Bài 1
Tổng quan về Spring MVC

Ôn luyện kiến thức chung

Hỏi và trao đổi về

JAVA

JEE

JSP, SERVLET, MySQL

Kiến trúc MVC

Mục tiêu

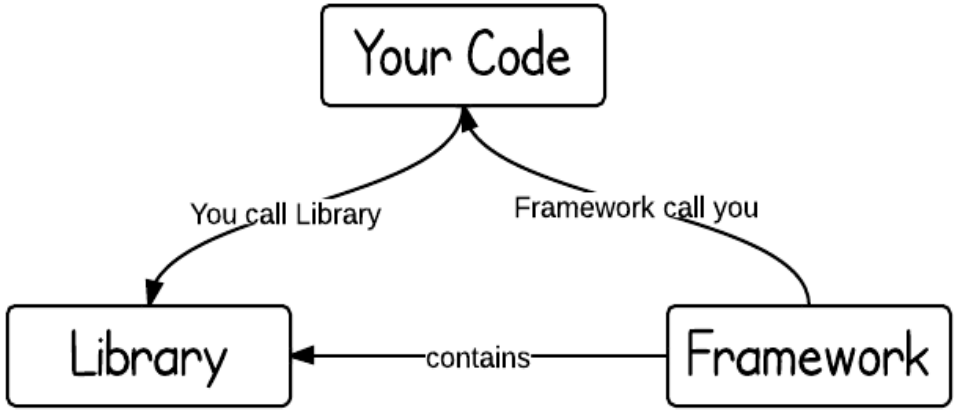
- Trình bày được ý nghĩa của các Framework
- Trình bày được tổng quan kiến trúc của Spring MVC
- Trình bày được cơ chế Dependency Injection
- Trình bày được ý nghĩa của Model trong Spring MVC
- Trình bày được DispatcherServlet
- Tạo được ứng dụng Spring MVC cơ bản
- Triển khai được ứng dụng Spring MCV cơ bản

Framework

- Framework là các ứng dụng phần mềm có tính trừu tượng (abstraction), cung cấp các tính năng chung và thông dụng, có thể tuỳ biến để tạo nên những ứng dụng cụ thể khác nhau
- Mỗi framework cung cấp một phương pháp riêng biệt để xây dựng và triển khai ứng dụng
- Mỗi framework bao gồm một môi trường tổng thể, tái sử dụng được nhằm cung cấp các chức năng và công cụ để hỗ trợ quá trình phát triển ứng dụng

Framework vs. Library

- Điểm khác biệt lớn nhất giữa Framework và Library đó chính là cơ chế “Inversion of Control”
- Với Library: Ứng dụng nắm quyền điều khiển (control)
- Với Framework: Framework nắm quyền điều khiển



Lợi ích của Framework

- Framework giải quyết các vấn đề thông dụng, giúp lập trình viên tập trung vào xử lý nghiệp vụ
- Giúp tăng tốc độ phát triển
- Cung cấp môi trường làm việc tiêu chuẩn, giúp dễ giao tiếp giữa các bên khi cùng tham gia phát triển
- Các framework thường có cộng đồng phát triển lớn, các giải pháp đã được đánh giá và thử nghiệm, hệ sinh thái đầy đủ giúp nhanh chóng xây dựng được các giải pháp tuỳ biến

Thảo luận

Spring Framework

Spring Framework

- Spring Framework cung cấp một mô hình đầy đủ cho việc phát triển và cấu hình các hệ thống Java lớn

- Các tính năng lõi:
  - **Core**: IoC container, Events, Resources, i18n, Validation, Data Binding, Type Conversion, SpEL, AOP.
  - **Testing**: Mock objects, TestContext framework, Spring MVC Test, WebTestClient.
  - **Data Access**: Transactions, DAO support, JDBC, ORM, Marshalling XML.
  - **Web Servlet**: Spring MVC, WebSocket, SockJS, STOMP messaging.
  - **Web Reactive**: Spring WebFlux, WebClient, WebSocket.
  - **Integration**: Remoting, JMS, JCA, JMX, Email, Tasks, Scheduling, Cache.
  - **Languages**: Kotlin, Groovy, Dynamic languages.

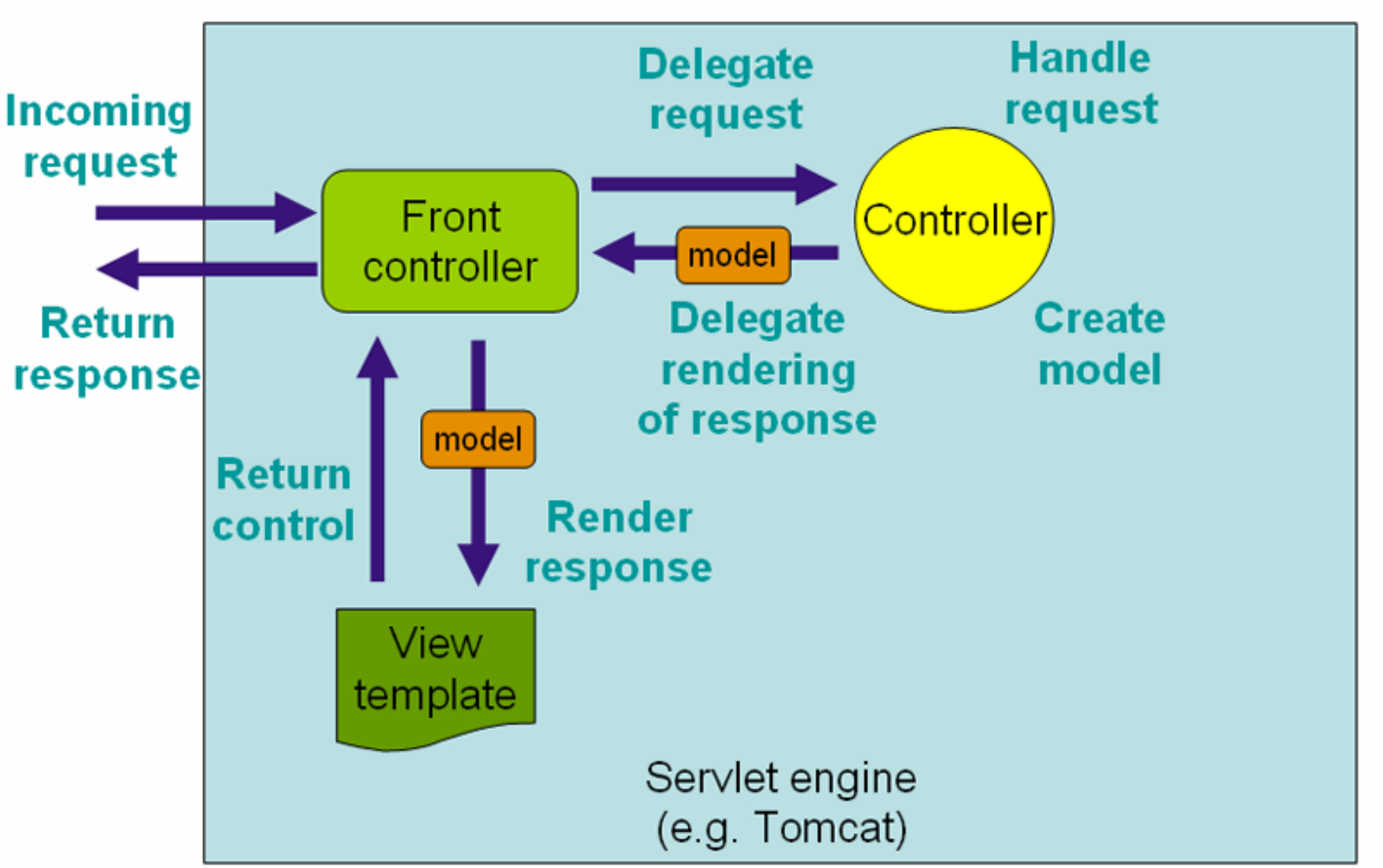
Các dự án Spring

- Spring IO Platform
- Spring Boot
- Spring Framework
- Spring Cloud Data Flow
- Spring Cloud
- Spring Data
- Spring Integration
- Spring Batch
- Spring Security
- …

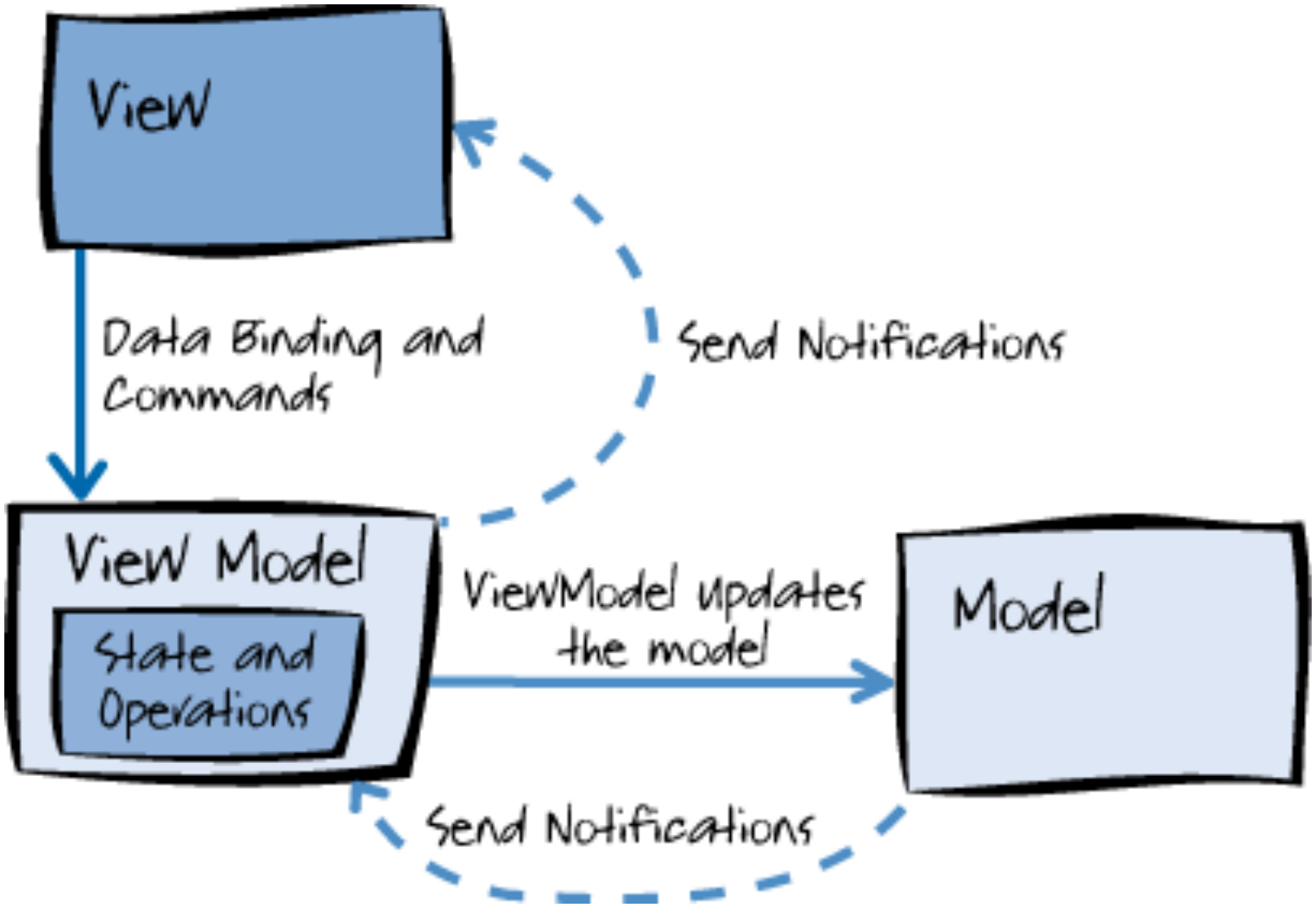
Giới thiệu Spring Web MVC

- Spring Web MVC Framework là một nền tảng mã nguồn mở phổ biến để phát triển ứng dụng Java EE
- Được cài đặt đầy đủ các đặc tính của MVC Pattern
- Cung cấp một **Front Controller** để xử lý hoặc lắng nghe mỗi khi có request tới ứng dụng

Spring MVC xử lý request



ModelView-ViewModel



DispatcherServlet

- Mỗi request đến sẽ được đón nhận và xử lý bởi Front Controller
- Kế thừa từ HTTPServlet
- DispacherServlet gửi các request tới các Controller và quyết định hồi đáp bằng cách gửi lại view
- Cấu hình của DispatcherServlet trong file web.xml/Cấu hình Java:

```
<servlet>
   <servlet-name>books</servlet-name>
   <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
</servlet>
<servlet-mapping>
   <servlet-name>books</servlet-name>
   <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

Controller

- Spring controller xử lý các request để thực hiện các logic nghiệp vụ
- Annotation @RequestMapping ánh xạ một URL đến một phương thức của controller
- Ví dụ:

```
@Controller
public class CustomerController {

    @RequestMapping(value = "/input-customer")
    public String inputCustomer() {
        // do something here
        return "CustomerForm";
    }
}
```

ViewResolver

- ViewResolver là cơ chế để xử lý tầng view của Spring MVC
- ViewResolver ánh xạ tên của view sang đối tượng view tương ứng
- Có nhiều implementation khác nhau của ViewResolver:
  - *AbstractCachingViewResolver*
  - *XmlViewResolver*
  - *ResourceBundleViewResolver*
  - *UrlBasedViewResolver*
  - *InternalResourceViewResolver*
  - *FreeMarkerViewResolver*
  - *ContentNegotiatingViewResolver*

ModelAndView

- ModelAndView đại diện cho một view cùng với các dữ liệu sử dụng trong view đó
- ModelAndView có thể kèm theo status của Response
- Ví dụ:

```
@GetMapping("/goToViewPage")
public ModelAndView passParametersWithModelAndView() {
    ModelAndView modelAndView = new ModelAndView("viewPage");
    modelAndView.addObject("message", "Baeldung");
    return modelAndView;
}
```

Inversion of Control Pattern

- IoC là một nguyên lý trong phát triển phần mềm, trong đó việc điều khiển các đối tượng hoặc các thành phần của hệ thống được thực hiện bởi framework hoặc các container
- IoC cho phép framework nắm giữ quyền điều khiển luồng thực thi của hệ thống và gọi các mã nguồn khác
- Lợi ích:
  - Tách rời việc thực thi (execution) và việc triển khai (implementation)
  - Dễ chuyển đổi giữa các implementation
  - Dễ phân tách module hơn
  - Dễ kiểm thử hơn, bằng cách tách rời các thành phần riêng lẻ
- Có thể triển khai IoC thông qua một số cơ chế: Strategy design pattern, Service Locator pattern, Factory pattern, and Dependency Injection (DI)

Dependency Injection

- Dependency injection là một cơ chế để triển khai IoC
- Các dependency được cung cấp và điều khiển bởi container hoặc framework
- Thao tác "tiêm" các đối tượng vào trong đối tượng khác được thực hiện bởi container hoặc framework
- Ví dụ:

```
public class Store {
    private Item item;

    public Store() {
        item = new
ItemImpl1();
    }
}
```

```
public class Store {
    private Item item;
    public Store(Item item) {
        this.item = item;
    }
}
```

Dependency Injection: Ví dụ (1/3)

```
public class A {
        public void importantMethod() {
                B b = ... // get an instance of B
                b.usefulMethod();
                ...
        }
        ...
}
```

Dependency Injection: Ví dụ (2/3)

```
public class A {
    private B b;
    public void importantMethod() {
        // no need to worry about creating B anymore
        // B b = ... // get an instance of B
        b.usefulMethod();
        ...
    }

    public void setB(B b) {
        this.b = b;
    }
}
```

Dependency Injection: Ví dụ (3/3)

```
public class A {
    private B b;

    public A(B b) {
        this.b = b;
    }

    public void importantMethod() {
        // no need to worry about creating B
anymore
        // B b = ... // get an instance of B
        b.usefulMethod();
        ...
    }
}
```

Spring IoC Container

- IoC container được đại diện bởi interface *ApplicationContext*
- Spring container chịu trách nhiệm khởi tạo, cấu hình và tổ chức các đối tượng – còn được gọi là các *beans*
- Spring MVC cung cấp một số triển khai:
  - *ClassPathXmlApplicationContext*
  - *FileSystemXmlApplicationContext*
  - *WebApplicationContext*
- Ví dụ:

```
ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
```

Constructor-Based Dependency Injection

- Container sẽ gọi một constructor với các tham số, mỗi tham số đại diện cho một dependency
- Spring nhận biết các tham số thông qua kiểu dữ liệu, tên của thuộc tính và chỉ số của tham số
- Ví dụ:

```
@Configuration
public class AppConfig {
    @Bean
    public Item item1() {
        return new ItemImpl1();
    }
    @Bean
    public Store store() {
        return new Store(item1());
    }
}
```

Cấu hình bean bằng XML

- Ví dụ:

```
<bean id="store" class="org.baeldung.store.Store">
   <constructor-arg type="ItemImpl1" index="0" name="item"
ref="item1" />
</bean>
```

Setter-Based Dependency Injection

- Container gọi các phương thức setter để khởi tạo bean
- Ví dụ:

```
@Bean
public Store store() {
    Store store = new Store();
    store.setItem(item1());
    return store;
}
```

- Hoặc:

```
<bean id="store" class="org.baeldung.store.Store">
    <property name="item" ref="item1" />
</bean>
```

Field-Based Dependency Injection

- Sử dụng annotation *@Autowired* để tiêm các dependency

- Ví dụ:

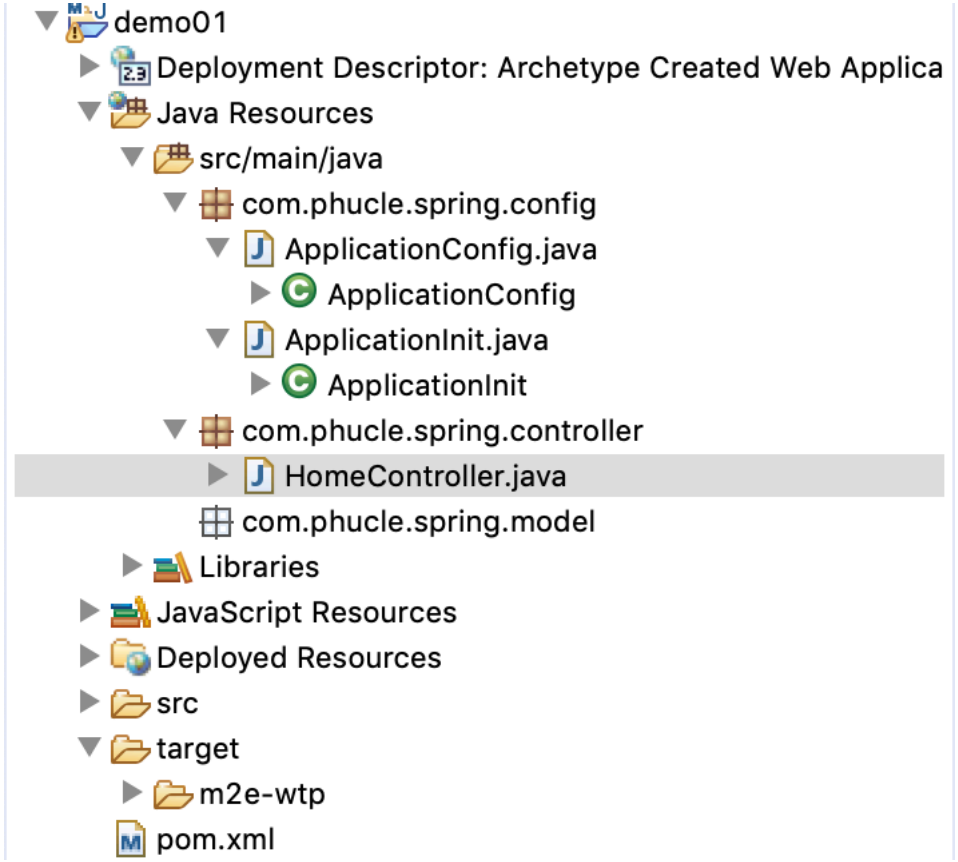
```
public class Store {
    @Autowired
    private Item item;
}
```

Tạo ứng dụng Spring MVC

Các cách tạo dự án Spring MVC

- Sử dụng IDE truyền thống
- **Sử dụng Maven**
  - Tạo với XML
  - **Tạo bằng Java class**
- Sử dụng Gradle
- Sử dụng Spring Web
- Sử dụng Spring Starter Project
- Sử dụng Spring Boot

Cấu trúc thư mục



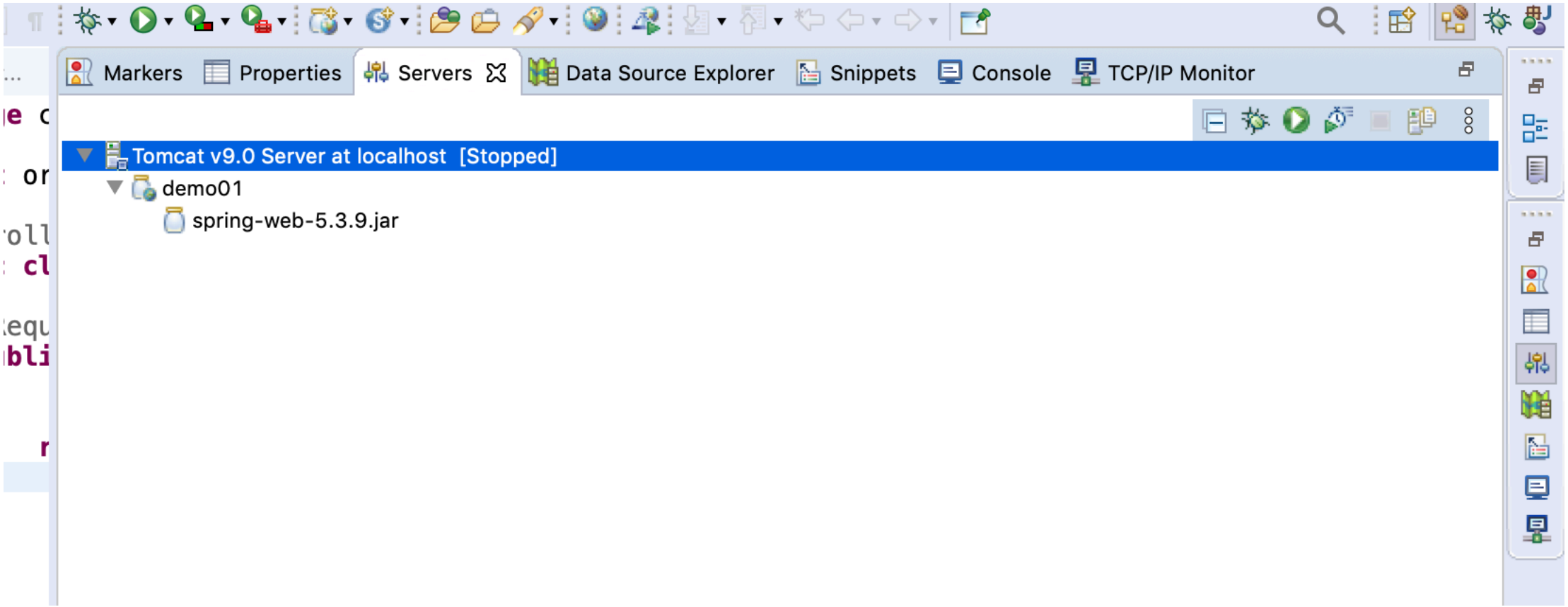
Controller

```
package controllers;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class GreetingController {
    @GetMapping("/greeting")
    public String greeting(@RequestParam String name, Model
modle){
        modle.addAttribute("name", name);
        return "index";
    }
}
```

View

```
<%@ page contentType="text/html;charset=UTF-8"
language="java" %>
<html>
<head>
  <title>Greeting</title>
</head>
<body>
  <h1>Hello: ${name}</h1>
</body>
</html>
```

Tomcat Server



Tổng kết

- Các framework giúp cho việc lập trình các ứng dụng trở nên dễ dàng hơn
- Dependency Injection là cơ chế các container hoặc framework tạo các đối tượng và cung cấp cho những nơi cần chúng
- @Autowired là annotation được sử dụng để tiêm các đối tượng
- DispatcherServlet là front controller của Spring MVC, nhận các request và chuyển đến controller tương ứng

Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập

Chuẩn bị bài tiếp theo: *Spring Controller*