L4. Spring MVC

Introduction to Spring MVC

Dr A Boronat

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

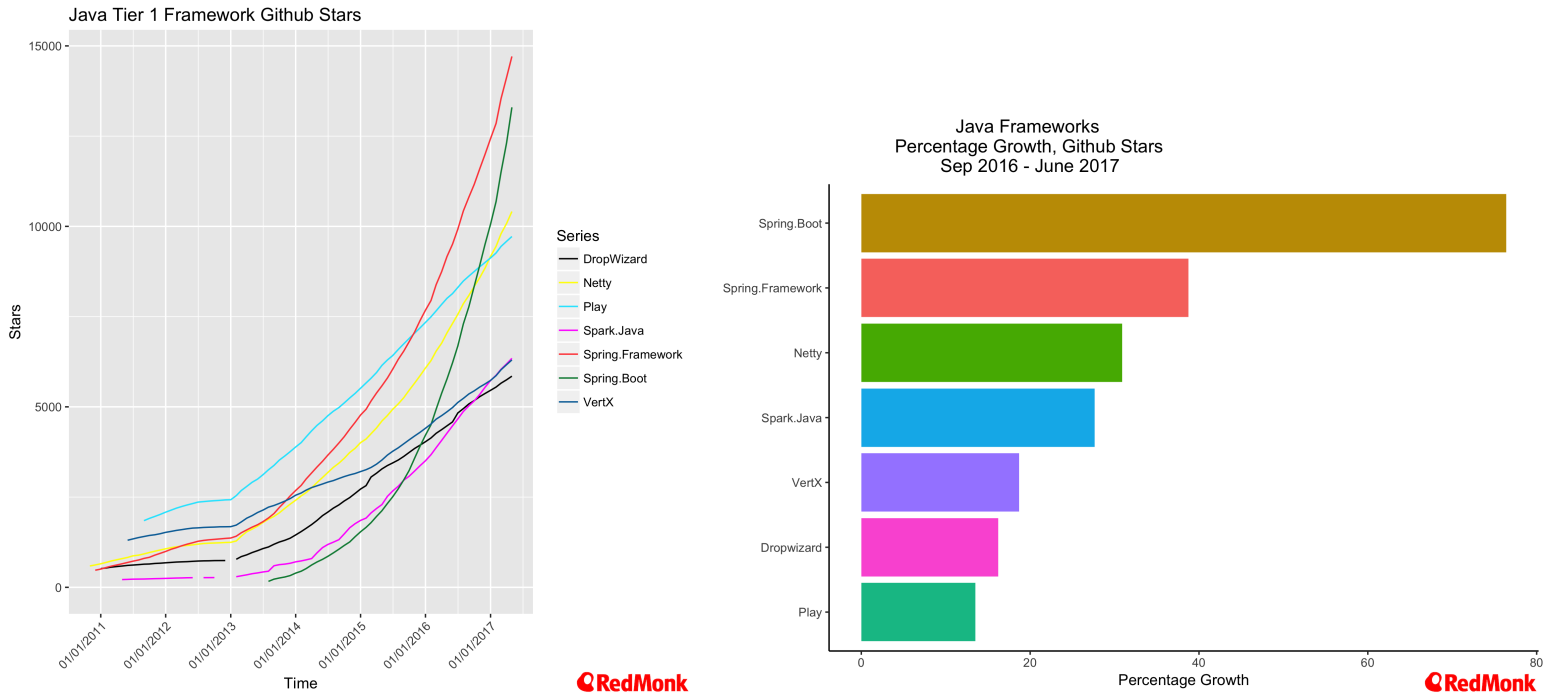
$$http://localhost:8080$$

by default

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

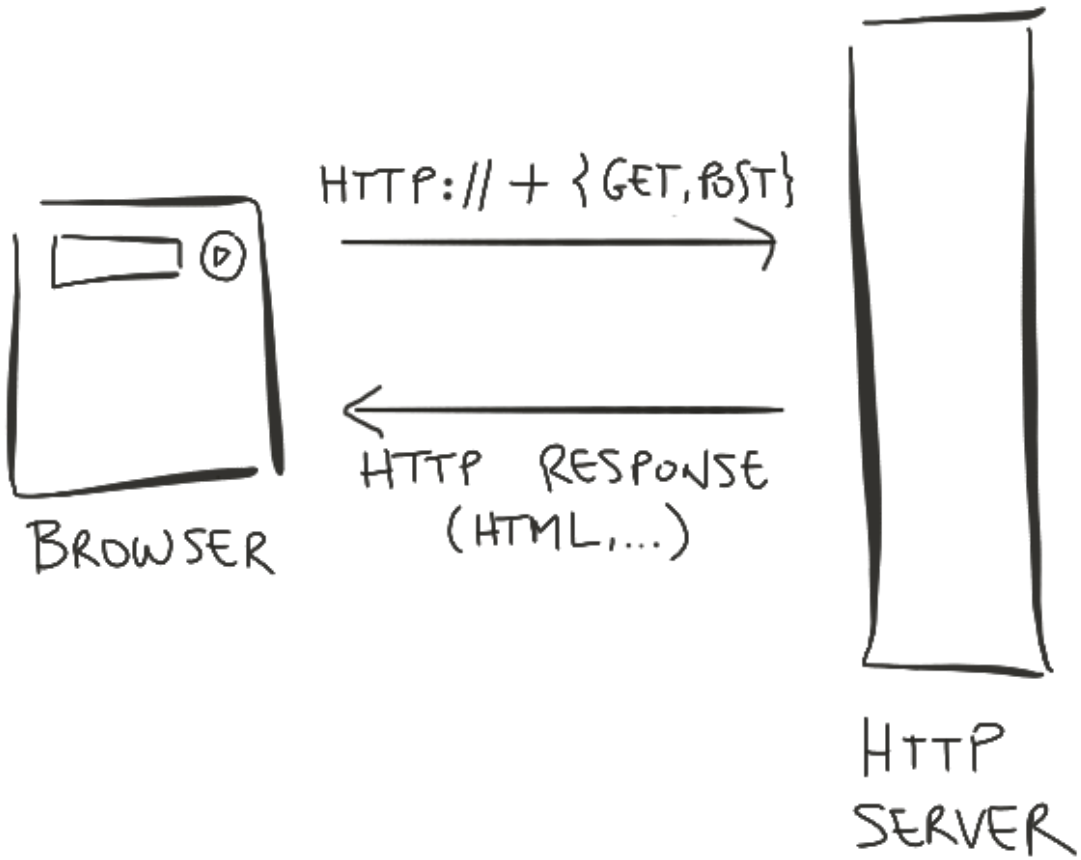
Web development frameworks (Java)

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



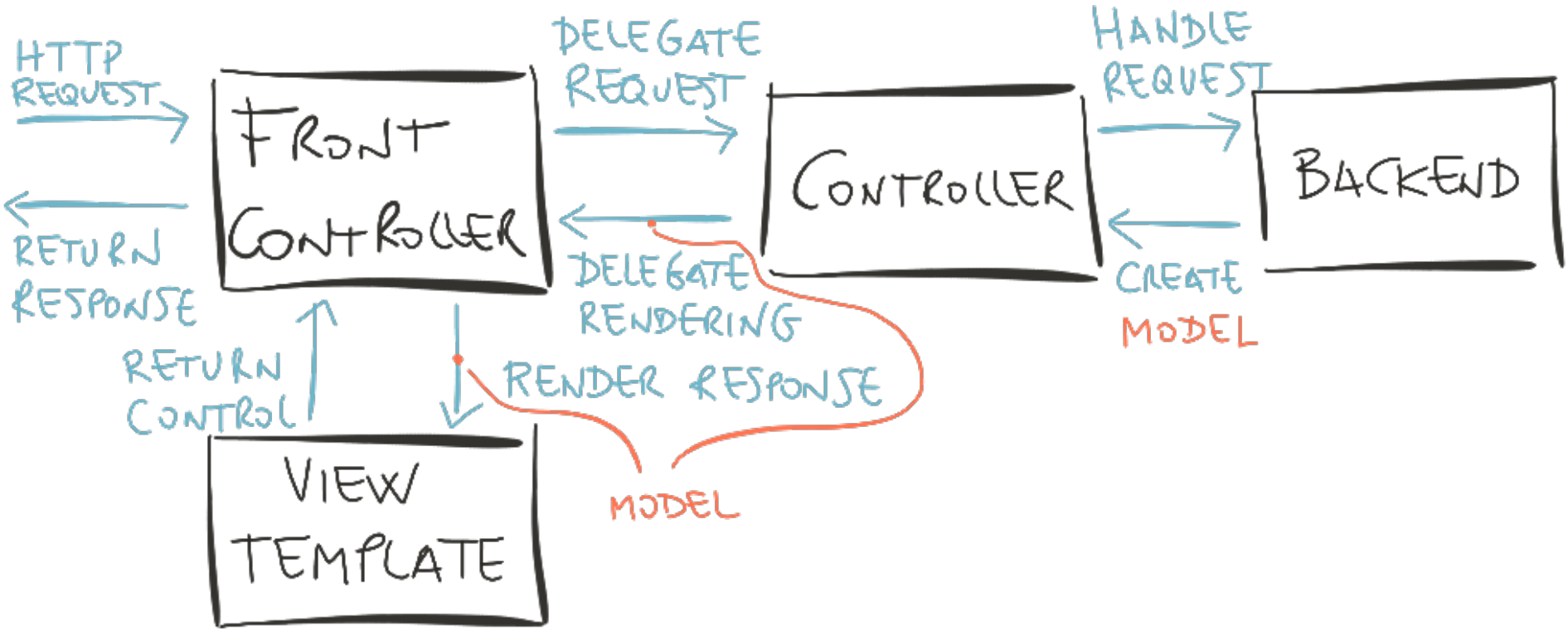
- Spring framework: facilitates the development of enterprise applications
  - can manage Java objects (beans) using dependency injection
  - offers a lot of functionality off-the-shelf (web development support)
- Spring MVC: web component of Spring, implementing MVC
- Spring Boot: convention-over-configuration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

DEALING WITH HTTP REQUESTS

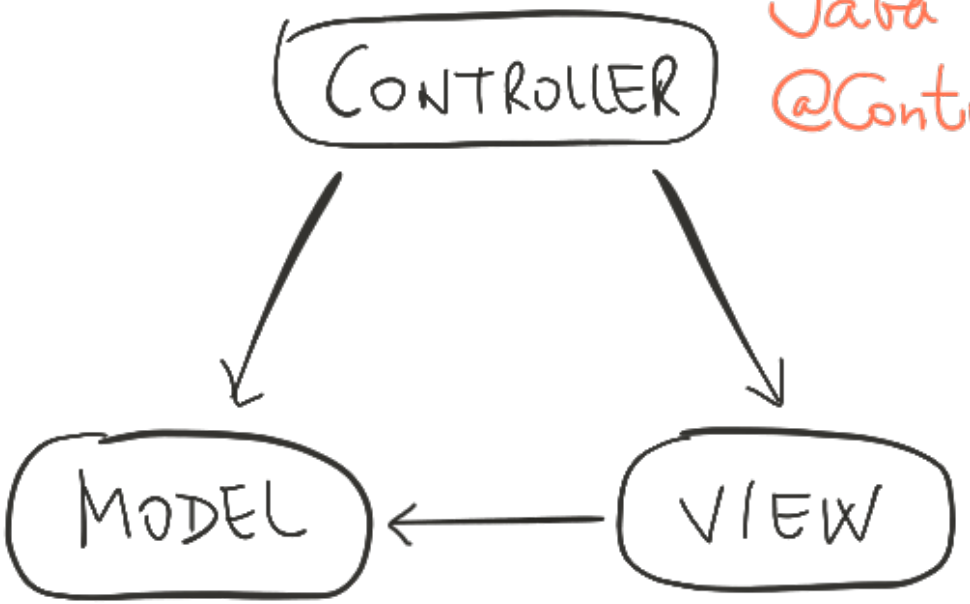


INSIDE THE HTTP SERVER:

HTTP REQUEST / RESPONSE LIFECYCLE



MODEL VIEW CONTROLLER



Java class
@Controller

MAP
attribute ↦ value

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

Model

Responsibility: encapsulate application data

- in general they will be POJOs
- as in the last exercise with Groovy

Controller

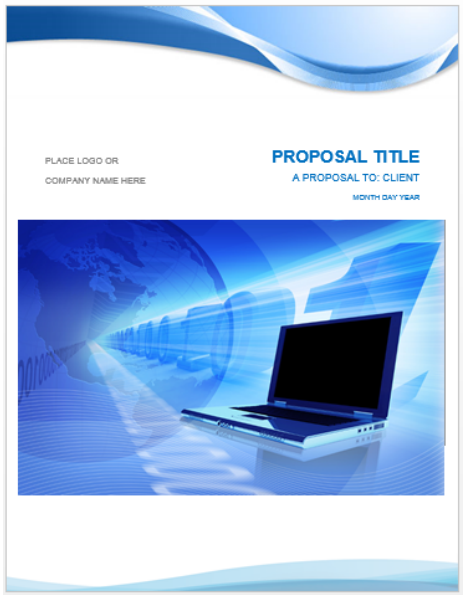
Responsibility: controls interactions with users

- 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
- main HTTP methods:
  - get: to fetch information from the server
  - post: to submit information to the server in a form

Views: JSPs (JavaServer Pages)

Responsibility: UI - displaying model data

- JSP files work as templates



- The controller chooses which template to apply by name (return value)
- The view resolver (configured in WebConfig.java) resolves the template:
  - instantiates the template: fills in gaps with information from model
  - generates code

Views: JSPs (Java Server Pages)

Generation of dynamic content (HTML)

- information from model, prepared by the controller
- tag libraries for controlling generation of HTML: loops, conditions
- tag libraries for forms: to post information

Ingredients

- Expression language: to fetch attribute values from model
- JSTL (JavaServer Pages Standard Tag Library): tags to define loops and conditions
- Spring form tag library: to design web forms that integrate well with Spring MVC

Views: Expression Language (EL)

EL

- language to evaluate expressions (returning a value)
- no loops, no conditions

How to use it

- `${expr}`: outputs the result of the expression in an HTML page
  - in view example.jsp: `<p>${product.getName()}</p>`
- we can refer to model attributes

```
// in the controller class
@RequestMapping(...)
public String productDetail(@ModelAttribute("product") Product product, ... ) {
  ...
  return "example"
}
```

- difference with GStrings in Groovy: the variables in expression `expr` are fetched from the model (as opposed to be local or global variables in the Groovy script)

Views: JSTL (JavaServer Pages Standard Tag Library)

JSTL

- collection of tags
- purpose: to program UI logic (how HTML is generated)

Tag lib directive

- added at the beginning of a JSP file
- to enable using tags from a tag library
- specifies the URI of the library (identifier for the library)
- prefix to be appended to tags within the library in order to use them

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

- to use a tag

```
<p><c:out value="Hello, World!"/></p>
```

Views: Spring Forms

spring-form tag library

- tags for including web forms in a web page
- integrate well with Spring MVC

Tag lib directive

- added at the beginning of a JSP file

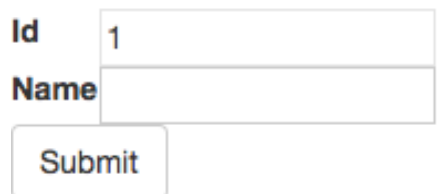
```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
```

Views: spring-form (common tags)

form:form

- http request for submitting the form:
  - action (URL)
  - HTTP method (POST)
- command object: object whose attributes can be used from a form (must be a model attribute)

```
<form:form method="POST" modelAttribute="product"
      action="/product/add">
  <table>
  <tr>
    <td><form:label path="id">Id</form:label></td>
    <td><form:input path="id" readonly="true"/></td>
  <tr>
    <td><form:label path="name">Name</form:label></td>
    <td><form:input path="name" /></td>
  </tr>
  <tr>
    <td colspan="2">
      <input type="submit" value="Submit"/>
    </td>
  </tr>
</table>
</form:form>
```



Form Validation

Fault prevention: Form Validation

- Report errors to users (in forms) when incorrect data is provided
- To avoid crashes at runtime

Components

- Validator class for command object: method `validate()`
- Controller class: checks command object
- JSP view: error tag next to each input element

Validator class

- Registers DTO class to be validated, e.g. Student
- Reports errors using
  - ValidationUtils: methods to reject empty fields
  - class Error: input element, error code (when message defined in a file), default error message

Example

```
public class StudentValidator implements Validator {
  public boolean supports(Class<?> clazz) {
    return Student.class.equals(clazz);
  }
  @Override
  public void validate(Object target, Errors errors) {
    Student dto = (Student) target;

    ValidationUtils.rejectIfEmptyOrWhitespace(errors, "id", "", "Field cannot be
        empty.");

  if ((dto.getId()!=null) && (dto.getId() < 0)) {
    errors.rejectValue("id", "", "Id invalid.");
  }
}
```

MVC Architectural Pattern

Properties

- separation of concerns
- decoupling application layers from UI

Advantages

- maintainability
- complexity management
- facilitates the re-use of business logic/data model/UI component
- multiple view support

During this Sprint...

Syllabus

- Configuring a Spring web app
- Controller: request mappings (GET)
- Views
- Views (forms) and request mappings (POST)
- Views (master/detail)
- Spring Validation

Assessment: miniproject

- effort in lab sessions: 5%
- checkpoint (23 Oct): 5%
- release (6 Nov): 90%