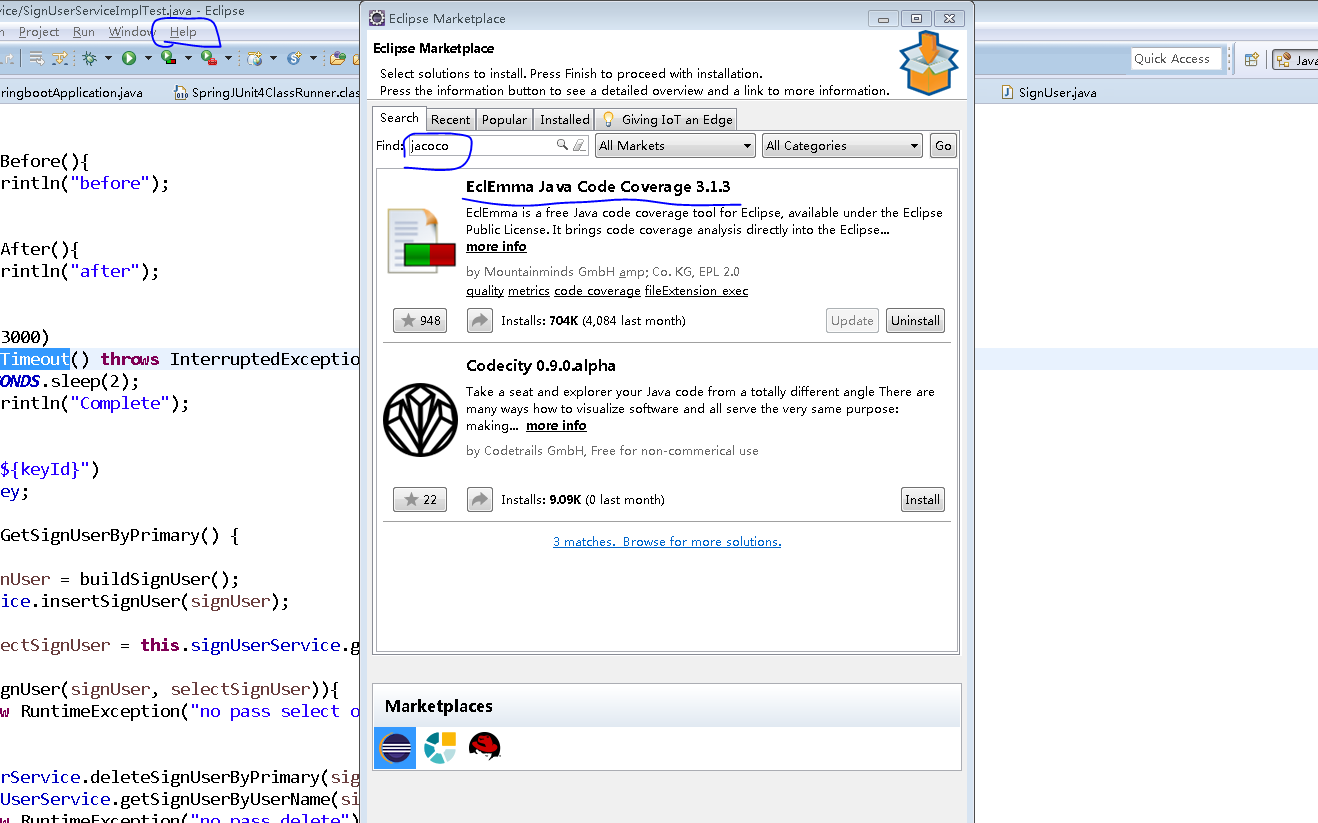
1 maven工程， jacoco插件， springboot工程

## eclipse配置

### 安装jacoco插件

点击help，选择eclipse MarketPlace



## 项目配置junit

### pom添加依赖

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

### Junit案例

@SpringBootTest(classes = {SpringbootApplication.**class**})

//@RunWith(SpringRunner.class)

@RunWith(SpringJUnit4ClassRunner.**class**)

@Transactional

JUnit中的注解

@BeforeClass：针对所有测试，只执行一次，且必须为static void

@Before：初始化方法，执行当前测试类的每个测试方法前执行。

@Test：测试方法，在这里可以测试期望异常和超时时间

@After：释放资源，执行当前测试类的每个测试方法后执行

@AfterClass：针对所有测试，只执行一次，且必须为static void

@RunWith:可以更改测试运行器 ，缺省值 org.junit.runner.Runner

一个单元测试类执行顺序为：

@BeforeClass –> @Before –> @Test –> @After –> @AfterClass

每一个测试方法的调用顺序为：

@Before –> @Test –> @After

超时测试

如果一个测试用例比起指定的毫秒数花费了更多的时间，那么 Junit 将自动将它标记为失败。timeout 参数和 @Test注释一起使用。现在让我们看看活动中的 @test(timeout)。

@Test(timeout = 1000)

public void testTimeout() throws InterruptedException {

TimeUnit.SECONDS.sleep(2);

System.out.println("Complete");

}

测试私有方法com.h3c.sign.service.SignServiceTest

//通过反射测试私有方法

// @Test(expected=Exception.class)

@Test(expected=NullSignImportException.class)

// @Test

public void testCheck() throws NoSuchMethodException, SecurityException, IllegalAccessException, IllegalArgumentException, InvocationTargetException{

try {

SignImportInfo signImportInfo = new SignImportInfo();

signImportInfo.setVersion("1");

Class<SignServiceImpl> impl = SignServiceImpl.class;

Method method = impl.getDeclaredMethod("check", SignImportInfo.class);

method.setAccessible(true);

method.invoke(signServiceImpl,signImportInfo);

//反射方法是InvocationTargetException 需要e.getTargetException() 获取真实异常

} catch (InvocationTargetException e) {

if(e.getTargetException() instanceof NullSignImportException){

throw new NullSignImportException(e.getTargetException().getMessage());

}

}

}

测试私有方法需要里面如果嵌套层级太多，由于spirng的依赖注入可能有的属性为空，这种情况可以通过自己构造适合的参数通过测试公共方法覆盖相关的私有方法

## 使用经验

对于重要的模块一定单元测试

对于特别简单的方法可以不进行单元测试

单元测试粒度要把握一下

通常测试进行\*\*\*service层的公共方法即可，

\*\*\*serviceImpl中重要的私有方法也要测试

单元测试尽量不要和API测试重复了，避免资源浪费

写代码的时候就写单元测试

代码每个函数功能最好独立，行数少点，如果一个函数的代码太长了，单元测试根本没法进行，可以提升代码质量

单元测试可以造数据，比如多个团队协作的项目，有时候只规定了数据格式，没有相关数据，写单元测试的同时就可以造相关的数据

防伪签名本地签名服务器就是如此。

工具类可以进行单元测试，不用在main方法中专门测试了，大家一般习惯于在工具类中写个main方法测试。

不要为了单元测试而单元测试

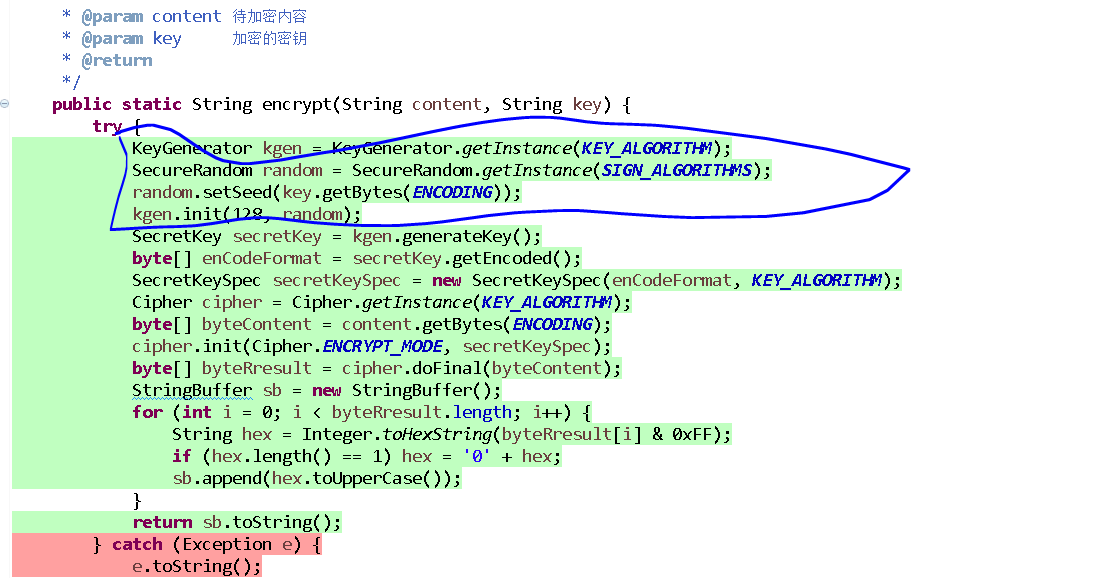
单元测试和写代码一样，要规范起来，要么后期维护单元测试成本也会很高

## 遇到的坑

Java ASS加密方法由于setSeed()的问题，在windows平台都可以，但是在linux下面就出问题，这种情况如果集成自动化就是相当有用的，问题可以提前暴露出来

com.h3c.sign.util.SecurityUtils

<SecurityUtils.java>



Mysql时区导致的问题，插入的时间和查询的时间差了8小时

这个需要设置mysql时区，配置文件或者命令

数据乱码问题