

# Lập trình web SPRING MVC



## BÀI 2: CONTROLLER

# Mục tiêu :

- ❖ Sử dụng thành thạo **@RequestMapping**
  - ⊙ Ánh xạ nhiều action
  - ⊙ Ánh xạ phân biệt POST|GET
  - ⊙ Ánh xạ phân biệt tham số
- ❖ **Nắm vững phương pháp nhận tham số**
  - ⊙ Sử dụng `HttpServletRequest`
  - ⊙ Sử dụng `@RequestParam`
  - ⊙ Sử dụng `JavaBean`
  - ⊙ Sử dụng `@PathVariable` để nhận dữ liệu từ URL
- ❖ Sử dụng **@CookieValue** để nhận cookie
- ❖ Hiểu rõ kết quả của phương thức action





# PHẦN 1

## @RequestMapping (1)

- ❑ Annotation **@RequestMapping** được sử dụng để ánh xạ một action đến một phương thức action trong Controller

```
@RequestMapping("say-hello")  
public String sayHello() {  
    return "hello";  
}
```

- ❑ Khi người dùng đưa ra yêu cầu **say-hello.htm** thì phương thức action sayHello() sẽ thực hiện
- ❑ Trong một lớp @Controller có thể chứa nhiều phương thức action.

## @RequestMapping (2)

- ❑ @RequestMapping("say-hello") là cách viết thu gọn của @RequestMapping(**value**="say-hello")
- ❑ @RequestMapping() có thể được sử dụng để **đặt trên lớp Controller** để ánh xạ chung cho nhiều action method

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



## @RequestMapping (3)

@Controller

```
public class HomeController{
```

```
    @RequestMapping("/home/index")
```

```
    public String index() {
```

```
        return "home/index";
```

```
    }
```

```
    @RequestMapping("/home/about")
```

```
    public String about() {
```

```
        return "home/about";
```

```
    }
```

```
}
```

home/index.htm

home/about.htm

@Controller

```
@RequestMapping("/home/")
```

```
public class HomeController{
```

```
    @RequestMapping("index")
```

```
    public String index() {
```

```
        return "home/index";
```

```
    }
```

```
    @RequestMapping("about")
```

```
    public String about() {
```

```
        return "home/about";
```

```
    }
```

```
}
```

❑ Hai cách ánh xạ này hoàn toàn tương đương nhau

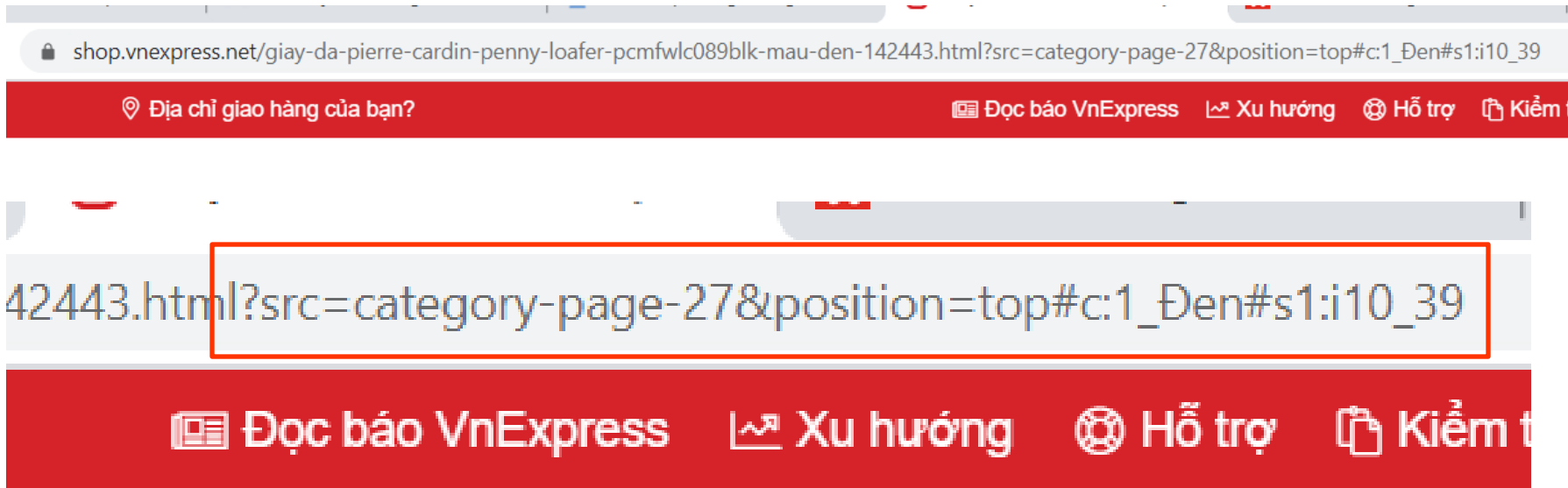
## PHÂN BIỆT POST|GET (1)

❑ Trong lập trình web. Để xử lý việc nhận gửi thông tin từ 1 form của người dùng nhập vào là việc rất thường xuyên. Chúng ta thường sử dụng 2 phương thức POST và GET. Tuy nhiên lúc nào sử dụng POST, lúc nào sử dụng GET? Sau đây là sự giống nhau và khác biệt giữa chúng.

❖ Phương thức GET rất dễ nhận thấy đó là trên URL sẽ kèm theo dữ liệu mà chúng ta muốn gửi. Client gửi lên Phương thức GET là phương thức gửi dữ liệu thông qua đường dẫn URL nằm trên thanh địa chỉ của Browser. Server sẽ nhận đường dẫn đó và phân tích trả về kết quả cho bạn. Server sẽ phân tích tất cả những thông tin đăng sau dấu hỏi (?) chính là phần dữ liệu mà Client gửi lên.

# PHÂN BIỆT POST|GET (2)

□ Ví dụ:



Với URL [trên](#) thì Server sẽ nhận được giá trị

src=**category-page-27&position=top#c:1\_Đen#s1:i10\_39**




## PHÂN BIỆT POST|GET (3)


- ❖ Phương thức POST có tính bảo mật hơn vì dữ liệu gửi phải thông qua một form HTML nên nó bị ẩn, nghĩa là chúng ta không thể thấy các giá trị đó được. Client Gửi Lên với phương thức GET thì dữ liệu được thấy trên URL thì phương thức POST thì hoàn toàn ngược lại, POST sẽ gửi dữ liệu qua một cái form HTML và các giá trị sẽ được định nghĩa trong các input gồm các kiểu (textbox, radio, checkbox, password, textarea, hidden) và được nhận dạng thông qua tên (name) của các input đó.

## PHÂN BIỆT POST|GET (4)

□ Ví dụ:

**Classy Login Form**



 User name

 .....

☒ Remember me [Forgot Password?](#)

Login

or

 Facebook  Twitter

# PHÂN BIỆT POST|GET (5)

## GET

## POST

### ❖ Giống nhau :

- 1.GET và POST đều là hai phương thức của giao thức HTTP.
- 2.Đều gửi dữ liệu về server xử lí, sau khi người dùng nhập thông tin vào form và thực hiện submit.
- 3.Trước khi gửi thông tin, nó sẽ được mã hóa bằng cách sử dụng một giản đồ gọi là url encoding. Giản đồ này là các cặp name/value được kết hợp với các kí hiệu = và các kí hiệu khác nhau được ngăn cách bởi dấu &. Các khoảng trống được xóa bỏ, thay thế bằng kí tự + và bất kì kí tự không phải dạng số và chữ được thay thế bằng giá trị hexa. Sau khi thông tin được mã hóa, nó sẽ được gửi lên Server

## PHÂN BIỆT POST|GET (6)

GET	POST
Phương thức GET gửi thông tin người dùng đã được mã hóa được phụ thêm vào yêu cầu trang, truyền thông tin thông qua url.	Phương thức POST truyền thông tin thông qua HTTP header
Dữ liệu của METHOD GET gửi đi thì hiện trên thanh địa chỉ (URL) của trình duyệt.	Dữ liệu được gửi đi với METHOD POST thì không hiển thị trên thanh URL
HTTP GET có thể được cache bởi trình duyệt	HTTP POST không cache bởi trình duyệt
HTTP GET có thể duy trì bởi lịch sử đó cũng là lý do mà người dùng có thể bookmark được.	HTTP POST không thể duy trì bởi lịch sử đó cũng là lý do mà người dùng không thể bookmark HTTP POST được.
Không bảo mật	Bảo mật
Thực thi nhanh hơn POST vì những dữ liệu gửi đi luôn được webbrowser cached lại.	Thực thi chậm hơn GET

## PHÂN BIỆT POST|GET (7)

GET	POST
Phương thức GET ứng với cùng một yêu cầu đó webbrowser sẽ xem trong cached có kết quả tương ứng với yêu cầu đó không và trả về ngay không cần phải thực thi các yêu cầu đó ở phía server.	Khi dùng phương thức POST thì server luôn thực thi và trả về kết quả cho client
Phương thức GET được giới hạn gửi tối đa chỉ 2048 ký tự	Phương thức POST không có bất kì hạn chế nào về kích thước dữ liệu sẽ gửi.
Không gửi được nhị phân.	Phương thức POST có thể sử dụng để gửi ASCII cũng như dữ liệu nhị phân.
Không bao giờ sử dụng phương thức GET nếu gửi password hoặc thông tin nhạy cảm lên Server.	Dữ liệu gửi bởi phương thức POST thông qua HTTP header, vì vậy việc bảo mật phụ thuộc vào giao thức HTTP. Bằng việc sử dụng Secure HTTP, bạn có thể chắc chắn rằng thông tin của mình là an toàn.

## PHÂN BIỆT POST|GET (8)

GET	POST
Gửi lại form Với form gửi đi bằng phương thức GET bạn có thể gửi lại bằng cách bấm phím F5 hoặc Ctrl + R	Nếu bạn muốn thực hiện việc gửi lại dữ liệu của form thì trình duyệt sẽ hiển thị một hộp thoại cảnh báo. Trở lại trang trước
Dữ liệu gửi đi được lưu lại trong lịch sử web và có thể xem lại	Không được lưu lại trong lịch sử
Trong trường hợp bạn đã gửi form dữ liệu đi rồi sau đó bấm phím Backspace để quay lại trang trước thì với phương thức GET bạn sẽ vẫn được cùng một nội dung (chứa form).	Với POST thì bạn sẽ thấy một trang trống.
Đối với dữ liệu ít thay đổi thường dùng phương thức GET để truy xuất và xử lý nhanh hơn.	Đối với những dữ liệu luôn được thay đổi thì thường sử dụng phương thức POST
Dữ liệu không cần bảo mật thì dùng phương thức GET	Dữ liệu bảo mật thì dùng phương thức POST.



## PHÂN BIỆT POST|GET (9)

❑ Trong Servlet khi yêu cầu từ người dùng gửi đến server với phương thức web là GET thì phương thức doGet() của Servlet được thực hiện, ngược lại nếu phương thức web là POST thì doPost() được thực hiện

❑ Chú ý:

❖ Trường hợp POST duy nhất là khi bạn submit một form có thuộc tính method="POST".

❖ Các trường hợp GET thường gặp

- Nhập url vào ô địa chỉ của trình duyệt web
- Nhấp vào liên kết
- Submit form với method="GET"

## PHÂN BIỆT POST|GET (10)

- ❑ Trong Spring MVC phân biệt POST|GET thông qua tham số method của phương thức action

```
@RequestMapping(value="login", method=RequestMethod.GET)  
public String login() {  
    return "user/login";  
}
```

```
@RequestMapping(value="login", method=RequestMethod.POST)  
public String login(ModelMap model, HttpServletRequest request) {  
    return "user/login";  
}
```

- ❑ Như vậy khi yêu cầu user/login.htm được gửi đến server, Spring MVC sẽ gọi phương thức login() nào là tùy thuộc vào phương thức web **GET** hay **POST**

## PHÂN BIỆT POST|GET (11)

- ❑ Thông thường GET là để vào giao diện còn POST được sử dụng để xử lý các nút chức năng

```
@RequestMapping(value="login", method=RequestMethod.GET)
public String login() {
    return "user/login";
}
```

```
@RequestMapping(value="login", method=RequestMethod.POST)
public String login(ModelMap model, HttpServletRequest request) {
    String id = request.getParameter("id");
    String pw = request.getParameter("password");
    if(id.equals("fpt") && pw.equals("polytechnic")){
        model.addAttribute("message", "Sai thông tin đăng nhập !");
    }
    else{
        model.addAttribute("message", "Sai thông tin đăng nhập !");
    }
    return "user/login";
}
```

## PHÂN BIỆT THAM SỐ (1)

- ❑ Trong Spring MVC không những hỗ trợ gọi phương thức action phân biệt theo phương thức web mà còn cho phép phân biệt theo tham số truyền theo

```
@RequestMapping(value="say-hello", params="mvc")  
public String sayHello() {  
    return "hello";  
}
```

**say-hello.htm?mvc**

- ❑ Với định nghĩa này khi gọi say-hello.htm phải có tham số mvc thì phương thức sayHello() mới được thực hiện

## PHÂN BIỆT THAM SỐ (2)

❑ Khi yêu cầu student.htm thì phương thức nào sẽ thực hiện?

```
@Controller
@RequestMapping("student")
public class StudentController {
    @RequestMapping()
    public String index(ModelMap model) {...}

    @RequestMapping(params="btnInsert")
    public String insert(ModelMap model) {...}

    @RequestMapping(params="btnUpdate")
    public String update(ModelMap model) {...}

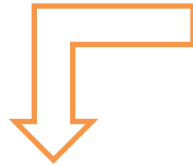
    @RequestMapping(params="btnDelete")
    public String delete(ModelMap model) {...}

    @RequestMapping(params="lnkEdit")
    public String edit(ModelMap model) {...}
}
```

- Nếu có tham số **btnInsert**, **btnUpdate**, **btnDelete** hoặc **lnkEdit** thì các phương thức **insert()**, **update()**, **delete()** hoặc **edit()** sẽ thực hiện
- Nếu không có các tham số trên thì **index()** sẽ thực hiện
- Nếu có nhiều hơn 1 tham số trên thì sẽ báo lỗi

# TRANG SAU SẼ GỌI STUDENT.HTM

- ❑ [Thêm] => insert()
- ❑ [Cập nhật] => update()
- ❑ [Xóa] => delete()
- ❑ [Nhập lại] => index()
- ❑ [Sửa] => edit()



Họ và tên	<input type="text"/>												
Điểm TB	<input type="text"/>												
Chuyên ngành	<input type="text"/>												
<input type="button" value="Thêm"/> <input type="button" value="Cập nhật"/> <input type="button" value="Xóa"/> <input type="button" value="Nhập lại"/>													
<table><thead><tr><th>Họ và tên</th><th>Điểm TB</th><th>Chuyên ngành</th><th></th></tr></thead><tbody><tr><td>Lê Phạm Tuấn Kiệt</td><td>8.5</td><td>CNTT</td><td><a href="#">Sửa</a></td></tr><tr><td>Bùi Minh Nhật</td><td>7.5</td><td>MUL</td><td><a href="#">Sửa</a></td></tr></tbody></table>		Họ và tên	Điểm TB	Chuyên ngành		Lê Phạm Tuấn Kiệt	8.5	CNTT	<a href="#">Sửa</a>	Bùi Minh Nhật	7.5	MUL	<a href="#">Sửa</a>
Họ và tên	Điểm TB	Chuyên ngành											
Lê Phạm Tuấn Kiệt	8.5	CNTT	<a href="#">Sửa</a>										
Bùi Minh Nhật	7.5	MUL	<a href="#">Sửa</a>										

```
<form action="student.htm" method="post">
    ...
    <button name="btnInsert">Thêm</button>
    <button name="btnUpdate">Cập nhật</button>
    <button name="btnDelete">Xóa</button>
    <button name="btnReset">Nhập lại</button>
</form>
<hr>
<table border="1" style="width:100%">
    <tr>
        <td>Lê Phạm Tuấn Kiệt</td>
        <td>8.5</td>
        <td>CNTT</td>
        <td><a href="student.htm?lnkEdit">Sửa</a> </td>
    </tr>
</table>
```





## PHẦN 2

## XỬ LÝ THAM SỐ NGƯỜI DÙNG

❑ Tham số là dữ liệu truyền đến server khi có yêu cầu từ người dùng dưới dạng các trường của form hoặc chuỗi truy vấn của liên kết

❑ Ví dụ

❖ Khi nhấp vào liên kết sau thì các tham số **mark** và **name** sẽ được truyền đến phương thức action

```
<a href="say-hello.htm?mark=5&name=Phương">Hello</a>
```

❖ Khi nhấp vào nút Hello của form sau thì các tham số **mark** và **name** sẽ được truyền đến phương thức action

```
<form action="say-hello.htm">
```

```
  <input name="mark">
```

```
  <input name="name">
```

```
  <button>Hello</button>
```

```
</form>
```

## XỬ LÝ THAM SỐ

- ❑ Spring MVC cung cấp các phương pháp nhận tham số sau đây
  - ❖ Sử dụng **HttpServletRequest** tương tự Servlet
  - ❖ Sử dụng **@RequestParam**
  - ❖ Sử dụng **JavaBean**
  - ❖ Sử dụng **@PathVariable** để nhận một phần trên URL

# SỬ DỤNG HTTPServletRequest

- ❑ Chỉ cần thêm đối số `HttpServletRequest` vào phương thức action là có thể nhận được tham số người dùng như `Servlet`

```
@RequestMapping(value="login", method=RequestMethod.POST)  
public String login(HttpServletRequest request) {  
    String id = request.getParameter("id");  
    String pw = request.getParameter("password");  
    // ...  
    return "login";  
}
```

## SỬ DỤNG @RequestParam (1)

- ❑ Sử dụng **@RequestParam** thể hiện tính chuyên nghiệp hơn và có thể chuyển đổi tự động sang kiểu mong muốn.
- ❑ Ví dụ sau được sử dụng để nhận các tham số có tên là id và password

```
@RequestMapping(value="login", method=RequestMethod.POST)  
public String login(@RequestParam("id") String id,  
                    @RequestParam("password") String password) {  
    // ...  
    return "login";  
}
```

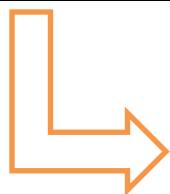
## SỬ DỤNG @RequestParam (2)

- ❑ **@RequestParam**(value, defaultValue, required) là dạng đầy đủ với ý nghĩa của các tham số:
  - ❖ value: chỉ ra tên tham số muốn nhận
  - ❖ defaultValue: là giá trị mặc định của tham số khi tham số không tồn tại
  - ❖ required: tham số có bắt buộc hay không
- ❑ Ví dụ với khai báo nhận tham số sau
  - ❖ **@RequestParam**(value="tuoi", defaultValue="20", required=false) **Integer age**
    - Tên tham số là **tuoi** sẽ được nhận vào đối số là **age**
    - Nếu không có tham số thì giá trị của **age** là 20
    - Tham số **tuoi** là không bắt buộc



## VÍ DỤ

```
@RequestMapping(value="login", method=RequestMethod.POST)
public String login(ModelMap model, HttpServletRequest request) {
    String id = request.getParameter("id");
    String pw = request.getParameter("password");
    if(id.equals("fpt") && pw.equals("polytechnic")){
        model.addAttribute("message", "Sai thông tin đăng nhập !");
    }
    else{
        model.addAttribute("message", "Sai thông tin đăng nhập !");
    }
    return "user/login";
}
```



```
@RequestMapping(value="login", method=RequestMethod.POST)
public String login(ModelMap model,
    @RequestParam("id") String id, @RequestParam("password") String pw) {
    if(id.equals("fpt") && pw.equals("polytechnic")){
        model.addAttribute("message", "Sai thông tin đăng nhập !");
    }
    else{
        model.addAttribute("message", "Sai thông tin đăng nhập !");
    }
    return "user/login";
}
```

# SỬ DỤNG JAVA BEAN (1)

- ❑ Lớp JavaBean là lớp thỏa mãn các qui ước sau
  - ❖ Phải được định nghĩa là public
  - ❖ Phải có constructor không tham số
  - ❖ Đọc ghi dữ liệu thông qua getter/setter

```
package nhatnghe.model;

public class User {
    private String id;
    private String password;

    public String getId() {return id;}
    public void setId(String id) {this.id = id;}
    public String getPassword() { return password;}
    public void setPassword(String password) {this.password = password;}
}
```

## SỬ DỤNG JAVABEAN (2)

- ❑ Thuộc tính của bean được xác định từ các getter và setter bằng cách
  - ❖ Bỏ get và set và đổi ký tự đầu tiên của phần còn lại sang ký tự thường
- ❑ Ví dụ lớp User có 2 thuộc tính cho phép đọc/ghi là id và password
  - ❖ Thuộc tính id được xác định từ get**Id**() và set**Id**()
  - ❖ Thuộc tính password được xác định từ get**Password**() và set**Password**()
- ❑ *Chú ý quan trọng: các trường dữ liệu không phải là thuộc tính của bean*

## SỬ DỤNG JAVA BEAN (3)

- ❑ Spring MVC cho phép sử dụng JavaBean để nhận các tham số **cùng tên** với các thuộc tính của bean.

```
@RequestMapping(value="login", method=RequestMethod.POST)  
public String login(ModelMap model, User user) {  
    if(user.getId().equals("fpt") && user.getPassword().equals("polytechnic")){  
        model.addAttribute("message", "Sai thông tin đăng nhập!");  
    }  
    else{  
        model.addAttribute("message", "Sai thông tin đăng nhập!");  
    }  
    return "user/login";  
}
```

- ❑ Với ví dụ này thì các thuộc tính id và password của đối số user sẽ nhận các giá trị từ các tham số cùng tên là id và password.

## SỬ DỤNG @PATHVARIABLE

- ❑ Spring MVC cho phép nhận một phần dữ liệu từ đường dẫn URL
- ❑ Ví dụ action edit() sau đây sẽ lấy được tên sinh viên từ URL student/**Nguyễn Văn Tèo**.htm

```
@Controller
@RequestMapping("/student")
public class StudentController {
    @RequestMapping(value="/{name}", params="!nkEdit")
    public String edit(ModelMap model, @PathVariable("name") String name) {
        ...
        return "student";
    }
}
```

`<a href="student/Nguyễn Văn Tèo.htm?nkEdit">Sửa</a>`

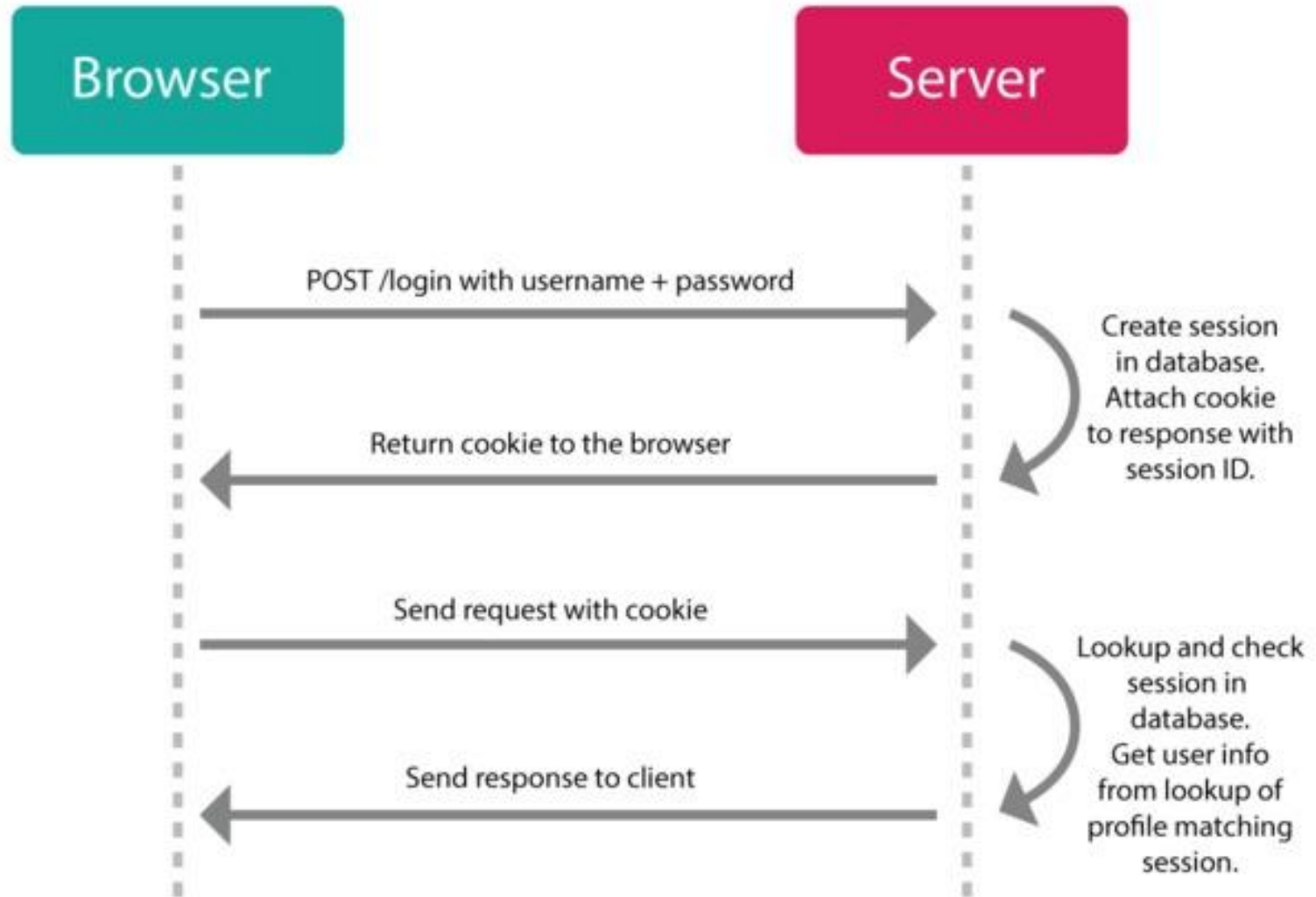
## NHẬN GIÁ TRỊ COOKIE

□ Session là gì? Cookie là gì?

- **Session** là một phiên làm việc là một khái niệm phổ biến được dùng trong lập trình web có kết nối với database. Đặc biệt các chức năng như đăng nhập, đăng xuất người dùng sẽ khó có thể thực hiện được nếu không sử dụng **session**.
- **Cookie** cũng được dùng để lưu những thông tin tạm thời. Nhưng tập tin cookie sẽ được truyền từ server tới browser và được lưu trữ trên máy tính của bạn khi bạn truy cập vào ứng dụng.



# NHẬN GIÁ TRỊ COOKIE



# NHẬN GIÁ TRỊ COOKIE

## ❑ So sánh giữa Cookie và Session

Cookie	Session
<b>Cookie</b> được lưu trữ trên trình duyệt của người dùng.	<b>Session</b> không được lưu trữ trên trình duyệt.
Dữ liệu <b>cookie</b> được lưu trữ ở phía client.	Dữ liệu <b>session</b> được lưu trữ ở phía server.
Dữ liệu <b>cookie</b> dễ dàng sửa đổi hoặc đánh cắp khi chúng được lưu trữ ở phía client.	Dữ liệu <b>session</b> không dễ dàng sửa đổi vì chúng được lưu trữ ở phía máy chủ.
Dữ liệu <b>cookie</b> có sẵn trong trình duyệt đến khi expired.	Sau khi đóng trình duyệt sẽ hết phiên làm việc (session)

## NHẬN GIÁ TRỊ COOKIE

- ❑ Trong Servlet bạn có thể nhận cookie thông qua `HttpServletRequest`. Phương pháp này viết mã khá dài dòng, phức tạp.
- ❑ Trong Spring MVC bạn có thể sử dụng **@CookieValue** để nhận dữ liệu từ cookie

```
@RequestMapping("say-hello")  
public String sayHello(@CookieValue("userid") String id) {  
    return "hello";  
}
```

- ❑ Ví dụ này cho phép sử dụng đối số **id** để nhận giá trị của cookie có tên là **userid**

## ❑ @CookieValue(**value**, **defaultValue**, **required**)

có 3 tham số và ý nghĩa như sau

- ❖ Value: tên cookie muốn nhận dữ liệu
- ❖ defaultValue: giá trị mặc định của cookie
- ❖ Required: có bắt buộc cookie userid có tồn tại hay không

## ❑ Ví dụ

❖ @CookieValue(value="userid", defaultValue="poly", required=false) String id

- Sử dụng đối số id để nhận giá trị của cookie có tên là userid
- Nếu cookie không tồn tại thì giá trị của id là poly
- Cookie này cho phép không tồn tại

## ĐẦU RA CỦA PHƯƠNG THỨC ACTION

❑ Return của phương thức action không đơn thuần phải là tên của view mà có thể là 1 trong 3 trường hợp sau

❖ Tên view => ViewResolver sẽ xử lý để xác định view

➤ return "<**tên view**>"

❖ Nội dung => được trả trực tiếp về client mà không qua ViewResolver. Trường hợp này phương thức action phải được chú thích bởi **@ResponseBody**


➤ return "<**Nội dung**>"

❖ Lời gọi một action khác

➤ return "**redirect:/<action>**"

# ĐẦU RA CỦA PHƯƠNG THỨC ACTION


```
@RequestMapping("say-hello")
public String sayHello() {
    return "hello";
```



→ /WEB-INF/views/hello.jsp

```
}
```

```
@ResponseBody
@RequestMapping("say-hello")
public String sayHello() {
    return "Hello World";
```




→ Hello World

```
}
```

Trường hợp này rất hữu ích  
cho tương tác JSON,  
JavaScript, XML...

```
@RequestMapping("say-hello")
public String sayHello() {
    return "redirect:/home/index.htm";
```



→ @RequestMapping("/home/index")

```
}
```

# TỔNG KẾT NỘI DUNG BÀI HỌC

- ☑ Sử dụng thành thạo @RequestMapping
  - ☑ Ánh xạ nhiều action
  - ☑ Ánh xạ phân biệt POST|GET
  - ☑ Ánh xạ phân biệt tham số
- ☑ Nắm vững phương pháp nhận tham số
  - ☑ Sử dụng HttpServletRequest
  - ☑ Sử dụng @RequestParam
  - ☑ Sử dụng JavaBean
  - ☑ Sử dụng @PathVariable để nhận dữ liệu từ URL
- ☑ Biết cách nhận Cookie với @CookieValue
- ☑ Hiểu rõ kết quả của phương thức action