Exception Handling

Exception handling in Spring Boot is a mechanism that allows developers to manage and respond to errors and exceptions in a structured and user-friendly way, especially in REST APIs. It provides tools and strategies to catch, handle, and respond to exceptions gracefully, allowing you to define custom messages or HTTP statuses rather than exposing stack traces to the user. This improves the user experience and application security.

Key Components of Exception Handling in Spring Boot

1. **Try-Catch Blocks**
    - Used within methods to handle exceptions locally. For example, if a specific error is expected in a service method, you can use a `try-catch` block to handle it right there.
2. **@ExceptionHandler Annotation**
    - This annotation is used at the method level within a `@Controller` or `@RestController`. It allows you to define specific exception-handling methods for individual controllers.
    - When an exception of a certain type is thrown, the corresponding `@ExceptionHandler` method is triggered.

```
@ExceptionHandler(EmployeeNotFoundException.class)
public ResponseEntity<String>
handleEmployeeNotFound(EmployeeNotFoundException ex) {
    return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
}
```

3. **@ControllerAdvice**
    - `@ControllerAdvice` is a more centralized approach to handle exceptions across all controllers.
    - It acts as a global exception handler, capturing exceptions thrown by any controller in the application.
    - Using `@ExceptionHandler` methods within a `@ControllerAdvice` class allows you to manage exceptions from a single place.

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(EmployeeNotFoundException.class)
    public ResponseEntity<String>
handleEmployeeNotFound(EmployeeNotFoundException ex) {
        return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleGenericException(Exception ex)
{
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                              .body("An error occurred: " +
ex.getMessage());
    }
}
```

4.
5. **ResponseEntityExceptionHandler**
    - This is a built-in Spring class that provides default handling for standard Spring MVC exceptions (like `HttpRequestMethodNotSupportedException`, `HttpMediaTypeNotSupportedException`, etc.).
    - You can extend this class to customize exception responses for specific Spring MVC errors.
6. **@ResponseStatus Annotation**
    - This annotation allows you to specify the HTTP status code for a specific exception. It can be applied directly to custom exception classes, so when the exception is thrown, the specified status code is returned.

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class EmployeeNotFoundException extends RuntimeException {
    public EmployeeNotFoundException(String message) {
        super(message);
    }
}
```

7.

Example: Exception Handling in a Spring Boot REST API

Let's say you have an endpoint that retrieves an employee by ID. If the ID is invalid, you want to throw a custom `EmployeeNotFoundException` and handle it gracefully.

Define the Custom Exception

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class EmployeeNotFoundException extends RuntimeException {
    public EmployeeNotFoundException(String message) {
        super(message);
    }
}
```

Use the Exception in the Controller or Service

```
@GetMapping("/employees/{id}")
public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id)
{
    Employee employee = employeeService.findById(id)
                        .orElseThrow(() -> new
EmployeeNotFoundException("Employee not found with ID: " + id));
    return ResponseEntity.ok(employee);
}
```

1.

Handle the Exception in @ControllerAdvice

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(EmployeeNotFoundException.class)
    public ResponseEntity<String>
handleEmployeeNotFound(EmployeeNotFoundException ex) {
        return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
    }
}
```

2. This setup allows you to:

- Return a `404 Not Found` HTTP status when an employee with the specified ID is not found.
- Show a custom error message to the client instead of a generic server error.

Benefits of Exception Handling in Spring Boot

- **Centralized error handling** for better code organization.
- **User-friendly error responses** by avoiding raw stack traces and providing custom messages.
- **Customizable HTTP status codes** that align with REST API conventions (like 404 for "not found," 400 for "bad request," etc.).
- **Enhanced security** by preventing sensitive information from being exposed in error messages.

With Spring Boot's exception handling features, you can create a more stable, user-friendly, and resilient application.

```
@GetMapping("/get/{id}")
        public ResponseEntity<?> getEmployee(@PathVariable Long id){
                try {
          Employee employee = service.getEmployee(id);
          return ResponseEntity.ok(employee);
      } catch (ResourceNotFoundException e) {
          return
ResponseEntity.status(HttpStatus.NOT_FOUND).body("Resource not
found");
      } catch (EmployeeNotFoundException e) {
                        // TODO Auto-generated catch block
          return
ResponseEntity.status(HttpStatus.NOT_FOUND).body("employee not
found");

                }
        }
```

```
public Employee getEmployee(Long id) throws
EmployeeNotFoundException {
                // TODO Auto-generated method stub
                 if (id <= 0) {
                   throw new EmployeeNotFoundException("Employee not
found with ID: " + id);
               }
                Employee byId = repository.findById(id).orElseThrow(()->new
ResourceNotFoundException("Employee not found with user id "+id));


                return byId;
        }
```

```
package com.example.exception;
public class EmployeeNotFoundException extends Exception {
        public EmployeeNotFoundException(String message) {
      super(message);
  }
}
```

```
package com.example.exception;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
@RestControllerAdvice
public class GlobalExceptionHandler {
        @ExceptionHandler(ResourceNotFoundException.class)
           public ResponseEntity<?>
handlerResourceNotFoundException(ResourceNotFoundException ex){
                String message = ex.getMessage();
                return new
ResponseEntity<>(message,HttpStatus.NOT_FOUND);
```

```
        }
    @ExceptionHandler(EmployeeNotFoundException.class)
        public ResponseEntity<String>
handleUserNotFoundException(EmployeeNotFoundException ex) {
            String message = ex.getMessage();
    return new ResponseEntity<>(message,HttpStatus.NOT_FOUND);
        }
}
```