1.单元测试

1.1概述【理解】

JUnit是一个 Java 编程语言的单元测试工具。JUnit 是一个非常重要的测试工具

1.2特点【理解】

- JUnit是一个开放源代码的测试工具。
- 提供注解来识别测试方法。
- JUnit测试可以让你编写代码更快，并能提高质量。
- JUnit优雅简洁。没那么复杂，花费时间较少。
- JUnit在一个条中显示进度。如果运行良好则是绿色；如果运行失败，则变成红色。

1.3使用步骤【应用】

- 使用步骤

    1. 将junit的jar包导入到工程中 junit-4.9.jar
    2. 编写测试方法该测试方法必须是公共的无参数无返回值的非静态方法
    3. 在测试方法上使用@Test注解标注该方法是一个测试方法
    4. 选中测试方法右键通过junit运行该方法

- 代码示例

```
xxxxxxxxxx

10
1

public class JunitDemo1 {

2

    @Test

3

    public void add() {

4

        System.out.println(2 / 0);

5

        int a = 10;

6

        int b = 20;

7

        int sum = a + b;

8

        System.out.println(sum);
```

```
9
    }
10
}
```

1.4相关注解【应用】

- 注解说明

注解	含义
@Test	表示测试该方法
@Before	在测试的方法前运行
@After	在测试的方法后运行

- 代码示例

```
xxxxxxxxxx
17
1
public class JunitDemo2 {
2
    @Before
3
    public void before() {
4
        // 在执行测试代码之前执行，一般用于初始化操作
5
          System.out.println("before");
6
    }
7
    @Test
8
```

```
    public void test() {
9
        // 要执行的测试代码
10
          System.out.println("test");
11
    }
12
    @After
13
    public void after() {
14
        // 在执行测试代码之后执行，一般用于释放资源
15
          System.out.println("after");
16
    }
17
}
```

2.日志

2.1概述【理解】

- 概述

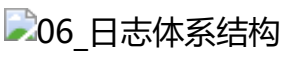
  程序中的日志可以用来记录程序在运行的时候点点滴滴。并可以进行永久存储。

- 日志与输出语句的区别

	输出语句	日志技术
取消日志	需要修改代码，灵活性比较差	不需要修改代码，灵活性比较好
输出位置	只能是控制台	可以将日志信息写入到文件或者数据库中
多线程	和业务代码处于一个线程中	多线程方式记录日志，不影响业务代码的性能

2.2日志体系结构和Log4J【理解】

- 体系结构

- Log4J

  Log4j是Apache的一个开源项目。

  通过使用Log4j，我们可以控制日志信息输送的目的地是控制台、文件等位置。

  我们也可以控制每一条日志的输出格式。

  通过定义每一条日志信息的级别，我们能够更加细致地控制日志的生成过程。

  最令人感兴趣的就是，这些可以通过一个配置文件来灵活地进行配置，而不需要修改应用的代码。

- Apache基金会

  Apache软件基金会（也就是Apache Software Foundation，简称为ASF），为支持开源软件项目而办的一个非盈利性组织。

2.3入门案例【应用】

- 使用步骤

  1. 导入log4j的相关jar包
  2. 编写log4j配置文件
  3. 在代码中获取日志的对象
  4. 按照级别设置记录日志信息

- 代码示例

```
xxxxxxxxxx

42
1

// log4j的配置文件,名字为log4j.properties, 放在src根目录下

2
```

```
log4j.rootLogger=debug,my,fileAppender
3


4
### direct log messages to my ###
5
log4j.appender.my=org.apache.log4j.ConsoleAppender
6
log4j.appender.my.ImmediateFlush = true
7
log4j.appender.my.Target=System.out
8
log4j.appender.my.layout=org.apache.log4j.PatternLayout
9
log4j.appender.my.layout.ConversionPattern=%d %t %5p %c{1}:%L - %m%n
10


11
# fileAppender��’
12
log4j.appender.fileAppender=org.apache.log4j.FileAppender
13
log4j.appender.fileAppender.ImmediateFlush = true
14
log4j.appender.fileAppender.Append=true
15
log4j.appender.fileAppender.File=D:/log4j-log.log
16
log4j.appender.fileAppender.layout=org.apache.log4j.PatternLayout
17
log4j.appender.fileAppender.layout.ConversionPattern=%d %5p %c{1}:%L - %m%n
18


19
// 测试类
20
public class Log4JTest01 {
```

```
21

22
    //使用log4j的api来获取日志的对象
23
    //弊端：如果以后我们更换日志的实现类，那么下面的代码就需要跟着改
24
    //不推荐使用
25
    //private static final Logger LOGGER = Logger.getLogger(Log4JTest01.class);
26

27
    //使用slf4j里面的api来获取日志的对象
28
    //好处：如果以后我们更换日志的实现类，那么下面的代码不需要跟着修改
29
    //推荐使用
30
    private static   final Logger LOGGER = LoggerFactory.getLogger(Log4JTest01.class);
31

32
    public static void main(String[] args) {
33
        //1.导入jar包
34
        //2.编写配置文件
35
        //3.在代码中获取日志的对象
36
        //4.按照日志级别设置日志信息
37
        LOGGER.debug("debug级别的日志");
38
        LOGGER.info("info级别的日志");
39
```

```
            LOGGER.warn("warn级别的日志");
40
            LOGGER.error("error级别的日志");
41
    }
42
}
```

2.4配置文件详解【理解】

- 三个核心
  - Loggers(记录器) 日志的级别

    Loggers组件在此系统中常见的五个级别：DEBUG、INFO、WARN、ERROR 和 FATAL。

    DEBUG < INFO < WARN < ERROR < FATAL。

    Log4j有一个规则：只输出级别不低于设定级别的日志信息。
  - Appenders(输出源) 日志要输出的地方

    把日志输出到不同的地方，如控制台（Console）、文件（Files）等。

    - org.apache.log4j.ConsoleAppender（控制台）
    - org.apache.log4j.FileAppender（文件）
  - Layouts(布局) 日志输出的格式

    可以根据自己的喜好规定日志输出的格式

    常用的布局管理器：

    org.apache.log4j.PatternLayout（可以灵活地指定布局模式）

    org.apache.log4j.SimpleLayout（包含日志信息的级别和信息字符串）

    org.apache.log4j.TTCCLayout（包含日志产生的时间、线程、类别等信息）
- 配置根Logger
  - 格式

    log4j.rootLogger=日志级别，appenderName1，appenderName2，...
  - 日志级别

    OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL或者自定义的级别。
  - appenderName1

    就是指定日志信息要输出到哪里。可以同时指定多个输出目的地，用逗号隔开。

    例如：log4j.rootLogger = INFO，ca，fa
- ConsoleAppender常用的选项
  - ImmediateFlush=true

    表示所有消息都会被立即输出，设为false则不输出，默认值是true。
  - Target=System.err

默认值是System.out。

- FileAppender常用的选项
  - ImmediateFlush=true

    表示所有消息都会被立即输出。设为false则不输出，默认值是true
  - Append=false

    true表示将消息添加到指定文件中，原来的消息不覆盖。

    false则将消息覆盖指定的文件内容，默认值是true。
  - File=D:/logs/logging.log4j

    指定消息输出到logging.log4j文件中
- PatternLayout常用的选项
  - ConversionPattern=%m%n

    设定以怎样的格式显示消息

     07_PatternLayout常用的选项

2.5在项目中的应用【应用】

- 步骤

  1. 导入相关的依赖
  2. 将资料中的properties配置文件复制到src目录下
  3. 在代码中获取日志的对象
  4. 按照级别设置记录日志信息

- 代码实现

```
xxxxxxxxxx

18
1

@WebServlet(urlPatterns = "/servlet/loginservlet")

2

public class LoginServlet implements HttpServlet{

3



4

    //获取日志的对象

5

    private static final Logger LOGGER = LoggerFactory.getLogger(LoginServlet.class);
```

```
6

7
    @Override
8
    public void service(HttpRequest httpRequest, HttpResponse httpResponse) {
9
        //处理
10
          System.out.println("LoginServlet处理了登录请求");
11

12
          LOGGER.info("现在已经处理了登录请求，准备给浏览器响应");
13

14
        //响应
15
          httpResponse.setContentTpye("text/html;charset=UTF-8");
16
          httpResponse.write("登录成功");
17
    }
18
}
```