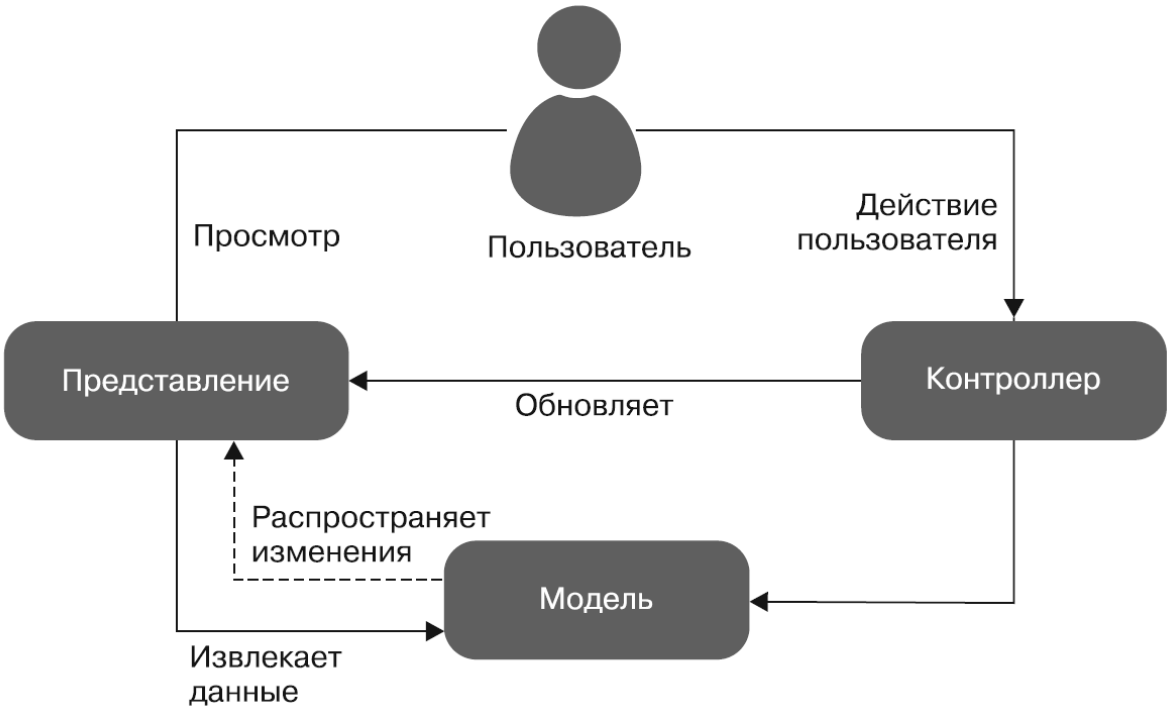
Spring MVC

Составил: Черниговский А.С.
Старший преподаватель кафедры “Информатика”
ИКИТ СФУ

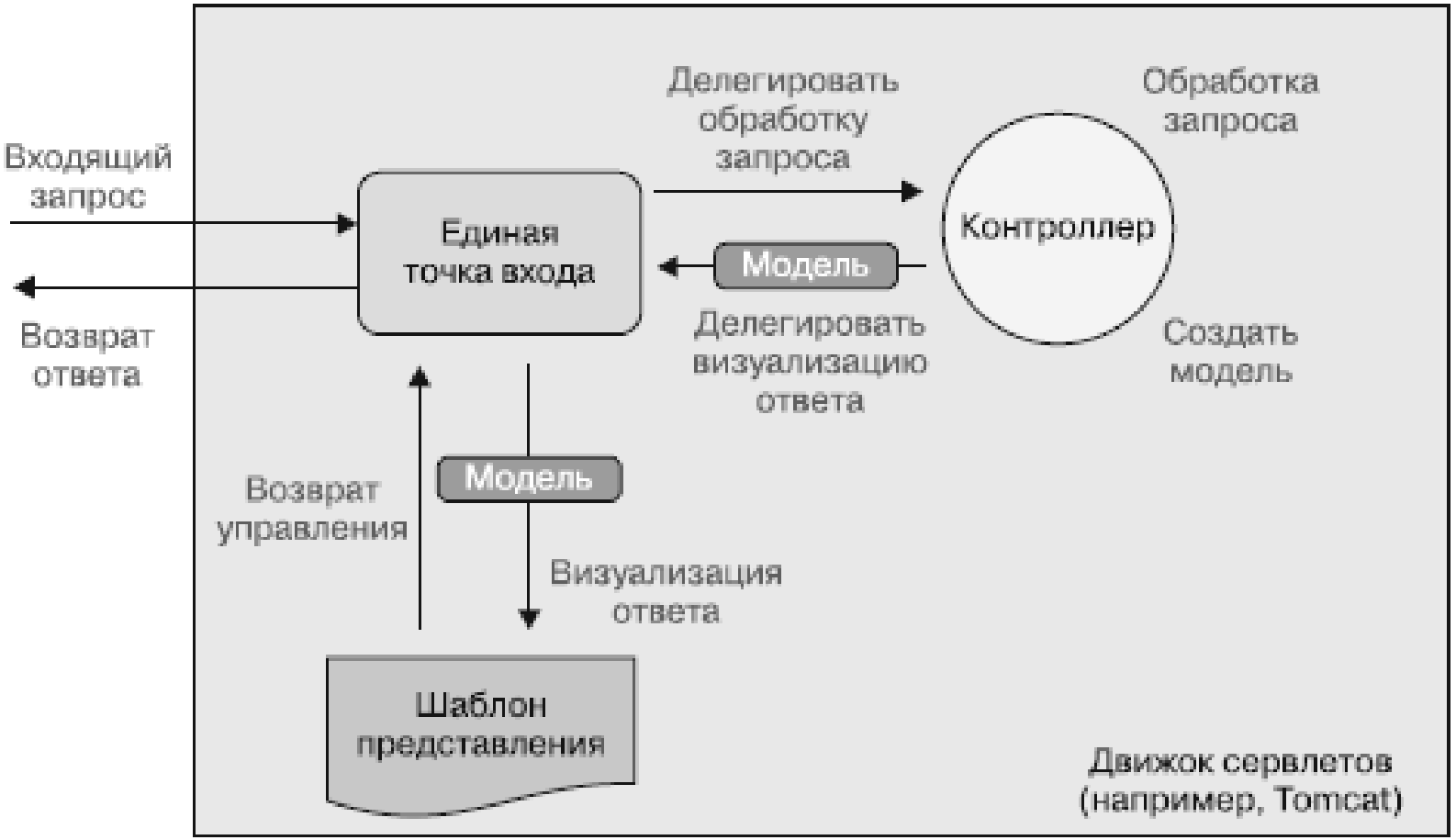
Принцип работы паттерна MVC



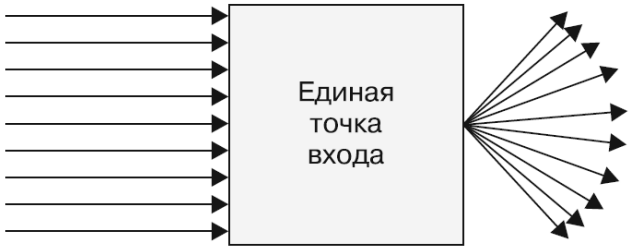
Принцип работы паттерна MVC

- *Модель.* Отвечает за данные для представления, с целью дальнейшей их визуализации в одном из шаблонов представления.
- *Представление.* Отвечает за визуализацию модели в веб-приложении в виде веб-страницы. Оно представляет данные модели в удобочитаемом для пользователя формате.
- *Контроллер.* Код контроллера управляет взаимодействием между представлением и моделью. Такие взаимодействия, как отправка формы или щелчок на ссылке, являются в корпоративном приложении частью контроллера. Кроме того, контроллер отвечает за создание и обновление модели, а также перенаправление модели представлению для визуализации.

Модель 2

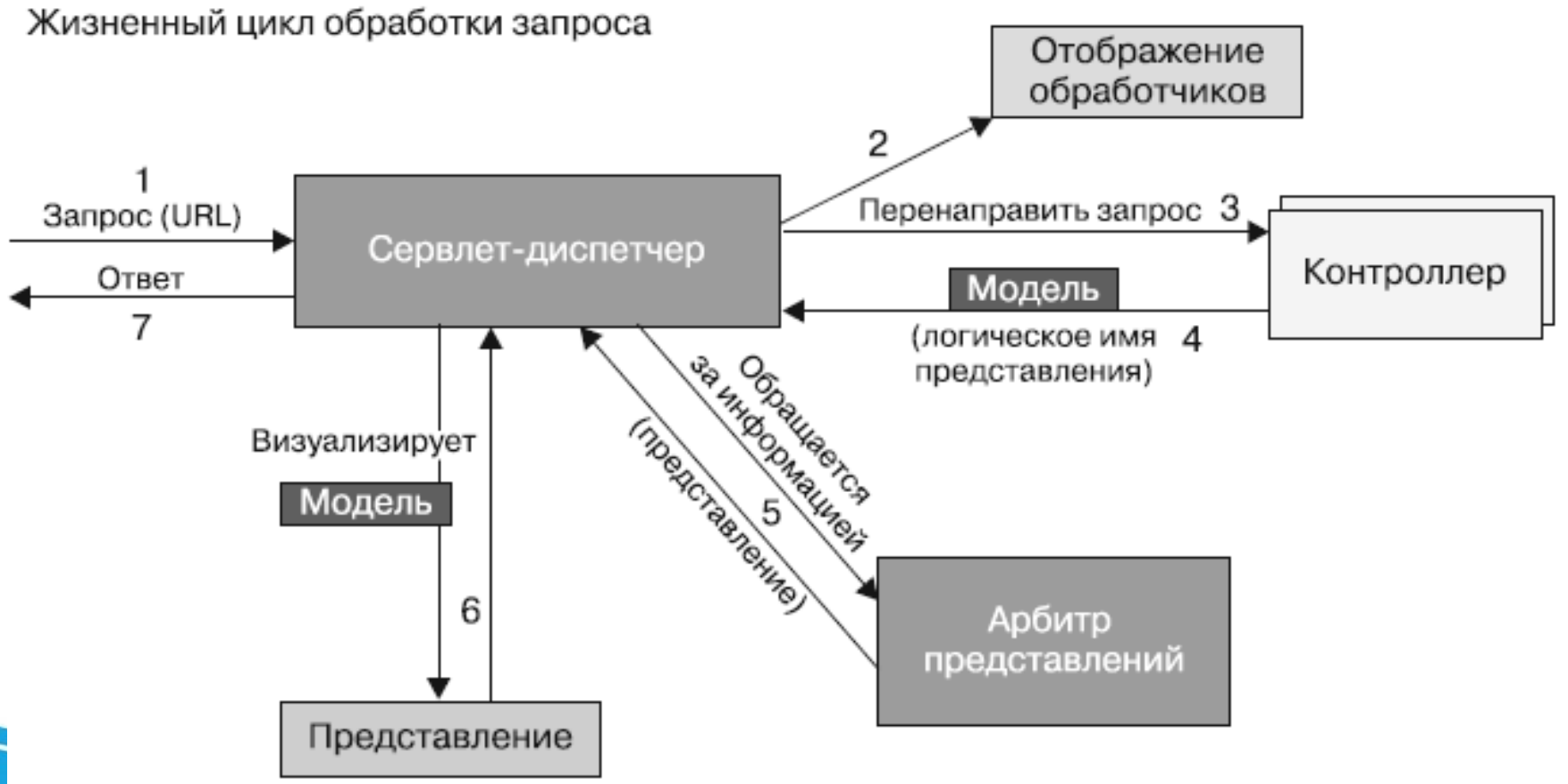

Входящий запрос
Возврат ответа
Единая точка входа
Делегировать обработку запроса
Обработка запроса
Контроллер
Модель
Делегировать визуализацию ответа
Создать модель
Возврат управления
Модель
Визуализация ответа
Шаблон представления
Движок сервлетов (например, Tomcat)

Единая точка входа



- Единая точка входа играет роль основного компонента, маршрутизирующего все запросы. Единая точка входа обеспечивает централизованное управление и расширяет возможности переиспользования и удобство управления, поскольку обычно в веб-контейнере регистрируется только ресурс. Эта точка входа не только распределяет/обрабатывает излишки запросов, но также отвечает за:
    - инициализацию обслуживающего запросы фреймворка;
    - загрузку ассоциативного массива всех URL и компонентов, отвечающих за обработку запроса;
    - подготавливает ассоциативный массив для представлений.

Технологический процесс жизненного цикла запроса



Шаг 1

- Пользователь нажимает на ссылку в браузере или отправляет веб-форму в приложении. Запрос, включающий или какую-то дополнительную, или просто основную информацию, покидает браузер и оказывается в объекте DispatcherServlet фреймворка Spring, представляющем собой просто класс сервлета, как и другие веб-приложения на основе Java. Он представляет собой единую точку входа фреймворка MVC Spring, через которую пропускаются все входящие запросы. Благодаря использованию этой единой точки входа фреймворк MVC Spring централизует управление всем потоком запросов.

Шаг 2

- После поступления запроса в объект DispatcherServlet фреймворка Spring последний передает этот запрос в контроллер MVC Spring, то есть контроллер приложения. Хотя в веб-приложениях Spring может быть несколько контроллеров, но каждый запрос должен попасть к одному из них. Для этого объект DispatcherServlet пользуется отображениями обработчиков, заданными в настройках веб-приложения. Отображение обработчиков определяет конкретный контроллер по URL и параметрам запроса.

Шаг 3

- После выбора нужного контроллера приложения объектом DispatcherServlet с помощью настроек отображения обработчиков DispatcherServlet направляет запрос этому контроллеру. Именно этот контроллер на самом деле отвечает за обработку информации в соответствии с запросом пользователя и его параметрами.

Шаг 4

- Контроллер фреймворка MVC Spring выполняет бизнес-логику, используя бизнес-сервисы приложения, и создает модель, в которую обертывается отправляемая обратно пользователю и отображаемая в браузере информация. Эта модель несет соответствующую запросу пользователя информацию, но она не форматирована, так что можно использовать любую технологию шаблонов представлений для визуализации в браузере содержащейся в ней информации. Именно поэтому контроллер MVC Spring возвращает, помимо модели, название логического представления. А делает он это потому, что контроллер MVC Spring не привязан ни к какой конкретной технологии представления — JSP, JSF, Thymeleaf и т. д.

Шаг 5

- И снова объект DispatcherServlet модуля MVC фреймворка Spring пользуется помощью арбитра представлений, который настроен в веб-приложении таким образом, чтобы разрешать представления. В соответствии с настройками объекта ViewResolver он выдает реальное имя представления вместо логического имени представления. Теперь у объекта DispatcherServlet есть необходимое для визуализации информации модели представление.

Шаг 6

- Объект DispatcherServlet модуля MVC фреймворка Spring визуализирует модель в представление и генерирует информацию модели в удобочитаемом для пользователя формате.

Шаг 7

- Наконец, на основе этой информации DispatcherServlet создает ответ и возвращает его браузеру пользователя.

Настройка Spring MVC

- web.xml

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.
    servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContextMVC.xml</param-
value>
    </init-param>
    <load-on-startup>1</load-on-startup>
 </servlet>
```

Настройка обработки адресов DispatcherServlet

```
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

Настройка поддержки аннотаций в Spring MVC

- DefaultAnnotationHandlerMapping
- <mvc:annotation-driven/>

Контроллер главной страницы

```
@Controller
public class HomeController {
    @RequestMapping("/")
    public String showHomePage() {
        return "home";
    }
}
```

```
<context:component-scan base-package="ru.testmvcapp"/>
```

Поиск представлений

```
bean id="templateResolver" class="org.springframework.web.servlet
.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"/>
    <property name="suffix" value=".jsp"/>
</bean>
```

FreeMarker

```
<bean id="freeMarkerConfig"
 class="org.springframework.web.servlet.view.
freemarker.FreeMarkerConfigurer">
        <property name="templateLoaderPath" value="/WEB-INF/views"/>
        <property name="defaultEncoding" value="UTF-8"/>
        <property name="freemarkerSettings">
            <props>
                <prop key="default_encoding">UTF-8</prop>
            </props>
        </property>
    </bean>
<bean id="templateResolver" class="org.springframework
        .web.servlet.view.
        reemarker.FreeMarkerViewResolver">
        <property name="suffix" value=".ftl"/>
        <property  name="contentType"  value="text/html;charset=UTF-8"/
>
        <property name="cache" value="false"/>
</bean>
```

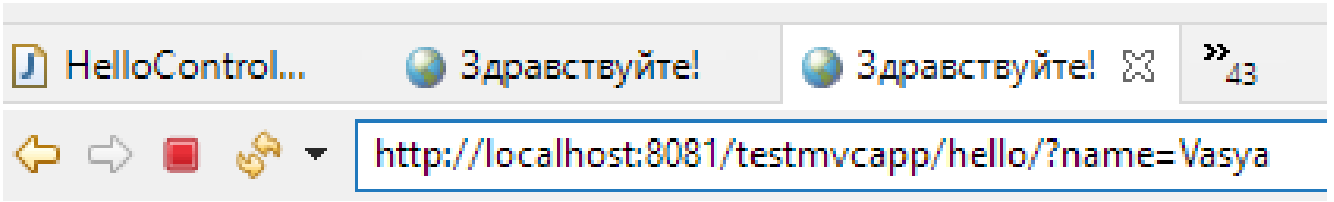
Контроллер обработки входных данных

```
@Controller
@RequestMapping({"/hello"})
public class HelloController {
    @RequestMapping(value="/", method=RequestMethod.GET)
    public String showHelloPage(@RequestParam(value = "name",
    required = false, defaultValue = "незнакомец")
    String name, Model model) {
    model.addAttribute("message", "Привет, " + name + "!");
    return "hello";
    }
}
```

Простейшее представление

```
    <meta charset="UTF-8">
<html>
    <head>
            <title>Здравствуйте!</title>
    </head>

    <body>
            <h2>${message}</h2>
    </body>
    </html>
```



Обработка форм

```
@Controller
@RequestMapping({"/add"})
public class AddStudentController {
    @RequestMapping(method=RequestMethod.GET, params="new")
    public String createStudent(Model model) {
    model.addAttribute(new StudentForm());
    return "studentform";
}
```

Представление формы

- Посмотрите в конспекте ;)

Обработка данных формы

```
@RequestMapping(method=RequestMethod.POST)
public String addStudentFromForm(@Valid StudentForm form,
    BindingResult bindingResult) {
    if(bindingResult.hasErrors()) {
        return "studentform";
    }
    // здесь могло быть сохранение объекта в бд
    return "redirect:add/" + form.getFirstName(); // Переадресовать
}

@RequestMapping(value="/{name}", method=RequestMethod.GET)
public String helloStudent(@PathVariable String name, Model model) {
    model.addAttribute("message", "Привет, " + name + "!");
    return "hello";
}
```

Проверка входных данных

```
    @NotEmpty(message="Поле не должно быть пустым")
private String firstName;

@NotEmpty(message="Поле не должно быть пустым")
private String lastName;

@NotEmpty(message="Поле не должно быть пустым")
private String patronymic;

@NotEmpty(message="Поле не должно быть пустым")
@Pattern(regexp="[2-5]\\.?[0-9]*", message="Проверье правильность ввода числа")
private String avgMark;
```

Некоторые примечания

- <a href="<@spring.url'/add?new'/>">Добавь нового!
- https://freemarker.apache.org/docs/ref_directive_list.html

Спасибо за внимание!