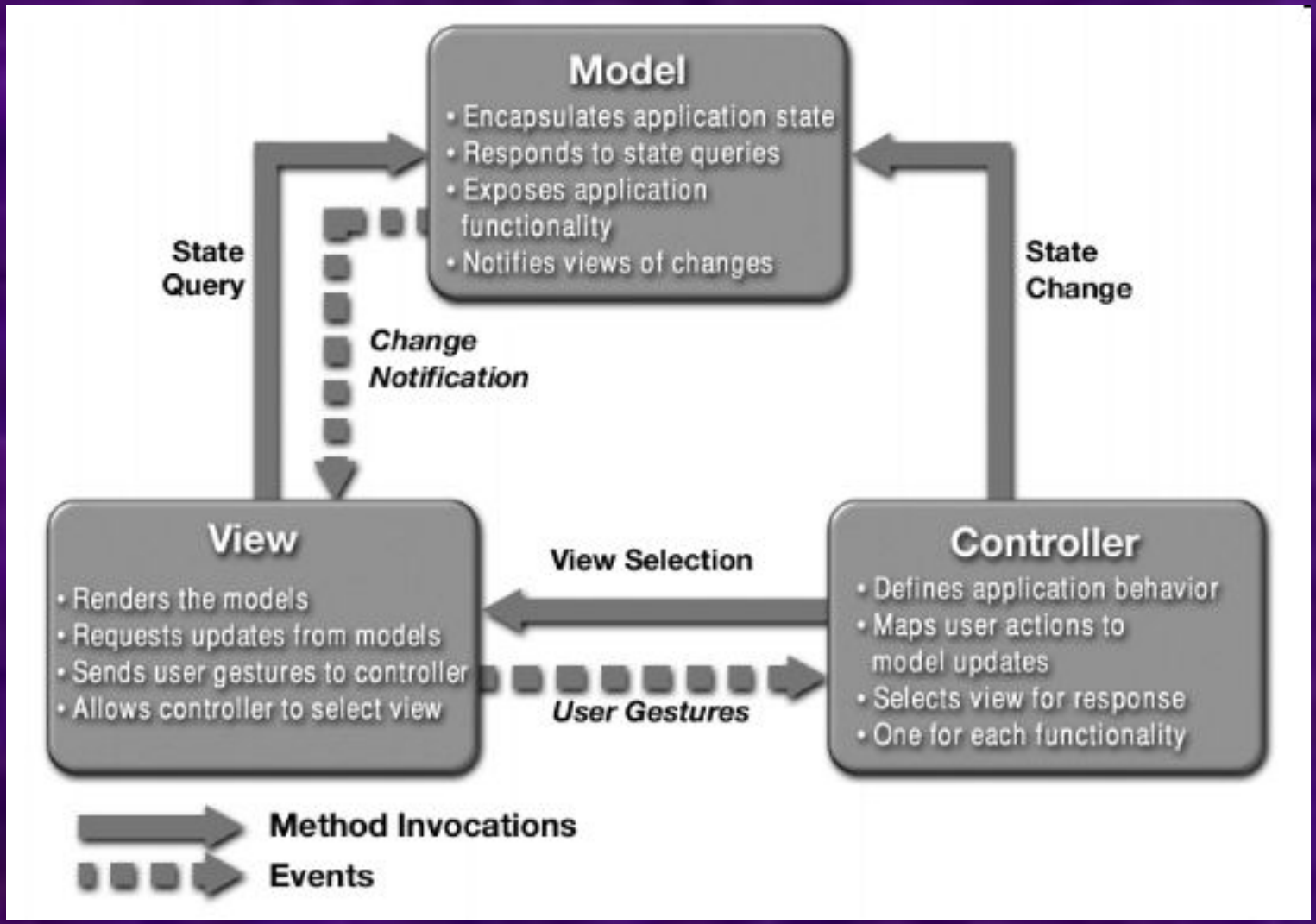
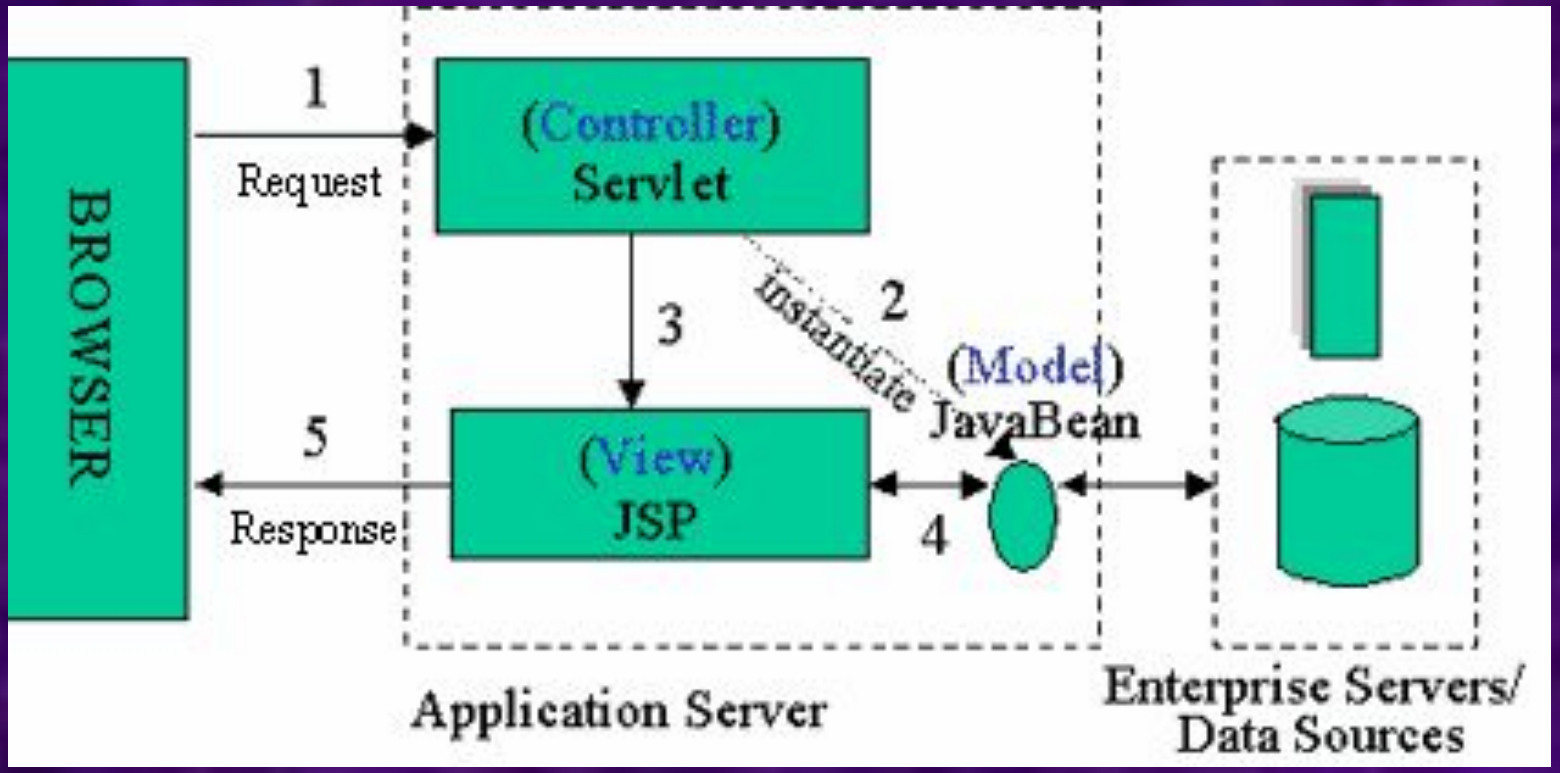


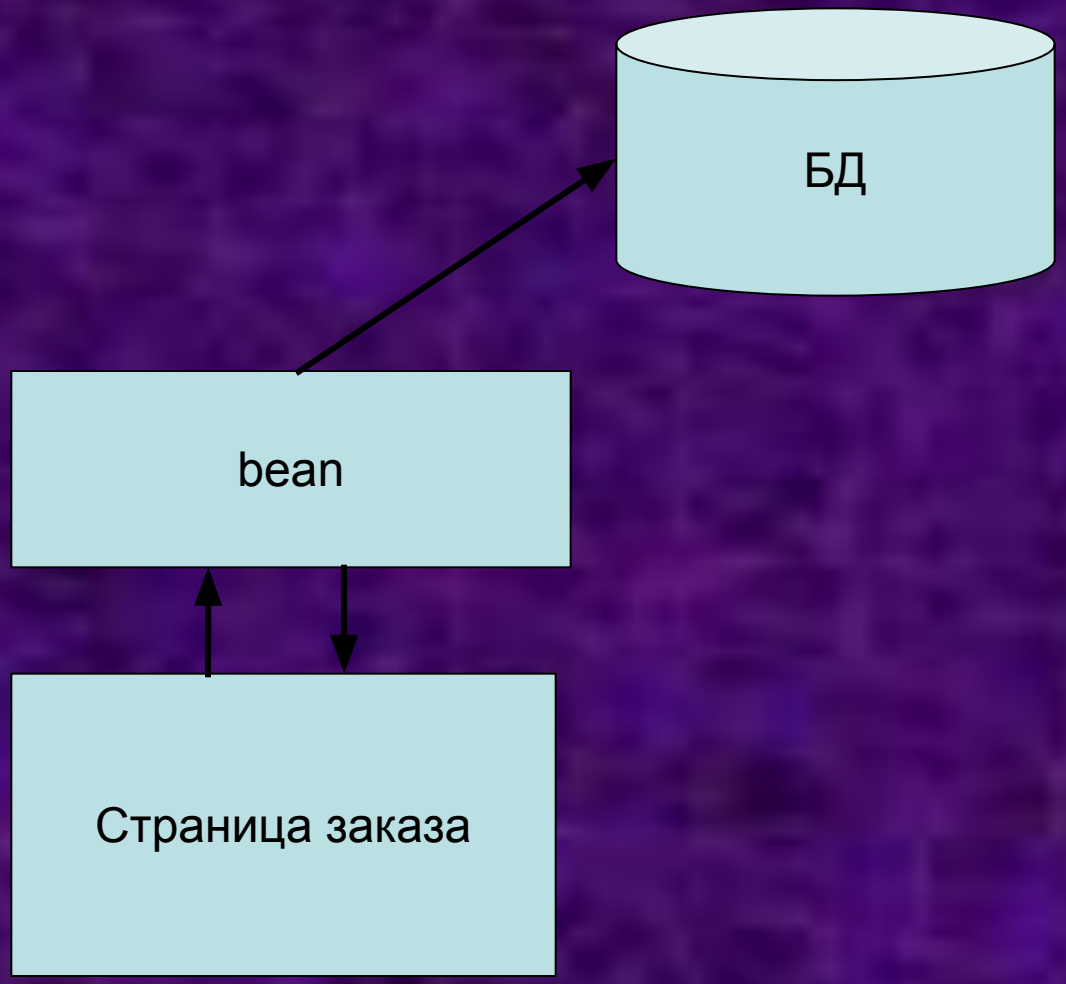
Схема Model 2 для Web-приложений



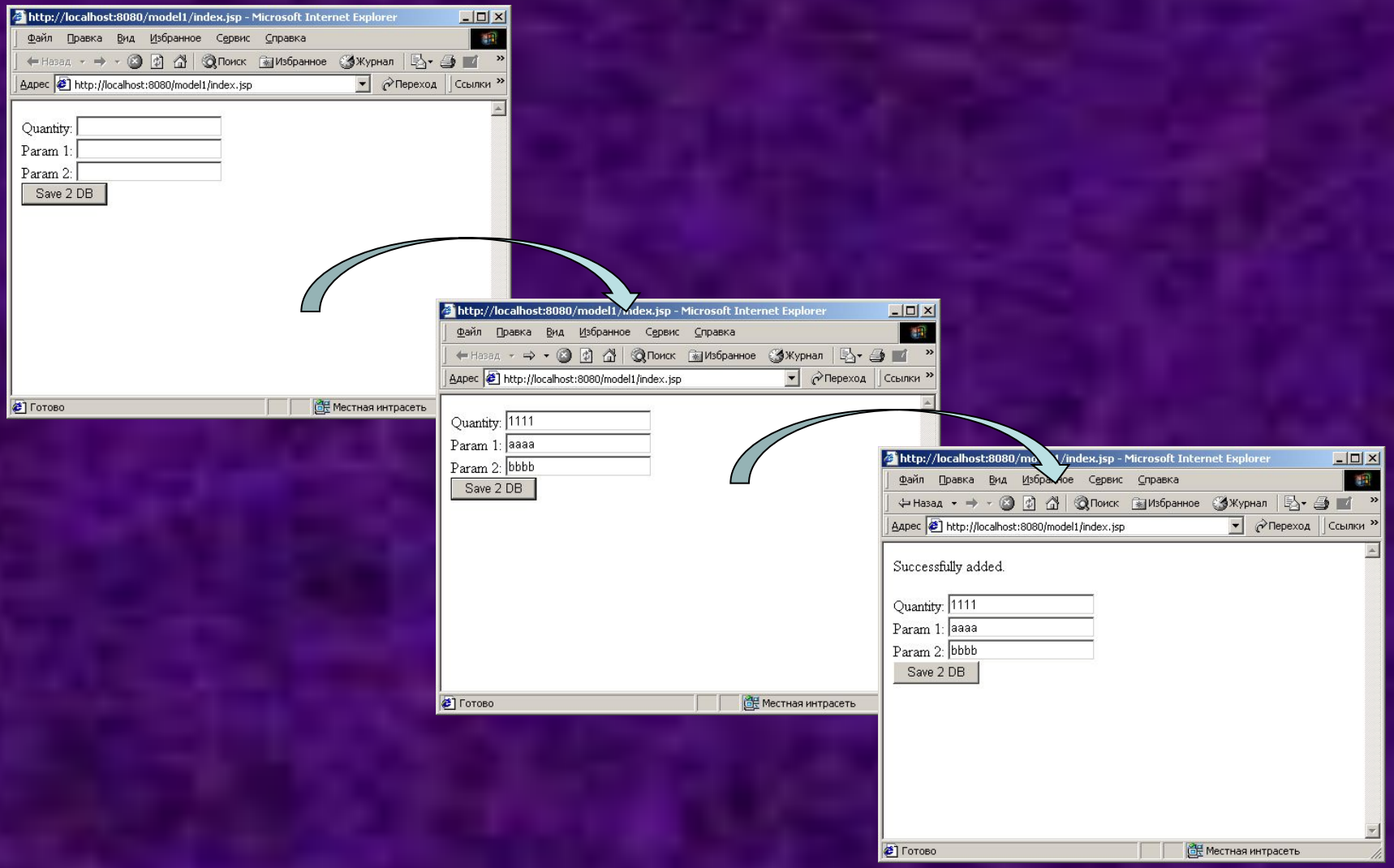
Особенности Model 2

- Возможность переиспользования компонентов Модели.
- Простая поддержка новых типов клиентов.
- Возросшая сложность разработки.

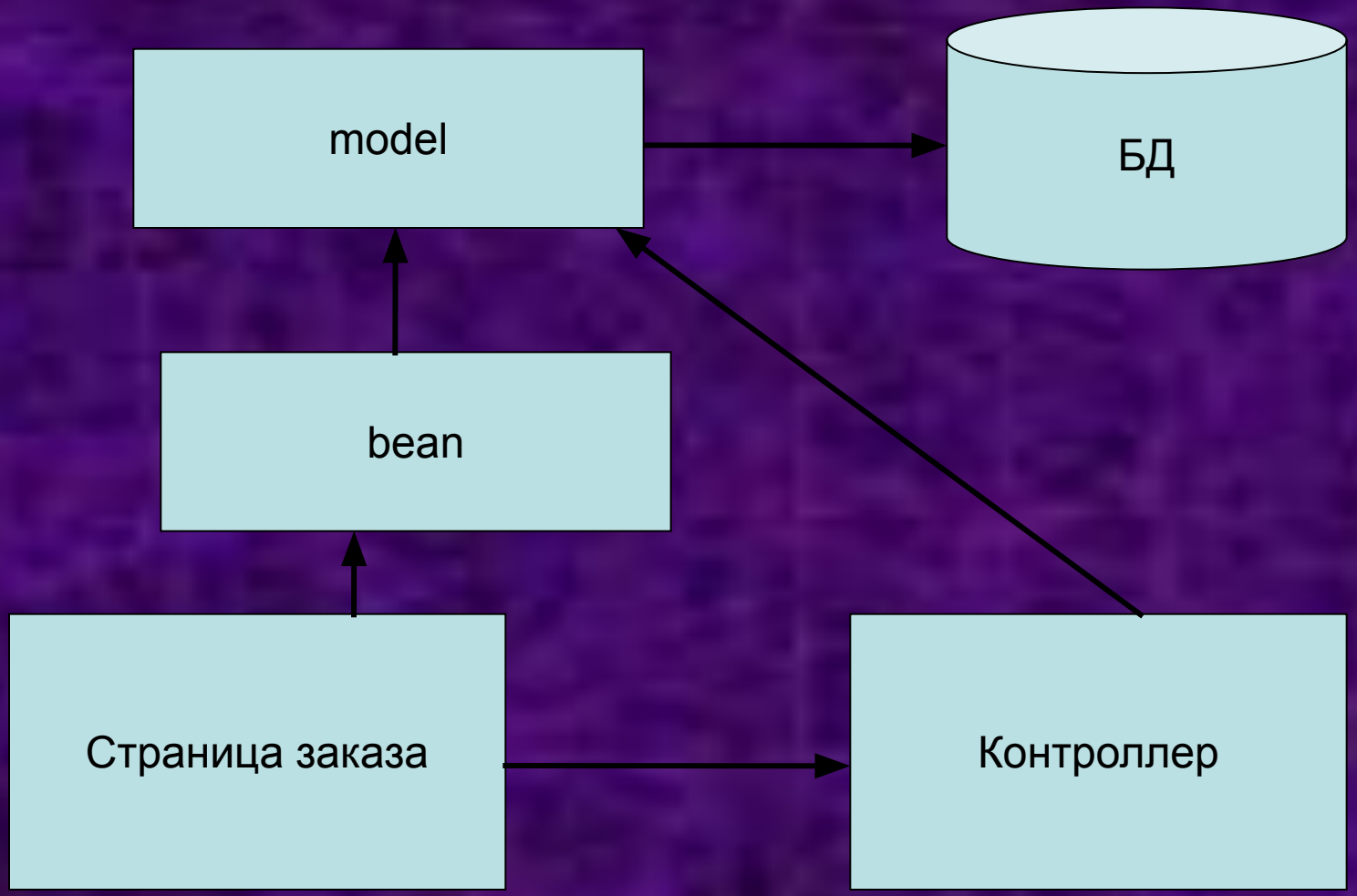
Пример приложения на model1



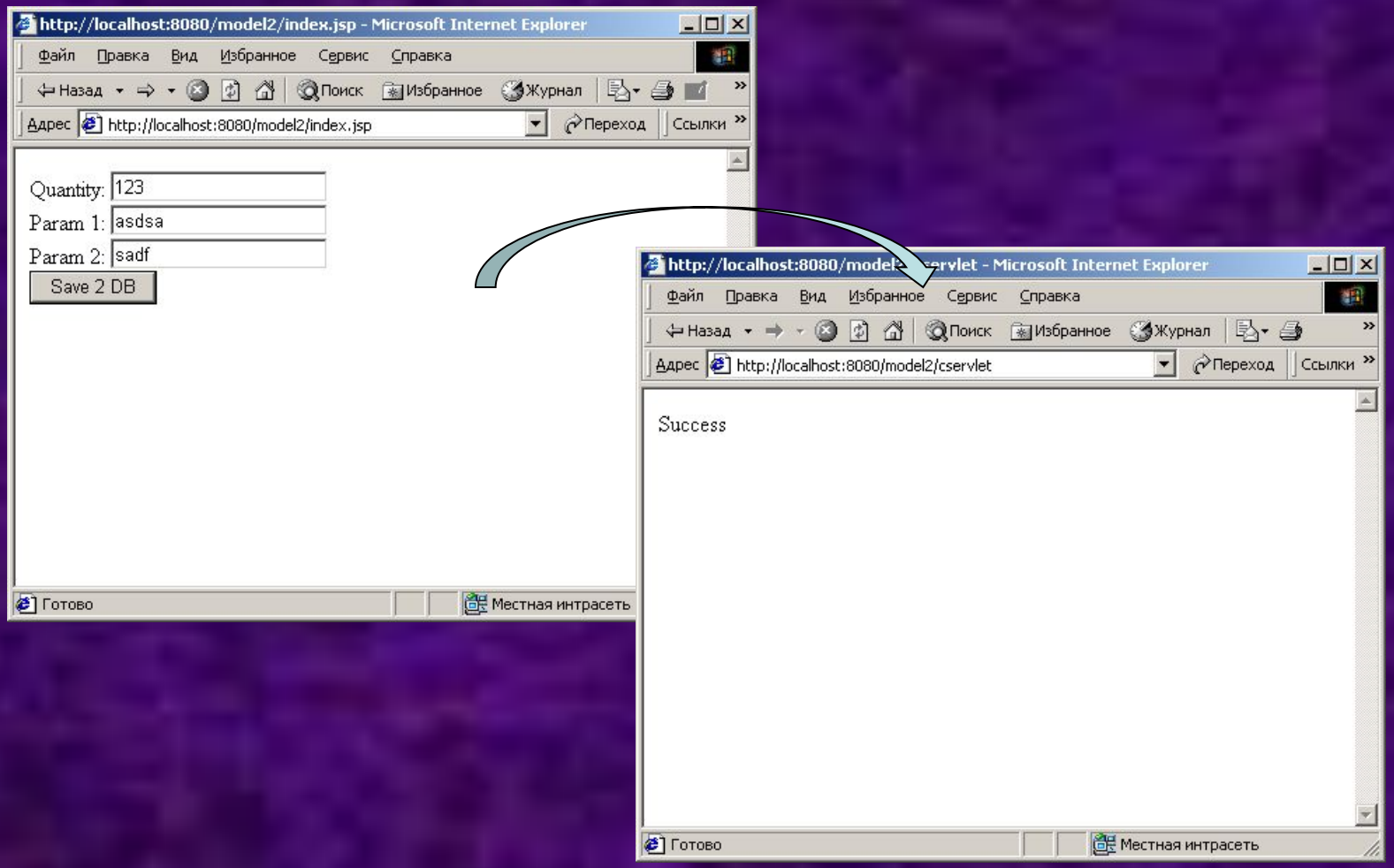
Результат


http://localhost:8080/model1/index.jsp - Microsoft Internet Explorer
Файл Правка Вид Избранное Сервис Справка
Назад Поиск Избранное Журнал
Адрес http://localhost:8080/model1/index.jsp
Переход Ссылки
Quantity:
Param 1:
Param 2:
Save 2 DB
Готово
Местная интрасеть
http://localhost:8080/model1/index.jsp - Microsoft Internet Explorer
Файл Правка Вид Избранное Сервис Справка
Назад Поиск Избранное Журнал
Адрес http://localhost:8080/model1/index.jsp
Переход Ссылки
Quantity: 1111
Param 1: aaaa
Param 2: bbbb
Save 2 DB
Готово
Местная интрасеть
Файл Правка Вид Избранное Сервис Справка
Назад Поиск Избранное Журнал
Адрес http://localhost:8080/model1/index.jsp
Переход Ссылки
Successfully added.
Quantity: 1111
Param 1: aaaa
Param 2: bbbb
Save 2 DB
Готово
Местная интрасеть

Пример приложения на model2



Результат


http://localhost:8080/model2/index.jsp - Microsoft Internet Explorer
Файл Правка Вид Избранное Сервис Справка
Назад Поиск Избранное Журнал
Адрес http://localhost:8080/model2/index.jsp
Переход Ссылки
Quantity: 123
Param 1: asdsa
Param 2: sadf
Save 2 DB
Готово
Местная интрасеть
Файл Правка Вид Избранное Сервис Справка
Назад Поиск Избранное Журнал
Адрес http://localhost:8080/model2/cservlet
Переход Ссылки
Success
Готово
Местная интрасеть