L4. Spring MVC

Introduction to Spring MVC

Dr A Boronat

Sprint 2: developing a web application

- Goal: mini project - duration ~2 weeks
    - agile development a functional feature for an online shop

Sprint 2: schedule

- Schedule: calendar on Blackboard for sessions
- Todo list:
  - week a: foundations of a web application
    - Spring MVC
    - Spring Boot
  - week b: development of more complex web pages with different views
    - Java Server Pages
    - Validation

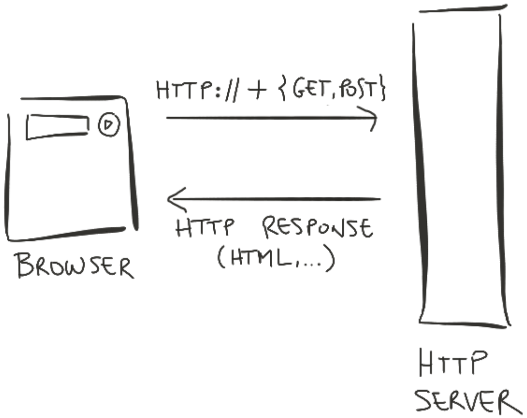
Web application

- Web application: client-server software application in which
  - the client (or user interface) runs in a web browser
  - the application server listens at some URL (base URL) and a port
    - when developing a web application this will be

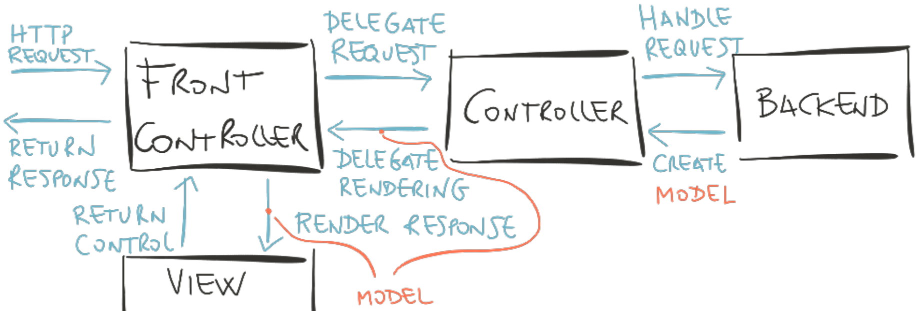
      $$http://localhost:8080$$

      by default
- web applications may contain
  - static content: HTML, images
  - dynamically generated content: HTML produced by JSPs after querying a database

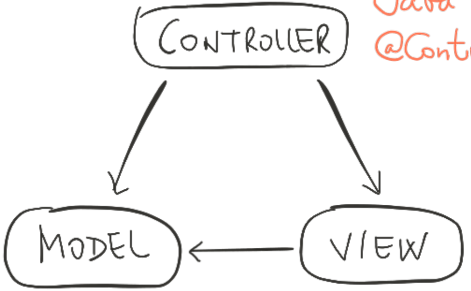
DEALING WITH HTTP REQUESTS



INSIDE THE HTTP SERVER: HTTP REQUEST / RESPONSE LIFECYCLE



MODEL VIEW CONTROLLER



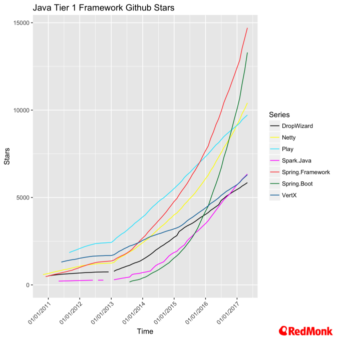
Java class
@Controller

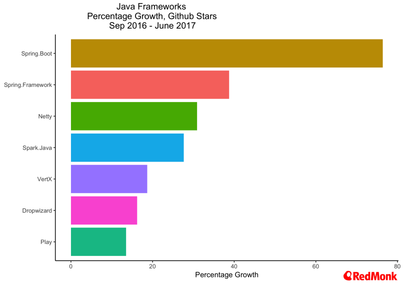
MAP
attribute $\mapsto$ value

Java Server Pages (JSPs)
- HTML
- Expression language
- Tag libraries (JSTL, Spring)

Web development frameworks (Java)

- RedMonk report on Java-based framework popularity (22/06/2017):

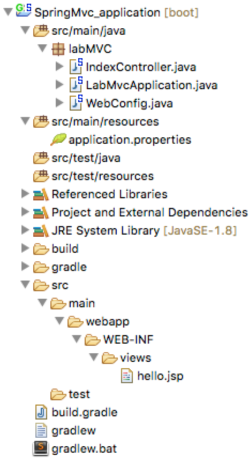




- Spring framework: facilitates the development of enterprise applications
  - can manage Java objects (beans) using dependency injection
  - offers a lot of functionality off-the-shelf (web development support)
- Spring MVC: web component of Spring, implementing MVC
- Spring Boot: convention-over-configuration rapid application development
  - configures Spring wherever possible automatically (opinionated approach)
  - ideal for beginners (no XML configuration)

Spring Boot web application

Project structure



```
▼ SpringMvc_application [boot]
  ▼ src/main/java
    ▼ labMVC
      ▶ IndexController.java
      ▶ LabMvcApplication.java
      ▶ WebConfig.java
  ▼ src/main/resources
      application.properties
    src/test/java
    src/test/resources
  ▶ Referenced Libraries
  ▶ Project and External Dependencies
  ▶ JRE System Library [JavaSE-1.8]
  ▶ build
  ▶ gradle
  ▼ src
    ▼ main
      ▼ webapp
        ▼ WEB-INF
          ▼ views
              hello.jsp
      test
    build.gradle
    gradlew
    gradlew.bat
```

Spring Boot web application

SpringBootApplication class

```
@SpringBootApplication
public class LabMvcApplication {

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

Spring Boot web application

Configuration

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

  @Bean
  public InternalResourceViewResolver viewResolver() {
    InternalResourceViewResolver viewResolver =
      new InternalResourceViewResolver();
    viewResolver.setViewClass(JstlView.class);
    viewResolver.setPrefix("/WEB-INF/views/");
    viewResolver.setSuffix(".jsp");
    viewResolver.setOrder(2);
    return viewResolver;
  }
}
```

Controller

Responsibility: HTTP request handling

- links a HTTP request to a method with an annotation @RequestMapping
- method parameters: get user input
- method body: population of the model
    - business logic (access to database, computations, etc.)
    - determines view
    - interprets exceptions arisen from business logic
- return value: view name

Handling HTTP requests (GET)

- attached to a class: defines relative url http://localhost:8080/hello/

```
@RequestMapping("/hello")
public String hello(Model model) { .. }
```

- attached to a class: defines relative url http://localhost:8080/index/hello/

```
@RequestMapping("/index")
public class IndexController {
  @RequestMapping("/hello")
  public String hello(Model model) { .. }
}
```

Handling HTTP requests (GET): parameters

- using path variables with @PathVariable:
  http://localhost:8080/hello/World

  ```
  @RequestMapping("/hello/{value}")
  public String hello(@PathVariable String value, Model model)
      {...}
  ```

- request parameters: http://localhost:8080/hello?value=World

  ```
  @RequestMapping("/hello")
  public String greetingParam(
    @RequestParam(value="value", required=false,
        defaultValue="World") String value,
    Model model) { ... }
  ```

Handling HTTP requests (GET): type conversion

- getting primitive datatypes: http://localhost:8090/hello?value=1

```
@RequestMapping("/hello")
public String primitive(@RequestParam Integer value, Model
    model) { ... }
```

- getting dates: http://localhost:8090/hello/2016-07-10

```
@RequestMapping("/hello/{value}")
public String date(@PathVariable
    @DateTimeFormat(iso=ISO.DATE) Date value, Model model) {
    ... }
```

- getting collections: http://localhost:8090/hello?values=1&values=2

```
@RequestMapping("/hello")
public String collection(@RequestParam Collection<Integer>
    values, Model model) { ... }
```

Handling HTTP requests (GET)

- Model parameter: allows us to access the model

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

- @ModelAttribute
  - fetches the object associated with the attribute user from the model

    ```
    @RequestMapping("/hello")
    public String hello(@ModelAttribute User user) {
      return "hello";
    }
    ```

  - if the entry is not present in the model, the object is instantiated and added to the model
  - the argument's fields are populated from all request parameters that have matching names

Executing the web application

With Gradle:

- from Mac/Linux: `./gradlew bootRun`
- from Windows: `gradlew.bat bootRun`

With STS: use Boot Dashboard

- enables debugging from STS
- fast application restarts/reload static content

To use resources from Pluralsight

- Slides and examples used in tutorial available under tab Exercises
- Follow the guide on GitHub to avoid XML configuration
  - Gradle: no need to deal with Maven POMs directly (XML)
  - Spring Boot: automated configuration

What's next?

- Resources from Pluralsight
- Exercise 1: setting up for first Spring Boot application
- Exercises from Pluralsight
- Exercise 2
- Mini project: get your web app up and running
- Lab session on Monday