

JAVA WEB

Unit 5: Spring MVC



Mục lục

1. Cấu trúc source code của Spring Boot, mô hình three layers, mô hình MVC, luồng xử lý trong Spring Boot.
2. 04 loại components: `@Component`; `@Service`; `@Repository`; `@Controller`
3. Các annotations trong Spring MVC : `@Controller`, `@RequestMapping`, `@PathVariable`, `@RequestParam`, `@ModelAttribute`, `@SessionAttributes`, `@ResponseBody` , `@RequestBody`
4. Các giao diện: `ModelAndView`, `ModelMap`, `Model`
5. Xử lý ngoại lệ trong Spring MVC: xử lý ngoại lệ trong một controller; Xử lý ngoại lệ trong toàn bộ ứng dụng.

Cấu trúc source code của Spring Boot

Cấu trúc source code của Spring Boot được dựa trên hai mô hình là **mô hình MVC** và **mô hình 3 lớp**.

Cấu trúc source code của Spring Boot được dựa trên hai mô hình là **mô hình MVC** và **mô hình 3 lớp**:

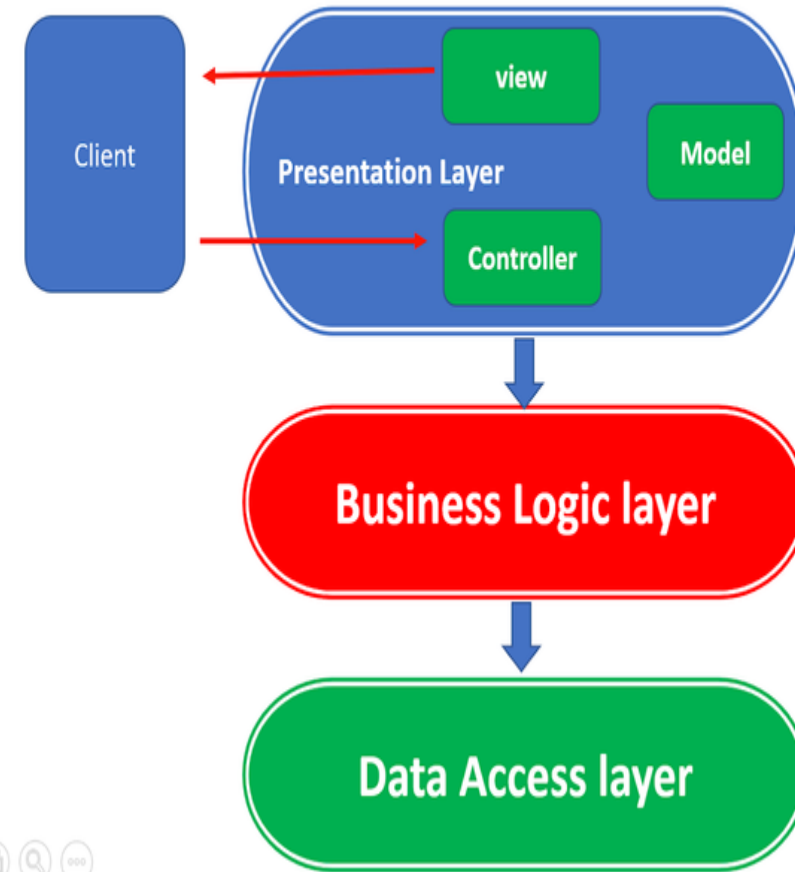
- **Presentation layer:** tầng này tương tác với người dùng, bằng View, Controller (trong MVC) hoặc API (nếu có).
- **Business logic layer:** Chứa toàn bộ logic của chương trình, các đa số code nằm ở đây
- **Data access layer:** Tương tác với database, trả về kết quả cho tầng business logic

Cấu trúc source code của Spring Boot

Trong Spring Boot:

- Service: chứa các business logic code
- Repository: đại diện cho tầng data access
- **Presentation layer** được chia thành 03 phần: Model, View, Controller.

Three-Tier architecture vs MVC pattern



Mô hình ba lớp (three tier)

Presentation layer: tầng này tương tác với người dùng, bằng View, Controller (trong MVC) hoặc API (nếu có).

Business logic layer: Chứa toàn bộ logic của chương trình, các đa số code nằm ở đây

Data access layer: Tương tác với database, trả về kết quả cho tầng business logic



Mô hình ba lớp (three tier)

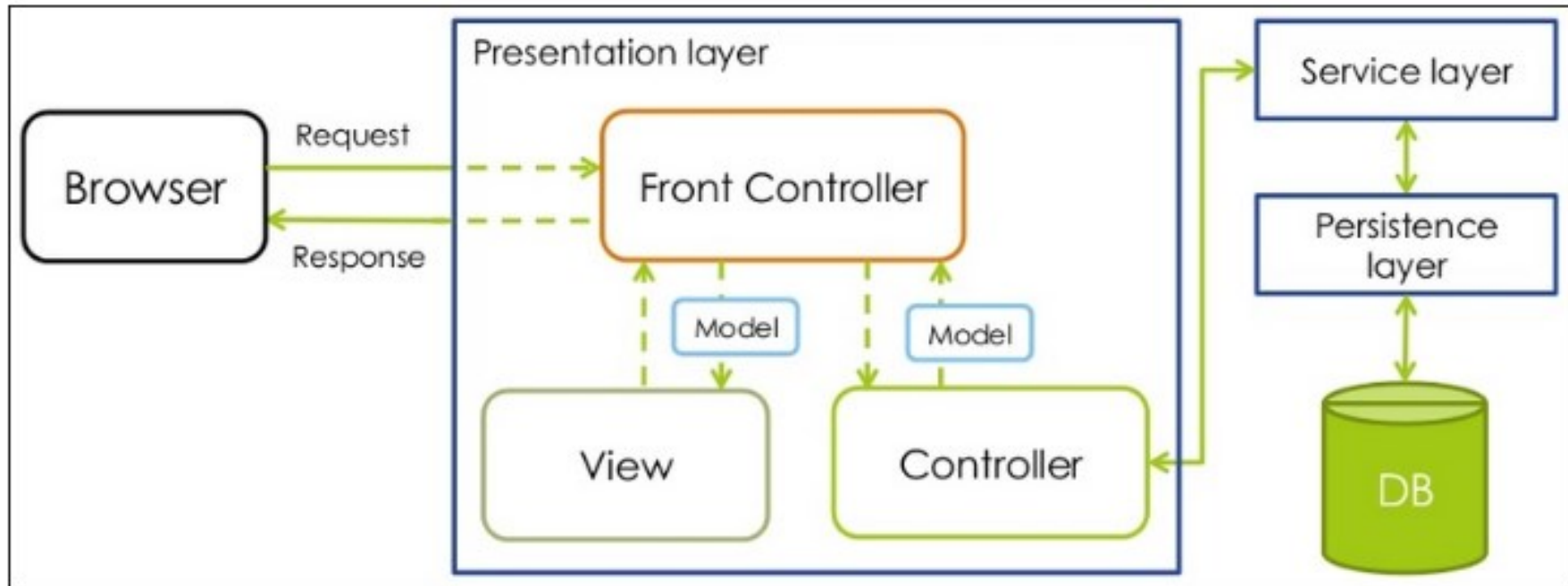
- Presentation Layer
 - Làm nhiệm vụ quản lý điều hướng, nhận HTTP request và chuyển đến Service Layer
 - Bao gồm cả các class Model
 - Không bao giờ giao tiếp trực tiếp với Data Layer
- Service Layer
 - Còn gọi là Business Logic Layer
 - Làm nhiệm vụ xử lý logic nghiệp vụ
 - Có thể bao gồm việc gọi các dịch vụ REST
 - Giao tiếp với Data Layer để đọc/ghi data...
 - Bao gồm các đối tượng DTO (Data Transfer Objects)

Mô hình ba lớp (three tier)

- Data Layer
 - Còn gọi là Persistence Layer
 - Chịu trách nhiệm giao tiếp với database
 - Thực hiện tất cả những thao tác database như create, update, delete, read.
 - Bao gồm các đối tượng Dao (Data Access Objects) hay còn gọi là Repository – liên quan đến database như câu truy vấn (query)
 - Bao gồm các đối tượng Entity – các đối tượng POJOs tương ứng với các bảng trong database
- *POJOs (Plain Old Java Objects): đối tượng chỉ bao gồm các private field, các hàm setter, getter, toString()*

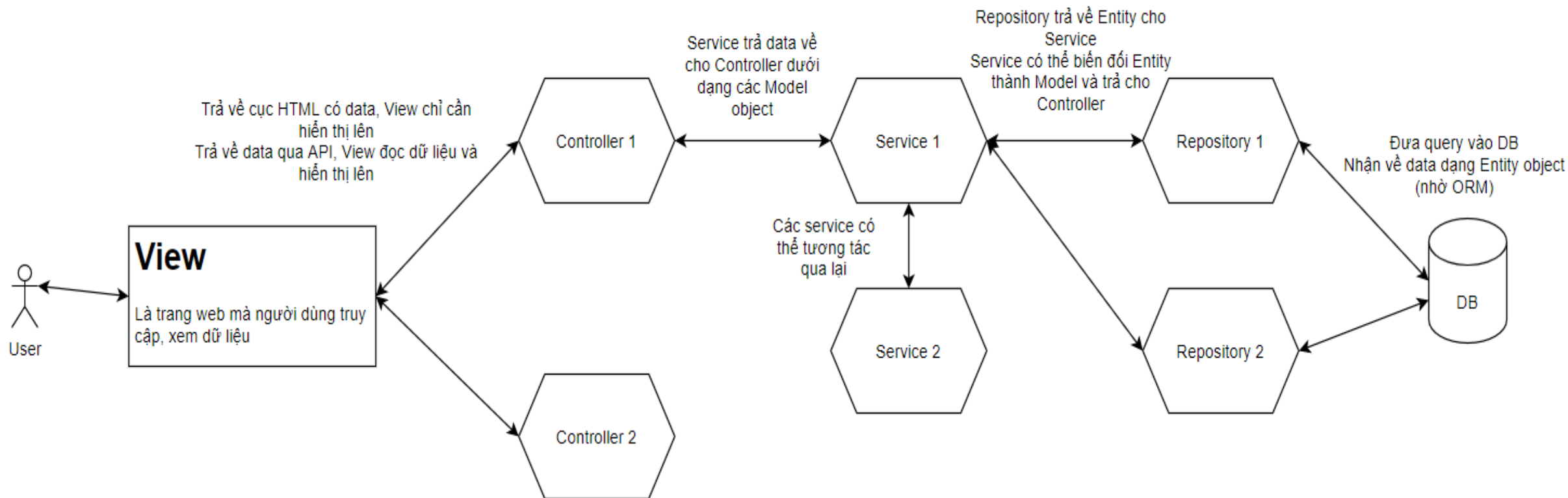
Luồng đi trong Spring Boot

- Tầng Presentation: Spring MVC, View là JSP/JSTL
- Tầng Persistence: Spring Data JPA, Hibernate
- DB là MySQL, SQL Server



Luồng đi trong Spring Boot

Luồng đi trong Spring Boot



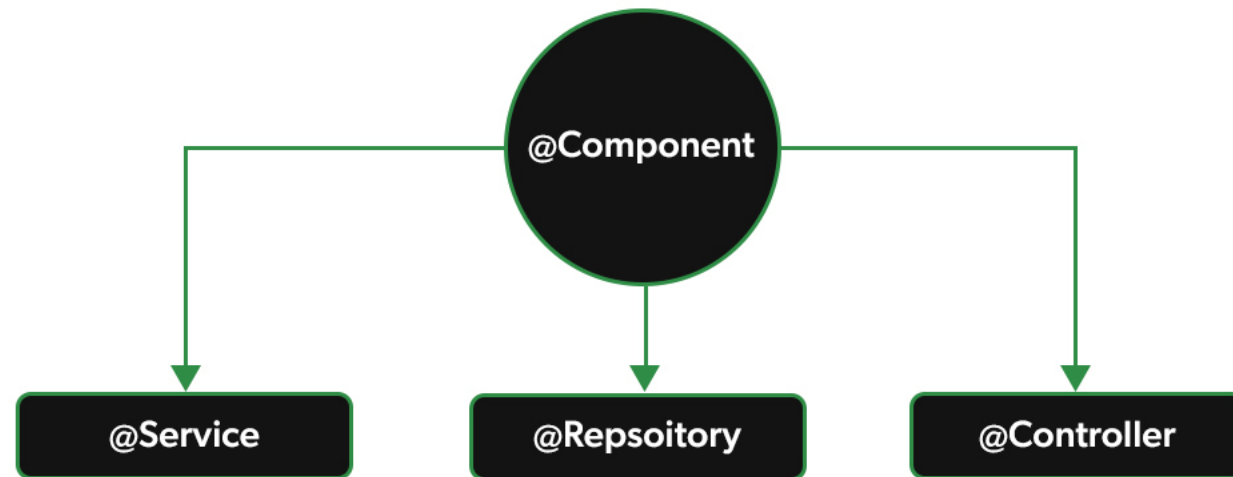
Component Types

@Controller: là tầng giao tiếp với bên ngoài và handler các request từ bên ngoài tới hệ thống.

@Service gắn cho các Bean đảm nhiệm xử lý logic

@Repository gắn cho các Bean đảm nhiệm giao tiếp với DB

@Component gắn cho các Bean khác.



Cấu trúc ứng dụng web

The screenshot displays an IDE interface with the following components:

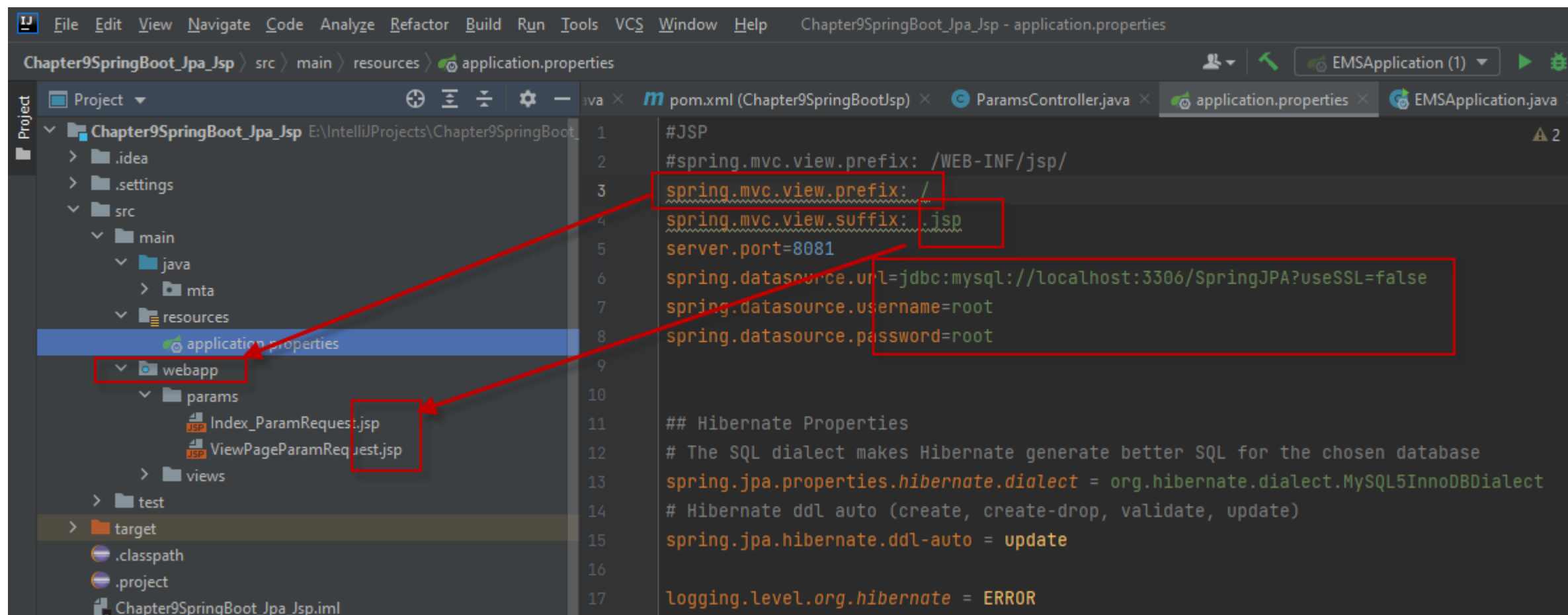
- Project Explorer (Left):** Shows the project structure for 'Chapter5SpringBootMVC_Jpa_Jsp'. The 'resources' folder under 'main' is expanded, and 'application.properties' is highlighted. A red box is drawn around the 'resources' folder, and a red arrow points from it to the code editor.
- Code Editor (Right):** Displays the content of 'application.properties'. The file is open in a tab labeled 'application.properties'. The code is as follows:

```
1 #JSP
2 #spring.mvc.view.prefix: /WEB-INF/jsp/
3 spring.mvc.view.prefix: /
4 spring.mvc.view.suffix: .jsp
5 server.port=8081
6 spring.datasource.url=jdbc:mysql://localhost:3306/SpringJPA?useSSL=false
7 spring.datasource.username=root
8 spring.datasource.password=root
9
10
11 ## Hibernate Properties
12 # The SQL dialect makes Hibernate generate better SQL for the chosen database
13 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
14 # Hibernate ddl auto (create, create-drop, validate, update)
15 spring.jpa.hibernate.ddl-auto = update
16
17 logging.level.org.hibernate = ERROR
18
19
```

Spring Boot JSP ViewResolver Configuration

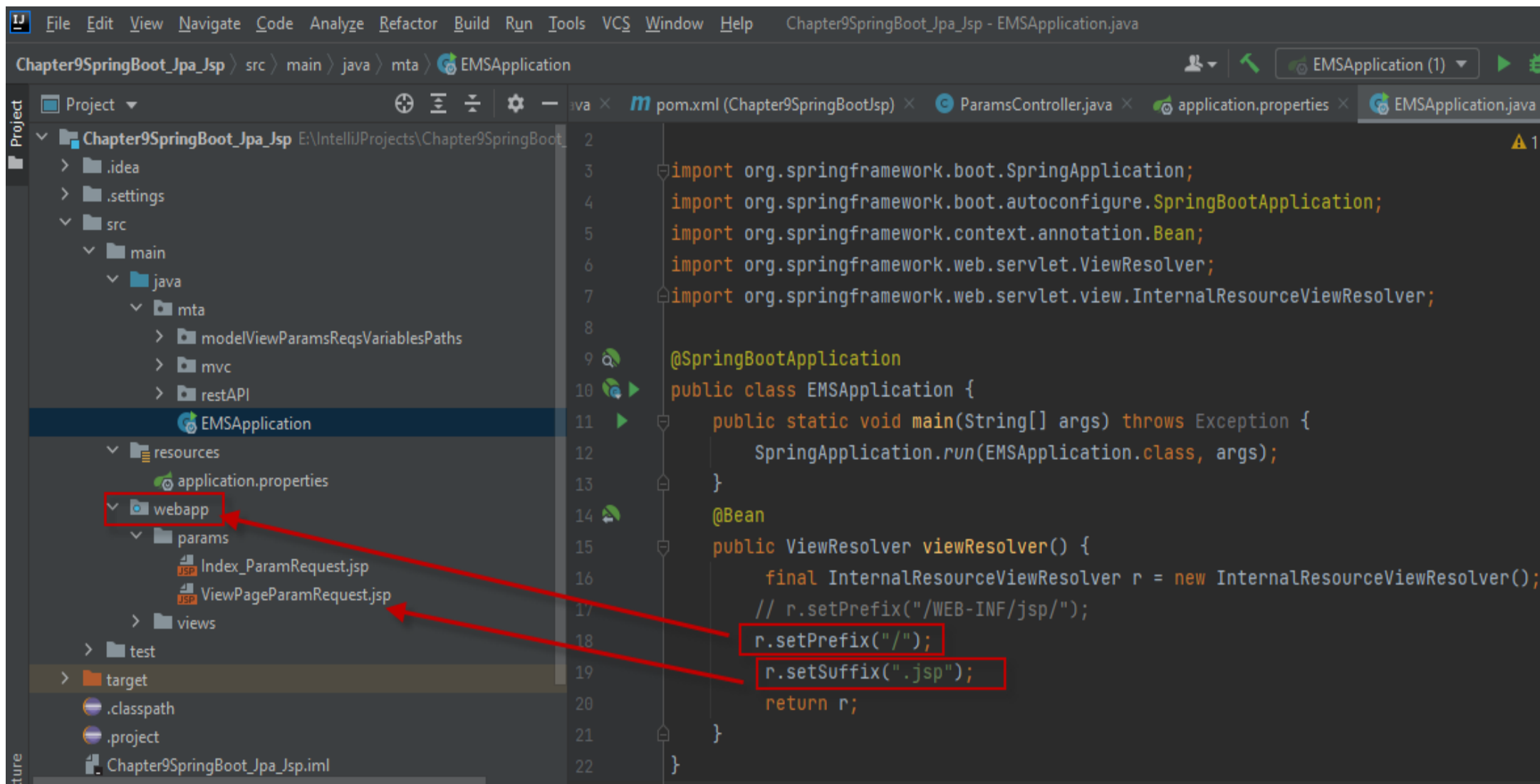
Có 02 cách sau đây để cấu hình tầng Presentation sử dụng JSP.

Cách 1:



Spring Boot JSP ViewResolver Configuration

Cách 2:



Spring MVC Annotations

@Controller tạo component ở tầng controller, thường hay được sử dụng cho Spring Controller truyền thống hay được sử dụng trong các phiên bản Spring từ 4.0 trở xuống.

@RestController được giới thiệu từ phiên bản Spring 4.0 để đơn giản hóa việc tạo ra các RESTful web service=> Nó là sự kết hợp của annotation **@Controller** và **@ResponseBody**.

@RequestMapping dùng để ánh xạ một phương thức đến một request (url). Mặc định là request kiểu GET.

Spring 4.3 giới thiệu một số annotation **RÚT GỌN** của **@RequestMapping** cho phép sử dụng các chức năng tương tự nhưng ngắn gọn và bao hàm một ý nghĩa trong đó, giúp mã nguồn dễ đọc hơn:

@GetMapping; **@PostMapping**; **@PutMapping**;

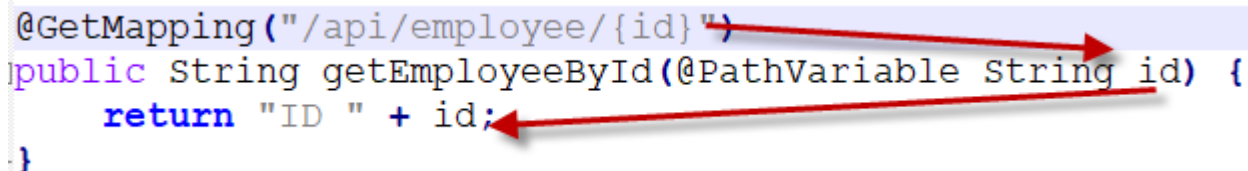
@DeleteMapping; **@PatchMapping**.

@PathVariable

@PathVariable được sử dụng để xử lý template variables trong request URI mapping.

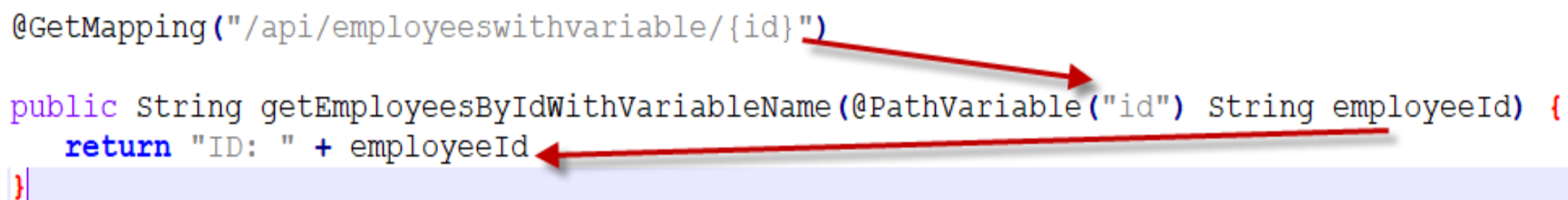
Simple mapping

```
@GetMapping("/api/employee/{id}")
public String getEmployeeById(@PathVariable String id) {
    return "ID " + id;
}
```



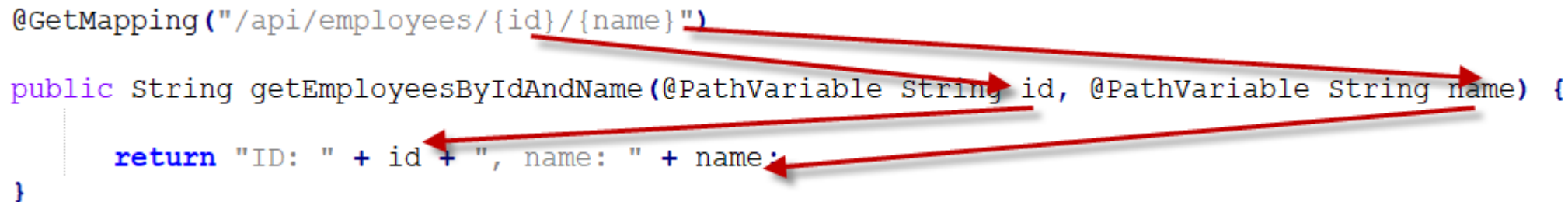
Path Variables Name

```
@GetMapping("/api/employeeswithvariable/{id}")
public String getEmployeesByIdWithVariableName(@PathVariable("id") String employeeId) {
    return "ID: " + employeeId;
}
```



Multiple Path Variables in a Single Request

```
@GetMapping("/api/employees/{id}/{name}")
public String getEmployeesByIdAndName(@PathVariable String id, @PathVariable String name) {
    return "ID: " + id + ", name: " + name;
}
```



@RequestParam

Cách 1: Truyền tham số từ form

```
//Index_ParamRequest.jsp
<form action="readRequestParamsWithName">
    UserName : <input type="text" name="name"/> <br><br>
    Password : <input type="text" name="pass"/> <br><br>
    <input type="submit" name="submit">
</form>

@GetMapping("/paramsRequestView")
public String paramsMethod() {
    return "params/Index_ParamRequest";
}

@RequestMapping("/readRequestParamsWithName") // From view: Index_ParamRequest.jsp
public String display(@RequestParam("name") String nameValue, @RequestParam("pass") String
passValue, Model m) {
    if (passValue.equals("admin")) {
        String msg = "Hello " + nameValue;
        m.addAttribute("message", msg);
        return "params/ViewPageParamRequest";
    } else {
        String msg = "Sorry " + nameValue + ". You entered an incorrect password";
        m.addAttribute("message", msg);
        return "params/ViewPageParamRequest";
    }
}
```


@RequestParam

Cách 2: Truyền tham số qua url

<http://localhost:8081/readRequestParamsWithName?name=Aa&pass=Bb>

```
@RequestMapping("/readRequestParamsWithName")
public String display(@RequestParam("name") String nameValue, @RequestParam("pass") String
passValue, Model m) {
    if (passValue.equals("admin")) {
        String msg = "Hello " + nameValue;
        m.addAttribute("message", msg);
        return "params/ViewPageParamRequest";
    } else {
        String msg = "Sorry " + nameValue + ". You entered an incorrect password";
        m.addAttribute("message", msg);
        return "params/ViewPageParamRequest";
    }
}
```

@ModelAttribute

views/Registration.jsp

```
<form:form method="POST" modelAttribute="employee" action="/home" name="fName">
  <table style="vertical-align: center; margin-left:20%;>
    <tr>
      <td><form:hidden path="id" /></td>
    </tr>
    <tr>
      <td>First Name :</td>
      <td><form:input path="firstName" id="fname"/></td>
    </tr>
    ...
  </table>
</form:form>
```

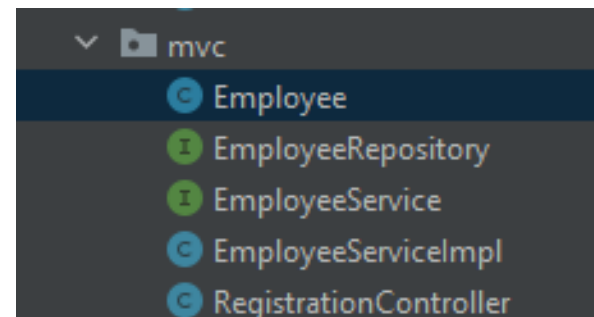
@Controller

```
public class RegistrationController {
    @Autowired
    private EmployeeService employeeService;
    @GetMapping("/registration")
    public String reg( Map<String, Object> model) {
        model.put("employee", new Employee ());
        return "views/Registration";// ViewSolver
    }
    @PostMapping("/home")
    public String createEmployee(
        @ModelAttribute("employee") Employee emp ) {
        employeeService.createOrUpdateEmployee(emp);
        return "redirect:/list";
    }
    ...
}
```

@Entity

```
public class Employee {
    @Id
    ...
    private Long id;
    private String firstName;
    private String lastName;
    ...
}
```

ute.



@SessionAttributes

@SessionAttributes được sử dụng ở cấp độ lớp. Thông thường, nó được sử dụng trên lớp @Controller. Giá trị của 'value' là 1 kiểu String có tên phù hợp với tên được sử dụng trong @ModelAttribute ở cấp phương thức hoặc ở cấp tham số phương thức.

```
@Controller
@SessionAttributes("visitor")
@RequestMapping("/trades")
public class TradeController {

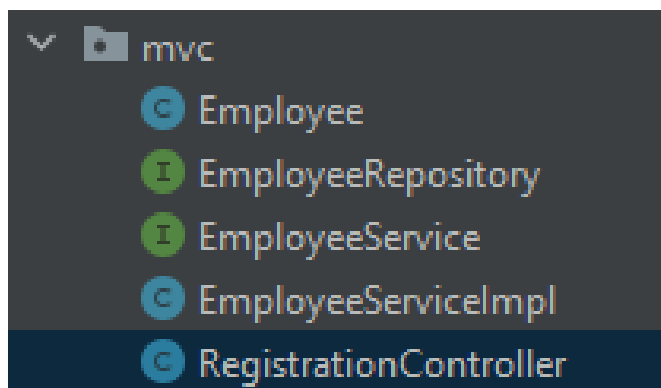
    @ModelAttribute("visitor")
    public Visitor getVisitor (....) {
        return new Visitor(....);
    }
    ....
}
```

Khi có yêu cầu, đầu tiên Spring sẽ cố gắng tìm giá trị của '**visitor**' trong javax.servlet.http.HttpSession. Nếu không tìm thấy thì phương thức với @ModelAttribute có cùng tên 'visitor' (phương thức getVisitor ()) sẽ được gọi. Giá trị được trả về từ phương thức đó sẽ được sử dụng để điền vào session có tên 'visitor'.

Model

Interface Model để truyền dữ liệu từ **Controller** sang **View** để hiển thị.

Spring cho phép chúng ta sử dụng Model như là một tham số trong method của Controller nên chúng ta dễ dàng lấy, chỉnh sửa dữ liệu để truyền qua cho View.



```
@Service
public class EmployeeServiceImpl implements EmployeeService {
    @Autowired
    private EmployeeRepository employeeRepository;
    public List<Employee> getAllEmployee() {
        List<Employee> list = employeeRepository.findAll();
        return list;
    }
    ...
}

@Controller
public class RegistrationController {
    @Autowired
    private EmployeeService employeeService;
    @GetMapping("/list")
    public String listOfEmployee(Model model) {
        List<Employee> employeeList = employeeService.getAllEmployee();
        model.addAttribute("empList", employeeList);
        return "views/employeeList";
    }
    ...
}
```

//views/employeeList.jsp

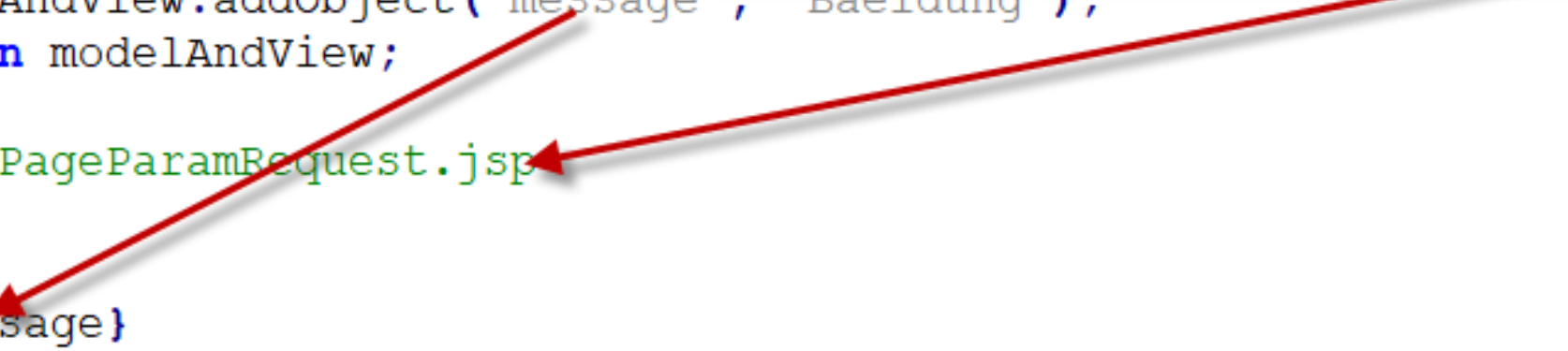
```
<table style="margin-top: 0px;margin-left: 100px; ">
...
</tr>
<c:forEach items="${empList}" var="emp">
<tr>
    <td>${emp.firstName}</td><td>${emp.lastName}</td>
    <td>${emp.userName}</td> <td>${emp.emailId}</td>
    <td>${emp.bloodGp}</td> <td>${emp.age}</td>
    <td>${emp.mobileNo}</td> <td>${emp.empId}</td>
    ..
</tr>
```

Diagram illustrating the flow of data: The `RegistrationController` (Controller) calls `employeeService.getAllEmployee()` to retrieve data. It then adds this data to the `Model` (specifically `model.addAttribute("empList", employeeList)`). This data is then passed to the `views/employeeList.jsp` (View) via the `empList` attribute, which is used in the JSP template to display the employee list.

ModelAndView

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

//params/ViewPageParamRequest.jsp
<html>
<body>
    ${message}
</body>
</html>
```



ModelMap

```
@GetMapping("/printViewPage")

public String passParametersWithModelMap( ModelMap map ) {
    map.addAttribute("welcomeMessage", "welcome");
    map.addAttribute("message", "Baeldung");

    return "params/ViewPageParamRequest";
}

//params/ViewPageParamRequest.jsp
<html>
<body>
    ${message}
</body>
</html>
```

The diagram illustrates the data flow from the Java code to the JSP view. A red box highlights the `ModelMap map` parameter in the `passParametersWithModelMap` method. Another red box highlights the `"params/ViewPageParamRequest"` string returned by the method. A red arrow points from the `return` statement to the `//params/ViewPageParamRequest.jsp` line, indicating the view to be rendered. A second red arrow points from the `return` statement to the `params/ViewPageParamRequest.jsp` file, indicating the view to be rendered.

Xử lý ngoại lệ trong Spring MVC

Chú ý: trong các ứng dụng nhiều lớp (nhiều layers) thì chỉ xử lý các ngoại lệ xuất hiện ở lớp trên cùng (Presentation); Các ngoại lệ xuất hiện ở lớp dưới phải được ném lên các lớp trên.

1> Xử lý ngoại lệ cục bộ trong một controller - **@ExceptionHandler** :

Chúng ta có thể thêm **các phương thức** với annotation là **@ExceptionHandler** vào một controller để xử lý các ngoại lệ xuất hiện trong các phương thức **@RequestMapping** của controller **hiện thời**.

2> Xử lý ngoại lệ toàn cục (Global Exception Handling) - **@ControllerAdvice**, **@ExceptionHandler**

Xử lý ngoại lệ trong Spring MVC - Xử lý ngoại lệ trong một controller

localhost:8081/uploadFile?a=1

I/O Error: Could not read upload file.

localhost:8081/uploadFile?a=2

I/O Error: Database exception!!!

The screenshot shows an IDE with the following components:

- Project Explorer:** Shows the project structure. The package `exceptionSpring.exceptionBasedController` is expanded, and `FileUploadController` is selected.
- Code Editor:** Displays the `FileUploadController` class with the following code:

```
13 @Controller
14 public class FileUploadController {
15     @RequestMapping(value = "/uploadFile", method = RequestMethod.GET)
16     public String doFileUpload(@RequestParam int a) throws IOException, SQLException {
17         // handles file upload stuff...
18         if (a == 1) {
19             throw new IOException("Could not read upload file.");
20         } else if (a == 2) {
21             throw new SQLException("Database exception!!!");
22         }
23         return "done";
24     }
25     @ExceptionHandler({IOException.class, SQLException.class})
26     public ModelAndView handleIOException(Exception ex) {
27         ModelAndView model = new ModelAndView("excepts/IError");
28
29         model.addObject("exception", ex.getMessage());
30         model.addObject("info", "extra info for the exception");
31         return model;
32     }
33 }
```
- Annotations:** `@ExceptionHandler({IOException.class, SQLException.class})` is highlighted with a red box.
- Browser:** Two browser windows are shown at the top. The left window shows an error message "I/O Error: Could not read upload file." with a red arrow pointing to the `throw new IOException("Could not read upload file.");` line in the code. The right window shows an error message "I/O Error: Database exception!!!" with a red arrow pointing to the `throw new SQLException("Database exception!!!");` line in the code.
- File Explorer:** The `excepts` folder is expanded, showing the `IError.jsp` file.

<http://localhost:8081/uploadFile?a=1>

Xử lý ngoại lệ toàn cục (Global Exception Handling)

```
@Data
@AllArgsConstructor
public class Employee {
    private String name;
    private int id;
    Employee() {
    }
}
```

```
import java.sql.SQLException;

public class MySQLException extends SQLException {
    public MySQLException(String st) {
        super(st);
    }
}
```

```
@Controller
public class EmployeeControllerTestException {
    @RequestMapping(value="/emp/{id}", method=RequestMethod.GET)
    public String getEmployee(@PathVariable("id") int id, Model model) throws Exception{
        if(id==1){
            throw new SQLException("SQLException, id="+id);
        }else if(id==2){
            throw new IOException("IOException---, id="+id);
        }else if(id==10){
            Employee emp = new Employee();
            emp.setName("Pankaj");
            emp.setId(id);
            model.addAttribute("employee", emp);
            return "home";
        }else {
            throw new Exception("Generic Exception, id="+id);
        }
    }
}
```

Xử lý ngoại lệ toàn cục (Global Exception Handling)

```
@Controller
public class EmployeeControllerTestException {
    @RequestMapping(value="/emp/{id}", method=RequestMethod.GET)
    public String getEmployee(@PathVariable("id") int id, Model model) throws Exception{
        if(id==1){
            throw new SQLException("SQLException, id="+id);
        }else if(id==2){
            throw new IOException("IOException---, id="+id);
        }else if(id==10){
            Employee emp = new Employee();
            emp.setName("Pankaj");
            emp.setId(id);
            model.addAttribute("employee", emp);
            return "home";
        }else {
            throw new Exception("Generic Exception, id="+id);
        }
    }
}
```

```
@ControllerAdvice
public class GlobalExceptionHandler {
    private static final Logger logger = LoggerFactory.getLogger(GlobalExceptionHandler.class);
    @ExceptionHandler({SQLException.class})
    public ModelAndView handleSQLException(HttpServletRequest request, Exception ex){
        logger.info("SQLException Occured:: URL="+request.getRequestURL());
        ModelAndView model = new ModelAndView("excepts/database_error");
        model.addObject("exception", ex.getMessage());
        return model;
    }
    @ExceptionHandler({Exception.class})
    public ModelAndView handleException(HttpServletRequest request, Exception ex){
        logger.info("SQLException Occured:: URL="+request.getRequestURL());
        ModelAndView model = new ModelAndView("excepts/error");
        model.addObject("exception", ex.getMessage());
        return model;
    }
}
```

```
webapp
├── excepts
│   ├── database_error.jsp
│   ├── error.jsp
│   └── IOError.jsp
├── params
│   ├── Index_ParamRequest.jsp
│   └── ViewPageParamRequest.jsp
└── views
    ├── employeeList.jsp
    ├── Registration.jsp
    └── Registration_RequestBody.jsp
```