

Bài 2

Spring MVC Controller

Kiểm tra bài trước

Hỏi và trao đổi về các khó khăn gặp phải trong bài “Introduction to Spring MVC”

Tóm tắt lại các phần đã học từ bài “Introduction to Spring MVC”

Mục tiêu

- Trình bày được ý nghĩa của Controller
- Trình bày được ý nghĩa của ModelAndView Interface
- Trình bày được ý nghĩa của ModelMap Interface
- Trình bày được ý nghĩa của ViewResolver Interface
- Định nghĩa được URI với các phương thức khác nhau như GET, POST, PUT, PATH, DELETE
- Thao tác được với form trong ứng dụng Spring MVC
- Trình bày được cơ chế Data Binding
- Sử dụng được các thẻ thành phần của form

Thảo luận

Annotated Controller

Request Mapping

Annotated Controller

- Spring MVC cung cấp các annotation như @Controller và @RestController để khai báo các controller
- Các annotation này thực hiện các nhiệm vụ như ánh xạ tới URL, khai báo tham số của request, xử lý ngoại lệ...
- Ví dụ khai báo controller:

```
@Controller
public class HelloController {

    @GetMapping("/hello")
    public String handle(Model model) {
        model.addAttribute("message", "Hello World!");
        return "index";
    }
}
```

Tự động phát hiện controller (1/2)

- Để có thể tự động phát hiện được controller, cần khai báo thuộc tính component-scan cho ứng dụng
- Ví dụ sử dụng lớp cấu hình Java:

@Configuration

@ComponentScan("com.phucle.spring")

public class WebConfig {

// ...

}

Request Mapping

- Annotation `@RequestMapping` được sử dụng để ánh xạ các request tới các action tương ứng của controller
- `@RequestMapping` bao gồm một số thuộc tính để ánh xạ đến:
 - URL
 - HTTP Method
 - Các tham số
 - Các headers
 - Media types
- `@RequestMapping` có thể áp dụng cho lớp hoặc phương thức

Các biến thể của @RequestMapping

- @RequestMapping còn có một số biến thể dành cho các HTTP Method cụ thể:
 - @GetMapping
 - @PostMapping
 - @PutMapping
 - @DeleteMapping
 - @PatchMapping

RequestMapping: Ví dụ

```
@RestController
@RequestMapping("/persons")
class PersonController {

    @GetMapping("/{id}")
    public Person getPerson(@PathVariable Long id) {
        // ...
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public void add(@RequestBody Person person) {
        // ...
    }
}
```

URI patterns

- Có thể ánh xạ tới các request bằng cách sử dụng các mẫu đại diện sau:

?	Trùng khớp một ký tự
*	Trùng khớp với 0 hoặc nhiều ký tự
**	Trùng khớp với 0 hoặc nhiều phần của đường dẫn (path segment)

Khai báo biến URI

- Sử dụng annotation `@PathVariable` để khai báo các biến của đường dẫn

- Ví dụ, khai báo ở phương thức:

```
@GetMapping("/owners/{ownerId}/pets/{petId}")
public Pet findPet(@PathVariable Long ownerId, @PathVariable Long petId)
{
    // ...
}
```

- Hoặc, khai báo ở lớp:

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class OwnerController {

    @GetMapping("/pets/{petId}")
    public Pet findPet(@PathVariable Long ownerId, @PathVariable Long petId) {
        // ...
    }
}
```

@PathVariable sử dụng Regex

- Có thể sử dụng biểu thức *{varName:regex}* để khai báo các biến đường dẫn sử dụng Regex
- Ví dụ, với đường dẫn *"/spring-web-3.0.5.jar"*, có thể tách thành các biến name, version và file extension như sau:

```
@GetMapping("/{name:[a-z-]+}-{version:\\d\\.\\d\\.\\d}{ext:\\.[a-z]+}")  
public void handle(@PathVariable String version, @PathVariable String ext)  
{  
    // ...  
}
```

Ánh xạ với Content-Type của Request

- Sử dụng thuộc tính consumes để ánh xạ đến Content-Type của request

• Ví dụ:

```
@PostMapping(path = "/pets", consumes = "application/json")
public void addPet(@RequestBody Pet pet) {
    // ...
}
```

- Hoặc phủ định:

```
@PostMapping(path = "/pets", consumes = "!text/plain")
public void addPet(@RequestBody Pet pet) {
    // ...
}
```

Ánh xạ với Accept của Request

- Sử dụng thuộc tính produces để ánh xạ đến Accept của Request

- Ví dụ:

```
@GetMapping(path = "/pets/{petId}", produces =  
"application/json;charset=UTF-8")
```

```
@ResponseBody
```

```
public Pet getPet(@PathVariable String petId) {
```

```
    // ...
```

```
}
```

Ánh xạ tới tham số của request

- Sử dụng thuộc tính params để ánh xạ tới tham số của đường dẫn
- Ví dụ:

```
@GetMapping(path = "/pets/{petId}", params = "myParam=myValue")  
public void findPet(@PathVariable String petId) {  
    // ...  
}
```

Ánh xạ tới header của Request

- Sử dụng thuộc tính headers để ánh xạ đến header của request
- Ví dụ:

```
@GetMapping(path = "/pets", headers = "myHeader=myValue")
public void findPet(@PathVariable String petId) {
    // ...
}
```


Thảo luận

Handler Method

Tham số của handler method

- Handler method có thể có nhiều tham số, chẳng hạn như:
 - *HttpServletRequest*
 - *HttpServletResponse*
 - *HttpSession*
 - *@PathVariable*
 - *@RequestParam*
 - *@RequestHeader*
 - *@RequestBody*
 - *@ModelAttribute*
 - *@RequestAttribute*
 - ... và các tham số khác

Giá trị trả về của handler method

- Handler method có thể trả về một trong các giá trị thuộc các loại sau:
 - *@ResponseBody*
 - *String*
 - *View*
 - *@ModelAttribute*
 - *ModelAndView*
 - *void*
 - *StreamingResponseBody*
 - *org.springframework.ui.Model*
 - ... và các loại giá trị trả về sau

@RequestParam

- Sử dụng @RequestParam để truy cập một tham số của URI với một tham số của handler method

- Ví dụ: **@Controller**

```
@RequestMapping("/pets")  
public class EditPetForm {
```

```
    // ...
```

```
    @GetMapping
```

```
    public String setupForm(@RequestParam("petId") int petId, Model  
model) {
```

```
        Pet pet = this.clinic.loadPet(petId);
```

```
        model.addAttribute("pet", pet);
```

```
        return "petForm";
```

```
    }
```

```
    // ...
```

```
}
```

@ModelAttribute

- Sử dụng @ModelAttribute để truy cập đến thuộc tính của một model (hoặc khởi tạo model nếu chưa có)
- Các thuộc tính của model được liên kết với các trường dữ liệu có cùng tên

- Ví dụ:

```
@PostMapping("/owners/{ownerId}/pets/{petId}/edit")
```

```
public String processSubmit(@ModelAttribute Pet pet)
```

- Hoặc:

```
@PutMapping("/accounts/{account}")
```

```
public String save(@ModelAttribute("account") Account account)
```

```
{
```

```
    // ...
```

```
}
```

Multipart

- Sử dụng MultipartFile để truy cập đến file được upload lên

• Ví dụ:

@Controller

```
public class FileUploadController {  
    @PostMapping("/form")  
    public String handleFormUpload(@RequestParam("name") String  
name,  
                                   @RequestParam("file") MultipartFile file) {  
        if (!file.isEmpty()) {  
            byte[] bytes = file.getBytes();  
            // store the bytes somewhere  
            return "redirect:uploadSuccess";  
        }  
        return "redirect:uploadFailure";  
    }  
}
```

Demo

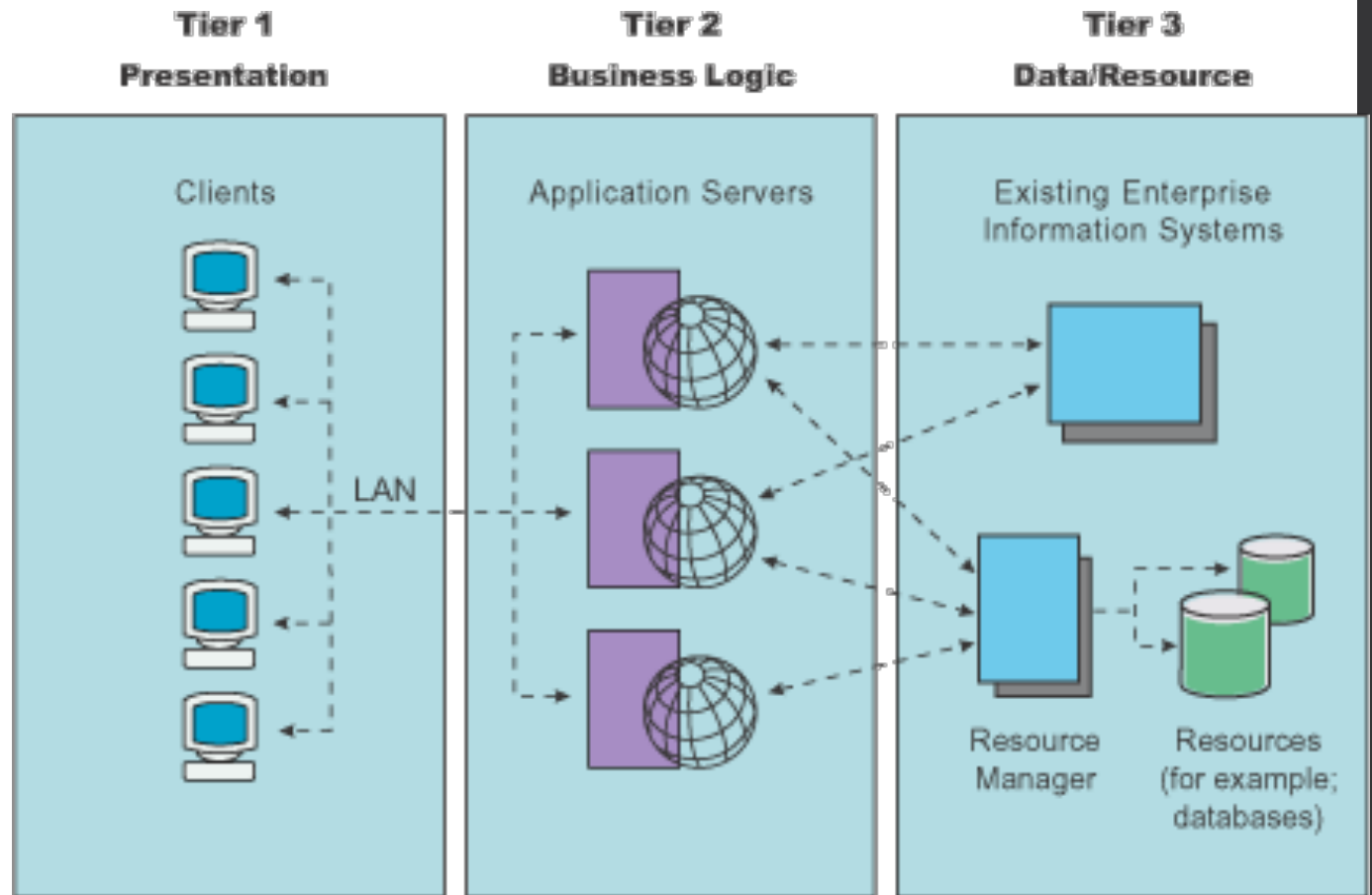
Handler Method

Thảo luận

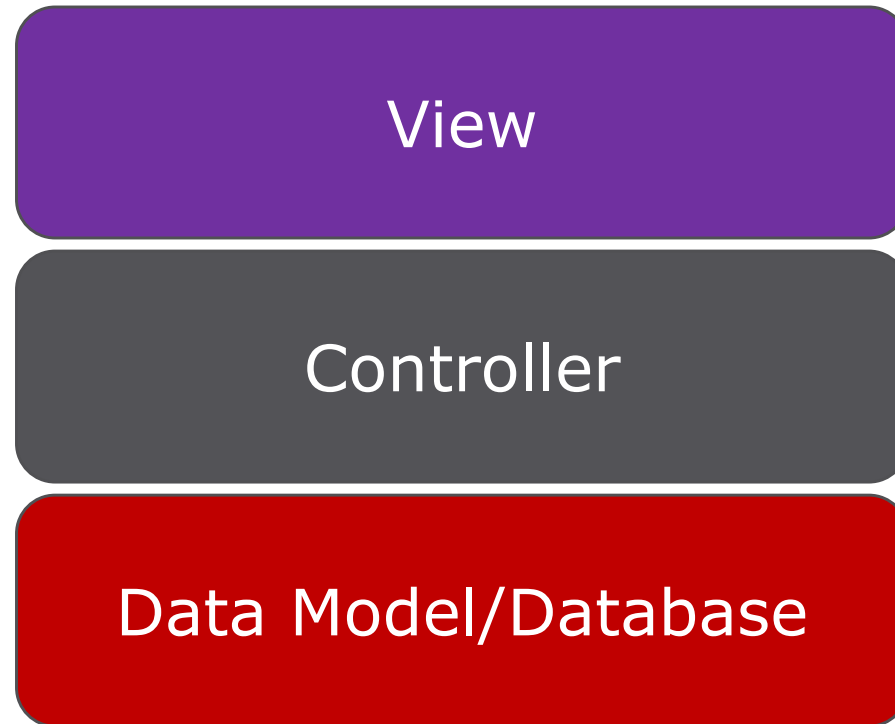
Kiến trúc của ứng dụng
Spring MVC

Kiến trúc 3-Tier và N-Tier

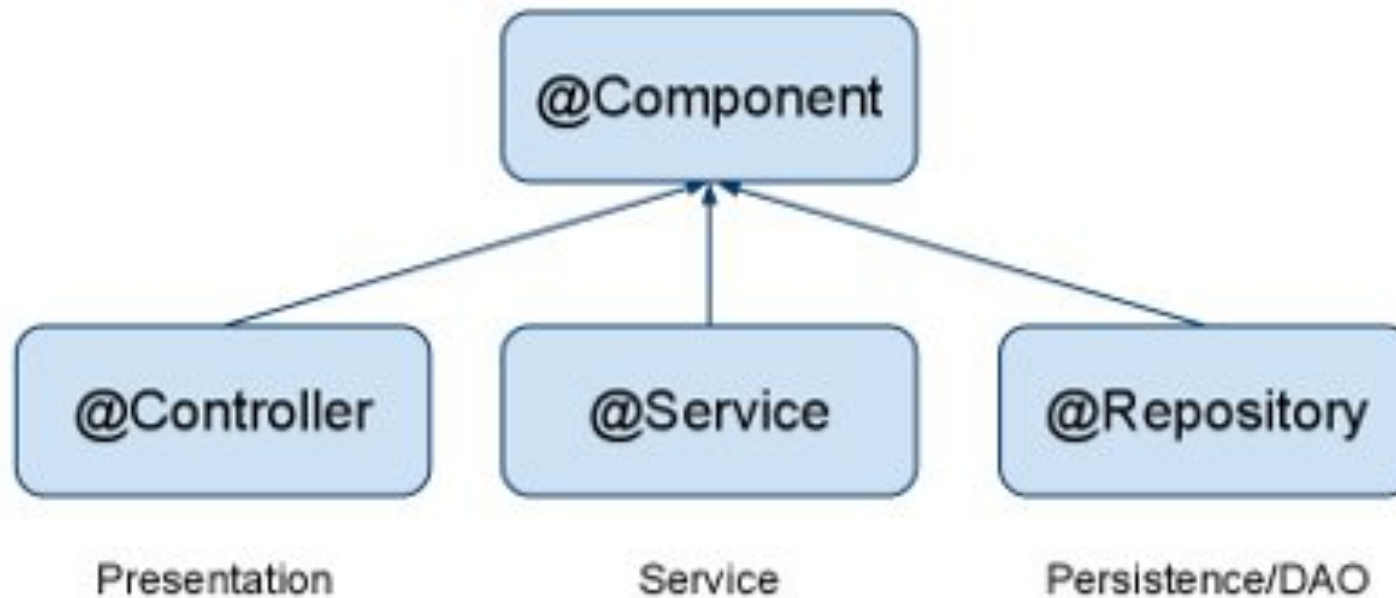
- Chia tách các nhiệm vụ
- Tái sử dụng các tầng
- Dễ bảo trì



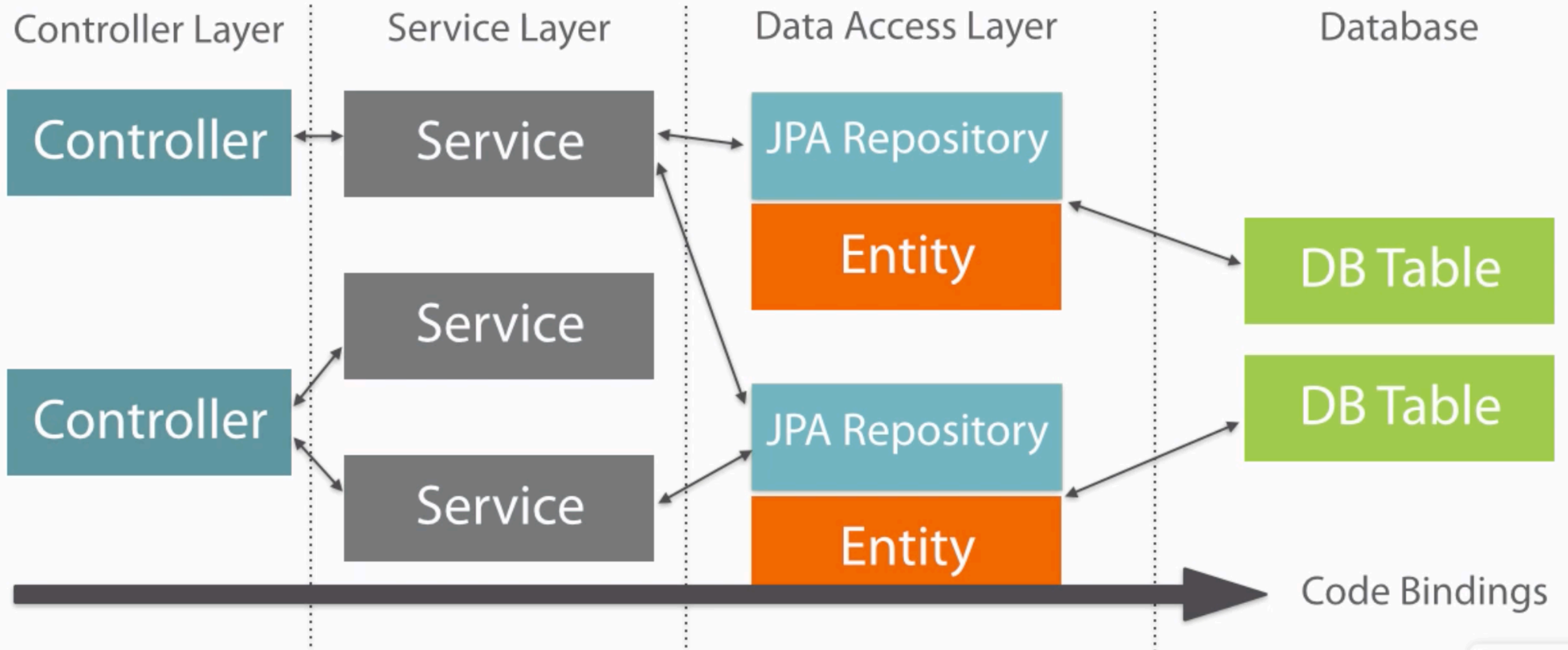
Các tầng của Spring MVC



Các component của ứng dụng Spring MVC

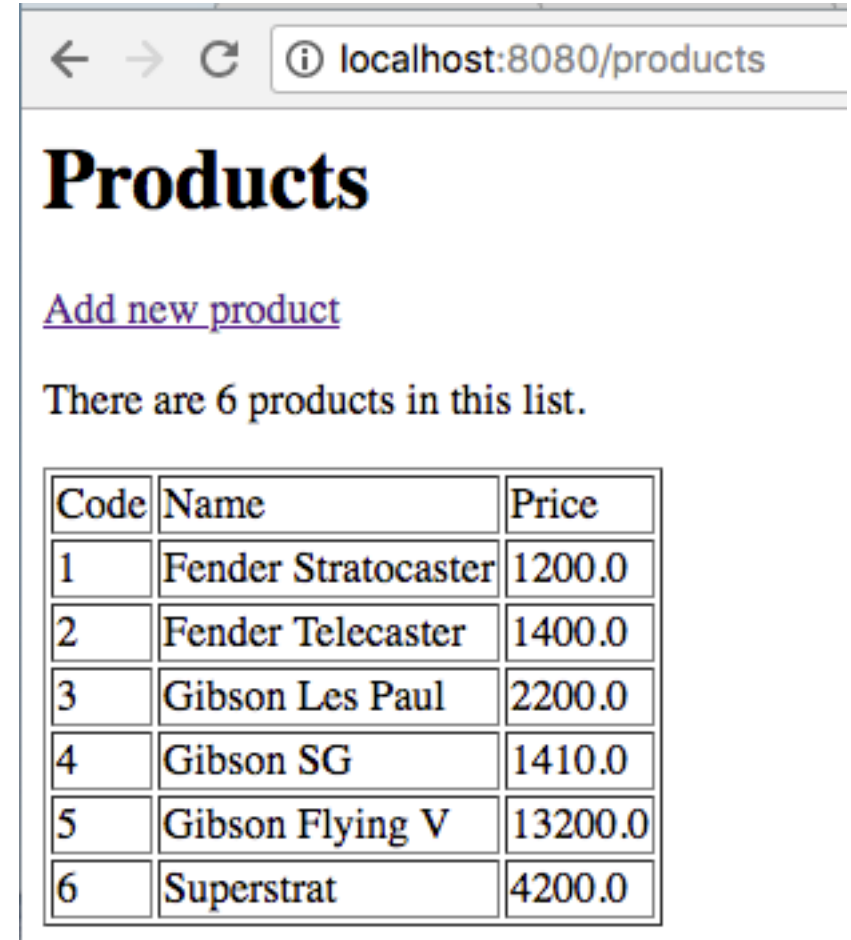


Repository Architecture



DEMO: Repository Architecture

- Model:
 - Class Product
- Repository:
 - ProductRepository
 - ProductRepositoryImpl
- Service:
 - ProductService
 - ProductServiceImpl
- Controller:
 - ProductController



← → ↻ ⓘ localhost:8080/products

Products

[Add new product](#)

There are 6 products in this list.

Code	Name	Price
1	Fender Stratocaster	1200.0
2	Fender Telecaster	1400.0
3	Gibson Les Paul	2200.0
4	Gibson SG	1410.0
5	Gibson Flying V	13200.0
6	Superstrat	4200.0

Thảo luận

Form và Data Binding

Tạo form

- Form là cơ chế để cho phép người dùng nhập dữ liệu
- Spring MVC cung cấp các thẻ trong thư viện *spring-form.tld* để thao tác với form

- Khai báo *spring-form.tld*:

```
<%@taglib prefix="form"
```

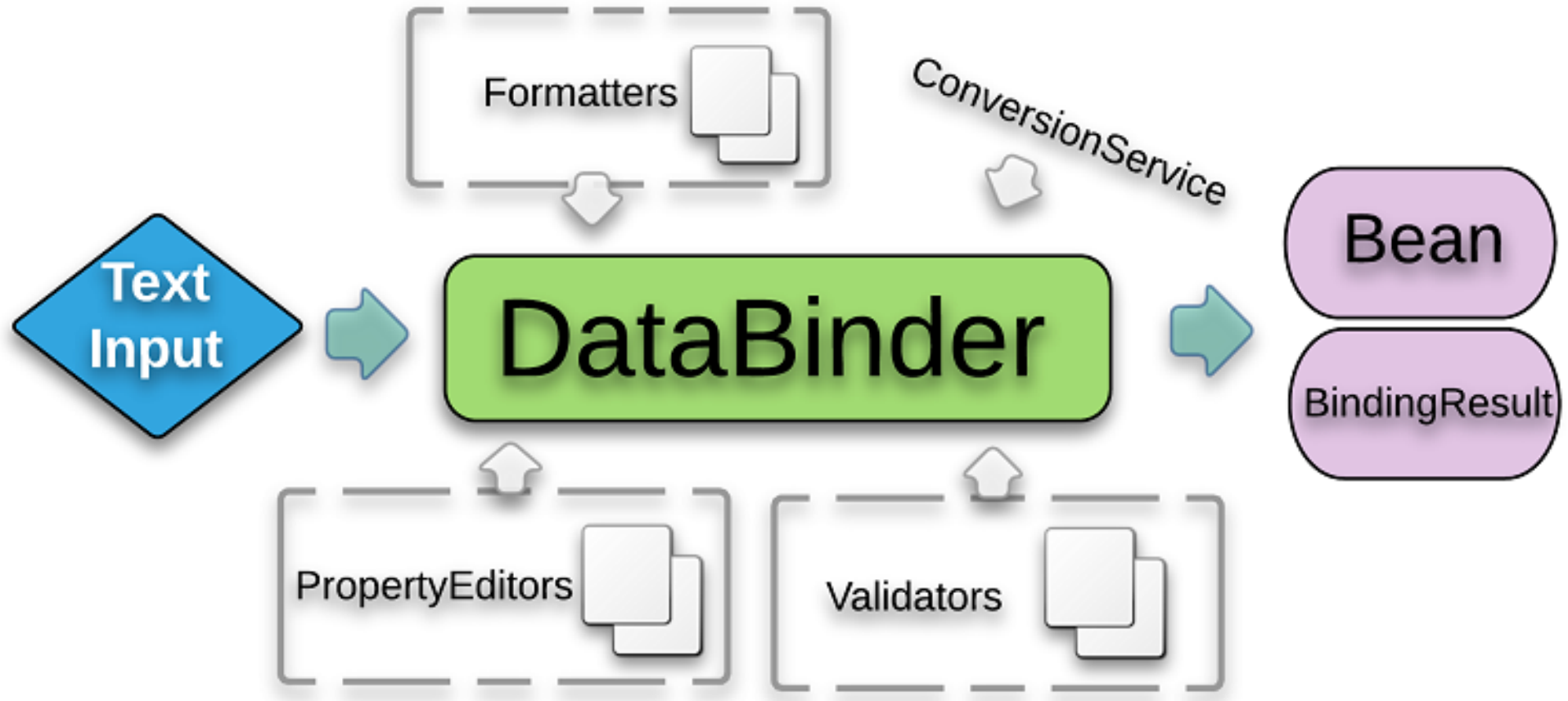
```
uri="http://www.springframework.org/tags/form"%>
```

- *spring-form.tld* cung cấp thẻ form và các thẻ thành phần khác để thao tác với form
- *spring-form.tld* cung cấp các cơ chế để kiểm tra lỗi, tùy biến giao diện và triển khai quốc tế hoá (i18n)

Data Binding

- Data Binding là cơ chế liên kết dữ liệu đầu vào (hoặc đầu ra) với các đối tượng model
- Data Binding giúp cho việc tương tác với dữ liệu trở nên dễ dàng
- Sử dụng Data Binding, các form đều được liên kết với một đối tượng biểu diễn dữ liệu ở phía sau
- Khi tương tác với form, dữ liệu trên form sẽ được tự động chuyển đổi thành các thuộc tính của đối tượng liên kết với nó
- Data Binding hỗ trợ chuyển đổi dữ liệu (data conversion) và validate dữ liệu
- Data Binding trong Spring hoạt động dựa trên Data Binder

Cơ chế hoạt động của Data Binding



Thẻ form

- Thẻ form được sử dụng để tạo một form
- Ngoài các thuộc tính cơ bản của một thẻ form trong HTML, thẻ form còn có thêm thuộc tính quan trọng:
 - *commandName* hoặc *modelAttribute*: Tên của model được liên kết với form
- Thuộc tính *commandName* và *modelAttribute* có tác dụng giống nhau (*commandName* là cách dùng cũ, *modelAttribute* là cách dùng mới)
- Ví dụ:
`<form:form modelAttribute="user" action="/create-user" method="post">`

`</form:form>`

Các thẻ của spring-form.tld

- spring-form.tld hỗ trợ các thẻ: button, checkbox, checkboxes, errors, form, hidden, input, label, option, options, password, radiobutton, radiobuttons, select, textarea
- Một số thuộc tính quan trọng:
 - *path*: tên của thuộc tính được liên kết với trường hiện tại
 - *items*: danh sách các hạng mục của các trường như select, options, radiobuttons, checkboxes

Các thẻ input, password, textarea, hidden

- Thuộc tính quan trọng
 - path: tên thuộc tính của model được liên kết với trường input
- Ví dụ:

```
<form:form modelAttribute="user" action="/create-user"
method="post">
  <form:hidden path="id"/>
  <form:input path="name"/><br/>
  <button type="submit">Create user</button>
</form:form>
```

```
public class User {
    private int id;
    private String name;
    private String gender;
    private List<String> hobbies;
```

```
//Constructors, Getters & Setters
}
```

Thẻ checkbox và checkboxes

- Ví dụ sử dụng checkbox và checkboxes:

```
<form:form modelAttribute="user" action="/create-user"
method="post">
```

```
    Name: <form:input path="name"/><br/>
```

```
    Is admin: <form:checkbox path="admin"/><br/>
```

```
    Hobbies: <form:checkboxes path="hobbies"
items="${hobbiesArray}"/><br/>
```

```
    <button type="submit">Create user</button>
</form:form>
```

```
public class User {

    private int id;
    private String name;
    private String gender;
    private boolean admin;
    private List<String> hobbies;
    // Constructors, Getters & Setters
}
```

Thẻ radiobutton và radiobuttons

- Ví dụ sử dụng thẻ radiobuttons:

```
<form:form modelAttribute="user" action="/create-user"
method="post">
```

```
...
```

```
    Type: <form:radiobuttons path="type" items="${typesArray}"/><br/>
```

```
...
```

```
</form:form>
```

Thẻ select, option, options

- Ví dụ sử dụng thẻ select:

```
<form:form modelAttribute="user" action="/create-user"
method="post">
```

...

```
Type: <form:select path="type" items="${typesArray}"/><br/>
```

...

```
</form:form>
```

Thảo luận

Upload file

Form hỗ trợ upload file

- Form hỗ trợ upload file thông qua input có type là file
- Form cần có thêm thuộc tính *enctype* với giá trị là *multipart/form-data*

```
<form:form modelAttribute="userForm" action="/create-user" method="post" enctype="multipart/form-data">  
    <form:input type="file" path="avatar"/>  
</form:form>
```

Lớp MultipartFile

- Lớp `org.springframework.web.multipart.MultipartFile` đại diện cho một file được upload lên
- Một số phương thức quan trọng:
 - `getOriginalFilename()`: Trả về tên của file
 - `getInputStream()`: Trả về Input Stream để thao tác với file
 - `getSize()`: Trả về kích thước của file
 - `transferTo(File)`: Chuyển file vừa upload đến một vị trí nhất định

MultipartResolver

- Spring cung cấp lớp MultipartResolver để làm việc với multipart form
- MultipartResolver không trực tiếp xử lý multipart form, mà có thể dựa vào 1 trong 2 cách sau:
 - Sử dụng thư viện Commons FileUpload
 - Sử dụng cơ chế xử lý multipart của Servlet 3.0

Cấu hình sử dụng Commons FileUpload

- Cấu hình dependency:

```
dependencies {  
    compile group: 'commons-fileupload', name: 'commons-fileupload',  
version: '1.3.3'  
}
```

- Cấu hình bean multipartResolver:

@Bean

```
public CommonsMultipartResolver multipartResolver(){  
    CommonsMultipartResolver multipartResolver = new  
CommonsMultipartResolver();  
    multipartResolver.setMaxUploadSizePerFile(10000000);  
    return multipartResolver;  
}
```

Cấu hình sử dụng Servlet 3.0 (1)

- Cấu hình dependency:

```
dependencies {  
    compile group: 'javax.servlet', name: 'javax.servlet-api', version:  
'3.1.0'  
}
```

- Cấu hình bean multipartResolver

```
@Bean  
public StandardServletMultipartResolver multipartResolver(){  
    return new StandardServletMultipartResolver();  
}
```

Cấu hình sử dụng Servlet 3.0 (2)

- Cấu hình Multipart:

```
public class AppInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected void customizeRegistration(ServletRegistration.Dynamic
registration) {
        MultipartConfigElement multipartConfigElement = new
MultipartConfigElement(FileUtils.UPLOAD_LOCATION);
        registration.setMultipartConfig(multipartConfigElement);
    }
}
```

Demo

Upload file

Tổng kết

- Controller là đối tượng nhận và xử lý các request
- Sử dụng annotation `@Controller` để khai báo các controller
- Sử dụng `@RequestMapping` để ánh xạ các URL tới các handler method của controller
- `@RequestMapping` có các biến thể khác như `@GetMapping`, `@PostMapping`, `@PutMapping`...
- 3 component quan trọng của Spring MVC là `@Controller`, `@Service` và `@Repository`
- Kiến trúc Repository là một kiến trúc tốt để xây dựng các ứng dụng web Spring MVC

Tổng kết

- Data Binding là cơ chế liên kết dữ liệu từ giao diện đến đối tượng ở phía sau
- Spring cung cấp spring-form.tld với các thẻ form thông dụng
- Có thể upload file sử dụng Commons FileUpload hoặc Servlet 3.0

Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập

Chuẩn bị bài tiếp theo: *Views and Thymeleaf*