**Junit**

## 一、Junit 是什么

JUnit是一个Java语言的单元测试框架。它由Kent Beck和Erich Gamma建立，逐渐成为源于Kent Beck的sUnit的xUnit家族中最为成功的一个JUnit有它自己的JUnit扩展生态圈。多数Java的开发环境都已经集成了JUnit作为单元测试的工具。注意：Junit 测试也是程序员测试，即所谓的白盒测试，它需要程序员知道被测试的代码如何完成功能，以及完成什么样的功能

## 二、Junit 能做什么

Junit 是一个单元测试框架，那么使用 Junit 能让我们快速的完成单元测试。通常我们写完代码想要测试这段代码的正确性，那么必须新建一个类，然后创建一个 main() 方法，然后编写测试代码。如果需要测试的代码很多呢？那么要么就会建很多main() 方法来测试，要么将其全部写在一个 main() 方法里面。这也会大大的增加测试的复杂度，降低程序员的测试积极性。而 Junit 能很好的解决这个问题，简化单元测试，写一点测一点，在编写以后的代码中如果发现问题可以较快的追踪到问题的原因，减小回归错误的纠错难度。

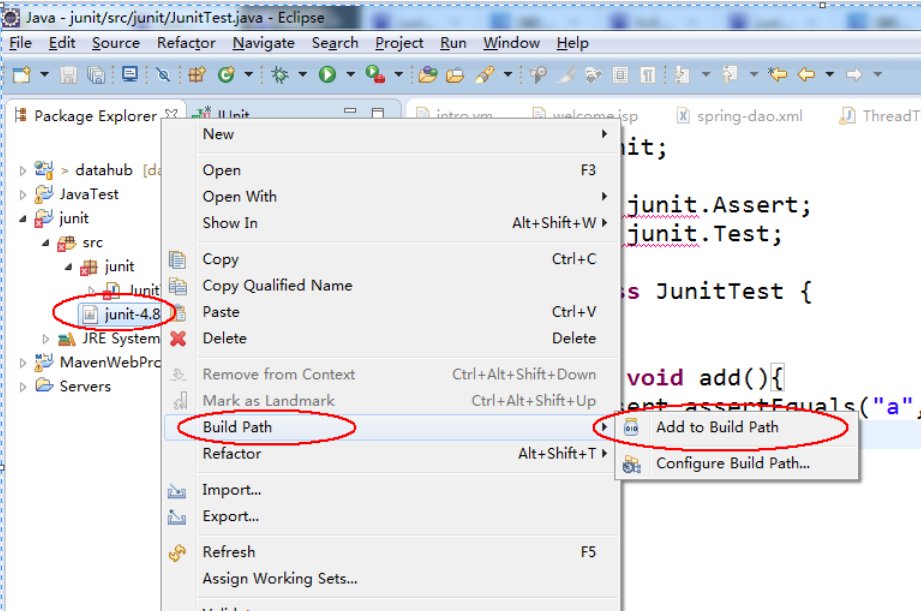
## 三、Junit 的用法

**1首先下载 Junit jar 包，这里给两个版本的百度云下载地址：**

　　①、Junit 4.12版本    链接：http://pan.baidu.com/s/1c2cRqdM 密码：hfix

　　②、Junit 4.8 版本     链接：http://pan.baidu.com/s/1qY8WVGK 密码：ma2u

**2下载完成之后，在项目中将 下载的 jar 包放进去，然后右键，Build--->Add Build Path  即可。**



**3 如果你是用 eclipse 开发，也可以不用下载那些jar包，eclipse内部集成了，我们只需要引入即可：**

①、选中项目，右键Build--->Add Library

②、弹出来的界面，选中 JUnit，点击 next

 ③、选中 Junit 的版本，一般我们都用 4.0 以上的。点击 Finish

**4 我们先看下面这个例子，看一下 Junit 的用法**

**（1）编写代码（需要测试的类）**

25 public class Calculator {

    public int add(int a,int b){

        return a+b;

    }

    public int sub(int a,int b){

        return a-b;

    }

}

**（2）编写测试类（不用Junit）**

18 public class CalculatorTest {

    public static void main(String[] args) {

        Calculator c = new Calculator();

        //测试 add()方法

        int result = c.add(1, 2);

        if(result == 3){

            System.out.println("add()方法正确");

        }

        //测试 sub()方法

        int result2 = c.sub(2, 1);

        if(result2 == 1){

            System.out.println("sub()方法正确");

        }

    }

}

那么我们可以看到，不用 Junit 只能写在 main（）方法中，通过运行结果来判断测试结果是否正确。这里需要测试的只有两个方法，如果有很多方法，那么测试代码就会变得很混乱。

**（3）编写测试类（用Junit）**

public class CalculatorTest {

    @Test

    //测试 add()方法

    public void testAdd(){

        Calculator c = new Calculator();

        int result = c.add(1, 2);

        Assert.assertEquals(result, 3);

    }

    @Test

    //测试 sub()方法

    public void testSub(){

        Calculator c = new Calculator();

        int result = c.sub(2, 1);

        Assert.assertEquals(result, 1);

    }

}

如何运行 Junit呢？鼠标放在需要测试的方法中，右键，Run As ---->JUnit Test。结果出现如下的绿色横条，则测试通过，红色横条，则测试失败。那么由上面可以看到，使用 Junit 不需要创建 main() 方法，而且每个测试方法一一对应，逻辑特别清晰。可能有读者会问，这样写代码量也并不会减少啊，那么你接着往下看：

**5 Junit 的几种注解**

（1）@Test: 测试方法

（2）@Ignore: 被忽略的测试方法：加上之后，暂时不运行此段代码

（3）@Before: 每一个测试方法之前运行

（4）@After: 每一个测试方法之后运行

（6）@BeforeClass: 方法必须必须要是静态方法（static 声明），所有测试开始之前运行，注意区分before，是所有测试方法

（7）@AfterClass: 方法必须要是静态方法（static 声明），所有测试结束之后运行，注意区分 @After

那么上面的例子，我们可以看到，每个 @Test 方法中都有 Calculator c = new Calculator();即类的实例化，那么我们可以将其放入到 @Before 中

**6 综合例子**

30 public class CalculatorTest {

    Calculator c = null;

    @Before

    public void testBeforeClass(){

        c = new Calculator();

    }

    @Test

    //测试 add()方法

    public void testAdd(){

        int result = c.add(1, 2);

        //Assert.assertEquals(result, 3);

        //等价于：

        if(result == 3){

            System.out.println("add()方法正确");

        }

    }

    @Test

    //测试 sub()方法

    public void testSub(){

        int result = c.sub(2, 1);

        //Assert.assertEquals(result, 1);

        //等价于：

        if(result == 1){

            System.out.println("sub()方法正确");

        }

    }

}

**7 同理：别的注解用法我们用一个类来看：**

35 public class JunitTest {

    public JunitTest() {

        System.out.println("构造函数");

    }

    @BeforeClass

    public static void beforeClass(){

        System.out.println("@BeforeClass");

    }

    @Before

    public void befor(){

        System.out.println("@Before");

    }

    @Test

    public void test(){

        System.out.println("@Test");

    }

    @Ignore

    public void ignore(){

        System.out.println("@Ignore");

    }

    @After

    public void after(){

        System.out.println("@After");

    }

    @AfterClass

    public static void afterClass(){

        System.out.println("@AfterClass");

    }

}

结果为：

**8 @BeforeClass**

构造函数

@Before

@Test

@After

@AfterClass

注意：编写测试类的原则：

**①测试方法上必须使用@Test进行修饰**

**②测试方法必须使用public void 进行修饰，不能带任何的参数**

**③新建一个源代码目录来存放我们的测试代码，即将测试代码和项目业务代码分开**

**④测试类所在的包名应该和被测试类所在的包名保持一致**

**⑤测试单元中的每个方法必须可以独立测试，测试方法间不能有任何的依赖**

**⑥测试类使用Test作为类名的后缀（不是必须）**

**⑦测试方法使用test作为方法名的前缀（不是必须）**

## 四、在Spring+Maven+Junit Test

**1 以前我是很讨厌写测试代码的，**总觉得测试用例是测试人员写的，现在想想自己真是Too yuong too simple，接触开发多了之后发现在开发中需要不断通过测试来发现某些路子的可行性，如果不写测试代码直接一股脑写下去，很可能代码写完了一运行发现到处都是坑。下面总结一下在SpringMVC和SpringBoot环境中的单元测试（Junit Test）的搭建和使用。

SpringMVC中的单元测试

**2 加入依赖包**

首先需要在pom.xml中加入junit依赖，其中scope限定了junit包的使用范围是test环境：

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<!-- 表示开发的时候引入，发布的时候不会加载此包 -->

<scope>test</scope></dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>4.0.2.RELEASE</version>

<scope>test</scope></dependency>

下图是我在IntelliJ IDEA中的项目包的架构，我们需要在test包下进行单元测试编写。

**3 编写测试基类**

由于我们在单元测试中可能需要对SpringMVC中的service、dao层进行调用，但是我们又知道在Spring中对象的创建时交给容器的，不需要我们手动创建、实例化对象，那么就需要引入配置文件进行上下文的初始化。所以为了避免每次写一个测试类都要进行相关的配置操作，我们直接写一个父类，然后每个测试类继承该类就可以达到简化的目的了。

BaseTest.java如下，其中@Before和@After注解都是junit提供的，其含义写在代码的注释中了：

@RunWith(SpringJUnit4ClassRunner.class)@ContextConfiguration({"classpath:spring-mvc.xml","classpath:spring-mybatis.xml"})public class BaseTest {

@Before

public void init() {

//在运行测试之前的业务代码

}

@After

public void after() {

//在测试完成之后的业务代码

}

}

**4 编写具体测试类**

写完测试基类后，就可以开始写真正的单元测试了，下面是一个简单的示例：

public class HelloTest extends BaseTest {

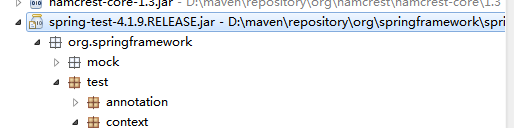
@Test

public void getTicketInfo() {

System.out.println("hello");

}

**5 我们可以看到在IDEA中可以直接对单元测试的某个方法进行运行，不用编译项目和启动服务器就可以达到对业务代码的功能测试，可见其便利程度。**



**package** com.yunnex.junit;

**import** org.junit.After;

**import** org.junit.Before;

**import** org.junit.runner.RunWith;

**import** org.springframework.test.context.ActiveProfiles;

**import** org.springframework.test.context.ContextConfiguration;

**import** org.springframework.test.context.junit4.AbstractTransactionalJUnit4SpringContextTests;

**import** org.springframework.test.context.transaction.TransactionConfiguration;

**import** org.springframework.test.context.web.WebAppConfiguration;

**import** com.yunnex.ops.erp.modules.sys.constant.SysConstant;

@ActiveProfiles(SysConstant.SystemEnvironment.***PROFLIE\_TEST***)

@RunWith(Junit4ClassRunner.**class**)

@WebAppConfiguration

@TransactionConfiguration(transactionManager = "transactionManager", **defaultRollback = true**)

@ContextConfiguration(value = {"classpath:config/spring/spring-context-env.xml", "classpath:config/spring/spring-context.xml", "classpath:config/spring/spring-context-shiro.xml", "classpath:config/spring/spring-context-activiti.xml", "classpath:config/spring/spring-context-ldap.xml", "classpath:config/spring/spring-context-jedis.xml"})

**public** **class** BaseTest **extends** AbstractTransactionalJUnit4SpringContextTests

{

@Before

**public** **void** before(){}

@After

**public** **void** after(){}

}

**package** com.yunnex.junit;

**import** org.activiti.engine.FormService;

**import** org.activiti.engine.HistoryService;

**import** org.activiti.engine.ProcessEngine;

**import** org.activiti.engine.RepositoryService;

**import** org.activiti.engine.RuntimeService;

**import** org.activiti.engine.TaskService;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** com.yunnex.ops.erp.modules.workflow.flow.service.WorkFlowMonitorService;

**public** **class** BaseActivitiTest **extends** BaseTest

{

@Autowired

**protected** ProcessEngine processEngine;

@Autowired

**protected** RepositoryService repositoryService;

@Autowired

**protected** RuntimeService runtimeService;

@Autowired

**protected** TaskService taskService;

@Autowired

**protected** HistoryService historyService;

@Autowired

**protected** FormService formService;

@Autowired

WorkFlowMonitorService workFlowMonitorService;

}

**public** **class** LeaveWorkflowServiceTest **extends** BaseActivitiTest

{

//业务key

String businessKey = "10006";

//流程定义key

String processDefineKey = "leave\_process";

String assingee = "wade";

String assingeeKey = "assingee";

**public** **class** LeaveWorkflowServiceTest **extends** BaseActivitiTest

{

String businessKey = "10006";

String processDefineKey = "leave\_process";

String assingee = "wade";

String assingeeKey = "assingee";

@Test

**public** **void** testJump( ) **throws** Exception

{

workFlowMonitorService.deploymentProcess("act/designs/oa/leave/leave.bpmn");

Map<String, Object> variables = Maps.*newHashMap*();

variables.put(assingeeKey, assingee);

workFlowMonitorService.startProcess(processDefineKey, businessKey, variables);

//A1--终止 OK

workFlowMonitorService.endProcess("36db492e001946aea5fa83ee36136817");

//A1--A6 OK

workFlowMonitorService.jump("dd2fd5821b6744c38dc328bbd732053d",**null**,"A6");

//A1--A5(并发任务) OK

workFlowMonitorService.jump("5ca2fd8c4b1f4c0ea2ce8d1357611ab4",**null**,"A5");

//A4|A5--终止 OK

workFlowMonitorService.endProcess("5ca2fd8c4b1f4c0ea2ce8d1357611ab4");

// 获取所有节点 OK

List<ActivityImpl> list = workFlowMonitorService.getProcessNodes("2ef52cd066e540fe9665dc707df7c2cb");

**for**(ActivityImpl activityImpl:list)

{

System.***out***.println(activityImpl.getId());

System.***out***.println(activityImpl.getProperties().get("name"));

}

// 获取所有任务

List<Task> tasks = workFlowMonitorService.getCurrentTasks("5ca2fd8c4b1f4c0ea2ce8d1357611ab4");

//获取子流程任务

tasks = workFlowMonitorService.getCurrentSubProcessTasks("33eed5dfdb3344d3864f9ab29de1daa8");

completeTask();

}

@Ignore

**public** **void** completeTask( )

{

Task task = taskService.createTaskQuery().active().processInstanceId("36db492e001946aea5fa83ee36136817").taskAssignee(assingee).list().get(0);

Map<String, Object> variables = Maps.*newHashMap*();

variables.put(assingeeKey, assingee);

variables.put("way", "A3");

taskService.complete(task.getId(), variables);

}

}

**6 SpringBoot中的单元测试**

SpringBoot中使用Junit和SpringMVC基本类似，只需要改动一些配置即可。

**（1）加入依赖**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-test</artifactId>

<version>1.5.2.RELEASE</version>

<scope>test</scope></dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>4.3.7.RELEASE</version>

<scope>test</scope></dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

**（2）编写测试基类**

@RunWith(SpringRunner.class)

@SpringBootTest

public class BaseTest {

@Before

public void init() {

}

@After

public void after() {

}

}