Lập trình ứng dụng mạng
Chương 3

Lập trình ứng dụng Web với Spring

Nội dung

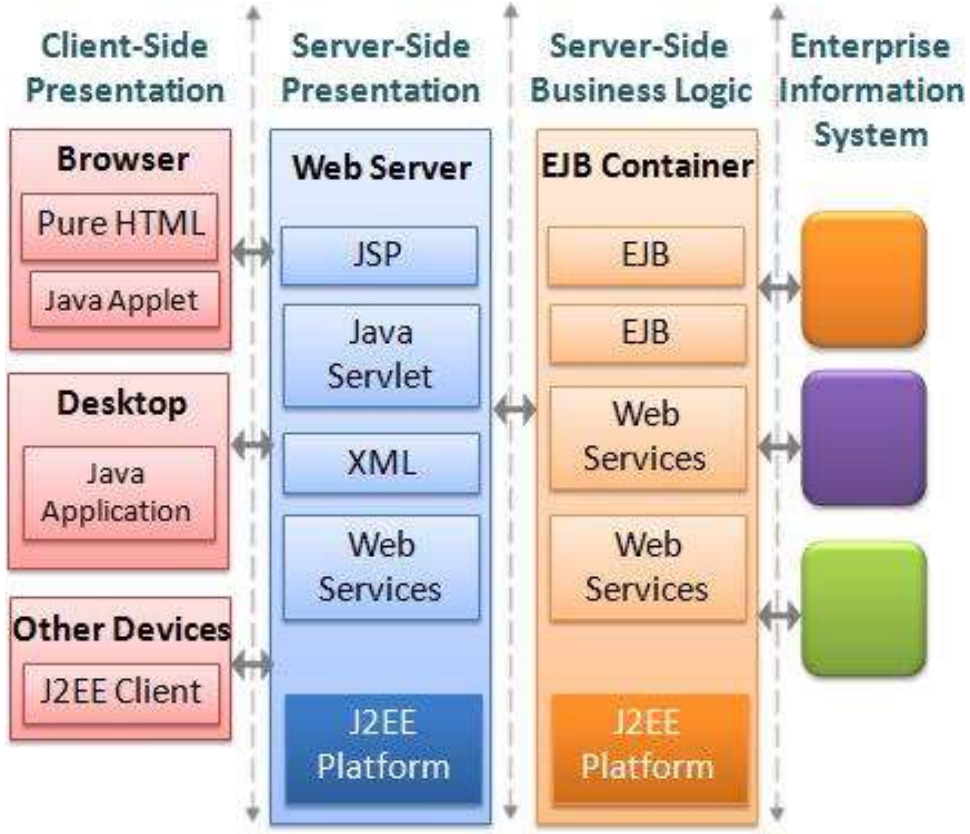
- Phần 1: Lập trình Java Web với JSP và SpringMVC
  - Servlet
  - JSP
  - SpringMVC
- Phần 2: Lập trình Web API với Spring Boot
  - Web API
  - Restful Webservice
  - Spring Data JPA

2 Giới thiệu J2EE

- J2EE (Java 2 Platform, Enterprise Edition) là nền tảng lập trình dành cho phát triển ứng dụng mạng doanh nghiệp: quy mô lớn, phân tán, đa tầng, tin cậy và bảo mật
- Các phiên bản: J2EE(1999), Java EE (2006), Jakarta EE (2019)

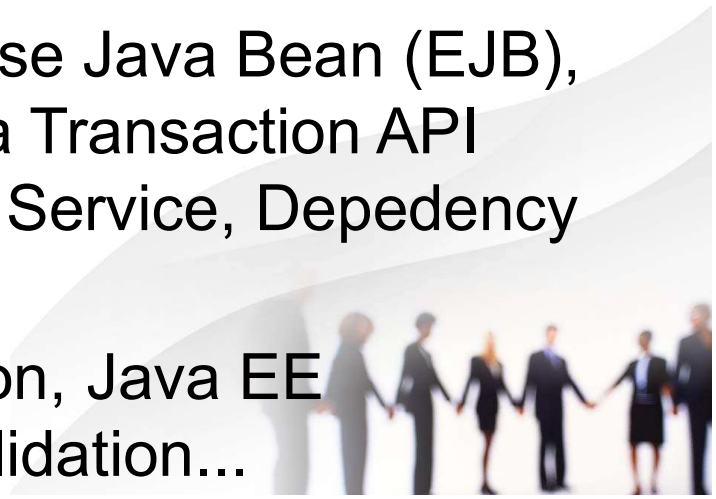
- J2EE gồm các đặc tả kỹ thuật cho nhiều mục đích khác nhau: xây dựng trang web, truy cập cơ sở dữ liệu, Web API, các hệ phân tán,...

- Các công nghệ của J2EE được các framework khác áp dụng: Spring, Hirbernate, Struts,...



2 Giới thiệu J2EE

- Các công nghệ được đặc tả trong J2EE:
  - Lập trình web: Servlet, WebSocket, Java Server Faces (JSF), Java Server Pages (JSP), Unifield Expression Language (EL)
  - Lập trình dịch vụ web: các Java API cho RESTful web services (JAX-RS), SOAP web services(JAX-WS), xử lý JSON và XML...
  - Các đặc tả công nghiệp: Enterprise Java Bean (EJB), Java Persistence API (JPA), Java Transaction API (JTA), JavaMail, Java Messenge Service, Depedency Injection...
  - Các đặc tả khác: Batch Application, Java EE Connector Architecture, Bean Validation...



2 Giới thiệu J2EE

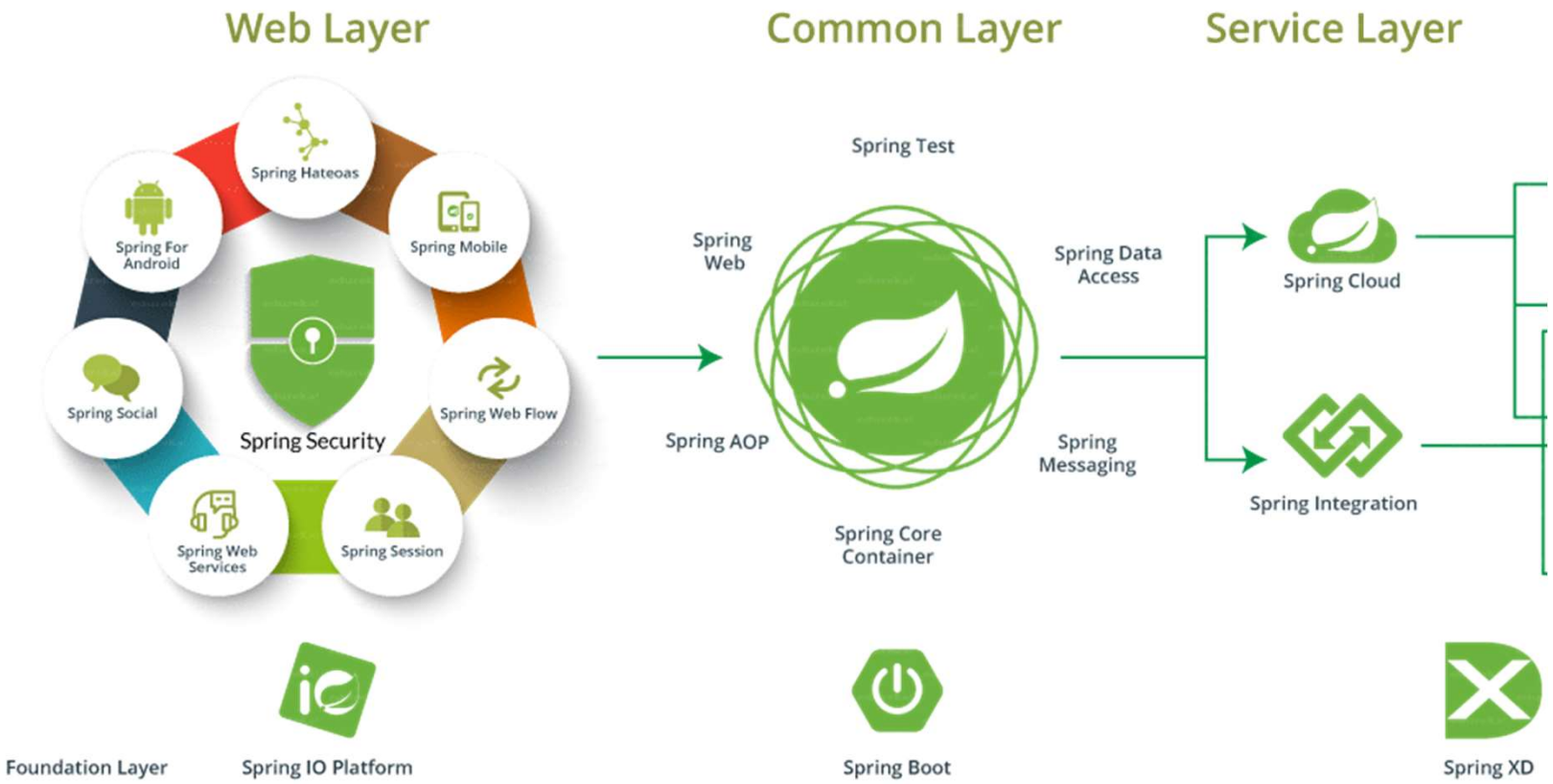
- Ưu điểm của J2EE:
  - J2EE là tập hợp các công nghệ phát triển ứng dụng Java toàn diện và đa năng
  - J2EE được phát triển bởi Oracle, nhà phát triển phầm mềm doanh nghiệp hàng đầu hiện nay
  - Các công nghệ được đặc tả trong J2EE là nền móng để xây dựng nhiều framework phổ biến khác
  - Phần lớn các hệ thống phần mềm doanh nghiệp (Enterprise middleware platform) được viết trên J2EE và Spring
- Vấn đề của J2EE:
  - Phức tạp: các đặc tả trong J2EE rất đa dạng và phức tạp
  - Chồng chéo: nhiều công nghệ trong J2EE có chức năng tương đương nhau gây khó khăn khi lựa chọn
  - Thư viện thiếu nhất quán: các thư viện Java nhìn chung được phát triển độc lập nên dễ gây lỗi khi phối hợp sử dụng

3. Giới thiệu Spring Framework

- Là framework phát triển ứng dụng Java được sử dụng rộng rãi nhất hiện nay.
  - Được viết bởi Rod Johnson (2002), phiên bản đầu tiên ra đời năm 2003.
  - Được phát triển nhằm đơn giản hóa việc phát triển ứng dụng với J2EE
- Đặc điểm:
  - Mã nguồn mở, gọn nhẹ
  - Được xây dựng trên nền tảng J2EE
  - Ứng dụng mô hình IoC (Inversion of Control) và lập trình hướng khía cạnh (AOP)

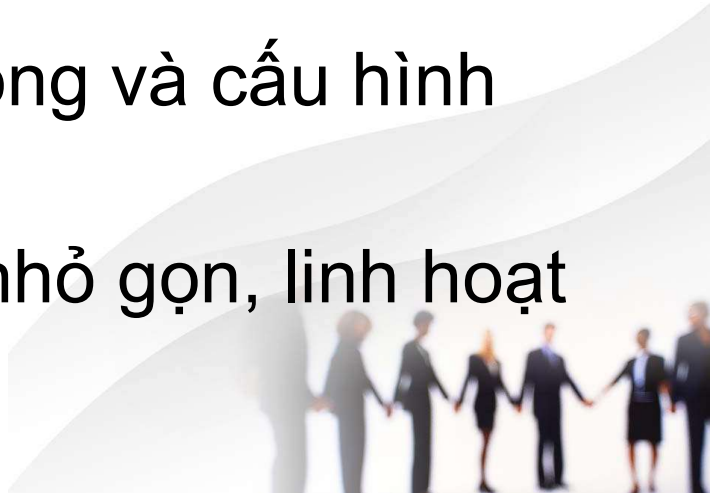
3. Giới thiệu Spring Framework

Các dự án trong Spring



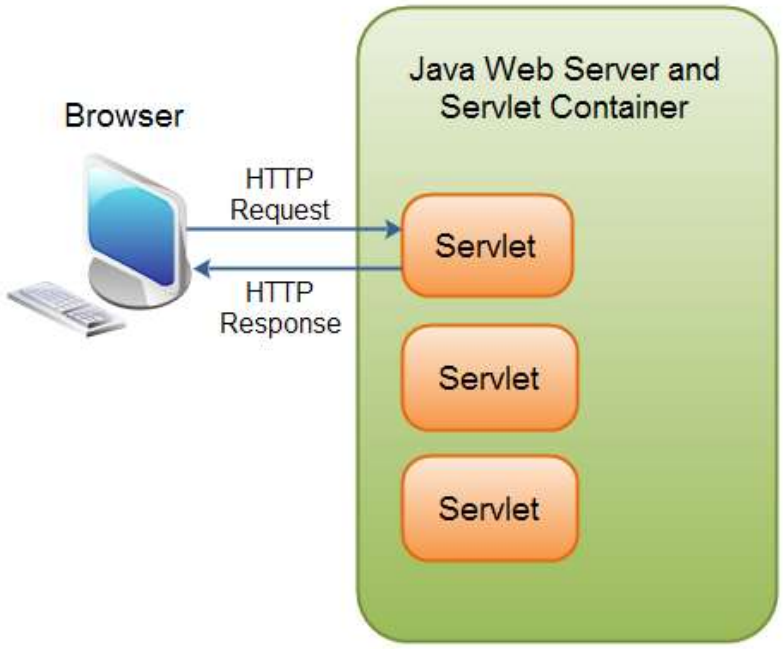
3.Giới thiệu Spring Framework

- Spring Boot là giải pháp giúp đơn giản hóa việc phát triển ứng dụng Spring:
  - Tạo ứng dụng Web có thể chạy độc lập (không cần phần mềm web server)
  - Tự động tạo cấu hình giữa các thư viện và thành phần trong Spring
  - Không chứa code sinh tự động và cấu hình XML
  - Phù hợp với các ứng dụng nhỏ gọn, linh hoạt và cần phát triển nhanh.



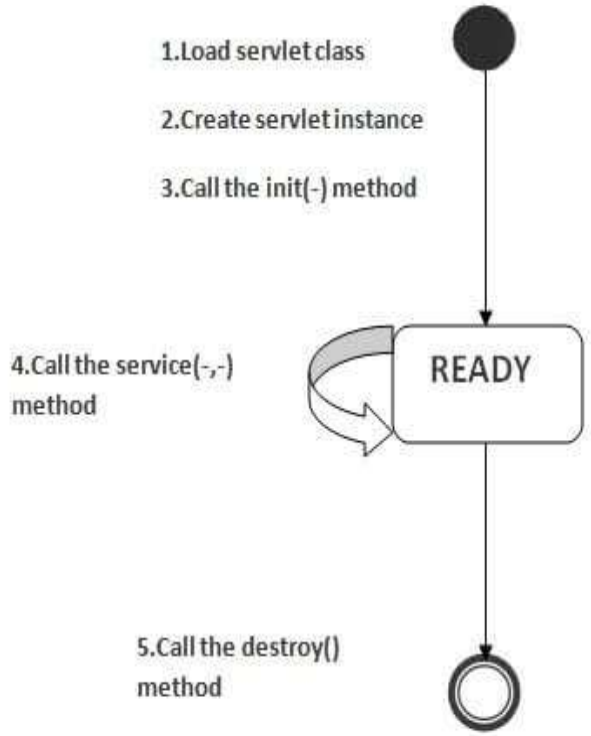
4.Servlet

- Servlet là gì?
  - Servlet là công nghệ của Java cho phép tạo các ứng dụng web theo mô hình request-response
  - Một class Servlet là một thành phần chạy trên máy chủ web :
    - Nhận request gửi lên từ client
    - Tạo phản hồi gửi trả lại client
  - Servlet thường được sử dụng để bổ sung khả năng xử lý truy vấn của ứng dụng web.



4.Servlet

- Vòng đời của Servlet:
  - Giai đoạn khởi tạo: một Servlet được khởi tạo khi có request đầu tiên gọi đến nó.
    - Nạp Servlet class
    - Tạo đối tượng Servlet
    - Gọi hàm init()
  - Giai đoạn làm việc: được thực hiện mỗi lần có request đến Servlet
    - Nhận Request
    - Gọi phương thức xử lý Request
    - Tạo Response
  - Hủy Servlet: hàm destroy() được gọi khi Servlet bị dừng triển khai (tắt ứng dụng web, shutdown,.)



4.Servlet

- Cách tạo một Servlet:
  - Tạo class thừa kế lại HttpServlet
  - Thiết lập chỉ dẫn @WebServlet với một mẫu URL

```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;

@WebServlet("/report")
public class MoodServlet extends HttpServlet {
    ...
}
```

Mẫu URL sẽ định nghĩa đường dẫn truy cập vào Servlet này.

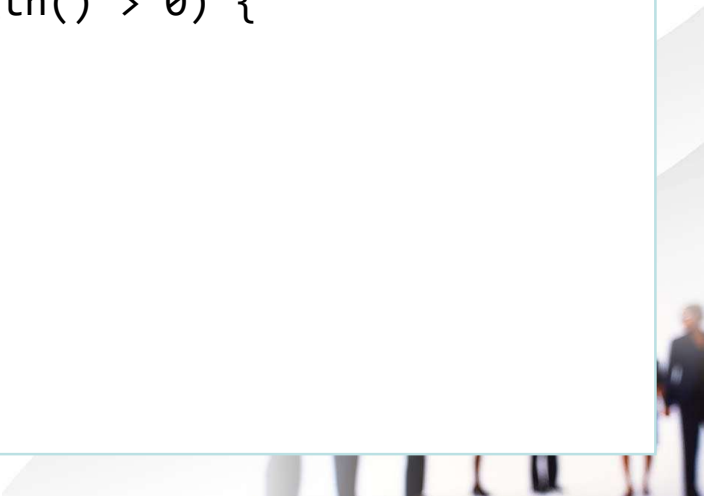
4.Servlet

- Tạo hàm dịch vụ của Servlet
  - Các hàm dịch vụ là các hàm xử lý các request của Servlet
  - Với HttpServlet, các hàm dịch vụ có dạng `doMethod()` với Method là các phương thức HTTP (GET, POST, PUT, DELETE,...)
    - VD: doGet(), doPost()
  - Lấy dữ liệu từ request: các dữ liệu gửi lên từ client đều có thể lấy được từ đối tượng HttpServletRequest:
    - Tham số gửi kèm URL
    - Dữ liệu dạng key-value theo dạng form
    - Các thông tin khác: giao thức, địa phương hóa,..
  - Tạo response gửi trả về client: bản tin phản hồi được tạo bằng đối tượng HttpServletResponse:
    - Dữ liệu được gửi bằng đối tượng PrintWriter
    - Các thông tin liên quan: ContentType, kích thước bộ nhớ đệm khi gửi,..

4.Servlet

- Ví dụ: tạo một servlet xử lý request theo phương thức GET với tham số là username, trả về chuỗi “Hello,”+username

```
@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)throws ServletException, IOException {
        response.setContentType("text/html");
        try (PrintWriter out = response.getWriter()) {
            String username = request.getParameter("username");
            if (username != null && username.length() > 0) {
                out.println("Hello,"+username);
            }
            else{
                out.println("Hello!");
            }
        }
    }
}
```



4.Servlet

- Chuyển hướng đến thành phần web khác trong Servlet:
  - Sử dụng RequestDispatcher để gọi thành phần web khác trong hàm dịch vụ của Servlet. VD: chuyển hướng từ Servlet Login đến Servlet quản lý tài khoản nếu login thành công
  - Khi chuyển hướng, RequestDispatcher có thể gửi kèm request và respone hiện tại của Servlet bằng cách dùng hàm *include*

  VD: chuyển hướng đến thành phần web khác có mẫu URL là "/response" và đính kèm theo request và response hiện tại

```
RequestDispatcher dispatcher =
                        getServletContext().getRequestDispatcher("/response");

                if (dispatcher != null) {
                    dispatcher.include(request, response);
                }
```

4.Servlet

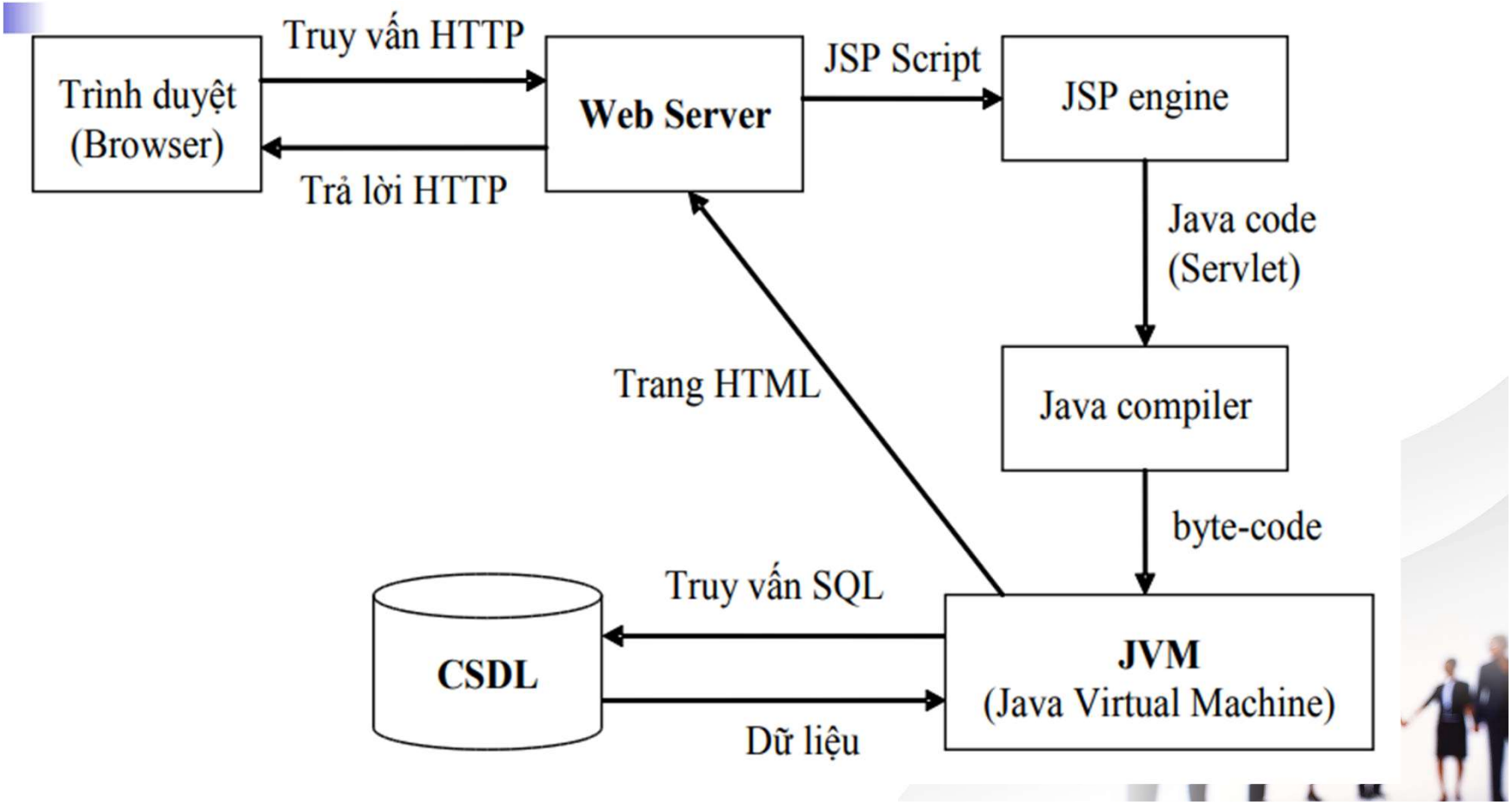
- Các vấn đề khác của Servlet (tự tìm hiểu):
  - Quản lý vòng đời của một Servlet: sử dụng ServletContextListener có thể bắt các sự kiện liên quan đến vòng đời của Servlet
  - Bộ lọc request và response: có thể thiết lập các bộ lọc nằm giữa Servlet và client để thực hiện một số hành động như : chặn request, chỉnh sửa thông tin header hoặc data,..
  - Duy trì phiên làm việc với client: sử dụng đối tượng HttpSession để lưu thông tin phiên làm việc của client với server. VD: giỏ hàng (shopping cart) cần lưu lại các mặt hàng khách hàng đã chọn khi duyệt qua các loại sản phẩm khác nhau.
  - Xử lý file tải lên với Servlet
  - Xử lý bất đồng bộ với Servlet
    - Thông thường, mỗi request của client sẽ phát sinh 1 thread xử lý bên server gắn với request đó -> không đủ tài nguyên nếu lượng request lớn
    - Xử lý bất đồng bộ cho phép Servlet tách tiến trình xử lý của một request ra một thread riêng để giải phóng thead đang gắn với request - > tăng khả năng đáp ứng request của hệ thống

5. Java Server Page

- JSP là công nghệ sử dụng để phát triển web động trên java
- JSP được phát triển dựa trên Servlet -> có thể coi là mở rộng của Servlet
- Một trang JSP là sự kết hợp của mã HTML và mã java (JSP tag)
- Các trang JSP thường đóng vai trò là tầng giao diện (View) của ứng dụng web.
- Để chạy được JSP cần Web Server hỗ trợ
  - VD: Tomcat, Glassfish, WebLogic,…

5. Java Server Page

Nguyên tắc hoạt động



5. Java Server Page

- Ưu điểm của Java Server Page
  - Khả năng biểu diễn trang web dễ dàng và mở rộng hơn so với Servlet nhờ các thẻ JSP
  - Tốc độ nhanh hơn (so với PHP) vì bytecodes thực thi trên JVM nhanh hơn so với mã PHP chạy trên trình thông dịch PHP.
  - Đồng bộ với các công nghệ Java:
    - Sử dụng chính code Java để nhúng vào các thẻ JSP
    - Tương tự servlet, trang JSP có thể tương tác trực tiếp về dữ liệu với các thành phần Java chạy trên web server.

5. Java Server Page

- Ví dụ: hello.jsp

```
<html>
    <head>
        <title>JSP Page</title>
    </head>
    <body>
        <h1>JSP demo</h1>
        <%  String username=request.getParameter("username");
            if(username==null){%>
                <h2>Hello</h2>
            <%} else {%>
                <h2>Hello <% out.print(username);%></h2>
            <%}%>
    </body>
</html>
```

Chạy website và thử gõ /hello.jsp hoặc /hello.jsp?username=yourname

5.1 Thẻ lệnh JSP

- Thẻ lệnh JSP (JSP Scriptlet tag): là các thẻ cho phép viết mã lệnh Java vào nội dung thẻ
- 3 loại thẻ lệnh:
  - Thẻ lệnh (scriptlet tag): **<% java code %>**
    - Các lệnh trong thẻ này được thực thi khi trang JSP được gọi
  - Thẻ diễn tả (expression tag): **<%=statement%>**
    - Giá trị bên trong thẻ này được in ra tại vị trí thẻ
  - Thẻ khai báo (declaration tag): **<%! Field or methods %>**
    - Khai báo biến toàn cục của file JSP và các hàm
    - Được khởi tạo 1 lần khi biên dịch JSP, không giống việc khai báo bên trong thẻ lệnh (các biến trong thẻ lệnh là cục bộ)

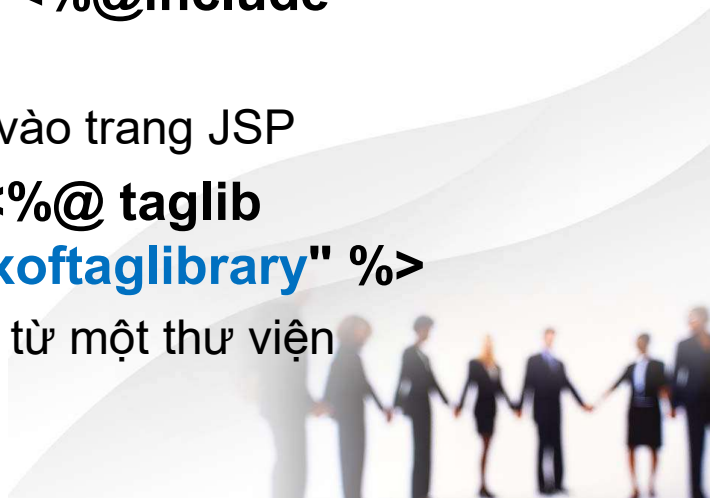
5.1 Thẻ lệnh JSP

- Ví dụ

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>JSP Scriplet tags</h1>
        <%! int count=0;
            int getCount(){return ++count;}
        %>
        <% int c=getCount();
          if(c<5){%>
          <h1>You accessed <%=c%> times</h1>
          <%}else{%>
          <h1>You accessed too much</h1>
          <%}%>
    </body>
</html>
```

5.2 Thẻ chỉ dẫn JSP

- Thẻ chỉ dẫn JSP (JSP directives): là các thẻ thông báo cho web server cách biên dịch trang JSP
- 3 loại thẻ chỉ dẫn:
  - Thẻ chỉ dẫn biên dịch trang (page directive): **<%@ page attribute="value" %>**
    - Chỉ dẫn một số tính chất biên dịch cho toàn trang JSP
    - Một số thẻ : import, contentType,…
  - Thẻ chỉ dẫn bao gồm(Include directive): **<%@include file="file"%>**
    - Nhúng nội dung một file (jsp, html, text) vào trang JSP
  - Thẻ chỉ dẫn Taglib (Taglib directive):): **<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>**
    - Khai báo việc sử dụng các thẻ tùy chỉnh từ một thư viện



5.2 Thẻ chỉ dẫn JSP

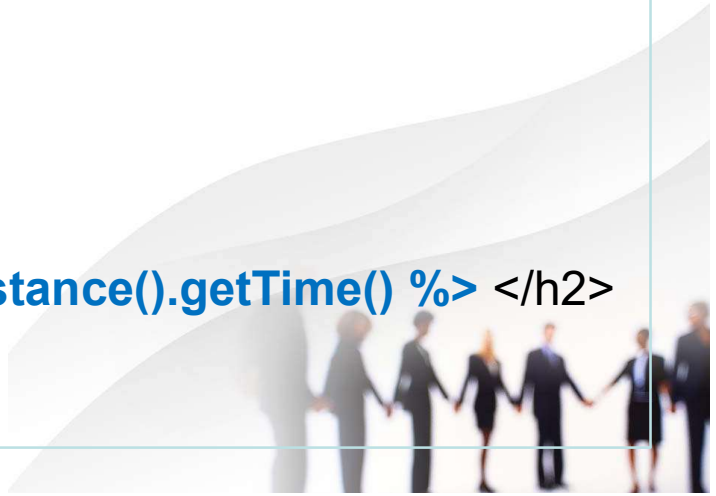
- Ví dụ

header.html

```
<div>
    <h1>This is header from header.html</h1>
</div>
```

hello.jsp

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <%@ page import="java.util.Date" %>
        <%@ include file="header.html" %>
        <h2>Now is <%= java.util.Calendar.getInstance().getTime() %> </h2>
    </body>
</html>
```



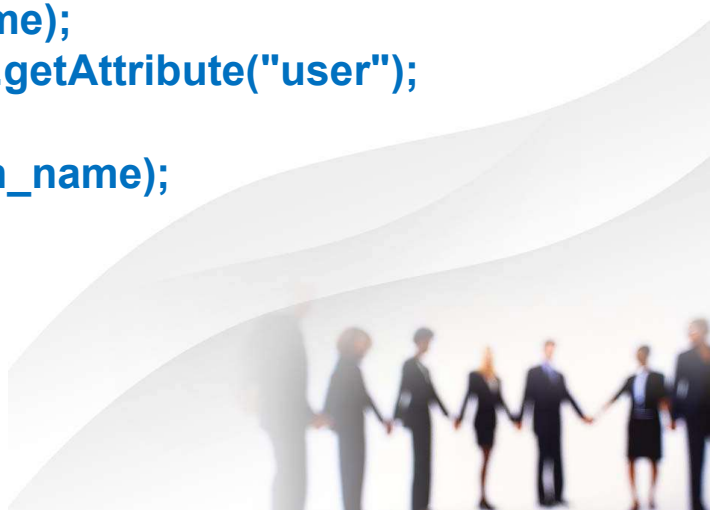
5.3 Các đối tượng ngầm định

- Có 9 đối tượng ngầm định được tạo sẵn và có thể được sử dụng trong mã nhúng JSP:
  - out (JspWriter): in ra một nội dung trên trang web
  - request (HttpServletRequest) : lấy các thông tin về request (tham số, header, contentype,…)
  - response (HttpServletResponse): sử dụng để thêm vào dữ liệu trả về hoặc chuyển hướng sang trang khác.
  - application (ServletContext): lấy các thông tin cấu hình khởi tạo (vd từ web.xml)
  - session (HttpSession): lấy, thêm hoặc xóa các thông tin về session
  - Các đối tượng khác: page, pageContext, exception

5.3 Các đối tượng ngầm định

- VD:

```
<html>
   <head>
            <title>JSP Implicit Objects</title>
   </head>
   <body>
      <h1><% out.print("Hello");%> </h1>
      <%
         String username=request.getParameter("user");
         if(username!=null)
            session.setAttribute("user",username);
         String session_name=(String)session.getAttribute("user");
         if(session_name!=null)
            out.print("Welcome back, "+session_name);
         else
            out.print("Welcome, Stranger");
      %>
   </body>
</html>
```



5.4 Các thẻ hoạt động

- Các thẻ hoạt động là các thẻ luồng điều khiển giữa các trang và tương tác với Java Bean:
  - Thẻ chuyển hướng **<jsp:forward** page="relativeURL | <%= expression %>" **/>**
    - Dùng để chuyển hướng kèm tham số đến trang khác (jsp, html,..)
  - Thẻ bao gồm **<jsp:include page** ="relativeURL | <%= expression %>" **/>**
    - Dùng để nhúng nội dung vào trang JSP
    - Khác với thẻ chỉ dẫn nhúng trang ngay khi biên dịch, thẻ hành động **include** nhúng nội dung khi trang được yêu cầu
  - Nhóm các thẻ làm việc với Java Bean
    - Java Bean là một lớp Java, thường chứa các phương thức get và set với các thuộc tính
    - jsp:useBean: chỉ ra hoặc khởi tạo lớp Java bean
    - Jsp:setProperty và jsp:setProperty : làm việc với các thuộc tính của bean

5.4 Các thẻ hoạt động

- VD

```
<html>
   <head>
         <title>JSP Page</title>
   </head>
   <body>
      <jsp:forward
page="hello.jsp" />
   </body>
</html>
```

Redirect.jsp

```
<html>
   <head>
         <title>JSP Actions</title>
   </head>
   <body>
      <jsp:include page="header.html" />
      <%
         String user=request.getParameter("user");
         out.print("Welcome , "+user);
      %>
   </body>
</html>
```

hello.jsp

Trang web hiển thị gì với đường dẫn /redirect.jsp?user=myname

5.4 Các thẻ hoạt động

```
package vn.nuce.ltudm.jspdemo;
public class User {
    private String username,password;
    public String getUsername() {
        return username;
    }
    public void setUsername(String
username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String
password) {
        this.password = password;
    }
}
```

User.java

```
 <form action="ProcessLogin.jsp"
method="post">
            Username<input type="text"
name="username"><br>
            Password:<input type="password"
name="password"><br>

            <input type="submit" value="Login">
  </form>
```

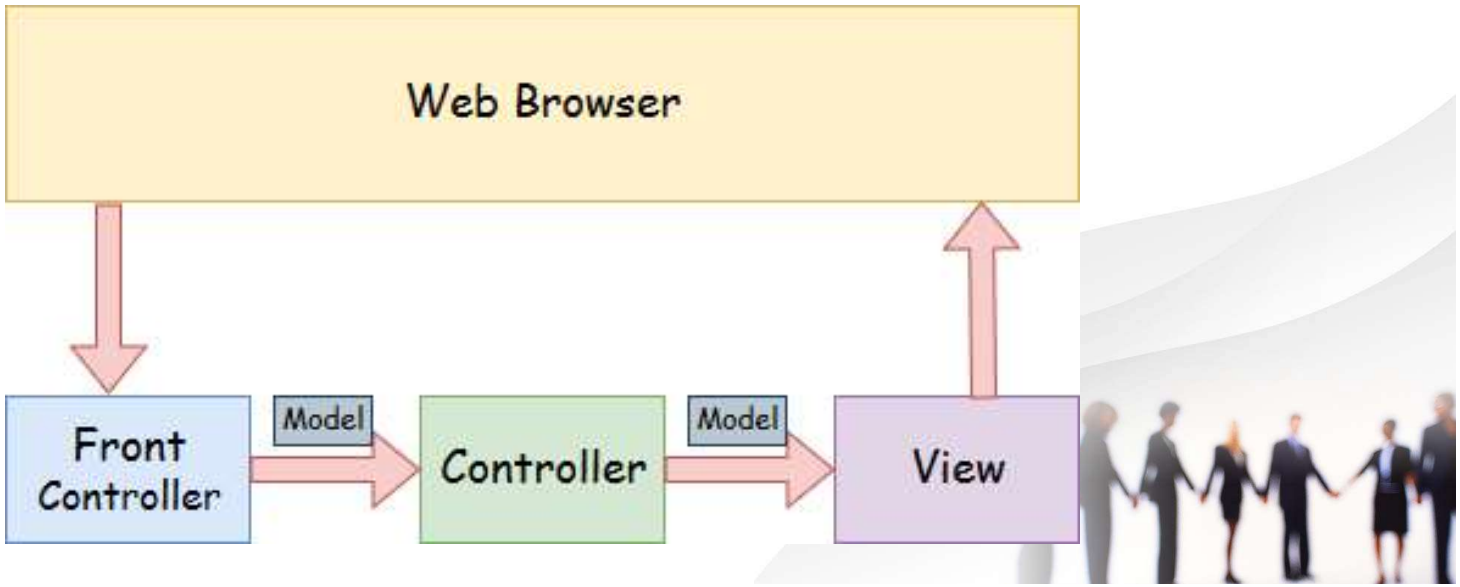
Form in login.html

```
<jsp:useBean id="u"
class="vn.nuce.ltudm.jspdemo1.User"><
/jsp:useBean>
        <jsp:setProperty property="*"
name="u"/>
        <h1>Login Info</h1>
        <jsp:getProperty
property="username" name="u"/><br>
        <jsp:getProperty
property="password" name="u"/><br>
```

ProcessLogin.jsp

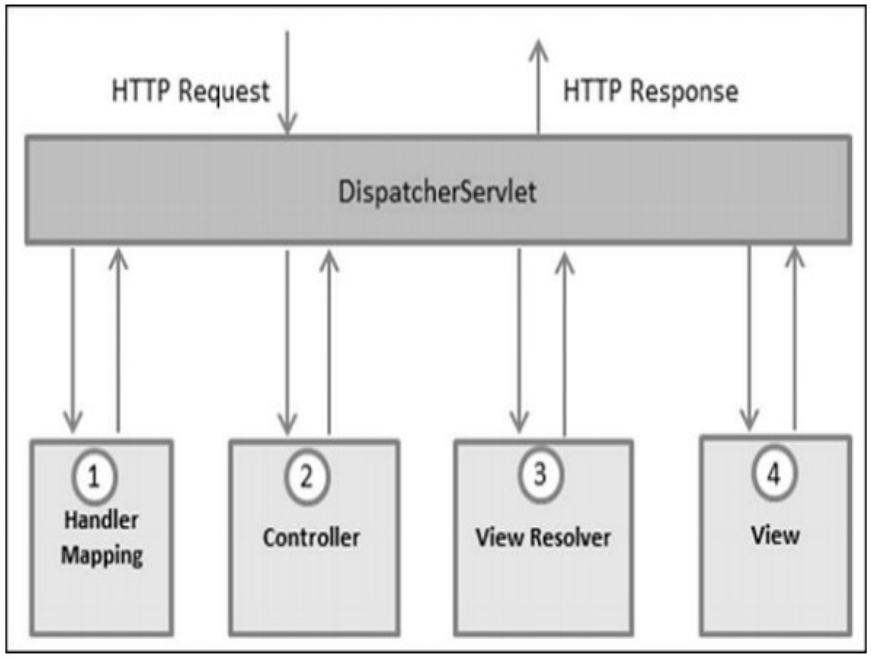
6. Spring Web MVC

- Spring Web MVC là một thành phần trong Spring Framewok
  - Là thành phần cốt lõi của Spring, xuất hiện ngay từ đầu trong Spring
  - Là framework để xây dựng ứng dụng web
  - Sử dụng mô hình MVC



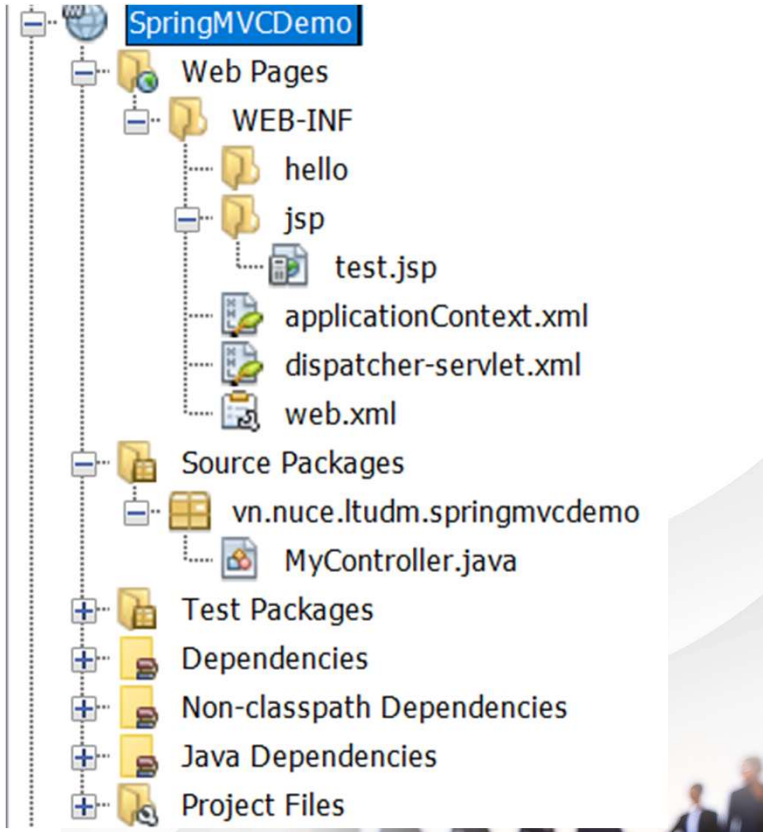
6. Spring Web MVC

- DispatcherServlet
  - Thành phần trung tâm của Spring Web MVC
  - Nhận truy vấn và điều phối các controller và view
  - Hoạt động:
    - 1 :Sử dụng Handler Mapping để gọi đến Controller tương ứng với request
    - 2: Controller xử lý xong request thì trả lại dữ liệu và tên view cho DispathcherServlet
    - 3: Sử dụng ViewResolver để gọi đến view theo tên
    - 4 :truyền dữ liệu cho view và nhận trang web đã render trả lại cho client



6. Spring Web MVC

- Cấu trúc project
  - Thư mục WEB-INF:
    - Tài nguyên web: html, images,css,js…
    - JSP files
    - Các file cấu hình:web.xml, servlet.xml
  - Thư mục source:
    - Controller
    - …
  - Dependencies



6. Spring Web MVC

- Các file cấu hình trong Spring Web MVC
  - web.xml: chỉ định DispatchServlet sẽ xử lý truy vấn dựa trên ánh xạ đường dẫn URL của truy vấn

VD:

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>*.jsp</url-pattern>
</servlet-mapping>
```

6. Spring Web MVC

- Các file cấu hình trong Spring Web MVC
  - [servlet name]-servlet.xml: cấu hình hoạt động của DispatcherServlet (Handler Mapping, Controller, View Resolver)

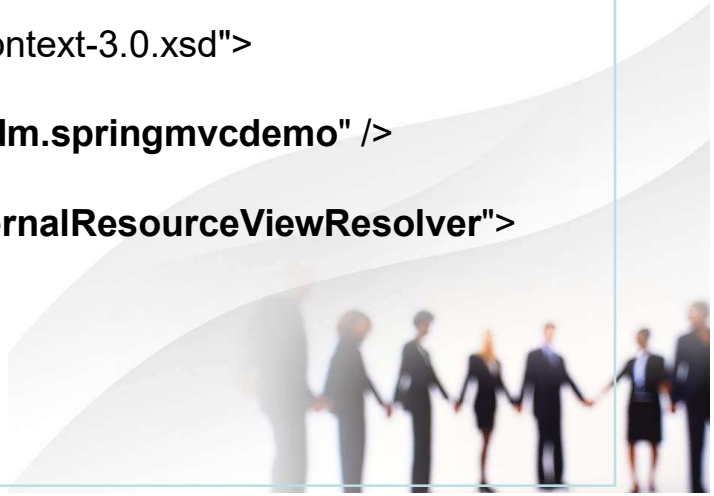
VD:

```
<beans xmlns = "http://www.springframework.org/schema/beans"
xmlns:context = "http://www.springframework.org/schema/context"
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation = "
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:component-scan base-package = "vn.nuce.ltudm.springmvcdemo" />

<bean class = "org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name = "prefix" value = "/WEB-INF/jsp/" />
  <property name = "suffix" value = ".jsp" />
</bean>

</beans>
```



6. Spring Web MVC

- Cấu hình Spring Web MVC trong Spring Boot:
  - Trong Spring Boot, việc cấu hình hoạt động của DispatcherServlet và Controller là tự động, tuy nhiên vẫn cần một số thiết lập để SpringBoot có thể hoạt động với Spring Web MVC và JSP:
    - Các gói cài đặt: web server mặc định trong Spring Boot không đủ để chạy được JSP, cần sử dụng webserver khác, ví dụ như tomcat-embed-jasper.

```
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <version>2.4.4</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.tomcat.embed</groupId>
        <artifactId>tomcat-embed-jasper</artifactId>
        <scope>provided</scope>
    </dependency>
```

6. Spring Web MVC

- Cấu hình Spring Web MVC trong Spring Boot:
  - Trong file Application , thừa kế lại lớp SpringBootServletInitializer

```
@SpringBootApplication
public class JSPDemoApplication extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
application) {
        return application.sources(JSPDemoApplication.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(JSPDemoApplication.class, args);
    }

}
```

6. Spring Web MVC

- Cấu hình Spring Web MVC trong Spring Boot:
  - Tạo thư mục WEB-INF để chứa tài nguyên web,lưu trong thư mục <thư mục project>/src/main/webapp
  - Trong file cấu hình application.properties, định nghĩa nơi chứa các view .jsp

```
spring.mvc.view.prefix: /WEB-INF/jsp/
spring.mvc.view.suffix: .jsp
```

6. Spring Web MVC

- Định nghĩa Controller: sử dụng các chỉ dẫn (annotation) trong Spring để đặc tả Controller
  - @Controller : cho biết lớp bên dưới là một Controller
  - @RequestMapping : đặc tả URL của lớp hoặc phương thức để khớp với URL của request
    - Đặt trước khai báo lớp: dùng để khớp với request dẫn đến Controller
    - Đặt trước khai báo hàm: dùng để khớp với request dẫn đến phương thức trong Controller

6. Spring Web MVC

- Định nghĩa Controller:

VD:

```
@Controller
@RequestMapping("/hello")
public class MyController {
@RequestMapping("/test", method = RequestMethod.GET)
    public String display()
    {
        return "test";
    }
}
```

Khớp với request có URL là /hello

Khớp với request có URL là /hello/test và phương thức là GET

Lưu ý: cần thiết kế URL sao cho đường dẫn đến mỗi hàm xử lý là duy nhất về URL và phương thức

6. Spring Web MVC

- Model:
  - Chứa dữ liệu truyền qua lại giữa Controller và View thông qua DispatcherServlet
  - Là các khối dữ liệu có thể chứa bất kỳ kiểu dữ liệu nào.
  - Các loại đối tượng Model:
    - Model: giao diện được định nghĩa để chứa dữ liệu dưới dạng các cặp thuộc tính – giá trị
    - ModelMap: mở rộng từ Model và Map, chứa các phương thức của Map
    - ModelAndView: đối tượng chứa cả view và ModelMap
  - Truy cập:
    - Controller: sử dụng các hàm addAttribute() để thiết lập giá trị cho các thuộc tính của model
    - View: sử dụng cú pháp ${attribute} để lấy ra các giá trị thuộc tính của model.

6. Spring Web MVC

- Model:
  - VD

controller

```
@RequestMapping("/hello")
public class MyController {
@RequestMapping("/test")
    public String display(Model m)
    {
        m.addAttribute("message", "hello");
        return "test";
    }
}
```

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
            <title>JSP Page</title>
    </head>
    <body>
        <h1>${message}</h1>
    </body>
</html>
```

test.jsp

6. Spring Web MVC

- Nhận tham số request trong Controller:
  - Sử dụng đối tượng **HttpServletRequest**

VD

```
@RequestMapping("/hello")
public class MyController {
@RequestMapping("/test")
    public String display(HttpServletRequest req,Model m)
    {
        String name=req.getParameter(("user"));
        m.addAttribute("message", "hello "+name);
        return "test";
    }
}
```

/hello/test?user=abc

6. Spring Web MVC

- Nhận tham số request trong Controller:
  - Sử dụng chỉ dẫn @RequestParam :
    - Gán tham số của request cho tham số của phương thức

VD:

```
<form action="login">
        UserName : <input type="text" name="name"/> <br><br>
        Password : <input type="text" name="pass"/> <br><br>
        <input type="submit" name="submit">
    </form>
```

```
@RequestMapping("/login")
  public String input(@RequestParam("name") String  username,
                      @RequestParam("pass") String  password ,Model m)
  {
     m.addAttribute("username", username );
     m.addAttribute("password", password );
     return "confirm";
  }
```

6. Spring Web MVC

- Nhận tham số request trong Controller:
  - Sử dụng chỉ dẫn thư viện Form Tag:
    - Spring MVC Form Tag là các khối được xây dựng sẵn có thể sử dụng trực tiếp trong JSP
    - Form Tag là các thẻ form được sử dụng để liên kết dữ liệu với Bean trong java code: input, password, textarea,…
    - Để sử dụng Form Tag cần khai báo thư viện trong JSP:

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

Cú pháp:

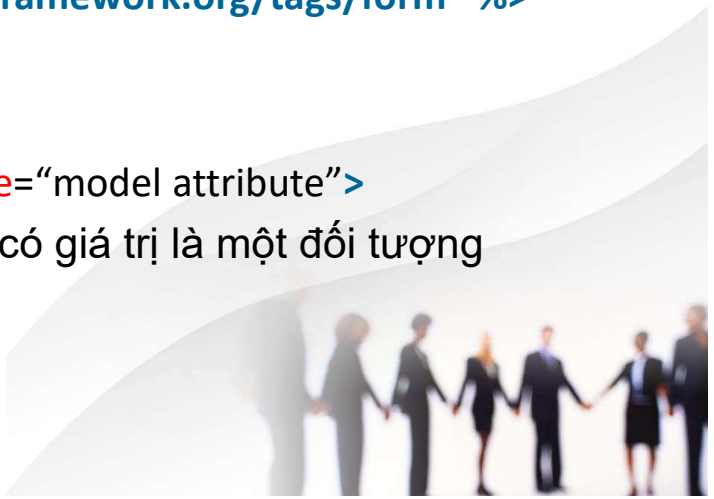
Khai báo form

```
<form:form action="controller/path" modelAttribute="model attribute">
```

Trong đó thuộc tính modelAttribute là thuộc tính có giá trị là một đối tượng Java Bean

Sử dụng các thẻ trong form

```
<form:tag path="tên thuộc tính của bean" />
```



6. Spring Web MVC

- Sử dụng chỉ dẫn thư viện Form Tag:
  - Nhận dữ liệu form dạng đối tượng java bean: sử dụng chỉ dẫn @ModelAttribute

  VD: gắn kết dữ liệu giữa bean và form

```
public class RegisterInfo {
    String username,password,email;
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

```
@RequestMapping("/register")
    public String register(Model m)
    {
        RegisterInfo reg=new RegisterInfo();
        m.addAttribute("reginfo", reg );
        return "register";
    }
```

```
<form:form method = "POST"
action="proccessRegister"
modelAttribute="reginfo">
        UserName : <form:input  path="username"/>
<br><br>
        Password : <form:password path="password"/>
<br><br>
        Email : <form:input type="email" path="email"/>
<br><br>
        <input type="submit" name="submit">
      </form:form>
```

6. Spring Web MVC

- Sử dụng chỉ dẫn thư viện Form Tag:

  VD: nhận dữ liệu submit

```
<form:form method = "POST"
action="proccessRegister"
modelAttribute="reginfo">
        UserName : <form:input
path="username"/>
<br><br>
        Password : <form:password
path="password"/>
<br><br>
        Email : <form:input type="email"
path="email"/>
<br><br>
        <input type="submit" name="submit">
    </form:form>
```

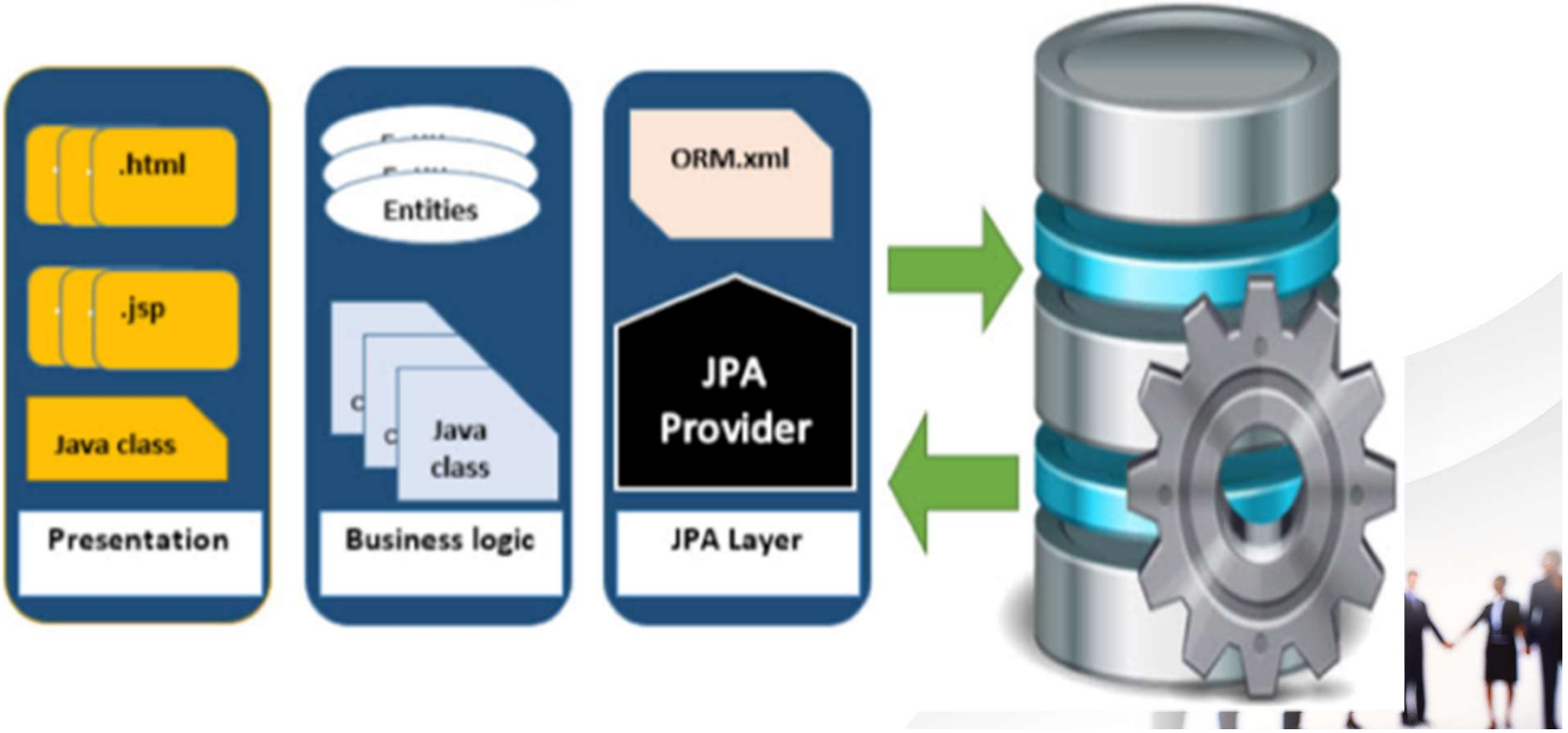
```
@RequestMapping(value = "/proccessRegister",
method = RequestMethod.POST)
   public String processRegister(
@ModelAttribute("reginfo") RegisterInfo reg)
   {
      //do something with reg
      return "confirm";
   }
```

```
<body>
      <h1>${reginfo.username}</h1>
      <h1>${reginfo.password}</h1>
       <h1>${reginfo.email}</h1>
  </body>
```

confirm.jsp

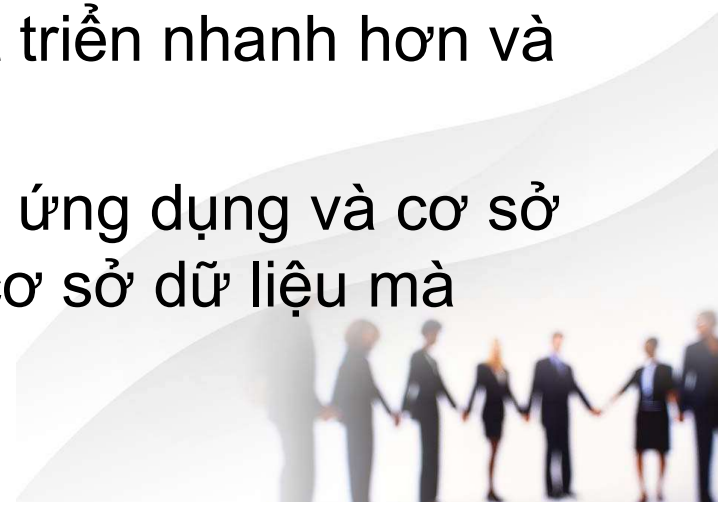
7. JPA

- JPA (Java Persistence API) là các đặc tả công nghệ kết nối cơ sở dữ liệu quan hệ với các đối tượng trong ứng dụng



7.JPA

- Vai trò của JPA trong phát triển ứng dụng:
  - Đảm nhận trung gian chuyển đổi giữa các bảng trong CSDL với các đối tượng sử dụng trong chương trình
    - Chức năng này gọi là ORM (Object Relation Mapping): ánh xạ các bảng trong CSDL với các lớp mô tả đối tượng trong chương trình.
  - Giảm đáng kể khối lượng code cần thiết để tiến hành kết nối và quản lý dữ liệu -> phát triển nhanh hơn và ít lỗi hơn
  - Tách biệt việc xử lý dữ liệu trong ứng dụng và cơ sở dữ liệu -> dễ dàng thay đổi loại cơ sở dữ liệu mà không cần sửa lại code

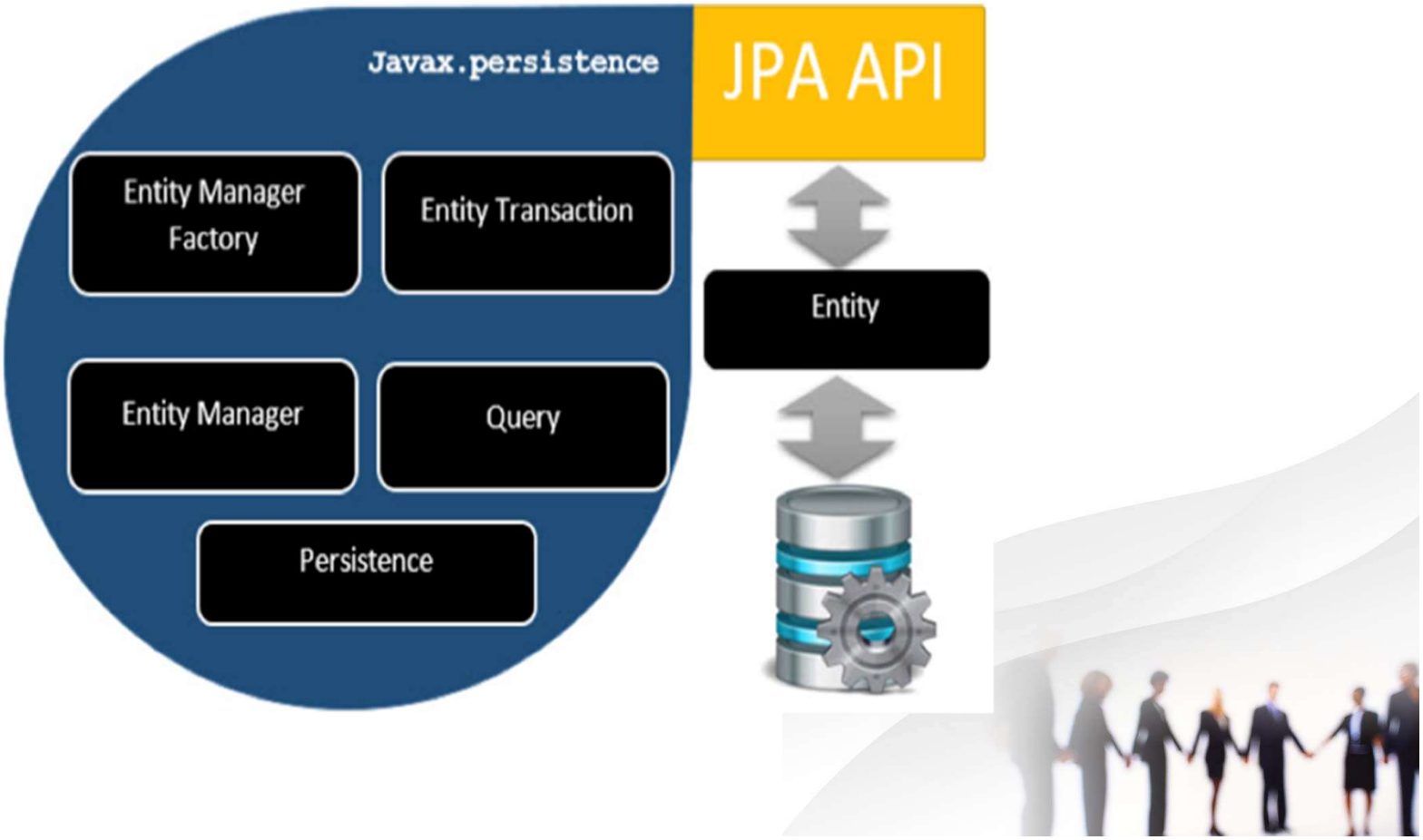


7.JPA

- JPA Provider:
  - JPA chỉ là các đặc tả (chuẩn) mà J2EE cung cấp
  - JPA Provider là các framework triển khai JPA để sử dụng trong thực tế:
    - Hirbernate
    - Eclipselink
    - Toplink
    - Spring Data JPA
    - Kodo JDO
    - ...

7.JPA

- Các thành phần của JPA:



7.JPA

- Các thành phần của JPA:
  - Entity: đại diện cho 1 đối tượng dữ liệu
  - EntityManager: đối tượng thực hiện các thao tác quản lý dữ liệu (CRUD-thêm, sửa, xóa, truy vấn) với các Entity
  - EntityManagerFactory : đối tượng sinh ra EntityManager
  - EntityTransaction: đi kèm với EntityManager để quản lý các thao tác dữ liệu
  - Persistence: cung cấp EntityManagerFactory
  - Query: giao diện cung cấp khả năng tạo truy vấn dữ liệu theo điều kiện

8. Spring Data JPA

- Là một phần của Spring Data, giúp đơn giản hóa thực thi JPA trong ứng dụng
- Hỗ trợ các đặc điểm của JPA:
  - Các chỉ dẫn trong Entity
  - Ngôn ngữ truy vấn JPQL
  - …

8.Spring Data JPA

- Cấu hình Spring Data JPA trong Spring Boot:
  - Sử dụng các dependency sau khi khởi tạo project: **spring-boot-starter-data-jpa**
  - Chọn connector tương ứng với CSDL , ví dụ: **mysql-connector-java** cho CSDL MySQL

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
</dependency>
```

8.Spring Data JPA

- Cấu hình Spring Data JPA trong Spring Boot:
  - Cấu hình application.properties về cách truy cập CSDL. VD:

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/mydb
spring.datasource.username=myuser
spring.datasource.password=mypass
```

8. Spring Data JPA

- Entity:
  - Là đối tượng mang dữ liệu trong hệ thống
    - Chính là các đối tượng có được khi phân tích hệ thống theo phương pháp hướng đối tượng (biểu đồ lớp và biểu đồ domain model)
    - Thường đại diện cho 1 bảng trong CSDL, trong đó mỗi đối tượng entity cụ thể có thể coi như 1 dòng trong bảng
  - Trong JPA, Entity là một lớp có các thuộc tính được đánh dấu tương ứng với tên các cột trong bảng bằng các anotation

8. Spring Data JPA

- Entity:
  - Định nghĩa một lớp là một Entity:
    - Lớp đó có annotation là @Entity
    - Lớp đó phải có một hàm khởi tạo không tham số
    - Lớp đó có các biến dữ liệu private và các hàm thuộc tính (các hàm setter-getter)
      - Các hàm thuộc tính setter-getter được viết dạng :
        - » getProperty()
        - » setProperty(Type value)

8. Spring Data JPA

- Các anotation được sử dụng trong Entity:
  - @Entity: cho biết lớp đó được định nghĩa là một Entity
  - @Table: cho biết entity đang định nghĩa tương ứng với bảng nào trong CSDL
  - @Column: cho biết thuộc tính bên dưới tương ứng với cột nào trong bảng
  - @Id: cho biết thuộc tính bên dưới tương ứng với cột là khóa chính trong bảng
  - @GeneratedValue: cho biết cách sinh giá trị tự động của thuộc tính bên dưới (thường áp dụng với khóa chính của bảng)

8. Spring Data JPA

- Các anotation được sử dụng trong Entity:
  - @OneToOne, @OneToMany, @ManyToOne, @ManyToMany: cho biết thuộc tính bên dưới là các entity khác có quan hệ với entity hiện tại
  - @NamedQuery: định nghĩa các câu query để truy vấn hoặc cập nhật dữ liệu cho entity

8. Spring Data JPA

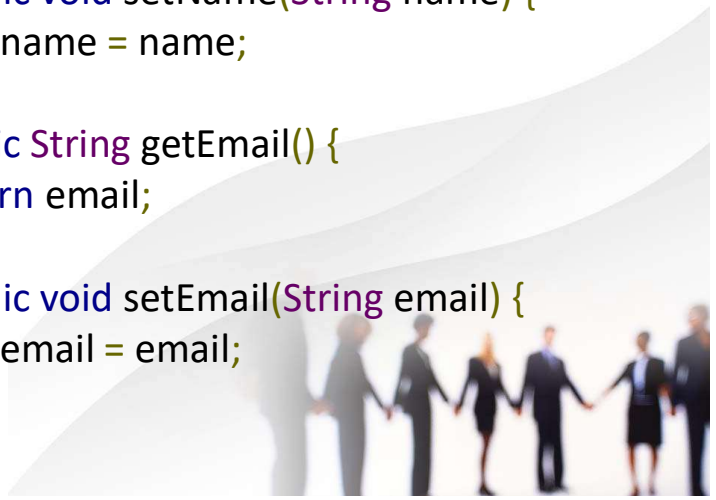
- Cách tạo các Entity: 2 phương pháp
  - Tạo CSDL trước:
    - Tạo CSDL
    - Tạo các Entity từ các bảng trong CSDL
    - Tạo các lớp dịch vụ sử dụng JPA để truy cập dữ liệu
  - Tạo các entity trước:
    - Tạo các entity và các liên hệ giữa các entity
    - Tạo CSDL từ entity
    - Tạo các lớp dịch vụ sử dụng JPA để truy cập dữ liệu

8. Spring Data JPA

- VD: Entity User

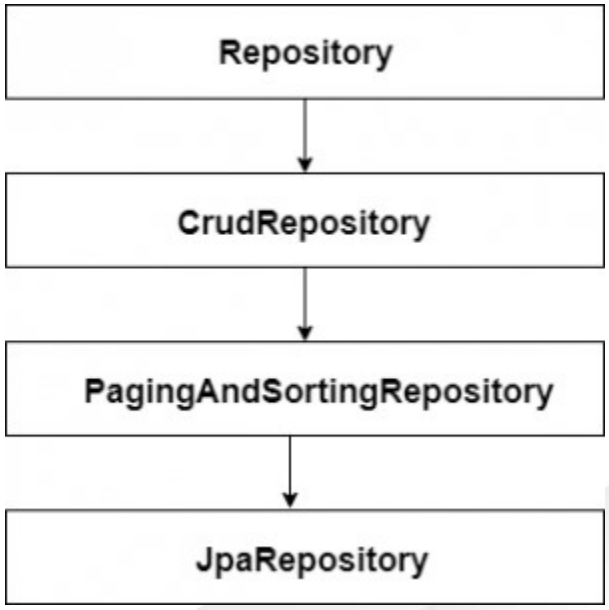
```
import javax.persistence.Entity;
import
javax.persistence.GeneratedValue;
 import
javax.persistence.GenerationType;
 import javax.persistence.Id;
@Entity // This tells Hibernate to make a
table out of this class
public class User {
 @Id
@GeneratedValue(strategy=GenerationT
ype.AUTO)
 private Integer id;
private String name;
 private String email;
```

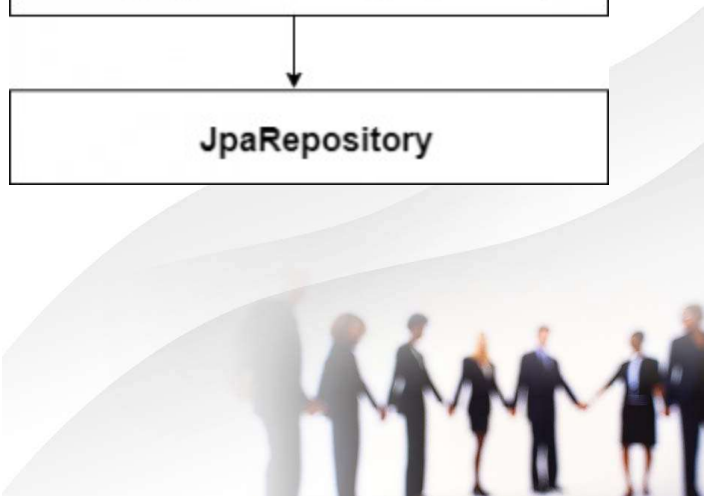
```
public Integer getId() {
return id;
 }
public void setId(Integer id) {
this.id = id;
}
 public String getName() {
 return name;
}
 public void setName(String name) {
 this.name = name;
 }
public String getEmail() {
 return email;
}
 public void setEmail(String email) {
 this.email = email;
 }
}
```



8. Spring Data JPA

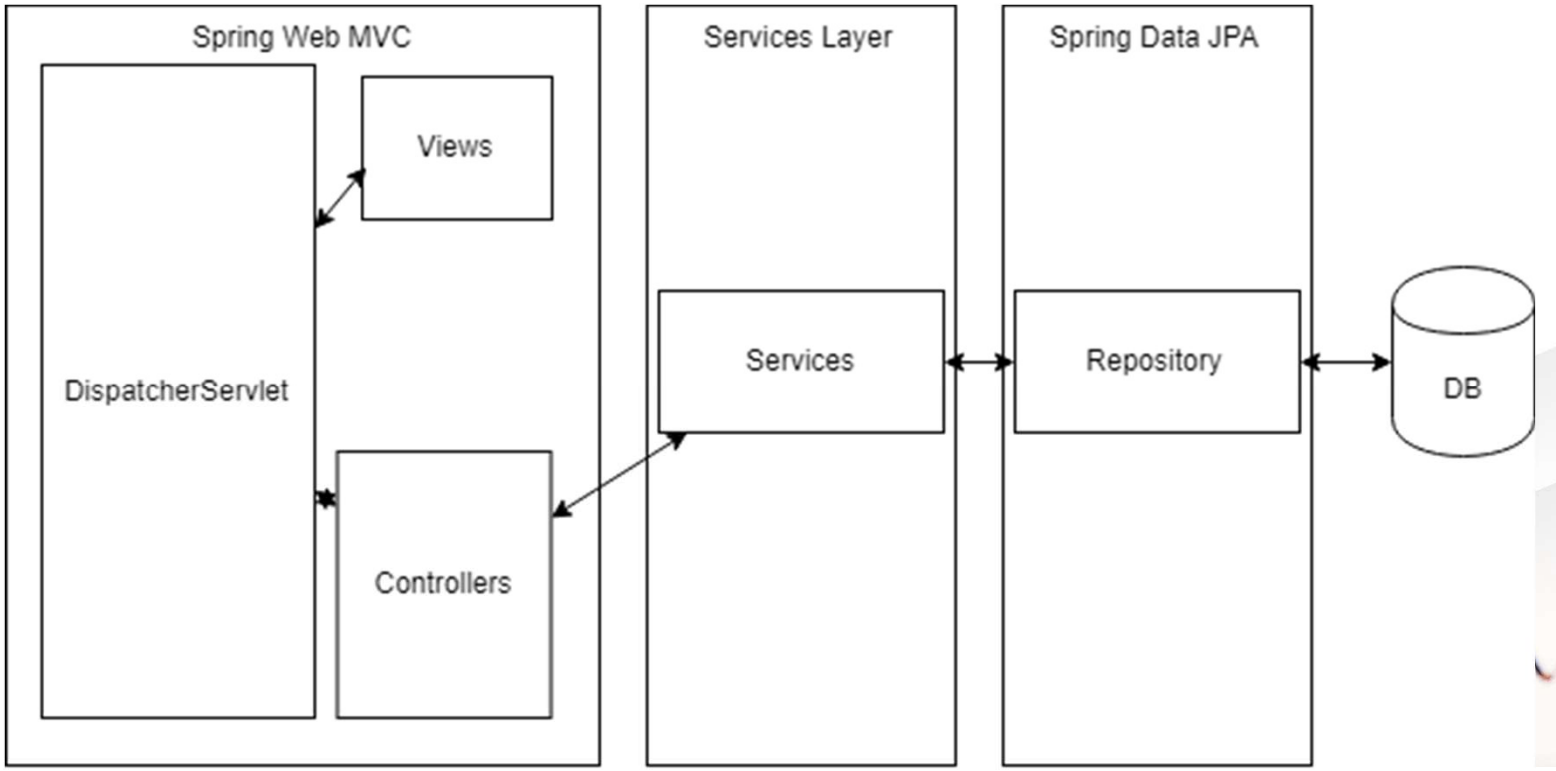
- Respository trong Sping Data JPA
  - Các giao diện hỗ trợ tạo lập lớp truy vấn dữ liệu
    - **CrudRepository** : cung cấp các phương thức truy vấn CRUD cơ bản
    - **PagingAndSortingRepository** : cung cấp thêm các phương thức truy vấn phân trang và sắp xếp
    - **JpaRepository** : cung cấp thêm các phương thức liên quan đến JPA: flush, batch…
  - Lớp thực thi giao diện được tự động tạo bởi Spring





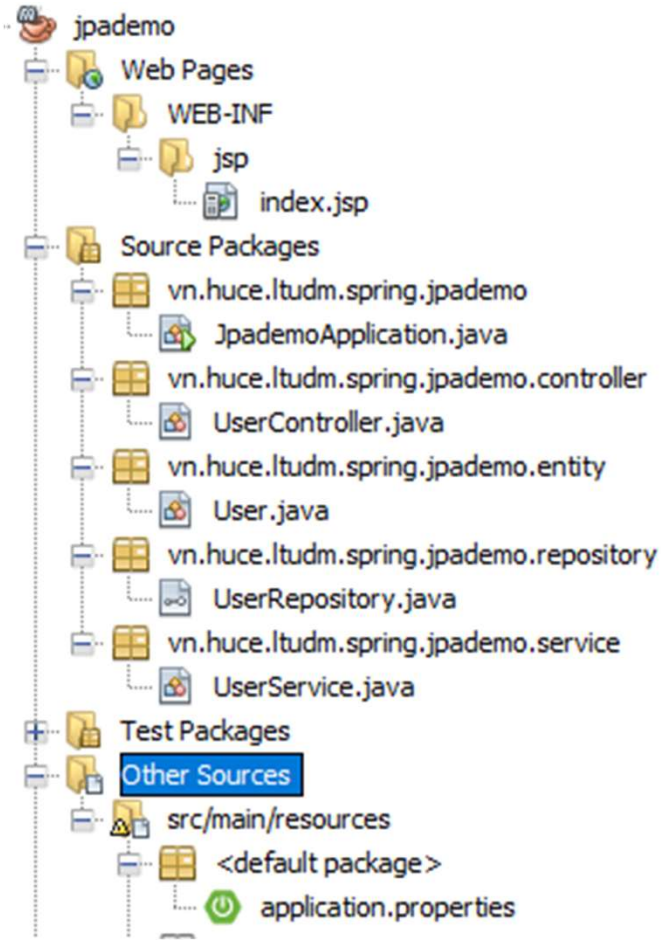
8.Spring Data JPA

- Hình mẫu thiết kế với Sping Data JPA và Spring Web MVC
  - Các lớp repository thường không được sử dụng trực tiếp trong các controller mà sẽ thông qua các lớp Service



8.Spring Data JPA

- Ví dụ về mô hình Spring Web MCV + Spring Data JPA



8.Spring Data JPA

- Ví dụ về mô hình Spring Web MCV + Spring Data JPA

```
import org.springframework.data.repository.CrudRepository; import
com.example.accessingdatamysql.User;
// This will be AUTO IMPLEMENTED by Spring into a Bean called
userRepository
// CRUD refers Create, Read, Update, Delete
public interface UserRepository extends CrudRepository<User, Integer> { }
```

UserRepository.java

```
@Service
@Transactional
public class UserService {
   @Autowired
   UserRepository userrepo;
   public List<User> listAll() {
      return userrepo.findAll();
   }
}
```

UserService.java

8.Spring Data JPA

- Ví dụ về mô hình Spring Web MCV + Spring Data JPA

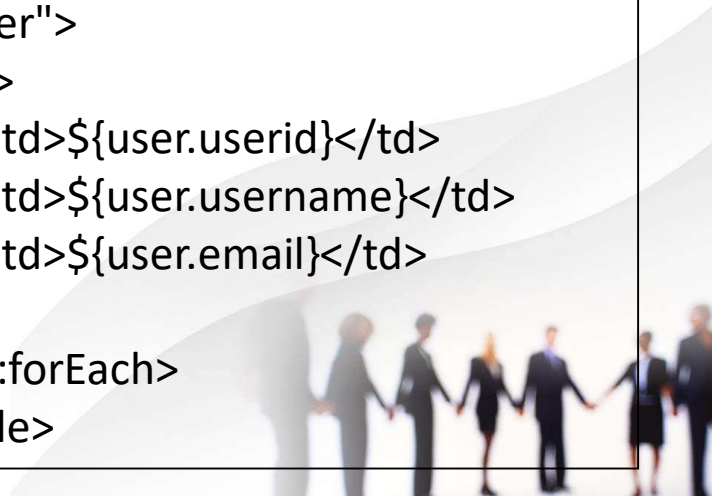
```
public class UserController {
   @Autowired
   private UserService userService;

   @RequestMapping("/index")
   public String home(Model m) {
      List<User> listUser = userService.listAll();
      m.addAttribute("listuser", listUser);
      return "index";
   }
}
```

Controller

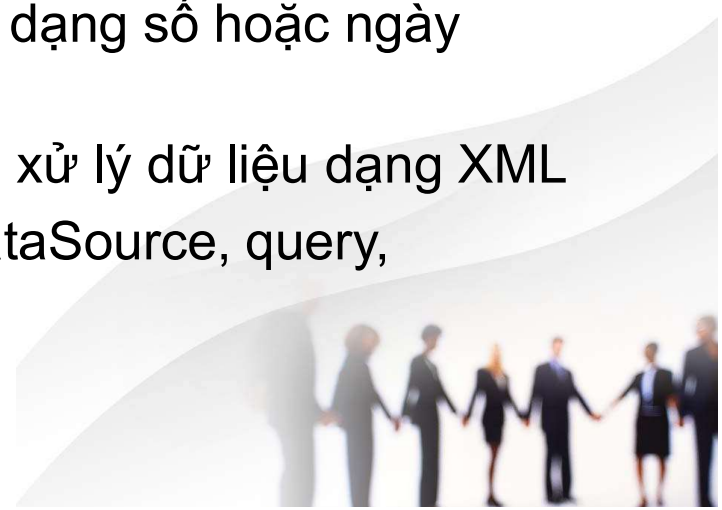
```
<%@ taglib prefix = "c" uri =
"http://java.sun.com/jsp/jstl/core" %>
<table border="1" cellpadding="5">
      <tr>
         <th>ID</th>
         <th>Name</th>
         <th>E-mail</th>
</tr>
      <c:forEach items="${listuser}"
var="user">
      <tr>
         <td>${user.userid}</td>
         <td>${user.username}</td>
         <td>${user.email}</td>
</tr>
      </c:forEach>
   </table>
```

Index.jsp



8.Spring Data JPA

- JSTL (JSP Standard Tag Library):
  - Thư viện tag nhằm đơn giản hóa phát triển JSP
  - Các kiểu tag:
    - Core tag: các tag hỗ trợ xuất dữ liệu và luồng điều khiển (out, if, forEach,…)
    - Function tag: các tag là các hàm hỗ trợ xử lý dữ liệu trình bày (contain(), trim(), replace(),…)
    - Formatting tag: các tag hỗ trợ định dạng số hoặc ngày tháng
    - XML tag: các tag hỗ trợ việc tạo và xử lý dữ liệu dạng XML
    - SQL tag: các tag hỗ tợ SQL (setDataSource, query, update,…)



8.Spring Data JPA

- Truy vấn Entity:
  - Sử dụng để tạo các truy vấn tùy ý trong JPA
  - Có 2 cách để tạo các truy vấn entity :
    - Sử dụng JPQL (Java Persistence query language): lệnh truy vấn được viết dưới dạng một chuỗi ký tự (giống các lệnh SQL):
      - Đơn giản, dễ đọc, quen thuộc với cú pháp tương tự SQL
    - Sử dụng Criteria API: các lệnh truy vấn được tạo bởi các lệnh java
      - Tốc độ nhanh hơn JPQL, khó đọc và thiết lập

8.Spring Data JPA

- Truy vấn Entity:
  - VD

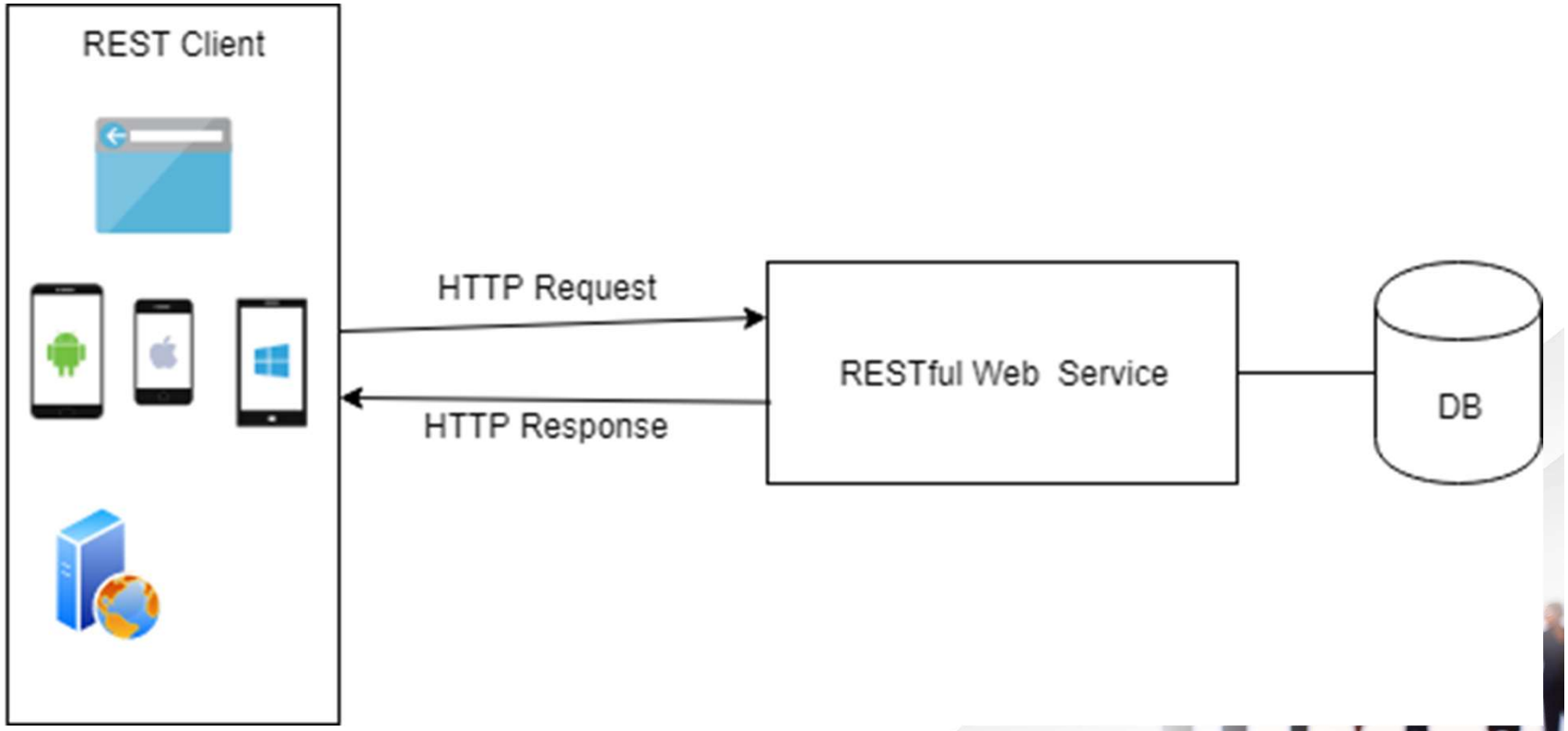
```
@Query(value = "SELECT u FROM User u ORDER BY id")
Page<User> findAllUsersWithPagination(Pageable pageable);
```

9. RESTful webservice với Spring Web

- Web Service : là hệ thống các quy tắc (chuẩn) định nghĩa cách client-server kết nối với nhau thông qua giao thức HTTP.
- 2 loại hình Web Service tương ứng với 2 chuẩn được sử dụng:
  - SOAP (Simple Object Access Protocol) + XML: dịch vụ web chuẩn mực cho các ứng dụng doanh nghiệp, có độ tin cậy và an toàn cao.
    - Nhược điểm: phức tạp và đòi hỏi sự chặt chẽ và tính tương thích cao giữa client và server
  - REST (Representational State Transfer) : dịch vụ web hoạt động trên chính các phương thức của HTTP do vậy đơn giản và dễ sử dụng, có tính tương thích tốt trên nhiều nền tảng
  - Bản chất: SOAP là dịch vụ web theo kiểu gọi hàm từ xa, còn REST hoạt động theo kiểu truy vấn tài nguyên

9. RESTful webservice với Spring Web

- Mô hình RESTFul Web Service



9. RESTful webservice với Spring Web

- Thiết kế RESTful API:
  - Đặc tả một RESTful API:
    - Một RESTful API là một *truy vấn tài nguyên*, mội lời gọi API là một hoạt động trao đổi tài nguyên giữa client và server
    - Mỗi API có một URI duy nhất, đóng vai trò định danh cho API
    - Dữ liệu trao đổi thường là dữ liệu dạng JSON
    - Các thông tin cần đặc tả:
      - Request: URI, Method, dữ liệu
      - Response: mã lỗi, kiểu dữ liệu trả về, cấu trúc dữ liệu trả về

9. RESTful webservice với Spring Web

- Thiết kế RESTful API:
  - Đặc tả một RESTful API:
    - VD:

```
POST https://adventure-works.com/orders HTTP/1.1 Content-Type: application/json; charset=utf-8 Content-Length: 57
{"Id":1,"Name":"Gizmo","Category":"Widgets","Price":1.99}
```

request

```
HTTP/1.1 200 OK
Content-Type: application/json
 { "status":"In progress",
 "link": { "rel":"cancel",
 "method":"delete",
 "href":"/api/status/12345" } }
```

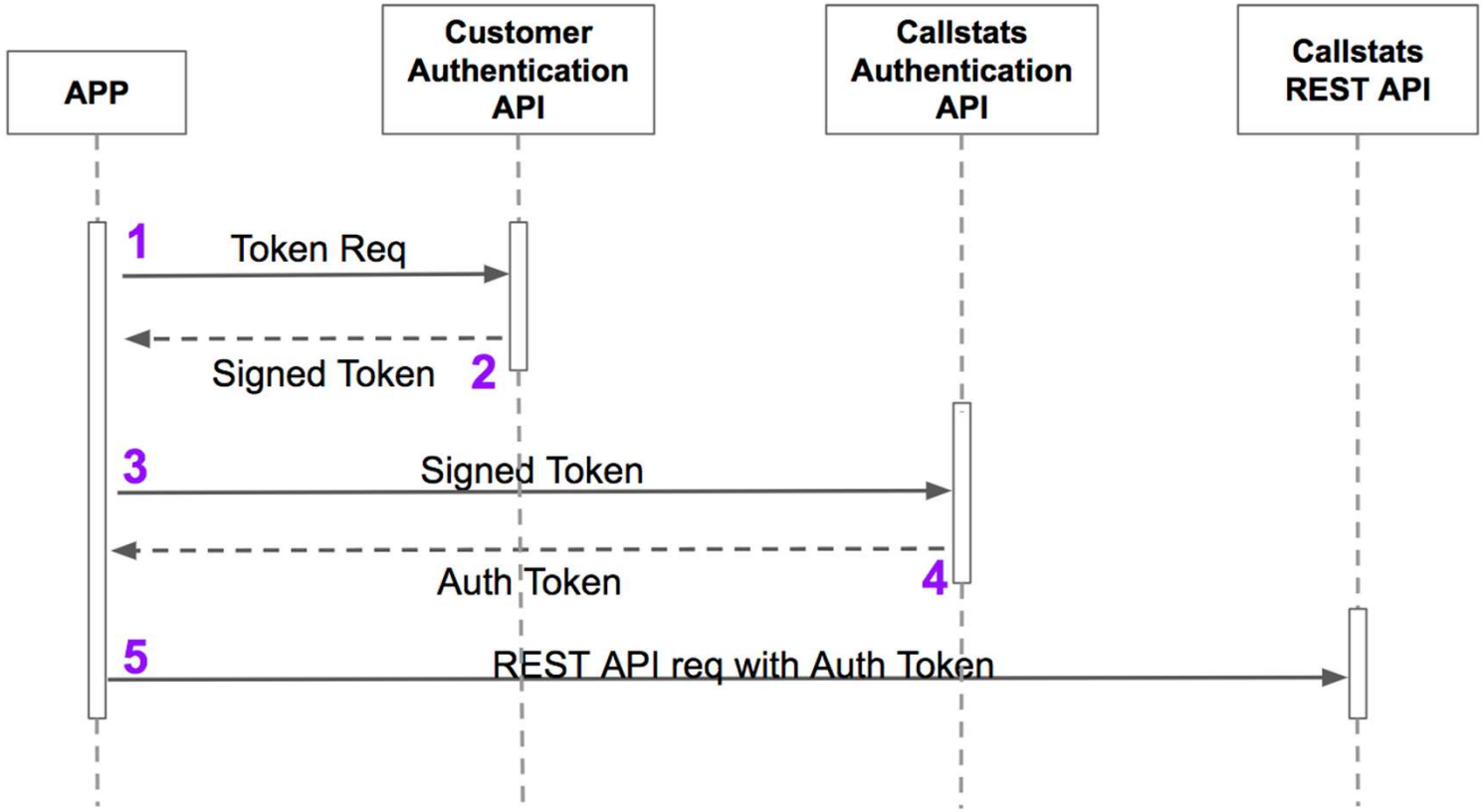
response

9. RESTful webservice với Spring Web

- Thiết kế RESTful API:
  - Chú ý:
    - Thiết kế RESTful API cần xoay quanh *tài nguyên* , là dữ liệu và dịch vụ được truy cập bởi client
    - Mỗi API cần có đặc tả (phương thức + URI) là duy nhất và không trùng với các API khác
    - REST API là truy vấn không trạng thái
      - Mỗi API nên hoạt động độc lập với các API khác, tránh lưu giữ các thông tin ràng buộc giữa các lời gọi API trên server
      - Quá trình truy vấn và nhận kết quả từ REST API đòi hỏi thời gian xử lý nhất định.

9. RESTful webservice với Spring Web

- Thiết kế RESTful API:
  - VD



https://help.callstats.io/hc/en-us/articles/360002330774-REST-API-Authentication

9. RESTful webservice với Spring Web

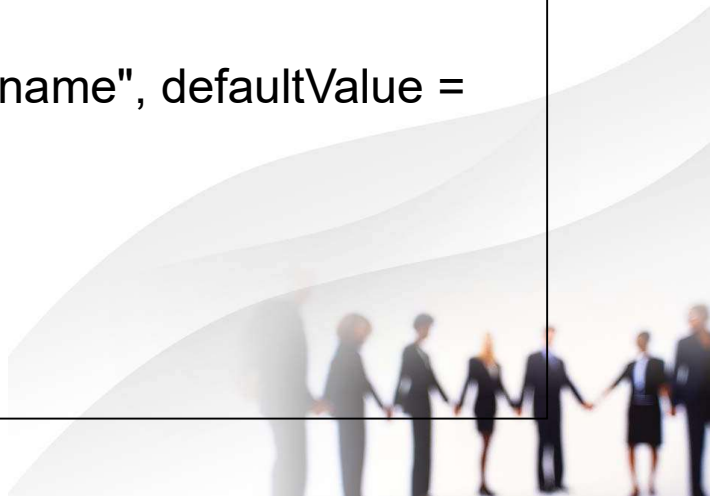
- Tạo lập RESTful API với Spring Web
  - **Resource Controller**: Lớp chứa các hàm phản hồi các request của client
  - Các chỉ dẫn (Annotation) sử dụng trong REST controller:
    - @RestController: chỉ dẫn cho biết lớp bên dưới là một Controller có các phương thức trả về đối tượng dữ liệu
    - @GetMapping (/path) : gắn hàm bên dưới với request GET theo đường dẫn **/path**
    - @PostMapping (/path) : gắn hàm bên dưới với request POST theo đường dẫn **/path**
    - @RequestParam : gắn tham số trong chuỗi truy vấn với tham số hình thức của hàm

9. RESTful webservice với Spring Web

Request (GET method) http://localhost:8080/greeting?name=User

Ví dụ Controller xử lý request trên

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class StudentController {
   @GetMapping("/greeting")
   public String greeting(@RequestParam(value = "name", defaultValue =
“noname") String name) {
                        return “Hi, “+name;
   }
}
```



9. RESTful webservice với Spring Web

- Nhận dữ liệu trong request:
  - Dữ liệu dạng tham số ở URL (POST và GET):
    - Sử dụng @RequestParam cho tham số *query string params*
    - Sử dụng @ PathVariable cho tham số ở *path*
  - Dữ liệu ở phần body (POST):
    - Tham số dạng urlencoded (form):
      - Sử dụng @RequestParam
    - Dữ liệu dạng đối tượng JSON
      - Sử dụng @RequestBody

9. RESTful webservice với Spring Web

- Nhận dữ liệu với request GET:

VD

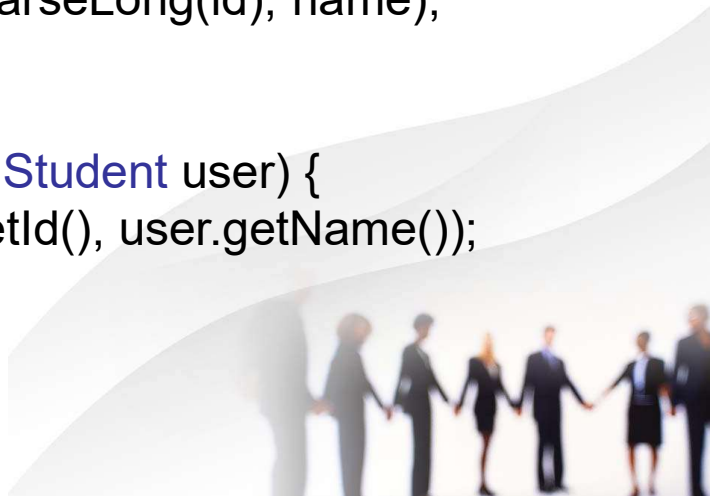
```
@GetMapping("/product")
  public String getProduct(@RequestParam(value = "id", defaultValue =
"0") String id) {
                return "get product by id="+id;
  }
  @GetMapping("/product/{category}/{sort}")
  public String getCategory(@PathVariable String category,
@PathVariable String sort) {
                return "get category="+category+" and sort by "+sort;
  }
```

9. RESTful webservice với Spring Web

- Nhận dữ liệu với request POST:

VD

```
@PostMapping("/add")
public Student addStudent(@RequestParam(value = "id", defaultValue =
"0") String id,
@RequestParam(value = "name", defaultValue = "noname") String name)
{
                return new Student(Long.parseLong(id), name);
}
@PostMapping("/edit")
public Student editStudent(@RequestBody Student user) {
                return new Student(user.getId(), user.getName());
}
```



9. RESTful webservice với Spring Web

- Trả về dữ liệu:
  - Dữ liệu trả về dạng text:

http://localhost:8080/say?name=User

```
@GetMapping("/say")

  public String sayHi(@RequestParam(value = "name", defaultValue = "World")
String name) {
                    return "hi "+name;
  }
```

Hi User

9. RESTful webservice với Spring Web

- Trả về dữ liệu đối tượng:
  - Khi trả về một đối tượng dữ liệu (domain model), đối tượng đó được tự động chuyển đổi sang JSON nhờ thư viện Jackson có sẵn trong Spring Web starter.

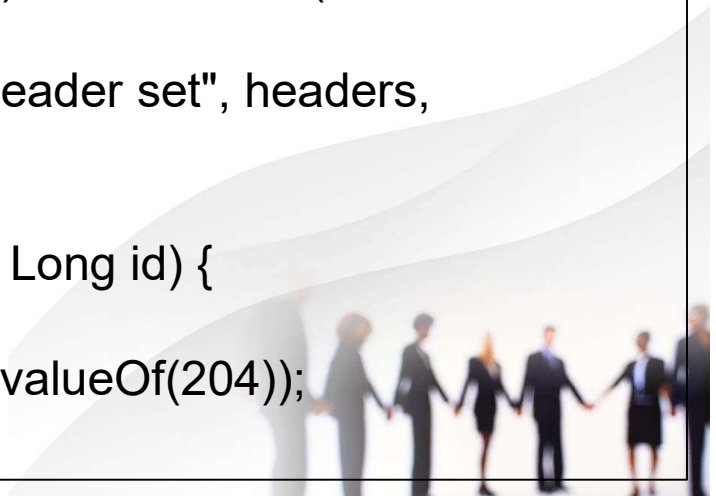
```
@GetMapping("/greeting")
  public Student greeting(@RequestParam(value = "name", defaultValue =
“noname") String name) {
                    return new Student(1, name);
  }
```

Domain model (đối tượng chứa dữ liệu)

9. RESTful webservice với Spring Web

- Trả về dữ liệu tùy chỉnh:
  - Khi muốn tùy chỉnh sâu hơn về dữ liệu trả về (kiểu dữ liệu, mã lỗi) ta sử dụng đối tượng ResponseEntity để cấu trúc phản hồi.

```
@GetMapping("/customHeader")
ResponseEntity<String> customHeader() {
        HttpHeaders headers = new HttpHeaders(); headers.add("Custom-
Header", "foo");
        return new ResponseEntity<>( "Custom header set", headers,
HttpStatus.OK); }
@DeleteMapping("/{id}")
   public ResponseEntity<?> delete(@PathVariable Long id) {
      …
      return new ResponseEntity<>(null, HttpStatus.valueOf(204));
   }
```



9. RESTful webservice với Spring Web

- Kết hợp RESTful API trong Spring Web với Spring Data JPA: