OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

13 – Spring MVC,
Spring MVC на Spring Boot

Дворжецкий Юрий

Как меня слышно && видно?

Если нет – напишите, если слышите – смайлик в чат.

Цели:

- Познакомиться с MVC,
- С архитектурой Spring MVC
- Разберём Controller, Model

Планы:

- Наконец-то веб-приложения и самое интересное!
- Это Вы будете использовать не только на занятиях, но и в боевой жизни.
- ДЗ пока нет. Доделываем что есть.

Совсем чуть-чуть орг.вопросов:

- Задержка 5 дней – немного разгрёб, но не всё.
- Разгребаю.
- Со всех отзыв!

OTUS

Поехали?

OTUS

Spring MVC

Spring MVC

- Spring MVC это Web-framework, основанный на Model-View-Controller архитектуре.
- Spring MVC очень «lightweight».
- Spring MVC имеет простую и понятную структуру.
- Просто расширяется и настраивается.
- Очень популярный, множество примеров, простая документация.

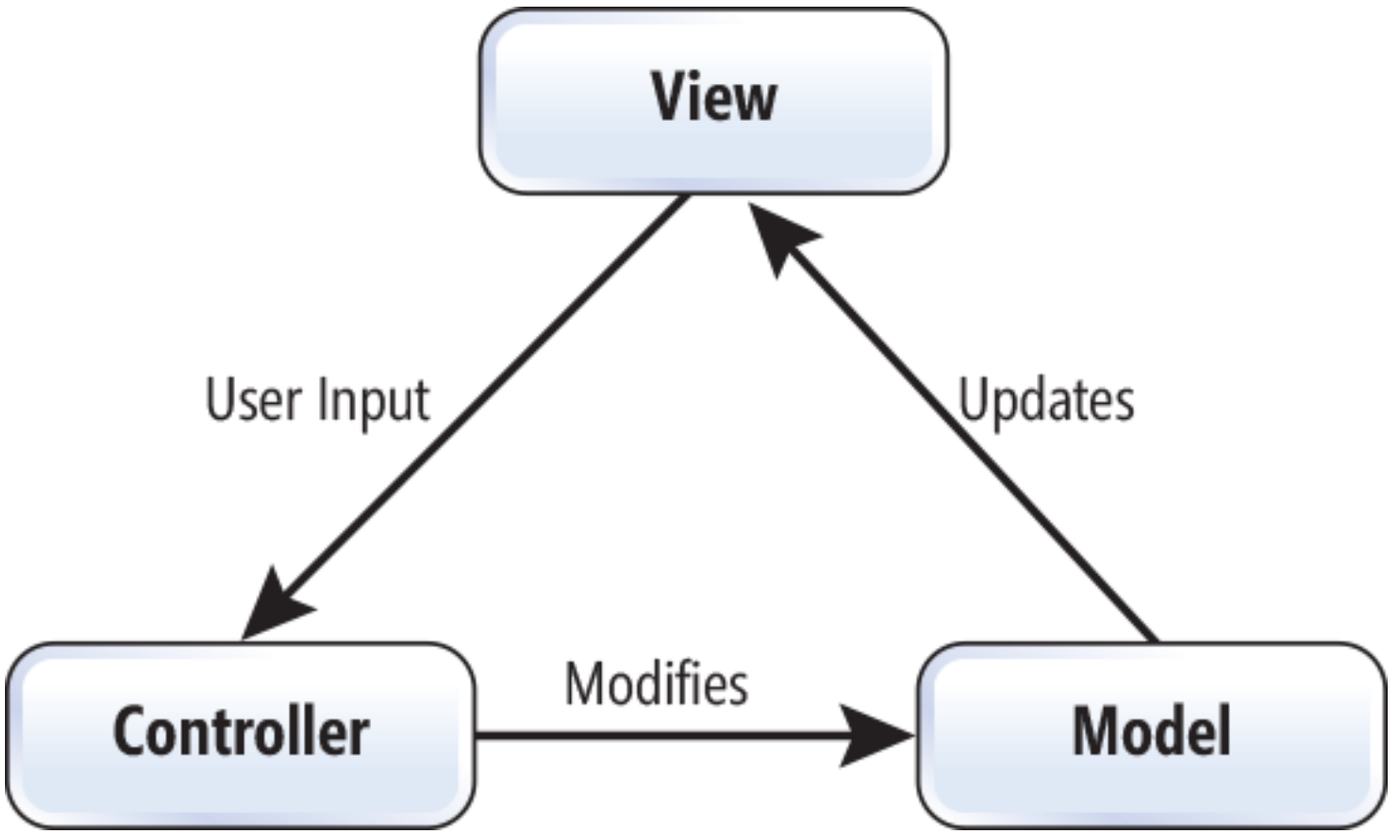
Spring MVC

- Web-framework построенный на принципах Spring
- Базируется на Dispatcher Servlet / Front Controller паттерне
- На нём ооочень просто писать микросервисы

Spring MVC

- Встроенная поддержка:
  - Темы
  - Locales/i18n
  - RESTful services
  - Annotation-based конфигурация
  - Простая интеграция со Spring-сервисами/бинами из контекста

Упражнение, расскажите что в MVC есть что?



MVC

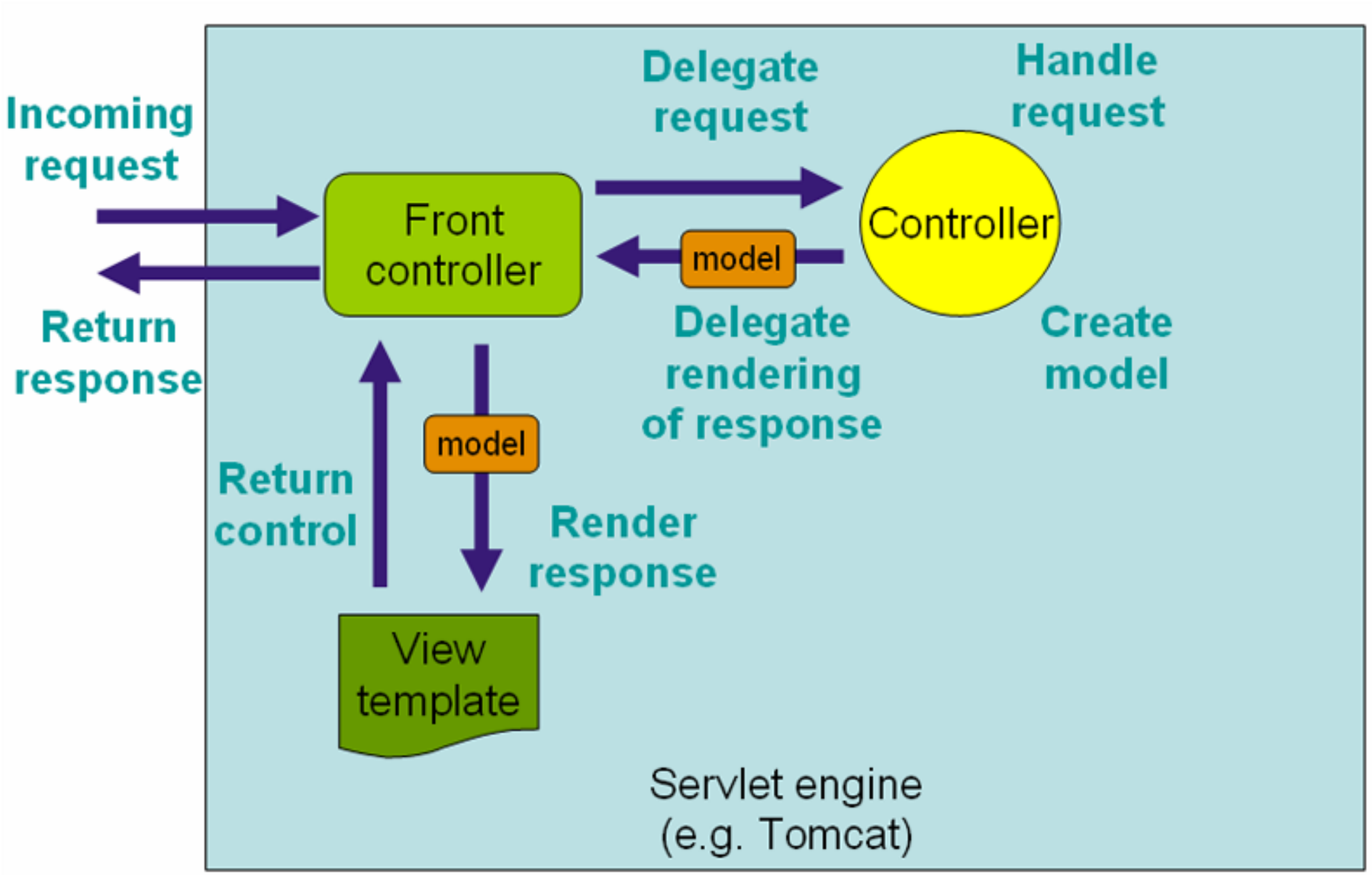
- **Model** представляет динамические данные приложения и методы для изменения состояния.

- **View** отвечает за представление (отображение) информации.

- **Controller** предоставляет канал для общения пользователя и системы. Обрабатывает, что ввёл пользователь на View и предоставляет View данные из Model.

Понятно ли куда класть классы? ☺

Spring's MVC


Incoming request
Delegate request
Handle request
Front controller
model
Controller
Return response
Delegate rendering of response
Create model
Return control
model
Render response
View template
Servlet engine (e.g. Tomcat)

Spring’s MVC: Преимущества

- Чёткое разделение controller-ов, JavaBean model-ей и views.
- Spring's MVC более гибок.
- Spring MVC не зависит от технологии View: технология View может быть заменена множеством альтернатив.
- Лёгкая конфигурация всего необходимого через Spring Bean-ы.

Структура Spring MVC

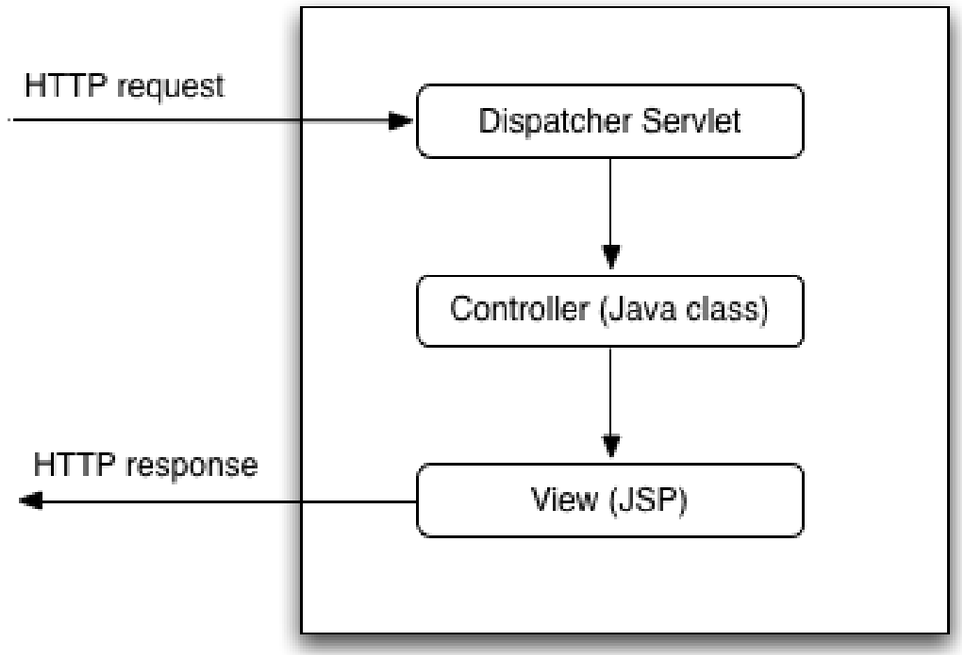
- WebApplicationContext: ApplicationContext адаптированный для Web.
- DispatcherServlet: сервлет для получения всех пользовательских запросов. Это паттерн "Front Controller".
- Model: org.springframework.ui.Model, java.util.Map, хранит данные в виде пар key-value.
- View: класс, реализующий интерфейс View, и соответствующий шаблон.
- Controller: класс, реализующий интерфейс Controller или помеченный @Controller.

Spring MVC Controller

- Spring MVC Controller – не классический MVC Controller.

- Роль классического MVC Controller-а выполняется DispatcherServlet-ом.

- DispatcherServlet получает все запросы и вызывает методы @Controller-ов.



Spring MVC Controller

- Этот контроллер обрабатывает /order?id=1 и показывает страницу заказов.
- И инкапсулирует сервис с бизнес-логикой.

```
@Controller
public class OrdersController {
    @Autowired
    private OrderRepository repo;

    @RequestMapping("/order")
    public String viewOrder(
        @RequestParam(value="id")
        String id,
        Model model       // Model
    ) {
        Order order = repo.get(id);
        model.addAttribute("order", order);
        return "orderView";  // View name
    }
}
```

Spring MVC Controller

- Annotation-based
  - @Controller
  - @RestController (это просто @Controller + @ResponseBody)
- Обычно соответствует какой-то области программы
- Путь запросов задаётся с помощью:
  - @RequestMapping
  - @GetMapping, @PostMapping, и т.д.

JSP

- Здесь пример JSP view.
- JSP нет в Spring Boot с embedded servlet container.
- Но Вы можете использовать JSP, если запакуете Spring Boot приложение в классический WAR

```jsp
<%@ page language="java"
    contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    ...
</head>
<body>
    <p>Hello, ${name}!</p>
</body>
</html>
```

Thymeleaf

- Здесь пример View на Thymeleaf.

- Thymeleaf рекомендованная технология для Spring Boot.

- Но Вы можете использовать и другие:

  - Groovy Markup Templates;

  - Freemarker;

  - Mustache;

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    ...
</head>
<body>
    <p th:text="'Hello, ' + ${name} + '!'">
        Hello, John!
    </p>
</body>
</html>
```

Spring MVC Context

- WebApplicationContext - расширение ApplicationContext я дополнительными возможностями для Web-приложения.

- Добавляет три скоупа бинов: request, session, global (только для портлетов).

- Некоторые бины могут существовать только в WebApplicationContext.

Spring MVC Context

- WebApplicationContext – это интерфейс и имеет несколько реализаций:
  - XmlWebApplicationContext – XML-based
  - AnnotationConfigWebApplicationContext – Java-based
  - GroovyWebApplicationContext – Groovy-based
  - XmlPortletApplicationContext –XML-based для portlet-ов
  - И т.д.

Специальные бины

Бин	Описание
Controller	Обрабатывает пользовательские запросы
Handler mapping	Анализирует URL и направляет запросы контроллерам
View resolver	Определяет какой View должен отобразить ответ пользователю
Locale resolver	Определяет какую локаль использовать
Theme resolver	Определяет тему (visual look)
Multipart file resolver	Поддерживает загрузку файлов
Handler exception resolver	Определяет, как должны обрабатываться исключения

Корневой и сервлет-контексты

- Бины приложения определяются в корневом контексте (services, DAOs, data source, и т.д.)
- Все бины, относящиеся к web (controllers, views, и т.д.) определяются в контексте сервлета.
- Бины из корневого контекста могут использоваться в контекстах сервлетов.
- В спринг буте обычно один сервлет

Конфиигурация

```
@Configuration
@EnableWebMvc
public class WebConfig extends
    WebMvcConfigurerAdapter {
  @Override
  public void addResourceHandlers(
      ResourceHandlerRegistry registry) {
    registry
      .addResourceHandler("/WEB-INF/pages/**")
      .addResourceLocations("/pages/");
  }

  @Bean
  public MyService myService() {
    return new MyService();
  }
}
```

Spring Boot
Какие бины там?)

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(
            Application.class, args
        );
    }

    @Bean
    public MyService myService() {
        return new MyService();
    }
}
```

Spring Boot configuration

- Можно зарегистрировать несколько сервлетов
- А также фильтры и listener-ы и даже с помощью аннотаций:
  - @WebServlet;
  - @WebFilter;
  - @WebListener;
  - @ServletComponentScan.

```
// в "/myServlet/" !
@Bean
public Servlet myServlet() {
    return new MyServletClass();
}

// в "/myServlet/*"
@Bean
ServletRegistrationBean registration() {
    return new ServletRegistrationBean(
        new MyServletClass(), "/myServlet/*"
    );
}
```

Spring Boot configuration

- Есть несколько дополнительных классов, чтобы конфигурировать Spring Boot web-приложение:
  - WebMvcConfigurerAdapter – для MVC-приложение.
  - EmbeddedServletContainerCustomizer – для настройки embedded servlet container.
  - И другие. См. документацию.

Maven (Boot)
изначально spring-mvc

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Вопросы?

Controller

- Controller-ы предоставляют доступ к логике приложения, обычно через интерфейс сервиса.
- Controller-ы могут обрабатывать исключения из бизнес-слоя.
- Controller-ы получают пользовательский ввод, сохраняют данные в модель и выбирают View для представления этих данных в модели.
- Spring MVC реализует Controller-ы абстрактным путём, поэтому Вы можете их написать разными способами.

Пример Controller-a

- Все контроллеры обычно имеют аннотацию @Controller стереотипа (или @RestController).

- Методы, которые получают запросы обычно имеют аннотацию @RequestMapping.

```
@Controller
public class OrdersController {
    @Autowired
    private OrdersRepository repo;

    @RequestMapping("/order")
    public String viewOrder(
        @RequestParam(value="id")
        String id,
        Model model            // Model
    ) {
        Order order = repo.get(id);
        model.addAttribute("order", order);
        return "orderView";  // View name
    }
}
```

Упражнение

- https://github.com/ydvorzhetskiy/spring-framework-13

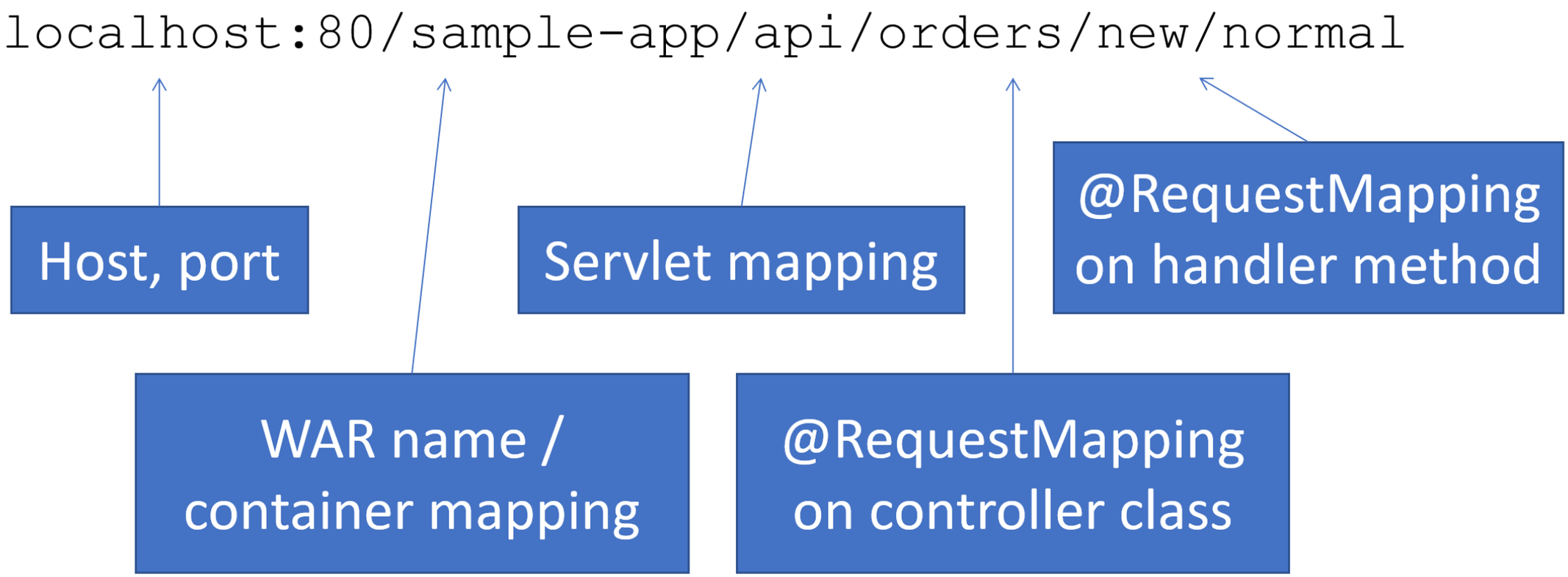
- Сделайте Контроллер (@RestController)

@RequestMapping

- @RequestMapping может стоять на:
  - Controller-классе (опционально);
  - Handler-методе (обязательно).

```
@Controller
@RequestMapping("/orders") // optional
public class OrdersController {

    @RequestMapping(method = RequestMethod.GET)
    public String orders(...) {...}

    @RequestMapping("/view")
    public String viewOrder(...) {...}

    @RequestMapping("/new/normal")
    public String newNormalOrder(...) {...}
}
```

Путь может быть намного длинее…



@PathVariable

- С @RequestMapping и @PathVariable Вы можете передавать данные в пути URL.

- Это полезно для построения RESTful web-сервисов.

```
@Controller
public class OrdersController {

    @RequestMapping("/view/{orderId}")
    public String viewOrder(
        @PathVariable String orderId
    ) {
        ...
    }
}
```

HTTP-методы

```
@Controller
public class SampleController {
    // Все методы
    @RequestMapping("/all")
    public String all() { ... }

    // Только GET
    @RequestMapping("/get", method = RequestMethod.GET)
    public String get() { ... }
}
```

HTTP-методы

- В @RequestMapping Вы можете задавать разные HTTP методы:
  - GET, POST, PUT, PATCH, DELETE.
  - HEAD маппится неявно с GET методами.
  - Для OPTIONS задайте dispatchOptionsRequest = true в Вашем сервлете. Spring MVC автоматически его напишет.
  - TRACE отключён из соображений безопасности.

@RequestMapping

```
@Controller
public class OrdersController {
    @RequestMapping(
        value = {"/a", "b"},
        method = {
            RequestMethod.GET,
            RequestMethod.POST
        }
    )
    public String info() { ... }
}
```

@*Mapping

- Начиная со Spring MVC 4.3 Вы можете использовать следующие аннотации:
  - @GetMapping
  - @PostMapping
  - @PutMapping
  - @DeleteMapping
  - @PatchMapping

```
@Controller
public class OrdersController {

    @GetMapping("/order")
    public String info() { ... }
}
```

Аргументы: @RequestParam

```
@RequestMapping(method = RequestMethod.GET)
public String showOrder(
    @RequestParam("id") long id
) {
    // ...
    return "viewName";
}

@RequestMapping(method = RequestMethod.POST)
public void createOrder(
    @RequestParam(value = "amount", required = false) Integer amount
) {
    // ...
}
```

Аргументы: @RequestParam

- Для получения параметров запроса - используйте аннотацию @RequestParam.

- Spring MVC автоматически обработает параметры запроса и передаст в обработчик.

- Вы также можете задать какие параметры являются обязательными.

Аргументы: POJO

- Вы можете передать в параметр POJO.

- Spring MVC смаппит параметры запроса на поля POJO.

```
class MyRequest {
    private String id;
    private int amount;
    // ...
}


// method?id=1&amount=2


@RequestMapping("/method")
public void method(
    MyRequest request
)
```

Аргументы: @RequestBody

- Если Вы хотите в запросах получать сложные объекты (JSON / XML / другие), то можете использовать @RequestBody аннотацию в Вашем методе.

- Spring Boot имеет сконфигурированный Jackson для конвертации JSON-ов.

```
class User {
    private String name;
    private int age;
    // ...
}

// {"name": "John", "age": 45}

@RequestMapping(
    value = "/create",
    method = RequestMethod.POST
)
public void method(
    @RequestBody User user
) { ... }
```

Аргументы: Model

- Модель – мап
- Но возвращать надо имя вью
- Поэтмоу обычно её принимают в параметры

```
@RequestMapping("/order/view")
public String showOrderPage(
    @RequestParam String orderId,
    Model model
) {
    Order order =
        orderRepository.get(orderId);
    model.addAttribute("order", order);
    return "orderPage";
}
```

Возможные аргументы (полный список в доке)

Аргумент	Описание
@PathVariable	Параметр из пути
@MatrixVariable	Множественные параметры из пути
@RequestParam	Параметры запроса
POJO	Параметры запроса запишутся в поля POJO
@RequestBody	Объект, преобразуемый из тела HTTP запроса специальным конвертером.
@RequestHeader	Значение конкретного заголовка
Model, ModelMap, java.util.Map	Модель в виде пар key-value

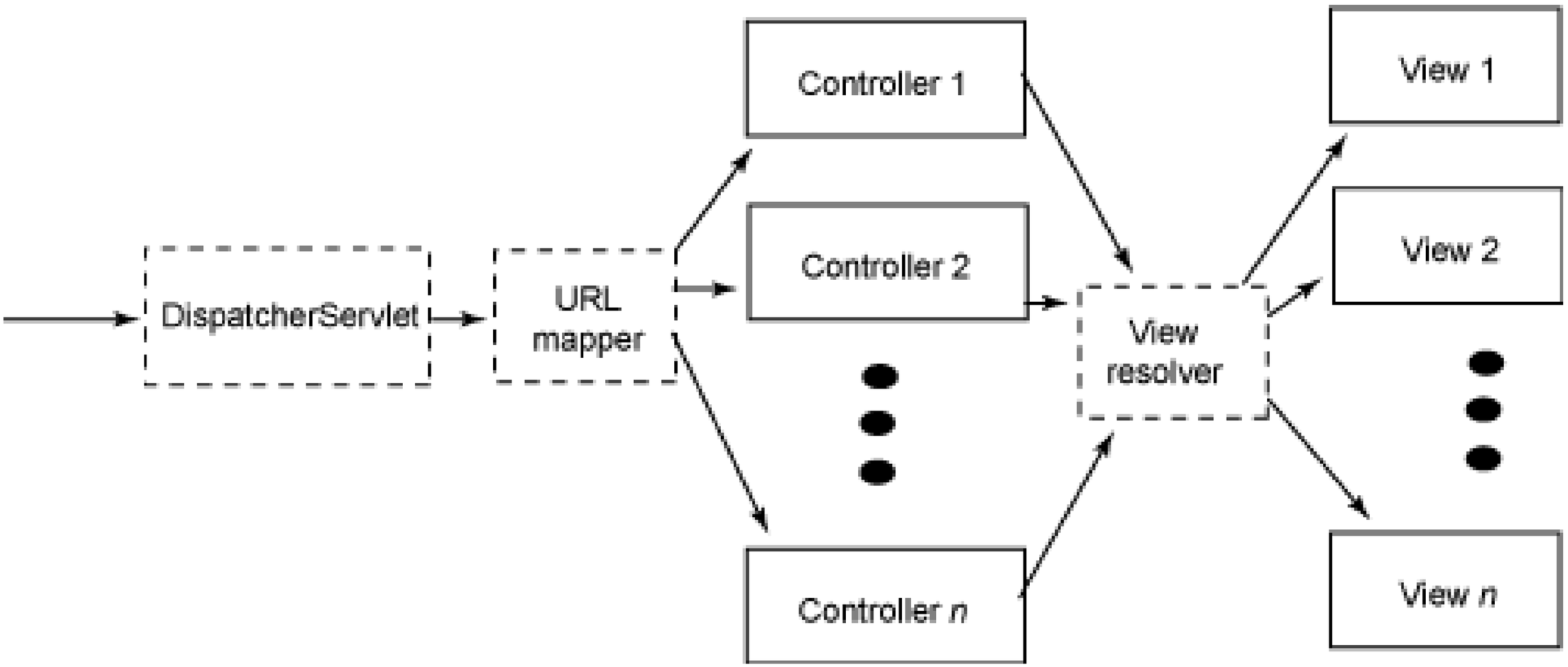
Упражнение

- https://github.com/ydvorzhetskiy/spring-framework-13

- Сделайте GET all

Вопросы?

Controller и View



Controller и View

```
@Controller
public class OrdersController

  @RequestMapping("/order/view")
  public String showOrderPage(
    @RequestParam String orderId,
    Model model
  ) {
    Order order =
      orderRepository.get(orderId);
    model.addAttribute("order", order);
    return "orderPage";
  }
}
```

Controller и View

- Обычно контроллеры страницы возвращают логическое имя View.

- Это имя транслируется ViewResolver-ом в имя шаблона и соответствует класс/технологии для рендеринга страница.

- "orderPage" может означать "orderPage.jsp".

@ResponseBody

- Если вы хотите вернуть данные (JSON / XML / Other) в ответе, Вы можете использовать @ResponseBody аннотацию на типе возвращаемому значению.

- Spring Boot автоконфигурирует Jackson для маршаллинга в/из JSON.

```
@Controller
public class OrdersController

  @RequestMapping("/order")
  public @ResponseBody Order get(
    @RequestParam String orderId,
  ) {
    Order order =
      orderRepository.get(orderId);
    return order;
  }
}
```

@RestController

- Для создания подобных REST контроллеров:
  - @Controller на классе;
  - @ResponseBody на каждом методе.
- Или просто поставить на класс @RestController.
- @RestController просто помечен @Controller и @ResponseBody

```
@RestController
public class OrdersController

  @RequestMapping("/order")
  public Order get(
    @RequestParam String orderId,
  ) {
    Order order =
      orderRepository.get(orderId);
    return order;
  }
}
```

@Controller и @RestController

@Controller

- Возвращает логическое имя View
- Получает чистые HTTP параметры
- Обычно отображает страницы и обрабатывает submit форм
- При ошибке отображается error page (404-page, 500-page с информацией об исключении)

@RestController

- Возвращает объект, который преобразуется в JSON
- получает HTTP параметры или JSON (@RequestBody) , который преобразуется в объект
- Обычно обрабатывает Ajax запросы с UI или запросы от микросервисов
- При ошибке возвращает JSON с кодом и сообщением

Упражнение

- https://github.com/ydvorzhetskiy/spring-framework-13

- GET by ID, POST, PUT, DELETE, PUT (change name)

ResponseEntity

```
@RequestMapping("/ping")
public ResponseEntity<String> ping() {
    return ResponseEntity.ok("OK");
}

@RequestMapping("/badRequest")
public ResponseEntity error() {
    return ResponseEntity
        .status(HttpStatus.BAD_REQUEST)
        .build();
}
```

ResponseEntity

- Можно сделать то же самое с ResponseEntity.
- ResponseEntity – это билдер ответа и может задавать:
  - HTTP status code;
  - response body;
  - Header-ы;
  - ETags

Возможные возвращаемые типы

Возвращаемый тип	Описание
String	Логическое имя view. ViewResolver определяет view по этому имени.
View	View объект.
ModelAndView	View и model.
Model, Map, void	Model, нужный view определится через RequestToViewNameTranslator
void	Когда ответ задаётся через HttpServletResponse аргумент

Возможные возвращаемые типы

Возвращаемый тип	Описание
@ResponseBody	Объект, который будет преобразовываться в JSON/XML/Other с помощью конвертера.
HttpEntity<?> / ResponseEntity<?>	Response entity, возможно, содержащее Ваш объект, который тоже будет конвертироваться
POJO	single-object model.
HttpHeaders	Для возврата response без тела.
Callable<?> / DefferedResult<?> / ListenableFuture<?>	Асинхронный ответ Spring MVC

@ExceptionHandler

- @ExceptionHandler аннотация задаёт exception, который будем обрабатывать.
- Должен соответствовать параметру метода.

@ExceptionHandler

```java
@Controller
public class MyController {

  @ExceptionHandler(NullPointerException.class)
  public ModelAndView handleNPE(NullPointerException e) {
      ModelAndView modelAndView = new ModelAndView("err500");
      modelAndView.addObject("message", e.getMessage());
      return modelAndView;
   }
}
```

MockMvc

```
@RunWith(SpringRunner.class)
@WebMvcTest(VehicleController.class)
public class ControllerTests {
  @Autowired
  private MockMvc mvc;

  @MockBean
  private VehicleService vehicleService;

  @Test
  public void test() throws Exception {
    given(vehicleService.getDetails("id"))
        .willReturn(new Vehicle("Honda"));
    this.mvc.perform(get("/id/vehicle"))
        .andExpect(status().isOk())
    .andExpect(content().string("Honda"));
  }
}
```

Вопросы?

Упражнение

- https://github.com/ydvorzhetskiy/spring-framework-13

- Exception handler

Вопросы?

Домашнее задание

нет

Вопросы?

Пожалуйста, пройдите опрос

https://otus.ru/polls/1644/

Спасибо
за внимание!

Spring MVC Вам!

OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

14 – Spring MVC View

Дворжецкий Юрий

Как меня слышно && видно?

Если нет – напишите, если слышите – смайлик в чат.

Цели:

- Поговорим за жинь веб-приложений
- Познакомиться с View в Spring MVC
- Посмотрим на JSP, Freemarker
- Изучим Thymeleaf

Планы:

- Наконец-то совсем веб-приложения!
- По идее это классический подход.
- Современный подход рассмотрим отдельно
- Но это встречается и в супер-современных приложениях ☺
- ДЗ есть.

Совсем чуть-чуть орг.вопросов:

- Оооочень нравятся работы!
- Задержки с проверкой нет.
- Напомню, что срок проверки 3 дня.
- Со всех отзыв!

«Наши звёздочки»

Sergey Skomorokhov

- Написал библиотеку книг

  с переключением JDBC, JPA и Spring JPA в рантайме

  Поздравляем!!!!!!

OTUS

Поехали?

OTUS

Spring MVC View

Классические приложения (jQuery)

1. HTML с данными - для каждой страницы (генерируется сервером –JSP/Thymeleaf, PHP)
2. JS - подключается в HTML, анимирует HTML с данными
3. CSS - подключается в HTML, задаёт стили отображения

Шаблонные фреймворки

1. Небольшая статическая HTML
2. Данные загружаются с сервера отдельно AJAX
3. HTML шаблоны – статические фрагменты компонентов HTML, отображающие данные
4. JS компоненты - подключаются в HTML, добавляют поведение к HTML шаблонам
5. CSS – подключается в главной HTML

Фреймворки типа

1. Минимальная HTML, только чтобы подключить JS и CSS (SPA – Single Page Application) и обычно статическая
2. Данные загружаются с сервера отдельно
3. JSX компоненты – содержат и JS код и подобие HTML разметки для каждого компонента
4. CSS – подключается в HTML
5. Переход по страницам осуществляется с помощью JS (!)

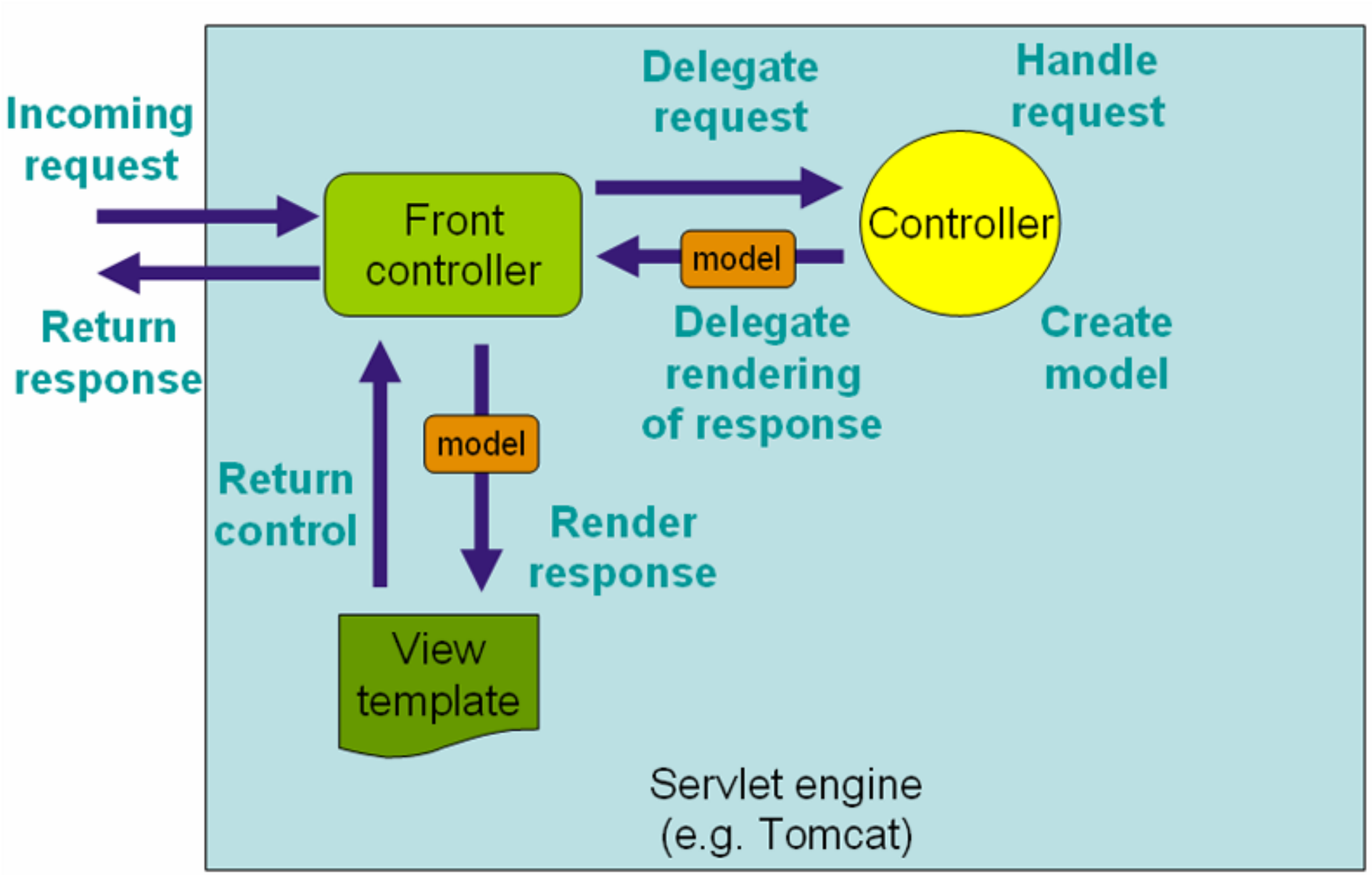
Классические приложения

- Сейчас мы с Вами будем проходить классические приложения.

- Это не устаревшее, они прекрасно решают свои задачи.

- Более того, то что пройдём сегодня, часто встречается и в очень современных приложениях.

Spring's MVC


Incoming request
Delegate request
Handle request
Front controller
model
Controller
Return response
Delegate rendering of response
Create model
Return control
model
Render response
View template
Servlet engine (e.g. Tomcat)

Classic Controller

```
@Controller
public class OrdersController {
    @Autowired
    private OrderRepository repo;

    @RequestMapping("/order")
    public String viewOrder(
        @RequestParam(value="id")
        String id,
        Model model        // Model
    ) {
        Order order = repo.get(id);
        model.addAttribute("order", order);
        return "orderView";  // View name
    }
}
```

Controller и View

- Обычно контроллеры страницы возвращают логическое имя View.

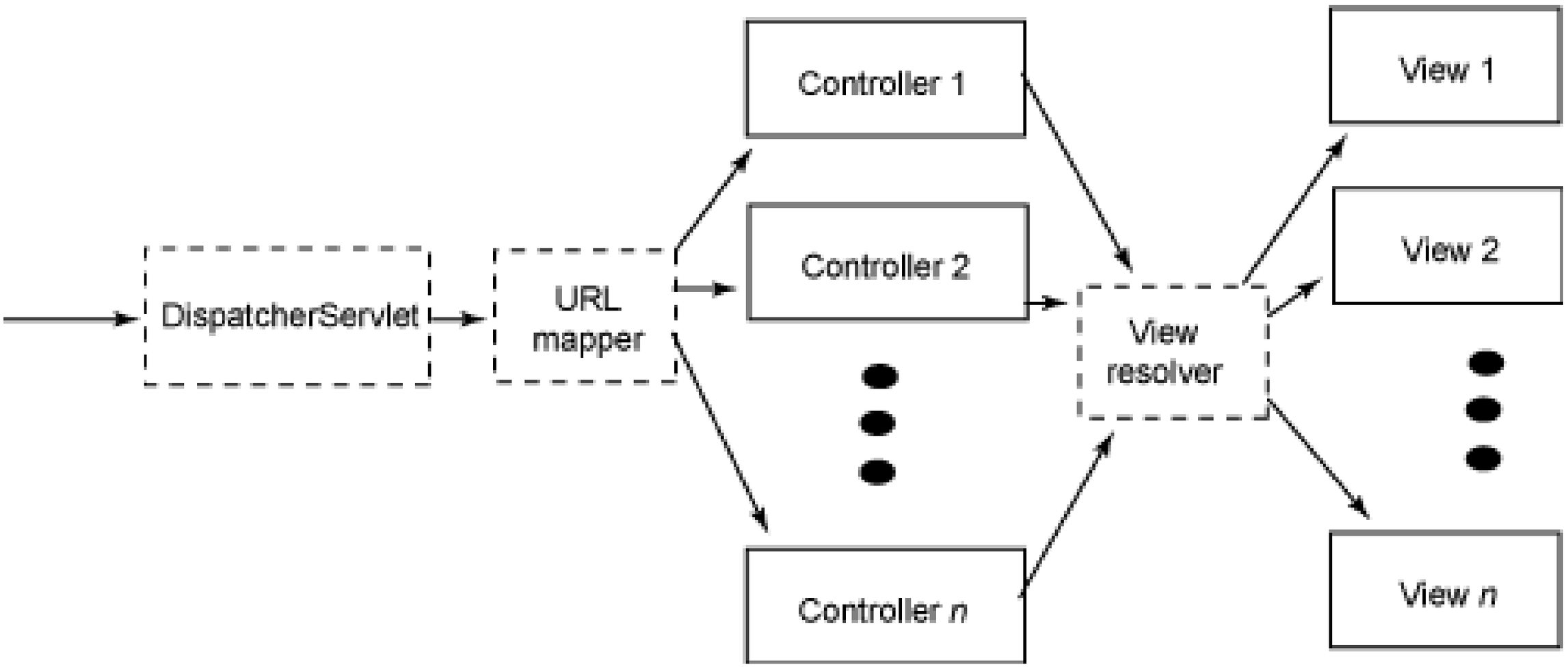
- Это имя транслируется ViewResolver-ом в имя шаблона и соответствует класс/технологии для рендеринга страница.

- "orderPage" может означать “WEB-INF/orderPage.jsp".

HTTP-методы

- Методом GET обычно получается страница (соответствующий хэндлер – возвращает имя вью)
- Методом POST обычно присылаются сабмитом данные формы (иногда делает редиректы)
- Поэтому обычно ограничиваются обработкой этих методов
- Часто страницы ещё используют AJAX, поэтому может понадобиться @ResrtController

Controller и View



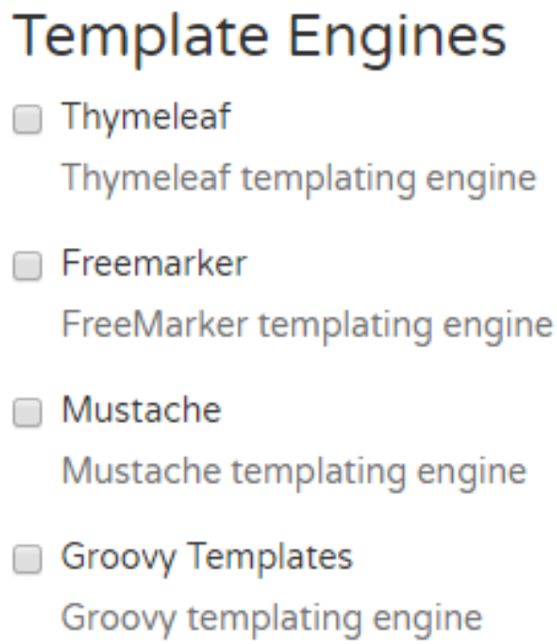
VIEW

- View предназначено для отображения модели

- Контроллер возвращает логическое имя View, а ViewResolver выбирает конкретный View.

View в Spring MVC

- Spring MVC поддерживает несколько технологий View:
  - JSP (с ограничениями в Spring Boot);
  - Thymeleaf;
  - Groovy Markup Templates;
  - Freemarker;
  - Velocity (old);



JSP

- Здесь пример JSP view.

- JSP нет в Spring Boot с embedded servlet container.

- Но Вы можете использовать JSP, если запакуете Spring Boot приложение в классический WAR

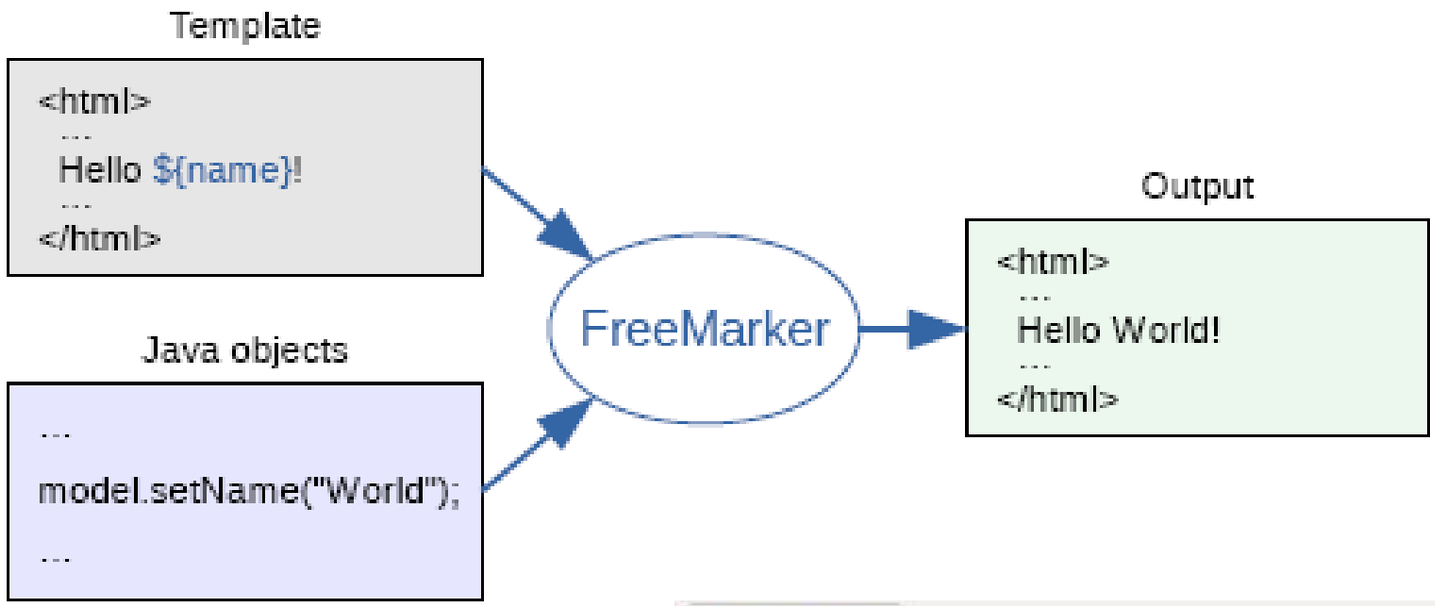
JSP

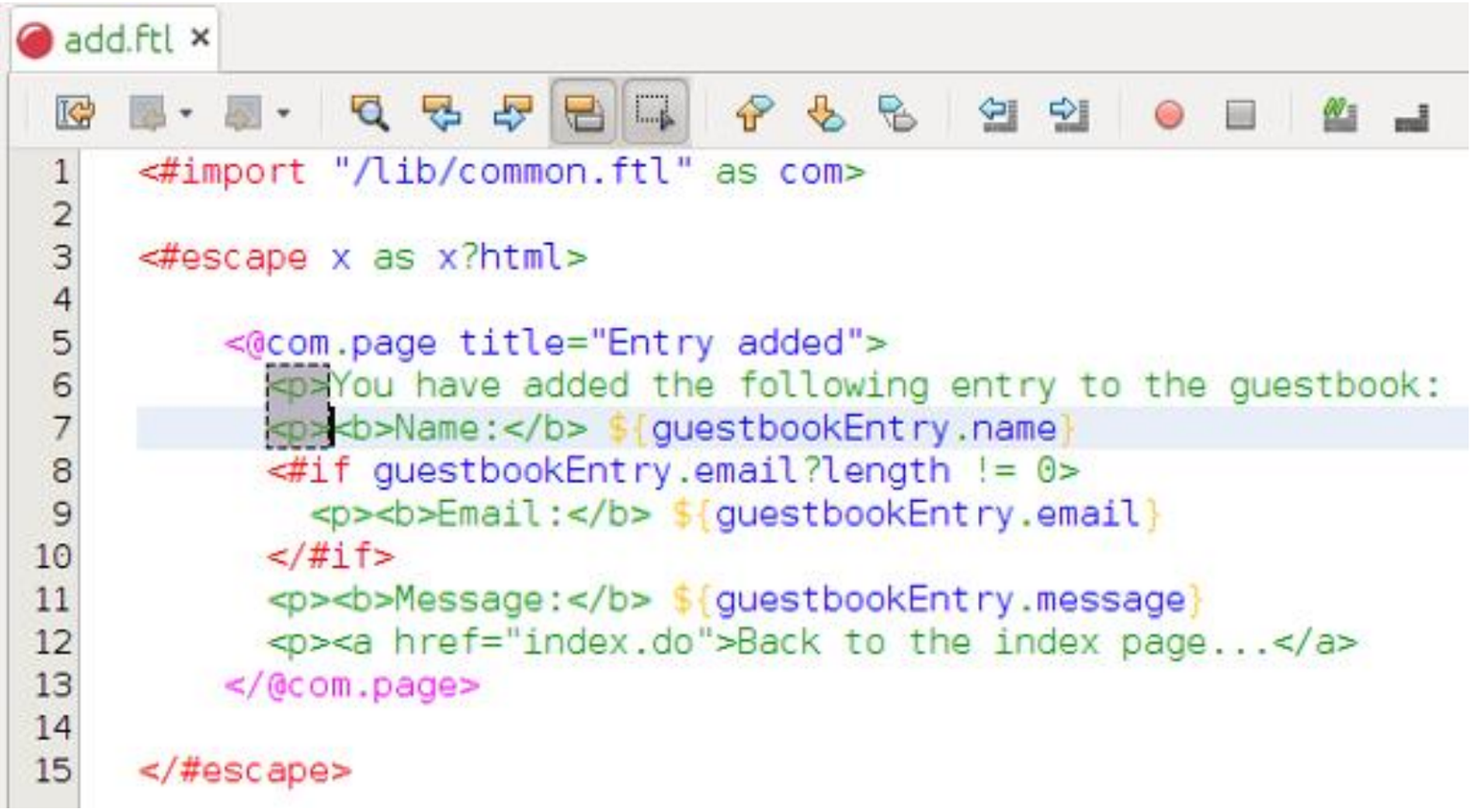
```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
        <title>JSP Page</title>
</head>
<body>
    <%! int result; %>

    <jsp:useBean id="calcul" scope="page" class="Beans.Calculator" />

    <jsp:setProperty name="calcul" property="lhs" param="lhs" />
    <jsp:setProperty name="calcul" property="rhs" param="rhs" />
    <jsp:setProperty name="calcul" property="oper" param="oper" />

    <h1>Resultado:</h1>
    <h2><%= calcul.getResult() %></h2>
    <br/>
```

JSP

- Джавовый аналог ASP, ну или PHP
- Старовата, реализуется (включается) App-сервером
- Но огромные возможности, и много где используется
- Есть ряд недостатков
- И она выпилена из embedded servlet container режима
- Возвращаться к нему – не очень.

Freemarker


Template
<html>
...
Hello ${name}!
...
</html>
Java objects
...
model.setName("World");
...
FreeMarker
Output
<html>
...
Hello World!
...
</html>


add.ftl
1 <#import "/lib/common.ftl" as com>
2
3 <#escape x as x?html>
4
5 <@com.page title="Entry added">
6 <p>You have added the following entry to the guestbook:
7 <p>Name: ${guestbookEntry.name}
8 <#if guestbookEntry.email?length != 0>
9 <p>Email: ${guestbookEntry.email}
10 </#if>
11 <p>Message: ${guestbookEntry.message}
12 <p>Back to the index page...
13 </@com.page>
14
15 </#escape>

Упраженение

- https://try.freemarker.apache.org/

- Вывести Hello, Ivan! Goodbye cruel world!
  Ivan и cruel – параметры.

Вопросы?

Thymeleaf

- Spring community рекомендует Thymeleaf как View для Spring Boot приложений.

- Кстати, используется и для ренедринга XML (!)

- Thymeleaf (да как и другие шаблонизаторы) обладают похожим на JSP синтаксисом

- HTML шаблоны (в Spring Boot) располагаются в src/main/resources/templates/

- А вот ключевой принцип Thymeleaf совсем другой

Thymeleaf и JSP

Thymeleaf Template

```
<!DOCTYPE HTML>
<html
xmlns:th="http://www.thymeleaf.org">

...

<span th:text="Hello, ${name}">
    Hello, Username
</span>
```

page.jsp (JSP)

```
<%@page ...>
<%@taglib ...>

...

<span>
    Hello, ${name}
</span>
```

Thymeleaf и JSP

Thymeleaf

- Валидная HTML с примерным текстом
- Файл корректно обрабатывается браузерами
- Просто дизайнить
- Stronger team collaboration

JSP

- Невалидная HTML
- Не может быть просмотрена не в приложении
- Очень сложно проводить профессиональный web-дизайн

Thymeleaf и JSP

Thymeleaf Template

```
<!DOCTYPE HTML>
<html
xmlns:th="http://www.thymeleaf.org">

...

<span th:text="Hello, ${name}">
    Hello, Username
</span>
```

page.jsp (JSP)

```
<%@page ...>
<%@taglib ...>

...

<span>
    Hello, ${name}
</span>
```

Thymeleaf Starter

- spring-boot-starter-thymeleaf:
  - thymeleaf-spring4
  - thymeleaf-layout-dialect

```
<dependency>
   <groupId>
        org.springframework.boot
   </groupId>
   <artifactId>
        spring-boot-starter-thymeleaf
   </artifactId>
</dependency>
```

Thymeleaf: Syntax

- th:text
  - Текст вместо текста в теге, когда приложение будет запущено
- #{localized.hello} локализационное сообщение

```
page.html:

<span th:text="#{localized.hello}">
    Hello!
</span>


messages.properties:

localized.hello=Hello!
```

Thymeleaf: Syntax

- Параметризованные сообщения.

- Что будет выводиться на странице?

page.html:

```
<span th:text="#{localized.hello('Mike')}">
    Hello, Username!
</span>
```

messages.properties:

```
localized.hello=Hello, {0}!
```

Вопросы?

Локализация Spring MVC

- Для того, чтобы локализовать приложение Spring MVC должен уметь осуществлять 3 действия:
  - Определять пользовательскую локаль
  - Менять пользовательскую локаль
  - Отображать локализованный текст/сообщения на странице
- Эти действия могут быть выполнены следующими компонентами:
  - LocaleResolver
  - LocaleChangeInterceptor
  - Собственно, локализационными сообщениями и MessageResolver-ом.

Локализация: LocaleResolver

Чтобы определить какая локаль у пользователя применяются:

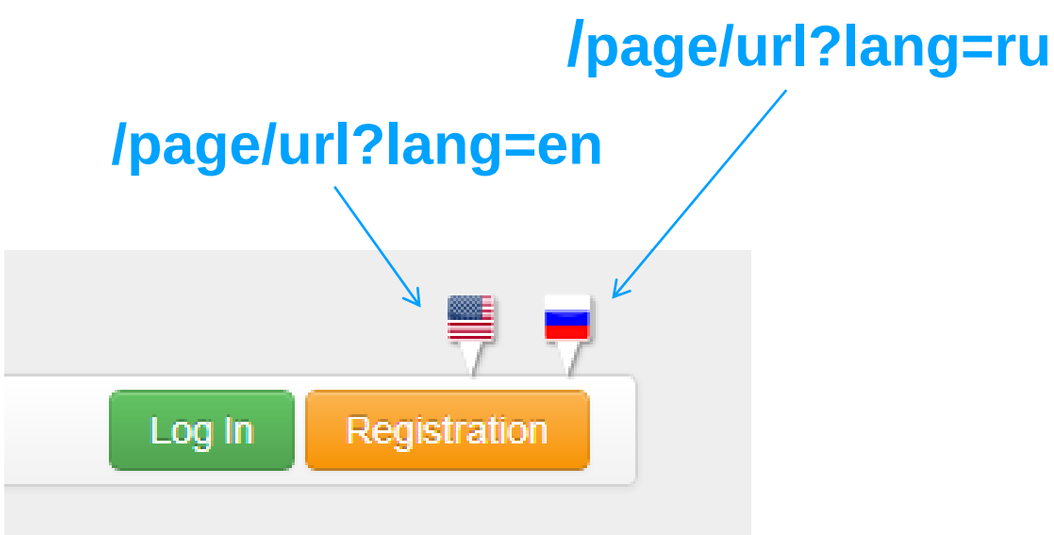
- AcceptHeaderLocaleResolver (на основании заголовка "Accept-Language")
- CookieLocaleResolver (текущая локаль сохраняется в cookie)
- SessionLocaleResolver (текущая локаль сохраняется в сессии)

```
// CookieLocaleResolver

@Bean
public LocaleResolver resolver() {
  CookieLocaleResolver resolver
    = new CookieLocaleResolver();
  resolver.setDefaultLocale(
    new Locale("en"));
  resolver.setCookieName("locale");
  return resolver;
}
```

Локализация: LocaleChangeInterceptor

- LocaleChangeInterceptor позволяет пользователю менять локаль через переходы по ссылкам:

```
@Bean
public LocaleChangeInterceptor lci()
{
  LocaleChangeInterceptor lci
    = new LocaleChangeInterceptor();
  lci.setParamName("lang");
  return lci;
}
```



Локализация: LocaleChangeInterceptor

- LocaleChangeInterceptor (как и другие interceptor-ы) должны добавляться в конфигурацию сервлета.

```
@Configuration
public class WebConfig extends
    WebMvcConfigurerAdapter {

  // ...

  private LocaleChangeInterceptor lci;

  @Override
  public void addInterceptors(
    InterceptorRegistry registry
  ) {
    registry.addInterceptor(lci);
  }
}
```

Локализация: MessageSource

```
@Bean
public MessageSource messageSource() {
  ReloadableResourceBundleMessageSource ms
    = new ReloadableResourceBundleMessageSource();
  ms.setBasename("/i18n/bundle");
  ms.setDefaultEncoding("UTF-8");
  return ms;
}
```

Локализация: сообщения

bundle_en.properties:

```
localized.hello=Hello!
```

page.html:

```
<span th:text="#{localized.hello}">
    Hello!
</span>
```

Локализация в Spring Boot

- Spring Boot автоконфигурирует LocaleResolver и MessageSource.
- По умолчанию MessageSource берёт бандлы в src/main/resources :
  - src/main/resources/messages.properties – default locale (required!)
  - src/main/resources/messages_en_US.properties – конкретные локали
- Все файлы читаются в UTF-8 (круто!)
- Вы можете поменять местонахождение бандлов в application.yml:
  - spring.messages.basename: path/to/i18n/messages

Thymeleaf: синтаксис

```
// Controller

@RequestMapping("/hello")
public String hello(
  @RequestParam String name,
  Model model
) {
  model.addAttribute(
    "username", name
  );
  return "helloPage";
}
```

helloPage.html:

```
<span th:text="${username}">
    Hello, Username!
</span>
```

Model и View

- Вы можете помещать в модель сложные объекты.

- ${user.name} – означает Spring Expression Language (SpEL).

- Со SpEL Вы можете получать поля объекта.

```
helloPage.html:

<span th:text="${user.name}">
    Hello, Username!
</span>


// Java code:

class User {
    private String name;
    ...
}
model.addAttribute("user", user);
```

Вопросы?

Упражнение

- https://github.com/ydvorzhetskiy/spring-framework-14

- Вывести информацию о пользователе

Thymeleaf: синтаксис

- Вы можете комбинировать выражения

- th:utext
  – unescaped text

```
page.html:

<span th:utext="
    #{localized.hello(${user.name})}
">
    Hello, John Doe!
</span>


messages.properties:

localized.hello=Hello, {0}!
```

SpEL

- SpEL предоставляет различные возможности получения данных

- И Вы можете вызывать методы у объекта

```
${person.father.name}

${person['father']['name']}

${personsArray[0].name}

${person.createCompleteName()}

${person.createWithSeparator('-')}
```

Thymeleaf: синтаксис

- *{name} то же что и ${user.name} но выражение считается в контексте user
- th:object задаёт контекст *{…}
- Если объект не задан, то ${…} и *{…} эквивалентны

```
page.html:

<div th:object="${user}">
    <span th:text="*{name}">
        John Doe
    </span>
</div>


class User {
    private String name;
    ...
}
```

Thymeleaf: *{} и ${}

```
<div th:object="${user}">
   <span th:text="*{name}">
      John Doe
   </span>
   <span th:text="*{age}">
      25
   </span>
</div>
```

```
<div>
   <span th:text="${user.name}">
      John Doe
   </span>
   <span th:text="${user.age}">
      25
   </span>
</div>
```

Упражнение

- https://github.com/ydvorzhetskiy/spring-framework-14

- Заюзать *{}

Thymeleaf: синтаксис

- С помощью @{/url} Вы можете задавать корректный URL в режиме работы приложения.

- Оригинальный URL будет доступен во время дизайна

```
page.html:

<a href="account.html"
   th:href="@{/account(id=${id})}">
      Account Page
</a>


/page in app:

<a href="/account?id=1">
      Account Page
</a>
```

Thymeleaf: синтаксис

- th:action – action формы
- th:value – значение input-a

page.html:

```
<form th:action="@{/main}">
   <input th:value="${name}"/>
</form>
```

Thymeleaf: синтаксис

- th:each – Thymeleaf foreach

- th:if – рендерится, если условие истинно

```
page.html:

<li th:each="account : ${accounts}">
   <span th:text="${account.owner}">
      John Doe
   </span>
</li>


<div th:if="${@myBean.hasErrors()}">
   <span>ERROR!</span>
</div>
```

Thymeleaf: синтаксис

- @beanName – Spring-бин с именем "beanName"
- #locale – текущая локаль
- #ctx – текущий контекст
- Конкатенация, арифметические и условные операции.
- И куча предопределённых констант и функций.

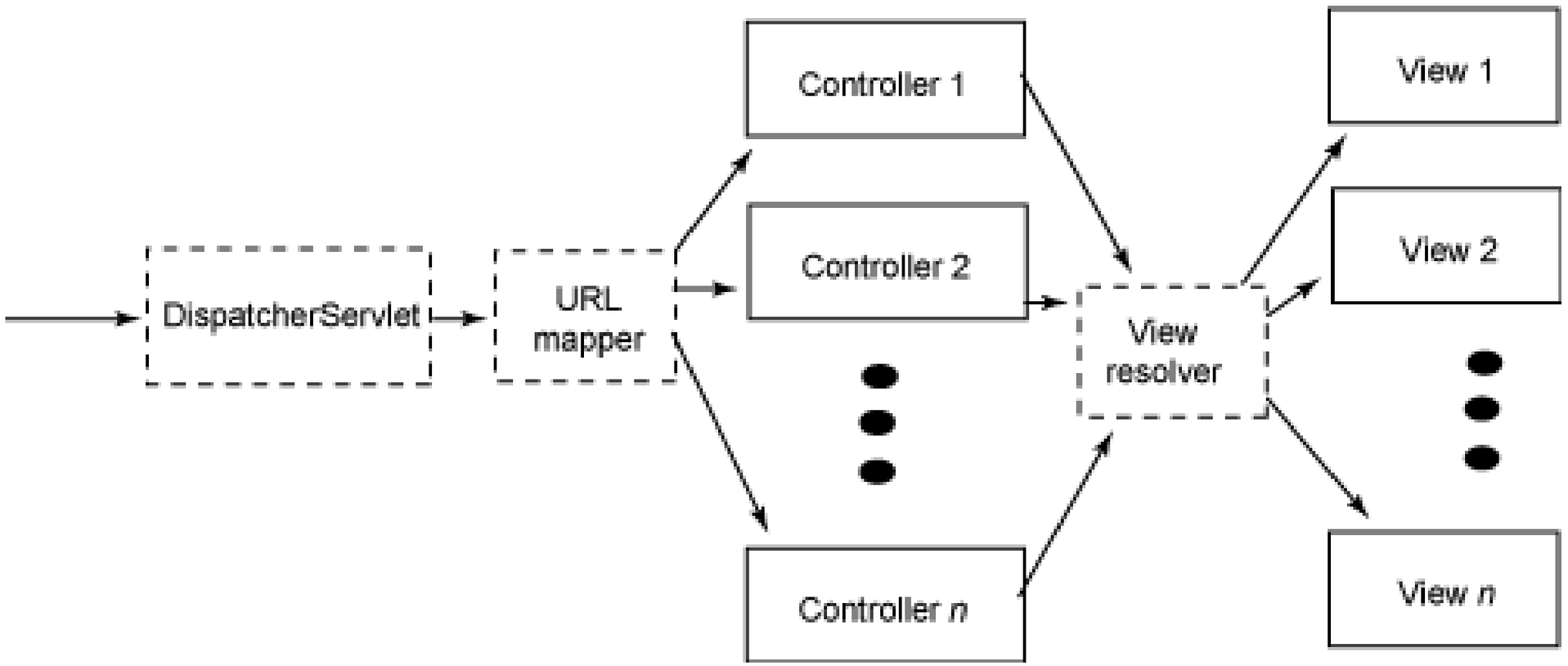
Упражнение

- https://github.com/ydvorzhetskiy/spring-framework-14

- Список Person-ов

Вопросы?

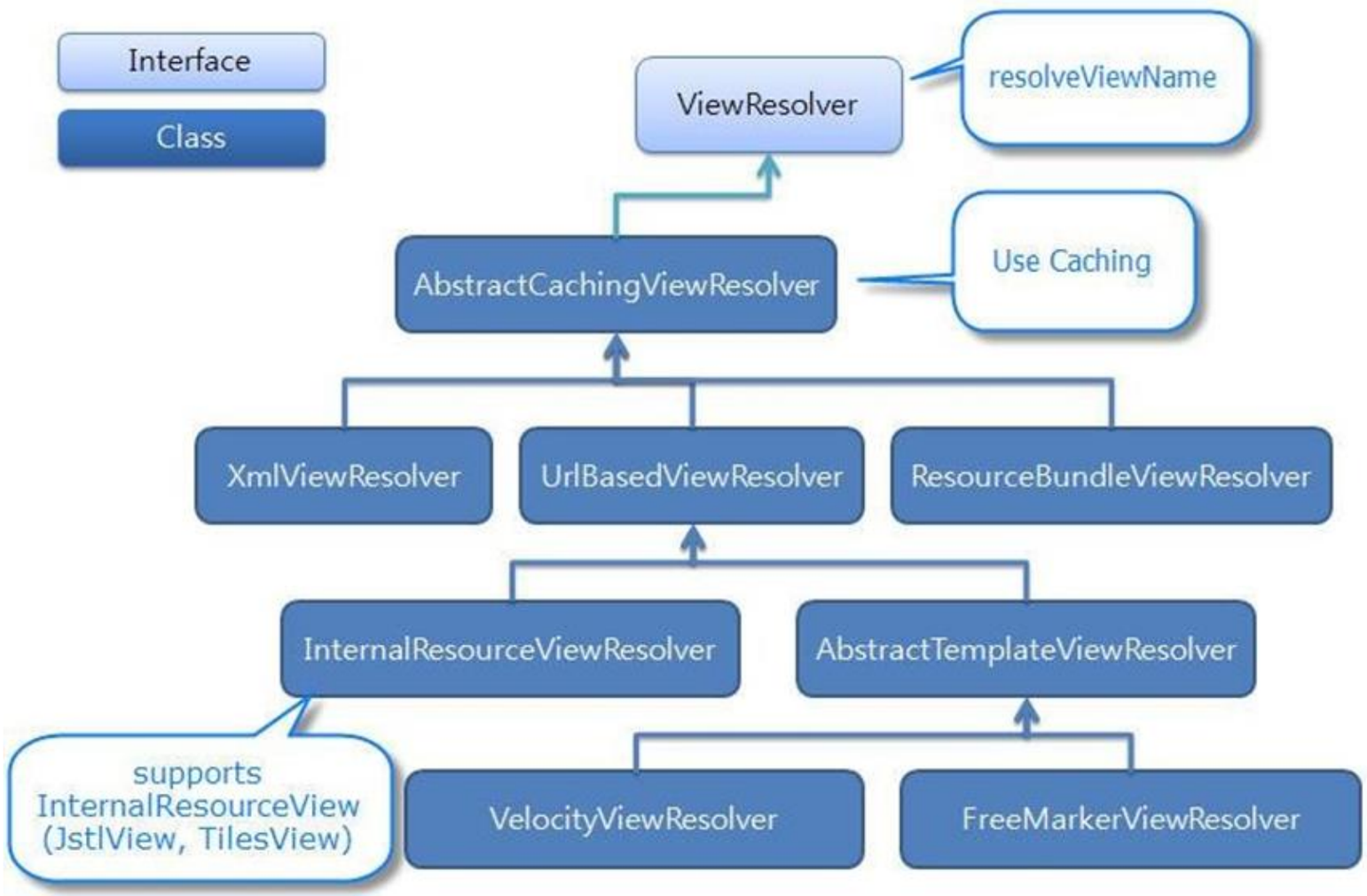
Spring MVC: ViewResolver



Spring MVC: ViewResolver

- ViewResolver – специальный интерфейс Spring MVC
- Они по логическому имени View определяют конкретный шаблон и технологию View, которые нужно рендерить
- В Spring Boot-е ViewResolver-ы автоматически конфигурируются.
- Если Вы добавите spring-boot-starter-thymeleaf в POM, Spring Boot создаст ViewResolver, который логическое имя "home" будет резолвить в "classpath:templates/home.html" и рендерить используя технологию Thymeleaf.

Spring MVC: ViewResolver-ы



ViewResolver: ручная конфигурация

- ViewResolver можно создать вручную.
- Вы можете создавать несколько ViewResolver-ов.
- Некоторые ViewResolver-ы имеют "order" – можно задавать приоритет.

```
@Bean
public ViewResolver viewResolver() {
    InternalResourceViewResolver r
            = new
  InternalResourceViewResolver();
  r.setPrefix("/template/");
     r.setSuffix(".html");
          return r;
          }
```

Статический Web-контент

- Вы можете разместить статический Web-контент (HTML, CSS, картнки) в
  - src/main/resources/static
  - src/main/resources/public
- Spring Boot автоматически смаппит эти ресурсы
  - src/main/resources/public/index.html -> /

Статический Web-контент

- Положив ссылки на CSS и JS Вы можете проставить href для дизайна и th:href для приложения
- Это позволит производить профессиональный дизайн приложения

Вопросы?

Упражнение

- https://github.com/ydvorzhetskiy/spring-framework-14

- * Трушный edit

Вопросы?

Домашнее задание

CRUD приложение с Web UI и хранением данных в БД

Создайте приложение с хранением сущностей в БД (можно взять DAOs из прошлых занятий)

Использовать классический View, предусмотреть страницу отображения всех сущностей и создания/редактирования.

View на Thymeleaf, classic Controllers.

Вопросы?

Пожалуйста, пройдите опрос

https://otus.ru/polls/1666/

Спасибо
за внимание!

Spring MVC Вам!