Junit

Junit就是测试工具，为什么需要测试，这是为了减少后期维护代码，检测代码的的时间

Junit4的快速入门
Junit4的使用详解
Junit4的深入使用
Junit4的在web中应用的
Junit是Xunit下的一个产品，
同时Xunit也有很多其他语言的测试框架
包括PythodUnit
CppUnit
同时还有一个hamcreat.jar来增强Junit.jar的测试功能的jar
断言机制：将程序预期的结果与程序运行的最终结果进行比对，确保对结果的可预知性
hamcrest-core设置匹配集规则的框架，可用来增强junit的功能
JUnit3：必须继承junit.framework.TestCase这个类，在方法前面必须加上test最为前缀
JUnit4：只要加上@Test注解即可，不需要继承任何类，命名没有限制

Junit可以帮助我们进行有目的的测试，能帮助我们最大限度的避免代码的BUG
Junit是使用断言机制可以直接

我们这里是可以使用断言机制的，关于断言的使用 在maven中是默认开启的
关于断言机制我们需要引入一个包
就是

```
import static org.junit.Assert.*;
```

把要被测试的代码与我们测试的类放在同一个包下是不好的
所以我们需要一个特殊的包来存放这些特殊用来测试的类 这样保证了测试代码的分离，
这样查看起来又容易看

右键要测试的类，选择“other”，输入junit，选择junit test case，选择测试存档的包，然后自动生成还测试类的方法进行测试
这是Eclipse 的快速生成测试类的方法
我在Idea中安装了一个关于Junit的插件，可以快速生成junit4的测试方法

测试方法上必须使用@Test进行修饰
测试方法必须使用public void进行修饰，不能带任何参数
要新建一个源代码目录来存放我们的测试代码
测试类的包应该与被测试的包保持一致
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
测试类使用test作为类名的后缀（不是必须） 不过这样看起来规范
测试方法使用test作为方法名的前缀（不是必须）

1.Failure一般由单元测试使用的断言方法判断失败所引起的，这将表示测试点发现了问题，就是说程序输出的结果和我们预期的不一样。
2.error是由代码异常引起的，它可以产生于测试代码本身的错误，也可以是被测试代码中的一个隐藏的bug
3.测试用例不是用来证明你是对的，而是用来证明你没有错。

JunitTest运行流程
beforeClass
before
test1
after

1.@BeforeClass修饰的方法会在所有方法被调用前被执行，而且该方法是静态的，所以当测试类被加载后接着就会运行它，而且在内存中它只会存在一份实例，它**比较适合加载配置文件**。
2.@AfterClass所修饰的方法通常用来对资源的清理，如关闭数据库的连接
3.**@Before和@After会在每个测试方法的前后各执行一次。**

Junit4的常用注解

@Test:将一个方法修饰成测试方法
@Test(expected=xxx（异常类）.class)：会抛出该异常
@Test(timeout=毫秒)：设置执行的时间，用于结束死循环或是性能测试
@BeforeClass：所有方法之前执行，且执行一次 static修饰
@AfterClass：所有方法之后执行 static修饰
@Before：每一个测试方法之前执行
@After：每一个测试方法后执行
@Ignore:所修饰的测试方法会被测试运行器忽略
@RunWith:可以更改测试运行器，通过继承org.junit.runner.Runner这个类来写自己的运行器

测试套件

```
@RunWith(Suite.class)
@Suite.SuiteClasses({TaskTest1.class,TaskTest2.class,TaskTest3.class})
public class SuitTest {
```

1.测试套件就是组织测试类一起运行的
2.写一个作为测试套件的入口类，这个类里不包含其他的方法。
3.更改测试运行器Suite.class.
4.将要测试的类作为数组传入到Suite.SuiteClasses({})

JUnit参数化设计
参数的测试，就是将一个方法的多组测试用例一起测试
1.更改默认的测试运行器为RunWith(Parameterized.class)
2.声明变量存放预期值和结果值
3.声明一个返回值为Collection的公共静态方法，并使用@Parameters进行修饰

```
import java.util.Arrays;
import java.util.Collection;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
import static org.junit.Assert.assertEquals;

/**
 * Created by han on 2016/8/23.
 */

@RunWith(Parameterized.class)
public class ParameterTest {
int expected = 0;
int input1 = 0;
int input2 = 0;

@Parameters
public static Collection<Object[]> t() {
return Arrays.asList(new Object[][]{
                {3,1,2},
                {4,2,2}
        }) ;
    }


public ParameterTest(int expected, int input1, int input2) {
this.expected = expected;
this.input1 = input1;
this.input2 = input2;
    }

@Test
public void testAdd() {
assertEquals(expected, new Calculate().add(input1, input2));
    }
}
```

Spring与Hibernate的整合测试

1，添加spring，hibernate，MySQL等jar包
2，添加spring配置文件，hibernate配置文件
3，@BeforeClass获得spring的配置文件ClassPathXmlApplicationContext("配置文件")
4，测试通过getBean获得spring管理的bean是否成功。