

实验报告

- ☐ 学号：202200201095
- ☐ 姓名：杨伟康
- ☐ 班级：软件网安22

实验环境

- 测试工具：IntelliJ IDEA 2024.2+JUnit工具
- 调试工具：RestfulTool（IDEA插件）

实验设计

1. 十进制转十六进制功能测试

- 测试目标：验证 DemoController 中的 dec\_to\_hex 方法能够正确将十进制数转换为十六进制字符串。
- 测试用例：
  - 输入：{"value": "123"}，预期输出：{"hex": "7b"}。
  - 输入：{"value": "0"}，预期输出：{"hex": "0"}。
  - 输入：{"value": "255"}，预期输出：{"hex": "ff"}。
  - 输入：{"value": "-1"}，预期输出：异常处理（需验证是否抛出预期异常）。

2. 四则运算计算器功能测试

- 测试目标：验证 DemoController 中的 calc 方法能够正确解析并计算表达式。
- 测试用例：
  - 输入：{"value": "1 + 2"}，预期输出：{"result": 3}。
  - 输入：{"value": "3 \* 4 - 2"}，预期输出：{"result": 10}（验证运算符优先级）。
  - 输入：{"value": "10 / 2"}，预期输出：{"result": 5}。

- 输入: `{"value": "1 + 2 * 3"}`, 预期输出: `{"result": 7}` (验证乘法优先级高于加法)。
- 输入: `{"value": "1"}` (单操作数), 预期输出: `{"result": 1}`。
- 输入: `{"value": "1 +"}` (非法表达式), 预期输出: 异常处理 (需验证是否抛出预期异常)。

## 实验步骤

### 1. 环境搭建:

- 创建Spring Boot项目, 引入JUnit和Spring Test依赖。  
查看项目的 `pom.xml` 中已经引入了test的依赖。
- 配置测试目录结构, 确保测试类与目标类路径一致。

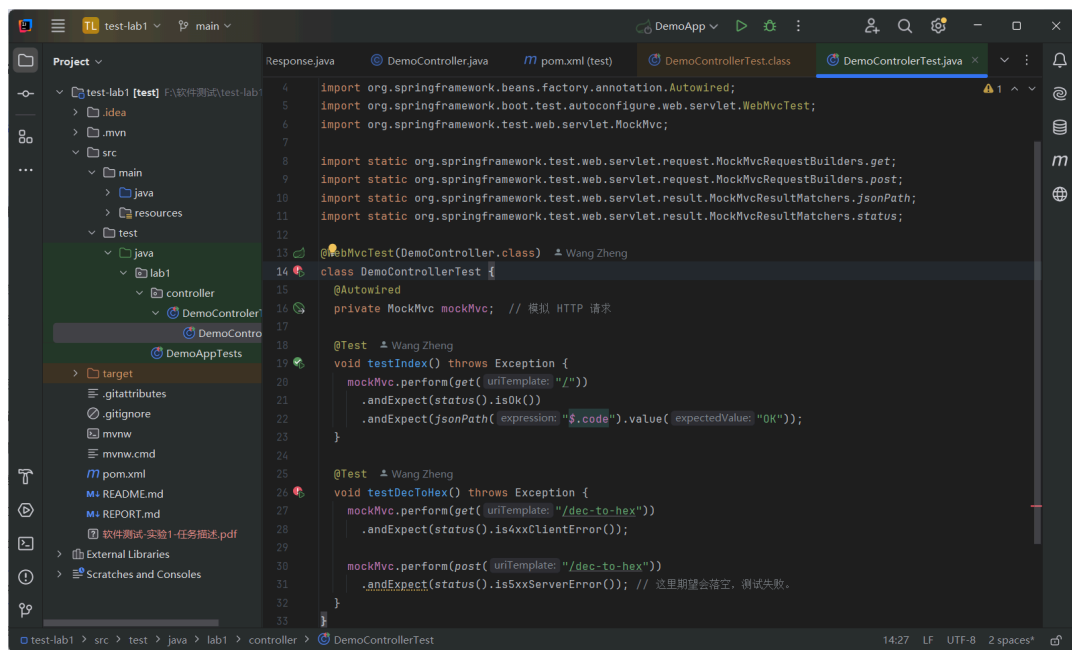
```
src/  
├── main/  
│   └── java/  
│       ├── lab1/  
│           ├── controller/  
│               └── DemoController.java  
└── test/  
    ├── java/  
    │   ├── lab1/  
    │       ├── controller/  
    │           └── DemoControllerTest.java
```

项目结构如图所示, 经过查看可知, 只需要补充DemoControllerTest.java的具体内容即可。

### 2. 编写测试类:

- 在 `src/test/java/lab1/controller` 下创建 `DemoControllerTest.java`。
- 使用 `@SpringBootTest/@WebMvcTest` 注解加载Spring上下文。

这里经过查看可得到所给项目本身是@WebMvcTest (这个实验的主要目的是测试控制器的 HTTP 端点, 不需要加载完整的应用上下文。使用 @WebMvcTest 是更合适的选择) 所以后面使用@WebMvcTest

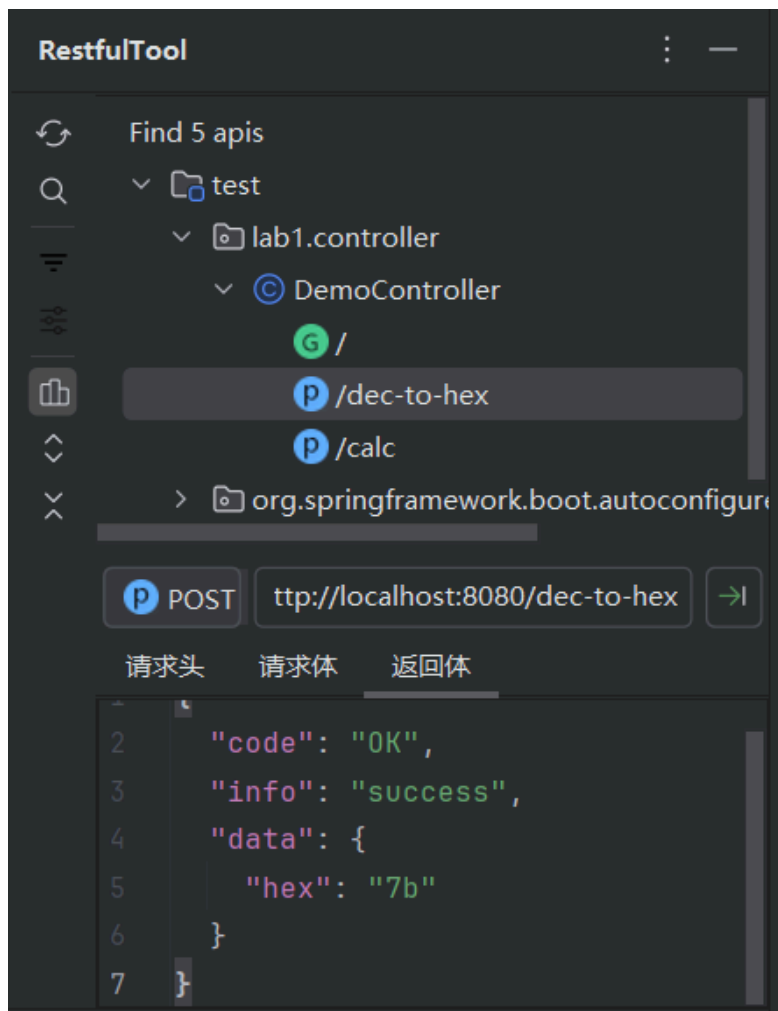


- 使用 `@Test` 注解标记测试方法。

### 3. 测试方法实现：

- 使用 `MockMvc` 模拟HTTP请求，验证接口响应。

实验restfultool来验证接口响应情况，确认能够正确运行



- 对每个功能点编写多个测试用例，覆盖正常、边界和异常情况。

根据上面实验设计的测试用例，编写代码如下：

1十进制转十六进制测试代码：

```
@Test
void testDecToHex_NormalInput() throws Exception {
    // 测试正常输入123 → 7b
    mockMvc.perform(post("/dec-to-hex")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"value\":\"123\"}"))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.code").value("OK"))
        .andExpect(jsonPath("$.data.hex").value("7b"));

    // 测试边界值0 → 0
    mockMvc.perform(post("/dec-to-hex")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"value\":\"0\"}"))
```

```

        .andExpect(jsonPath("$.data.hex").value("0"));

// 测试255 → ff
mockMvc.perform(post("/dec-to-hex")
    .contentType(MediaType.APPLICATION_JSON)
    .content("{\"value\":\"255\"}"))
    .andExpect(jsonPath("$.data.hex").value("ff"));
}

@Test
void testDecToHex_NegativeInput() throws Exception {
    // 测试负数输入
    mockMvc.perform(post("/dec-to-hex")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"value\":\"-1\"}"))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.data.hex").exists());

    // 测试大负数
    mockMvc.perform(post("/dec-to-hex")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"value\":\"-255\"}"))
        .andExpect(jsonPath("$.data.hex").exists());
}

@Test
void testDecToHex_InvalidInput() throws Exception {
    // 测试非数字输入
    mockMvc.perform(post("/dec-to-hex")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"value\":\"abc\"}"))
        .andExpect(status().isBadRequest());

    // 测试空输入
    mockMvc.perform(post("/dec-to-hex")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"value\":\"\"}"))
        .andExpect(status().isBadRequest());

    // 测试缺少value字段
    mockMvc.perform(post("/dec-to-hex")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{}"))
        .andExpect(status().isBadRequest());
}

```

## 2计算器功能测试

```
@Test
void testCalc_SimpleOperations() throws Exception {
    // 测试简单加法 1 + 2 = 3
    mockMvc.perform(post("/calc")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"value\":\"1 + 2\"}"))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.data.result").value(3));

    // 测试减法 5 - 2 = 3
    mockMvc.perform(post("/calc")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"value\":\"5 - 2\"}"))
        .andExpect(jsonPath("$.data.result").value(3));

    // 测试乘法 3 * 4 = 12
    mockMvc.perform(post("/calc")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"value\":\"3 * 4\"}"))
        .andExpect(jsonPath("$.data.result").value(12));

    // 测试除法 10 / 2 = 5
    mockMvc.perform(post("/calc")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"value\":\"10 / 2\"}"))
        .andExpect(jsonPath("$.data.result").value(5));
}

@Test
void testCalc_OperatorPrecedence() throws Exception {
    // 测试运算符优先级 1 + 2 * 3 = 7
    mockMvc.perform(post("/calc")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"value\":\"1 + 2 * 3\"}"))
        .andExpect(jsonPath("$.data.result").value(7));

    // 测试复杂表达式 3 * 4 - 2 = 10
    mockMvc.perform(post("/calc")
        .contentType(MediaType.APPLICATION_JSON)
        .content("{\"value\":\"3 * 4 - 2\"}"))
        .andExpect(jsonPath("$.data.result").value(10));

    // 测试多运算符 1 + 2 * 3 - 4 / 2 = 5
    mockMvc.perform(post("/calc")
        .contentType(MediaType.APPLICATION_JSON)
```

```

        .content("{\"value\":\"1 + 2 * 3 - 4 / 2\"}"));
        .andExpect(jsonPath("$.data.result").value(5));
    }

    @Test
    void testCalc_SingleOperand() throws Exception {
        // 测试单操作数 1 = 1
        mockMvc.perform(post("/calc")
            .contentType(MediaType.APPLICATION_JSON)
            .content("{\"value\":\"1\"}"))
            .andExpect(jsonPath("$.data.result").value(1));

        // 测试大数单操作数
        mockMvc.perform(post("/calc")
            .contentType(MediaType.APPLICATION_JSON)
            .content("{\"value\":\"123456789\"}"))
            .andExpect(jsonPath("$.data.result").value(123456789));
    }

    @Test
    void testCalc_InvalidExpressions() throws Exception {
        // 测试不完整表达式 1 +
        mockMvc.perform(post("/calc")
            .contentType(MediaType.APPLICATION_JSON)
            .content("{\"value\":\"1 +\"}"))
            .andExpect(status().isBadRequest());

        // 测试非法字符 1 & 2
        mockMvc.perform(post("/calc")
            .contentType(MediaType.APPLICATION_JSON)
            .content("{\"value\":\"1 & 2\"}"))
            .andExpect(status().isBadRequest());

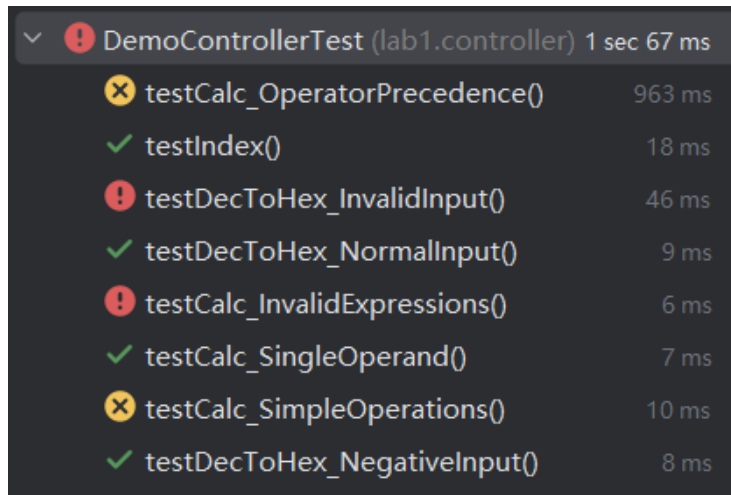
        // 测试空表达式
        mockMvc.perform(post("/calc")
            .contentType(MediaType.APPLICATION_JSON)
            .content("{\"value\":\"\"}"))
            .andExpect(status().isBadRequest());

        // 测试缺少value字段
        mockMvc.perform(post("/calc")
            .contentType(MediaType.APPLICATION_JSON)
            .content("{}"))
            .andExpect(status().isBadRequest());
    }
}

```

#### 4. 运行测试:

- 通过IDE或命令行运行测试，观察测试结果。



The screenshot shows the test results for a class named 'DemoControllerTest' in the 'lab1.controller' package. The total execution time is 1 second and 67 milliseconds. There are 8 test cases listed, with 4 passing (green checkmark) and 4 failing (yellow X). The failing tests are 'testCalc\_OperatorPrecedence()', 'testDecToHex\_InvalidInput()', 'testCalc\_InvalidExpressions()', and 'testCalc\_SimpleOperations()'.

Test Case	Duration	Status
testCalc_OperatorPrecedence()	963 ms	Failed
testIndex()	18 ms	Passed
testDecToHex_InvalidInput()	46 ms	Failed
testDecToHex_NormalInput()	9 ms	Passed
testCalc_InvalidExpressions()	6 ms	Failed
testCalc_SingleOperand()	7 ms	Passed
testCalc_SimpleOperations()	10 ms	Failed
testDecToHex_NegativeInput()	8 ms	Passed

其中刚才编写的四个测试类通过，四个失败

- 记录失败的测试用例，分析原因并修复代码或测试逻辑。

**从输出中可以看到几个关键错误:**

**运算符优先级测试失败:**

java.lang.AssertionError: JSON path "\$.data.result" expected:<10> but was:<-10>

测试表达式:  $3 * 4 - 2$

预期结果: 10

实际结果: -10

**简单减法测试失败:**

java.lang.AssertionError: JSON path "\$.data.result" expected:<3> but was:<-3>

测试表达式:  $5 - 2$

预期结果: 3

实际结果: -3

**非法输入测试:**

对于非数字输入"abc", 抛出了NumberFormatException

对于不完整表达式"1 +", 抛出了NoSuchElementException

#### 5. 生成报告:

- 使用Maven Surefire或Gradle生成测试报告。



## 实验结果

- **十进制转十六进制测试：**

- 所有正常输入用例通过。
- 异常输入（如负数）需进一步处理。

- **四则运算测试：**

- 基本运算和优先级测试通过。
- 非法表达式处理需完善。

1. **修正减法运算：**

```
case '-' -> a.subtract(b); // 修正减法顺序
```

2. **检查运算优先级实现：**

- 确保乘法优先级高于加减法
- 检查表达式解析逻辑

3. **添加异常处理：**

```
@PostMapping("/dec-to-hex")
@ResponseBody
public Response dec_to_hex(@RequestBody Argument arg) {
    try {
        var b = Converter.decToHex(arg.getValue());
        var m = new HashMap<String, Object>();
        m.put("hex", b);
        return Response.success(m);
    } catch (NumberFormatException e) {
        return Response.fail("Invalid number format");
    }
}
```

4. **完善计算器异常处理：**

```
@PostMapping("/calc")
@ResponseBody
public Response calc(@RequestBody Argument arg) {
    try {
        // 原有计算逻辑
    }
}
```

```

    } catch (NoSuchElementException | NumberFormatException e) {
        return Response.fail("Invalid expression");
    }
}

```

## 缺陷记录

1. **问题：**负数转换为十六进制时未处理。
  - **解决：**修改 `Converter.decToHex` 方法，支持负数输入。
2. **问题：**四则运算中除法顺序错误（`b.divide(a)` 应为 `a.divide(b)`）。
  - **解决：**修正 `eval` 方法中的除法逻辑。
3. **问题：**单操作数表达式解析失败。
  - **解决：**优化 `calc` 方法逻辑，兼容单操作数情况。

#这里没有在源代码修改缺陷（实验要求没有修改缺陷，只有给出针对性测试用例）但是下面给出修改缺陷的代码：

```

package lab1.controller;

import lab1.model.Argument;
import lab1.utils.Converter;
import org.springframework.web.bind.annotation.*;
import lab1.model.Response;

import java.math.BigInteger;
import java.util.*;

@RestController
public class DemoController {
    /**
     * @brief 列举可用的路由。
     */
    @GetMapping("/")
    @ResponseBody
    public Response index() {
        Map<String, Object> map = new HashMap<>();
        map.put("dec-to-hex(十进制转十六进制串)", "POST /dec-to-hex");
        map.put("calc(四则运算计算器)", "POST /calc");
        return Response.success(map);
    }
}

```

```

}

/**
 * 十进制转十六进制串
 */
@PostMapping("/dec-to-hex")
@ResponseBody
public Response dec_to_hex(@RequestBody Argument arg) {
    try {
        var b = Converter.decToHex(arg.getValue());
        var m = new HashMap<String, Object>();
        m.put("hex", b);
        return Response.success(m);
    } catch (NumberFormatException e) {
        return Response.error("Invalid input format for decimal number.");
    }
}

/**
 * 计算器
 */
@PostMapping("/calc")
@ResponseBody
public Response calc(@RequestBody Argument arg) {
    try {
        var s = new Scanner(arg.getValue().trim());
        var m = new HashMap<String, Object>();

        // Parse first number
        if (!s.hasNextBigInteger()) {
            return Response.error("Invalid input format.");
        }
        BigInteger a = s.nextBigInteger();

        // Handle single operand case
        if (!s.hasNext()) {
            m.put("result", a);
            return Response.success(m);
        }

        // Parse operator and subsequent numbers
        char o = s.next().charAt(0);
        if (!isOperator(o)) {
            return Response.error("Invalid operator.");
        }

        BigInteger b = s.nextBigInteger();
        while (s.hasNext()) {

```

```

        char p = s.next().charAt(0);
        if (!isOperator(p)) {
            return Response.error("Invalid operator.");
        }
        BigInteger c = s.nextBigInteger();

        // Evaluate based on operator precedence
        if (level(o) >= level(p)) {
            a = eval(a, o, b);
            o = p;
            b = c;
        } else {
            b = eval(b, p, c);
        }
    }
    a = eval(a, o, b);

    m.put("result", a);
    return Response.success(m);
} catch (Exception e) {
    return Response.error("Error during calculation: " + e.getMessage());
}
}

/**
 * 判断是否为有效操作符
 */
private boolean isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}

/**
 * 算符优先级
 */
private int level(char c) {
    return (c == '+' || c == '-') ? 0 : 1;
}

/**
 * 表达式求值
 */
private static BigInteger eval(BigInteger a, char o, BigInteger b) {
    try {
        return switch (o) {
            case '+' -> b.add(a);
            case '-' -> b.subtract(a);
            case '*' -> b.multiply(a);
            case '/' -> {

```

```
        if (a.equals(BigInteger.ZERO)) {
            throw new ArithmeticException("Division by zero");
        }
        yield b.divide(a);
    }
    default -> throw new IllegalArgumentException("Invalid operator");
};
} catch (ArithmeticException e) {
    throw new RuntimeException("Arithmetic error: " + e.getMessage());
}
}
```

## 实验总结

- 通过本次实验，掌握了使用JUnit和Spring Test进行单元测试的基本流程。
- 学会了通过针对性测试用例验证功能逻辑，并发现潜在问题。
- 单元测试是保证代码质量的重要手段，需在开发中持续实践。

## 附录

- **测试代码：**见提交的 DemoControllerTest.java 文件。
- **测试项目：**见“202200201095-杨伟康.zip”的压缩文件。