

1



BUILDING CONTROLLER

GIẢNG VIÊN:

REQUEST MAPPING

 @REQUESTMAPPING, @GETMAPPING, @POSTMAPPING

USER DATA HANDLING

 @REQUESTPARAM, @REQUESTPART, @PATHVARUABLE, @COOKIEVALU

DATA SHARING (MODEL)

 MODEL, @MODELATTRIBUTE

MAPPING METHOD RETURN

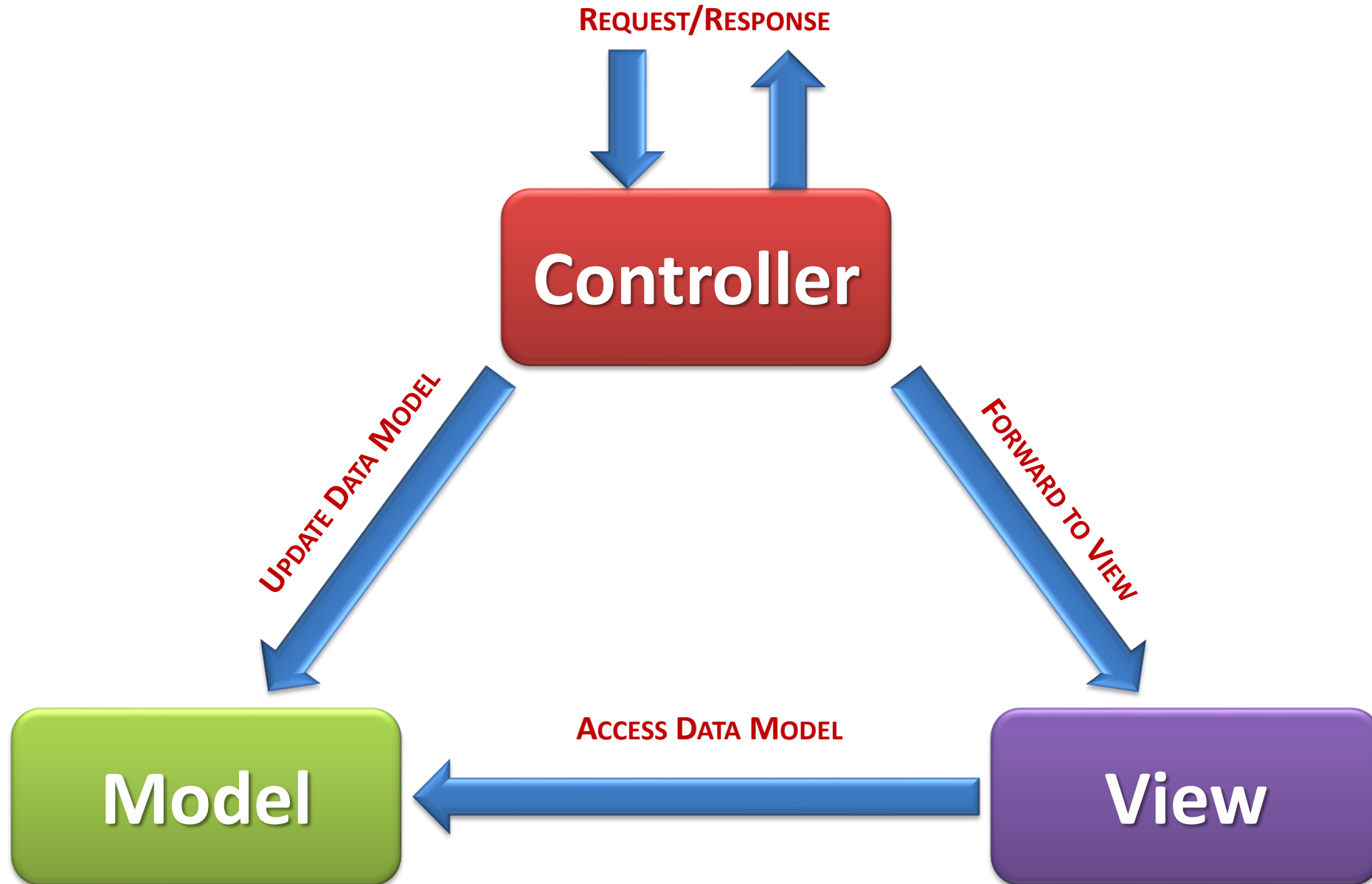
 VIEWNAME

 REDIRECT:

 FORWARD:

 @RESPONSEBODY (RAW DATA)







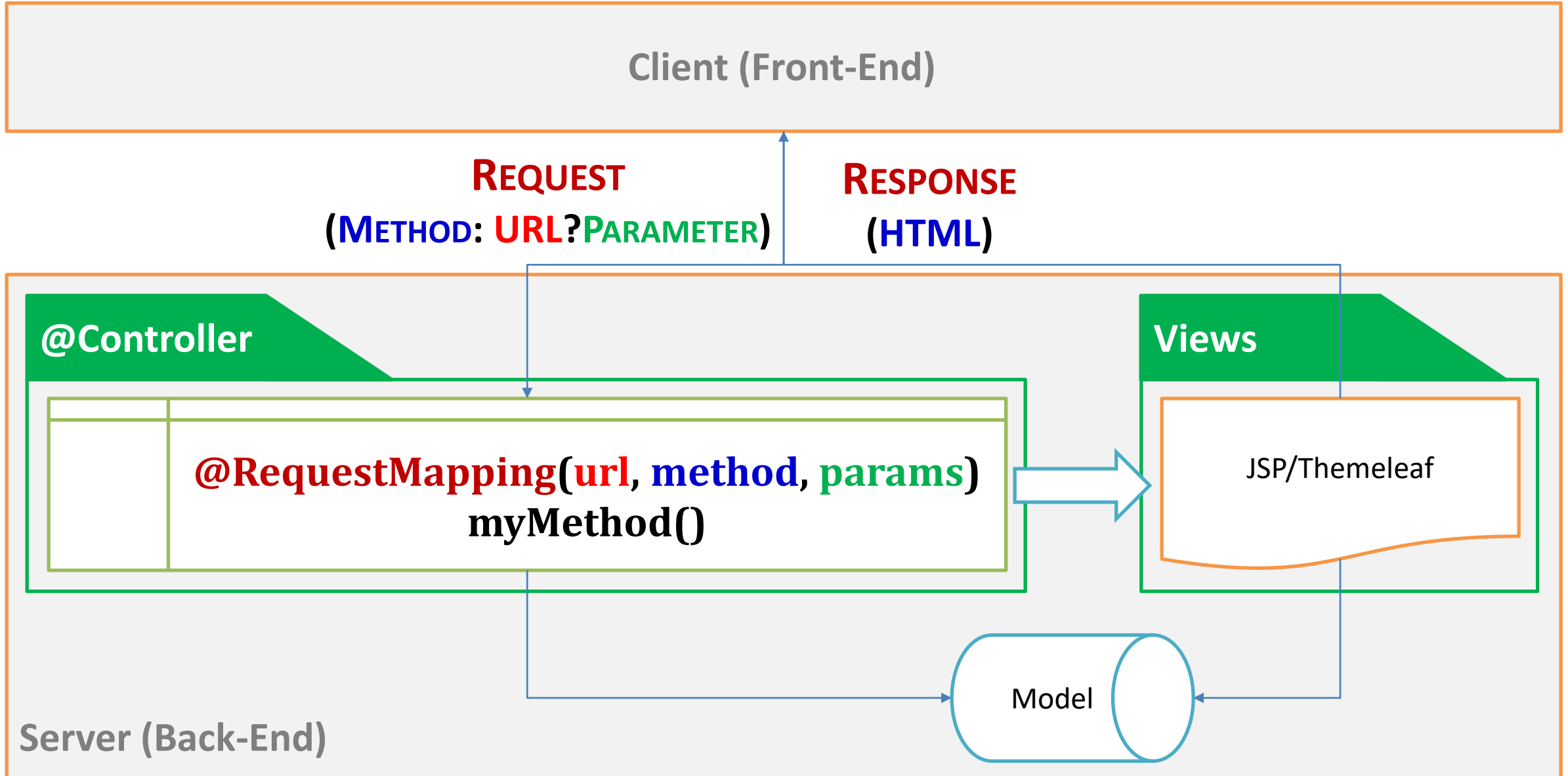
REQUEST MAPPING

```
@RequestMapping(URL, Method, Params)  
Controller.Method()
```



Request

Method, URL, Parameters



❑ Controller

- ❖ Controller chứa các mapping method điều khiển các request của người sử dụng.
- ❖ Để tạo ra request kích hoạt được mapping method thì request các chứa các thông tin định vị gồm (URL, method, params)

❑ View

- ❖ View là thành phần render giao diện phía back-end. Spring MVC hỗ trợ JSP hoặc html template (nếu sử dụng Thymeleaf). Thymeleaf được cấu hình mặc định.

❑ Model

- ❖ Model và Scopes (Request, Session, Application) chứa dữ liệu (các attribute) để chia sẻ với View.

```
@Controller
```

```
public class HomeController {  
    @RequestMapping(value = "/home/index", method = RequestMethod.GET)  
    public String index(Model model) {  
        model.addAttribute("message", "Spring MVC");  
        return "home/index";  
    }  
    @GetMapping("/home/about")  
    public String about(Model model) {  
        return "home/index";  
    }  
}
```

❑ @Controller

❑ @RequestMapping => GET:/home/index

❑ @GetMapping => GET:/home/about

❑ @RequestMapping được sử dụng để ánh xạ (Method , URL và Parameter) với một phương thức

❖ @RequestMapping(value, method, params)

❑ Mapping Annotation phân biệt Method

❖ @GetMapping(value, params)

➤ Chỉ với GET

❖ @PostMapping(value, params)

➤ Chỉ với POST

❖ @PutMapping(value, params)

➤ Chỉ với PUT (áp dụng cho REST API)

❖ @DeleteMapping(value, params)

➤ Chỉ với DELETE (áp dụng cho REST API)

❖ @PatchMapping(value, params)

➤ Chỉ với PATCH (áp dụng cho REST API)

REST API => Java6

```
@Controller
public class HomeController {
    @RequestMapping("home/index")
    public String index() {
        return "index";
    }

    @RequestMapping("home/about")
    public String about() {
        return "about";
    }
}
```

TƯỜNG MINH

```
@Controller
@RequestMapping("home")
public class HomeController {
    @RequestMapping("index")
    public String index() {
        return "index";
    }

    @RequestMapping("about")
    public String about() {
        return "about";
    }
}
```

TÁCH THÀNH 2 PHẦN

```
@Controller
@RequestMapping("account")
public class AccountController {
    @GetMapping("login")
    public String login() {
        return "login";
    }
    @PostMapping("login")
    public String login(MyBean bean) {
        return "login";
    }
}
```

GET: /account/login

POST: /account/login

❑ Xử lý thông thường

❖ GET: hiển thị form

❖ POST: xử lý form

```
@Controller
@RequestMapping("url")
public class MyController {
    @RequestMapping()
    public String method1() {
        return "index";
    }

    @RequestMapping(params="btnInsert")
    public String method2() {
        return "index";
    }

    @RequestMapping(params="btnUpdate")
    public String method3() {
        return "index";
    }
}
```

← **X: /url**

← **X: /url?btnInsert**

← **X: /url?btnUpdate**

Dựa vào param để phân biệt phương thức



USER DATA HANDLING

❑ @RequestParam được sử dụng để nhận các tham số từ người dùng

❑ Syntax

❖ @RequestParam(name[, defaultValue][, required]) Type value

- name: Tên tham số
- defaultValue: Giá trị mặc định
- required: Bắt buộc phải có hay không [true]

❑ Example

```
@PostMapping("/account/login")
```

```
public String login(
```

```
    @RequestParam("username") String un,
```

```
    @RequestParam("password") String pw,
```

```
    @RequestParam(name="remember", defaultValue = "false") boolean rm){
```

```
    // Xử lý tham số
```

```
    return "account/login";
```

```
}
```

- ❑ Kết hợp với `java.util.Optional<T>` để tạo ra các mapping method có độ tùy biến cao

```
@GetMapping("/product/list")
public String list(@RequestParam("category_id") Optional<Integer> cid){
    if(cid.isPresent()) {
        // Truy vấn sản phẩm theo loại
    }
    else {
        // Truy vấn tất cả sản phẩm
    }
    return "product/list";
}
```

❑ @RequestPart được sử dụng để nhận file upload từ client

❑ Syntax

❖ @RequestPart(name[, required]) MultipartFile value

➤ name: Tên tham số

➤ required: Bắt buộc phải có hay không [true]

❑ Example

```
@PostMapping("/upload/image")
```

```
public String upload(@RequestPart("photo_file") MultipartFile file){
```

```
    // Xử lý file upload
```

```
    return "upload/image";
```

```
}
```


- ❑ Spring MVC cho phép sử dụng `HttpServletRequest` để nhận tham số người dùng như trong lập trình Servlet
- ❑ Example

```
@Autowired
```

```
HttpServletRequest request;
```

```
@PostMapping("/account/login")
```

```
public String login(){
```

```
    String un = request.getParameter("username");
```

```
    boolean rm = Boolean.parseBoolean(request.getParameter("remember"));
```

```
    // Xử lý tham số
```

```
    return "account/login";
```

```
}
```

- ❑ Tạo bean class có các thuộc tính cùng tên với các tham số

```
public class Account {  
    String username;  
    String password;  
    boolean remember;  
    getters/setters  
}
```

- ❑ Sử dụng bean để tiếp nhận các tham số cùng tên thuộc tính

```
@PostMapping("/account/login")  
public String login(Account account){  
    // Xử lý tham số  
    return "account/login";  
}
```

□ HttpServletRequest

- ❖ Ưu: giống servlet (gần gũi)
- ❖ Nhược
 - Phải tự chuyển kiểu
 - Nhiều tham số

□ @RequestParam(name, defaultValue)

- ❖ Ưu:
 - Tự động chuyển đổi kiểu dữ liệu
 - Giá trị mặc định
- ❖ Nhược: Nhiều tham số

□ JavaBean

- ❖ Ưu: Code đơn giản, rõ ràng với nhiều tham số
- ❖ Nhược: Phải viết JavaBean

Tùy cơ mà ứng biến

Lời khuyên

□ Syntax

❖ @CookieValue(name[, defaultValue][, required]) String value

- name: Tên tham số
- defaultValue: Giá trị mặc định nếu không tồn tại
- required: Bắt buộc phải có hay không [true]

□ Example

```
@GetMapping("/account/login")
public String login(
    @CookieValue(name="account", defaultValue = "") String acc){
    // Xử lý cookie acc
    return "account/login";
}
```

□ Syntax

❖ @PathVariable(name, required) Type value

- name: Tên biến đường dẫn
- required: Có bắt buộc hay không [true]

□ Example

```
@GetMapping("/product/detail/{id}")  
public String detail(@PathVariable("id") Integer id){  
    // Truy vấn sản phẩm theo id  
    return "product/detail";  
}
```

2



DATA SHARING AND MAPPING METHOD RETURN

GIẢNG VIÊN:



DATA SHARING

- ❑ Model là nơi chứa dữ liệu do các thành phần khác tạo ra để chia sẻ với View
- ❑ Trong Controller có 3 cách để đưa dữ liệu (attribute) vào Model với tên là name và giá trị là value.
 - ❖ `Model.addAttribute(name, value)`
 - ❖ `MappingMethod(@ModelAttribute(name) Type value)`
 - ❖ `@ModelAttribute(name) Type method(){...return value;}`


```
@RequestMapping("/sharer/index")
public String index(Model model){
    model.addAttribute("message", "Hello Spring");
    model.addAttribute("Hello Spring");
    model.addAttribute("now", new Date());
    model.addAttribute(new Date());
    model.addAttribute("user", new Account());
    model.addAttribute(new Account());
    return "sharer/index";
}
```


Model

name	value
message	"Hello Spring"
string	"Hello Spring"
now	Date object
date	Date object
user	Account object
account	Account Object

❑ Nguyên tắc tạo tên là lấy tên kiểu của biến và đổi ký tự đầu tiên sang ký tự thường

```
@RequestMapping("/sharer/index")  
public String index(@ModelAttribute Date now,  
    @ModelAttribute("user") Account account){  
    return "sharer/index";  
}
```

Model



name	value
date	Date object
user	Account Object

- ❑ *Spring tự tạo đối tượng với constructor không tham số và gán cho đối số có @ModelAttribute*

```
@ModelAttribute("now")
public Date getNow() {
    return new Date();
}

@ModelAttribute
public Account getUser() {
    return new Account();
}
```



Model

name	value
now	Date object
account	Account Object

- ❑ *Spring lấy kết quả trả về của phương thức đưa vào Model với tên do người dùng định nghĩa hoặc tự sinh.*



MAPPING METHOD RESULT

MAPPING METHOD RESULTS

```
@RequestMapping("url")
public String method() {
    return "view";
}
```

Tên view

```
@RequestMapping("url")
public void method() {
}
```

Tên view = url

```
@ResponseBody
@RequestMapping("url")
public String method() {
    return "Hello Spring MVC";
}
```

Dữ liệu trực tiếp

```
@RequestMapping("url")
public String method() {
    return "forward:/other-url";
}
```

Chuyển tiếp sang url khác (same request)

```
@RequestMapping("url")
public String method() {
    return "redirect:/other-url";
}
```

Chuyển hướng sang url khác (other request)


```
@RequestMapping("/demo/1")  
public String method1(){  
    return "demo/form";  
}
```

demo/form.jsp

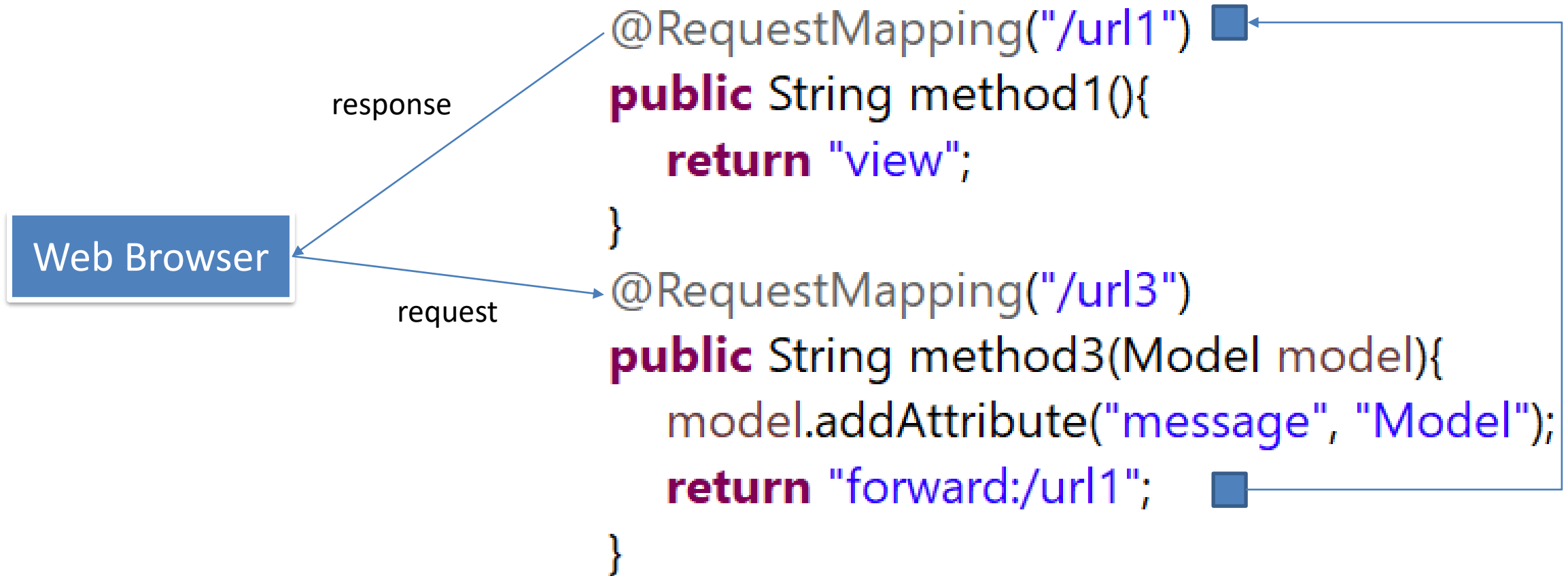


```
@RequestMapping("/demo/2")  
public void method2(){  
}
```

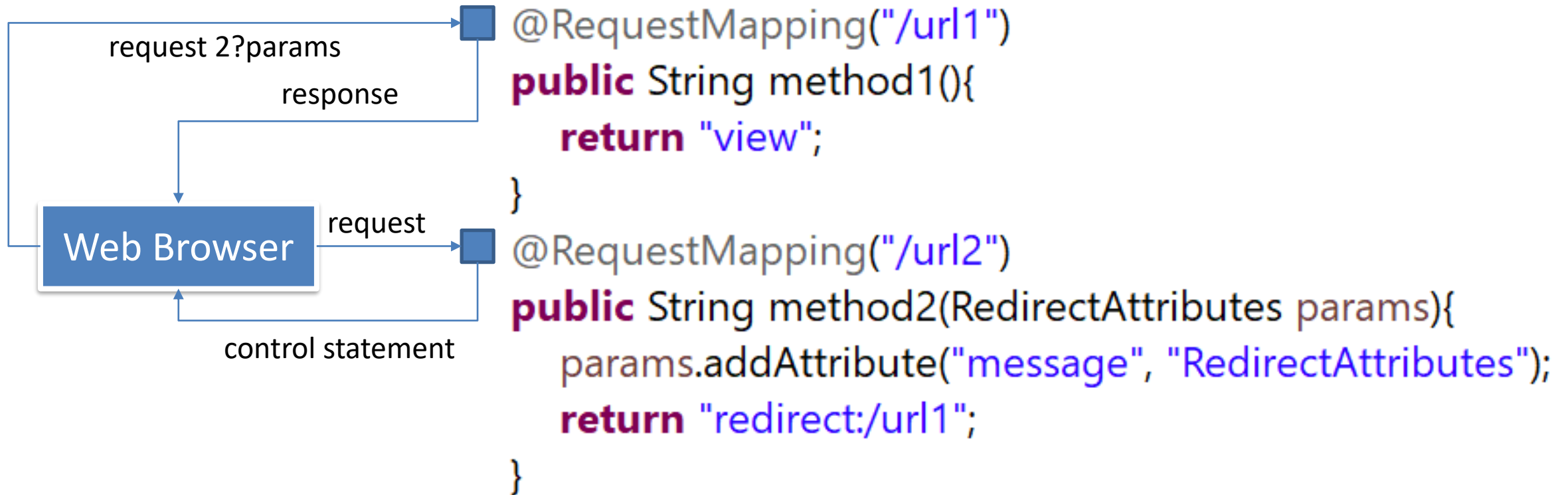
demo/2.jsp



❑ *Nếu kiểu trả về của mapping method là void thì Spring sẽ lấy url làm view name*



- ❑ *Forward xảy ra phía server trên cùng một request nên dữ liệu trong Model được tạo bởi url3 có thể chuyển sang url1*



- ❑ *Redirect sẽ trả về một lệnh điều khiển, yêu cầu trình duyệt tạo một request khác đến url1*
- ❑ *Dữ liệu Model không thể chia sẻ giữa url2 và url1 thay vào đó là các tham số đặt vào RedirectAttributes*


```
@ResponseBody  
@RequestMapping("/url1")  
public String method1(){  
    return "Chào quý vị đại biểu";  
}
```

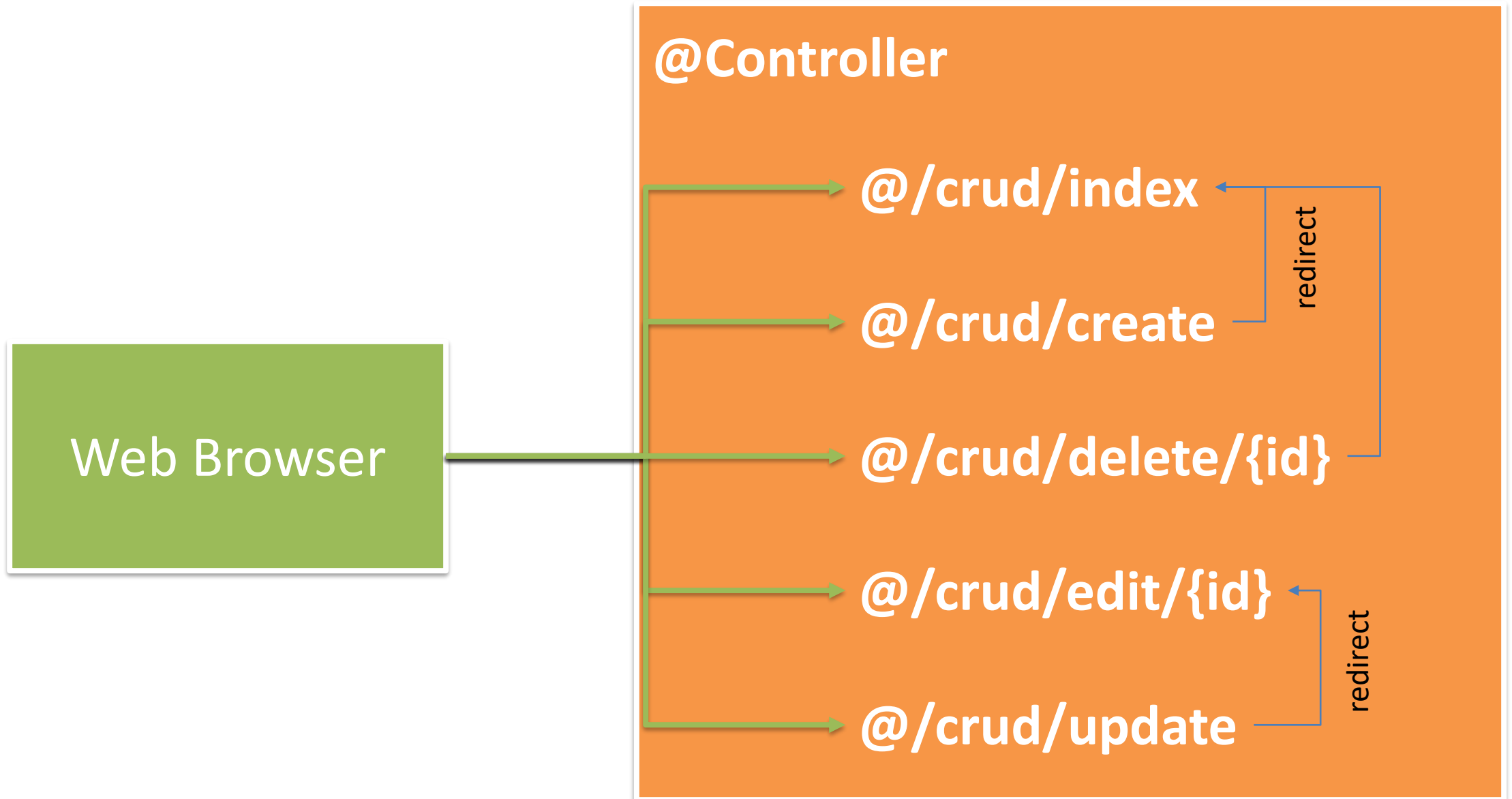
 “Chào quý vị đại biểu”

```
@ResponseBody  
@RequestMapping("/url2")  
public void method2(){  
}
```

 empty

```
@ResponseBody  
@RequestMapping("/url3")  
public Account method3(){  
    return new Account();  
}
```

 {JSON}



✓ Request Mapping

✓ @RequestMapping, @GetMapping, @PostMapping

✓ User Data Handling

✓ @RequestParam, @RequestPart, @PathVariable, @CookieValue

✓ Data Sharing (Model)

✓ Model, @ModelAttribute

✓ Mapping method return

✓ ViewName

✓ Redirect:

✓ Forward:

✓ @ResponseBody (Raw Data)





FPT Education

FPT POLYTECHNIC

Thank you