JUnit 实验手册（2）

第一部分 测试驱动开发（Test driven development, TDD）

一、相关理论

1. 传统的开发流程：编码->测试
测试驱动的开发（TDD）：测试->编码

2. **黑盒测试**和**白盒测试**的概念

3. **覆盖率**是单元测试质量的一个重要指标，覆盖率一方面跟单元测试的完备有关，另一方面跟被测方法的内聚有关。如果一个方法内聚性不好，就会存在很多分支，会增大测试的冗余工作量。所以从测试的角度来说，每个方法的设计也应该要高内聚。举例

```
addUser(){
    // 验证用户是否有增加用户的权限

    // 判断用户是否已存在

    // 将用户加入到数据库
}
```

上面的方法实现中，可以提取出：

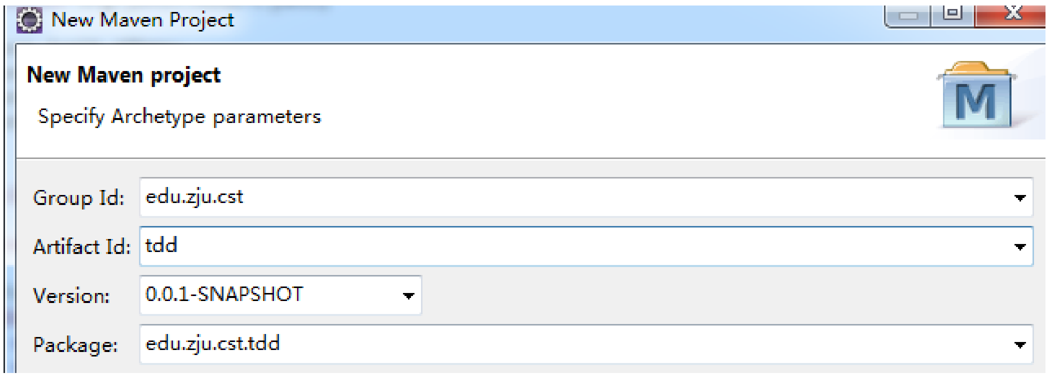
```
boolean authUser()
boolean checkUserExist()
```

4. 编写单元测试类时，一开始每个测试方法的默认实现逻辑：

```
@Test
public void testAddUser(){
    fail("Not implemented.");
}
```

二、案例演示

1. 创建一个项目，比如 tddproject


New Maven Project

New Maven project

Specify Archetype parameters

Group Id: edu.zju.cst

Artifact Id: tdd

Version: 0.0.1-SNAPSHOT

Package: edu.zju.cst.tdd

2. 创建一个 User 类，比如 edu.zju.cst.tdd.model.User
添加若干属性、getter/setter 方法及构造方法

3. 创建一个 IUserService 接口，比如 edu.zju.cst.tdd.service.IUserService
添加若干方法，比如：

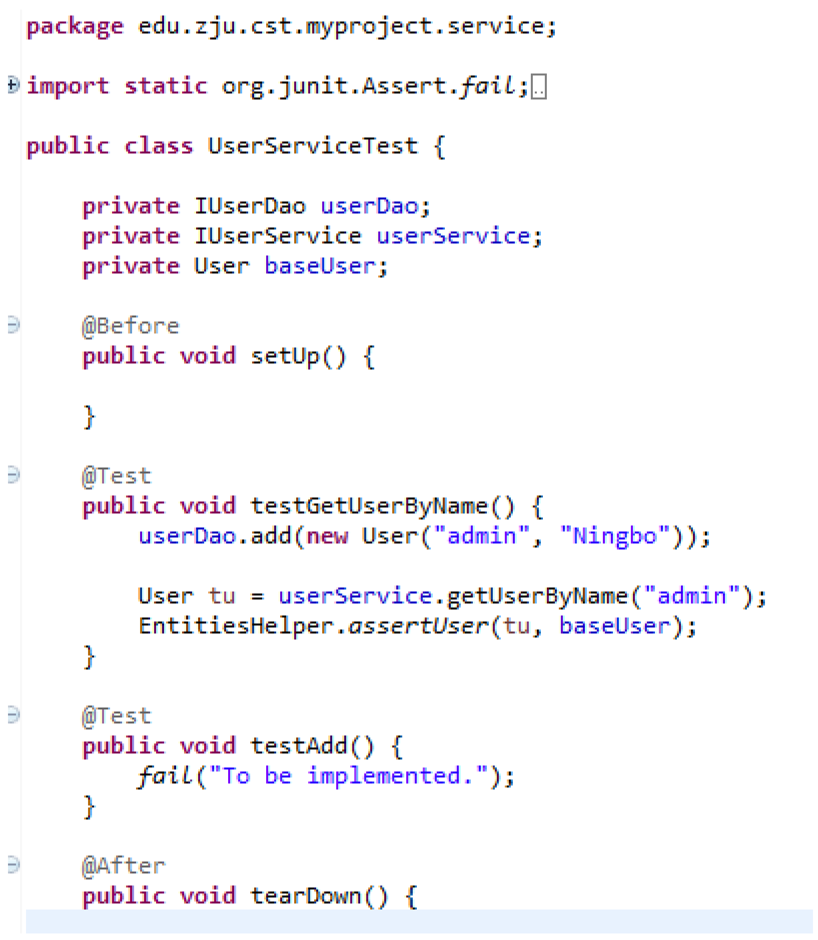
```
public interface IUserService {
    public void addUser(User user);
    public User getUser(String name);
    public void deleteUser(User user);
}
```

4. 创建一个 UserService 类，先不用实现代码。

5. 创建一个单元测试类，比如 edu.zju.cst.tdd.service.UserServiceTest

6. 在 UserServiceTest 中添加测试方法骨架。

7. 实现各个单元测试方法



```java
package edu.zju.cst.myproject.service;

import static org.junit.Assert.fail;

public class UserServiceTest {

    private IUserDao userDao;
    private IUserService userService;
    private User baseUser;

    @Before
    public void setUp() {

    }

    @Test
    public void testGetUserByName() {
        userDao.add(new User("admin", "Ningbo"));

        User tu = userService.getUserByName("admin");
        EntitiesHelper.assertUser(tu, baseUser);
    }

    @Test
    public void testAdd() {
        fail("To be implemented.");
    }

    @After
    public void tearDown() {
```

7. 运行单元测试类。这个时候当然是全部都通不过的。

8. 然后就可以开发 UserService 类了，只有当所有的单元测试都可以通过时，UserService 类才算完成。

第二部分 hamcrest

一、相关理论

1. Provides a library of matcher objects (also known as constraints or predicates) allowing 'match' rules to be defined declaratively, to be used in other frameworks. Typical scenarios include testing frameworks, mocking libraries and UI validation rules.

官方网站：http://hamcrest.org/JavaHamcrest/
Tutorial: https://code.google.com/p/hamcrest/wiki/Tutorial
简单地说，hamcrest 可以用来增强 JUnit 中的 assert 功能。

2. 要使用 JUnit 中的 assertThat 来进行断言

第一个参数表示实际值，第二个参数表示 hamcrest 的表达式

3. A tour of common matchers

Hamcrest comes with a library of useful matchers. Here are some of the most important ones.

- Core 核心
  - **anything** - always matches, useful if you don't care what the object under test is
  - **describedAs** - decorator to adding custom failure description
  - **is** - decorator to improve readability
- Logical 逻辑
  - **allOf** - matches if all matchers match, short circuits (like Java &&)
  - **anyOf** - matches if any matchers match, short circuits (like Java ||)
  - **not** - matches if the wrapped matcher doesn't match and vice versa
- Object 对象
  - **equalTo** - test object equality using Object.equals
  - **hasToString** - test Object.toString
  - **instanceOf**, isCompatibleType - test type
  - **notNullValue**, nullValue - test for null
  - **sameInstance** - test object identity
- Beans
  - **hasProperty** - test JavaBeans properties
- Collections 集合
  - **array** - test an array's elements against an array of matchers
  - **hasEntry**, hasKey, hasValue - test a map contains an entry, key or value
  - **hasItem**, hasItems - test a collection contains elements
  - **hasItemInArray** - test an array contains an element
- Number 数字
  - **closeTo** - test floating point values are close to a given value
  - **greaterThan**, **greaterThanOrEqualTo**, **lessThan**, **lessThanOrEqualTo** - test ordering
- Text 文本
  - **equalToIgnoringCase** - test string equality ignoring case
  - **equalToIgnoringWhiteSpace** - test string equality ignoring differences in runs of whitespace
  - **containsString**, **endsWith**, **startsWith** - test string matching

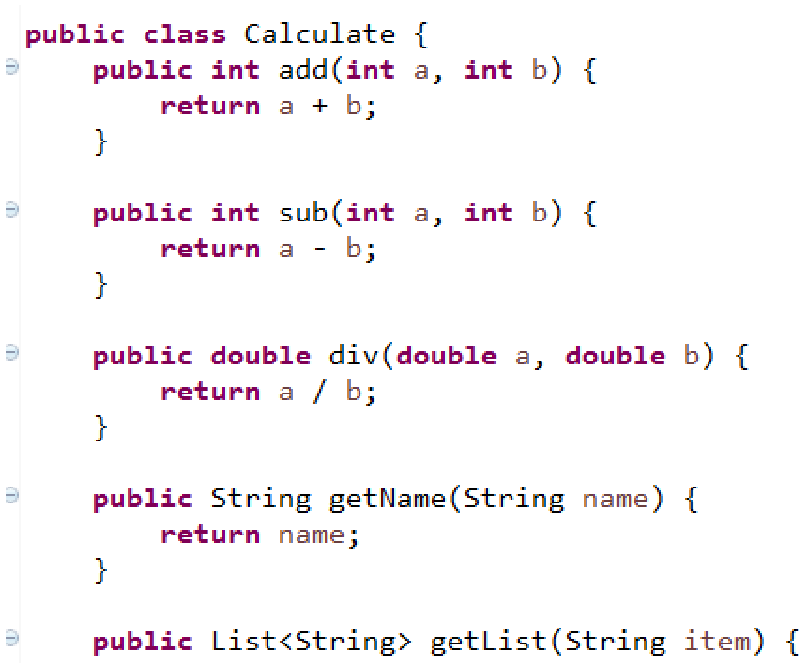
4. maven 格式

```
<dependency>
```

```
            <groupId>org.hamcrest</groupId>
            <artifactId>hamcrest-all</artifactId>
            <version>1.3</version>
        </dependency>
```

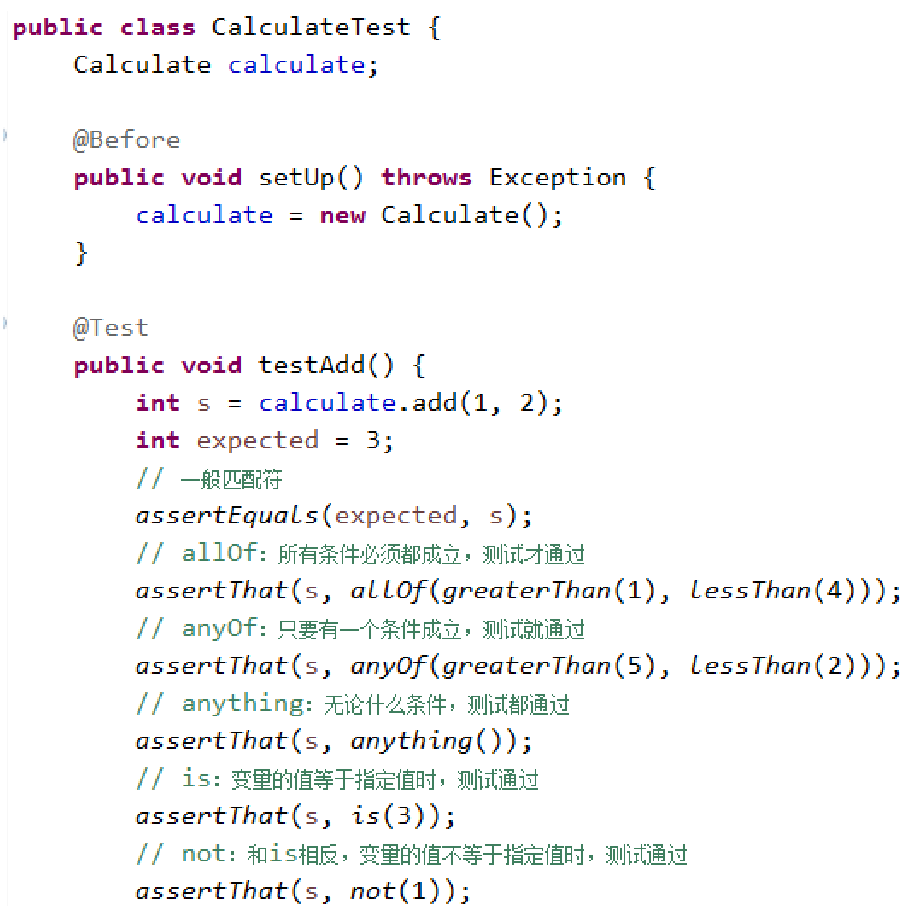
二、实际使用

1. 被测代码：



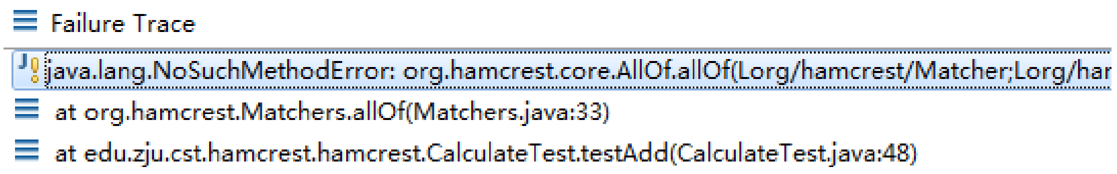
```
public class Calculate {
    public int add(int a, int b) {
        return a + b;
    }

    public int sub(int a, int b) {
        return a - b;
    }

    public double div(double a, double b) {
        return a / b;
    }

    public String getName(String name) {
        return name;
    }

    public List<String> getList(String item) {
```

测试代码：



```java
public class CalculateTest {
    Calculate calculate;

    @Before
    public void setUp() throws Exception {
        calculate = new Calculate();
    }

    @Test
    public void testAdd() {
        int s = calculate.add(1, 2);
        int expected = 3;
        // 一般匹配符
        assertEquals(expected, s);
        // allOf: 所有条件必须都成立，测试才通过
        assertThat(s, allOf(greaterThan(1), lessThan(4)));
        // anyOf: 只要有一个条件成立，测试就通过
        assertThat(s, anyOf(greaterThan(5), lessThan(2)));
        // anything: 无论什么条件，测试都通过
        assertThat(s, anything());
        // is: 变量的值等于指定值时，测试通过
        assertThat(s, is(3));
        // not: 和is相反，变量的值不等于指定值时，测试通过
        assertThat(s, not(1));
```

2. 特别注意：如果使用 JUnit4.10，必须把 hamcrest 的 jar 包移到 JUnit 的 jar 之前，否则组合条件 allOf, anyOf 等会有冲突

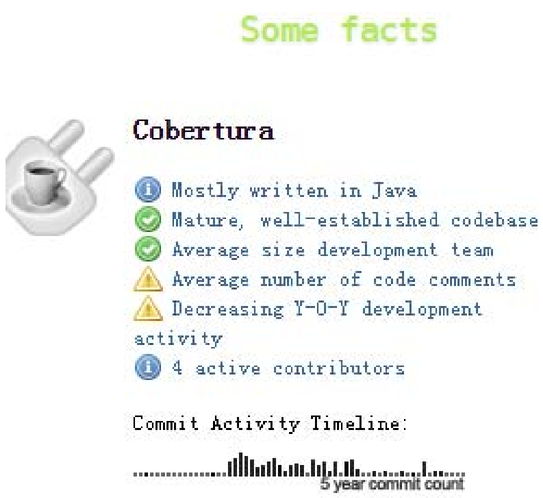


```
Failure Trace
java.lang.NoSuchMethodError: org.hamcrest.core.AllOf.allOf(Lorg/hamcrest/Matcher;Lorg/har
at org.hamcrest.Matchers.allOf(Matchers.java:33)
at edu.zju.cst.hamcrest.hamcrest.CalculateTest.testAdd(CalculateTest.java:48)
```

第三部分 cobertura

一、相关理论

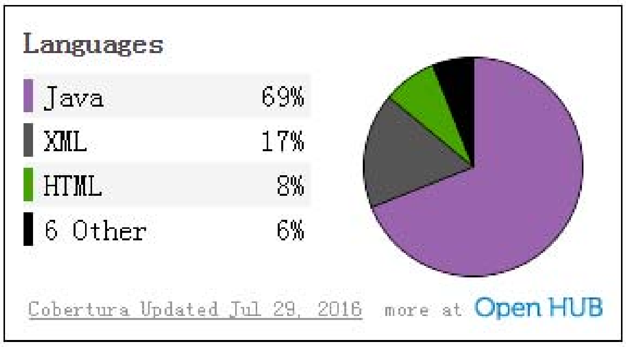
1. Cobertura is a free Java tool that calculates the percentage of code accessed by tests. It can be used to identify which parts of your Java program are lacking test coverage. It is based on jcoverage.
官方网站：http://cobertura.github.io/cobertura/


Some facts

Cobertura

Mostly written in Java
Mature, well-established codebase
Average size development team
Average number of code comments
Decreasing Y-O-Y development activity
4 active contributors

Commit Activity Timeline:


Languages

🟪	Java	69%
⬛	XML	17%
🟩	HTML	8%
⬛	6 Other	6%

Cobertura Updated Jul 29, 2016 more at Open HUB

2. 三种执行方式

- Execute via Ant
- Execute via Command Line
- Execute via Maven

二、案例演示

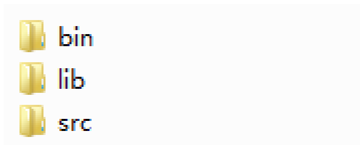
1. 下载安装 Cobertura
下载后直接解压，并将 cobertura 的目录路径设置到 path 中
下载地址：http://sourceforge.net/projects/cobertura/files/cobertura/

这里演示命令行里使用 cobertura

2. Instrumenting(生成 ser 文件)

```
cobertura-instrument.bat [--basedir dir] [--datafile file] [--auxClasspath classPath] [--destination dir] [--ignore regex] classes [...]
```

为方便起见，将要测试的源代码、字节码和相关的 jar 包拷贝到一个单独的文件夹

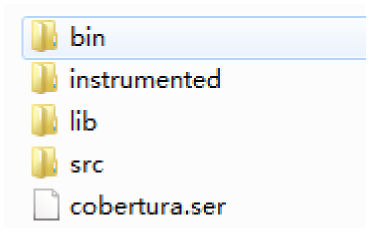


Examples:
cobertura-instrument.bat --destination instrumented bin

cobertura-instrument.bat --destination C:\MyProject\build\instrumented
C:\MyProject\build\classes

(最后一个目录是指字节码文件所在根目录)

运行完之后，目录如下所示：



3. Running Tests (基于 covertura.ser 文件运行测试)
Examples:
java -cp lib/junit-4.10.jar;lib/cobertura.jar;instrumented;bin;-Dnet.sourceforge.covertura.datafile=cobertura.ser org.junit.runner.JUnitCore edu.zju.cst.myproject.model.UserTest

java -cp C:\cobertura\lib\cobertura.jar;C:\MyProject\build\instrumented;C:\MyProject\build\classes;C:\MyProject\build\test-classes -Dnet.sourceforge.cobertura.datafile=C:\MyProject\build\cobertura.ser ASimpleTestCase

注意：文件的编码时 utf-8 的，在生成报告前需要为 covertura-report.bat 文件增加 -Dfile.encoding=utf-8

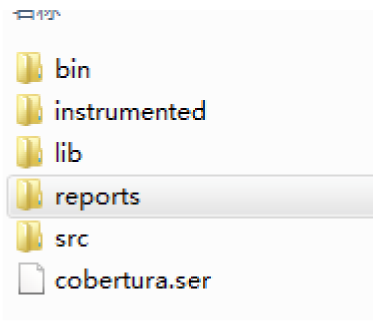
4. Reporting(基于 ser 文件生成测试覆盖率报告)
```
cobertura-report.bat [--datafile file] [--destination dir] [--format
(html|xml)] [--encoding encoding] source code directory [...] [--basedir
dir file underneath basedir ...]
```

Examples:
cobertura-report -format html --datafile cobertura.ser --destination reports src

cobertura-report.bat --format html --datafile C:\MyProject\build\cobertura.ser --destination

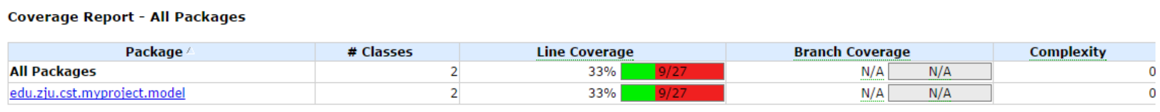
C:\MyProject\reports\coverage C:\MyProject\src

执行后目录：



bin
instrumented
lib
reports
src
cobertura.ser

覆盖率报告：



Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	2	33%	9/27	N/A	N/A	0
edu.zju.cst.myproject.model	2	33%	9/27	N/A	N/A	0

5. 练习
在 ant 里使用 cobertura：https://github.com/cobertura/cobertura/wiki/Ant-Task-Reference/
在 maven 里使用 cobertura：http://mojo.codehaus.org/cobertura-maven-plugin/

第四部分 stub & mock

一、相关理论

1. 关于测试里的 stub 和 mock

参考：
http://stackoverflow.com/questions/346372/whats-the-difference-between-faking-mocking-and-stubbing
http://martinfowler.com/articles/mocksArentStubs.html

二、案例演示

（可以重用之前的代码）

1. 创建 IUserDao 类
addUser
deleteUser
getUser

2. 创建 UserDao 类

3. 在数据库中创建表

4. 使用 JDBC 方式来连接数据库
导入 mysql-connect-java-XX.jar
引入 DBUtil.java

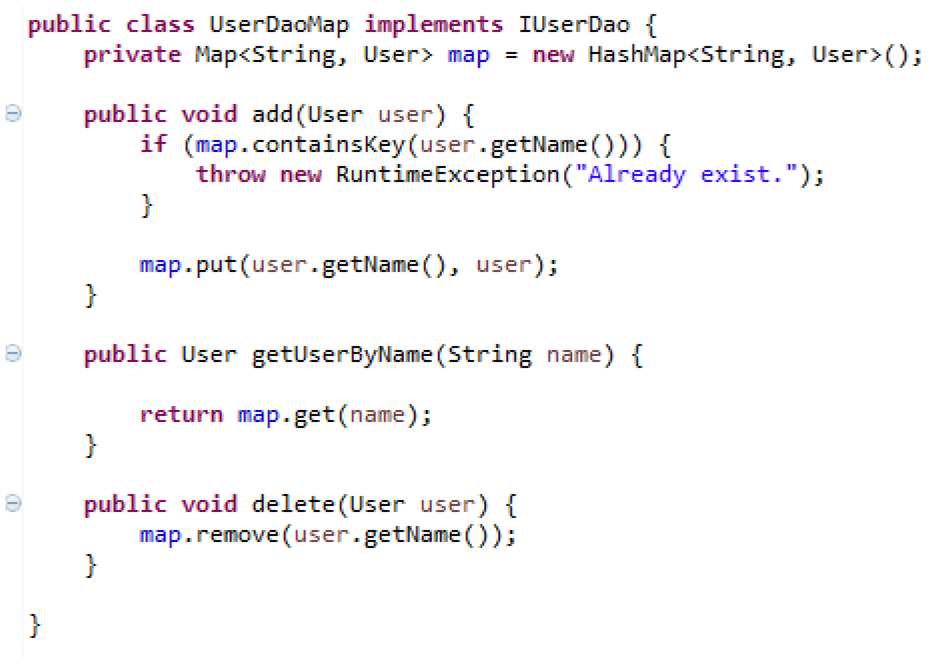
5. 编写 UserDAO

6. 创建 IUserService 类

7. 创建 UserService 类

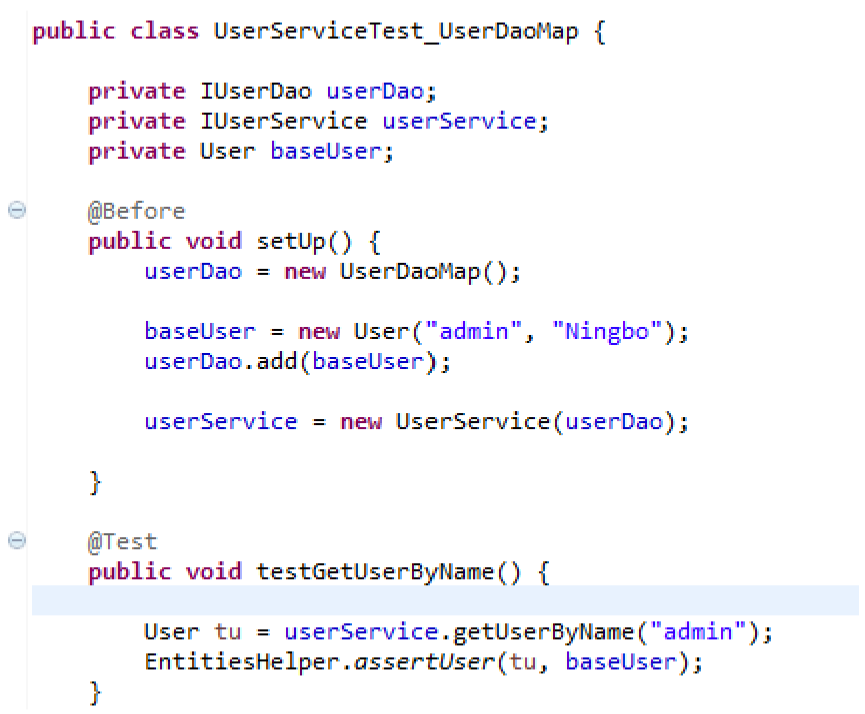
8. 运行单元测试
测试 UserService 类存在问题:(1)一个测试方法可能会影响其他单元测试方法;(2)对 service 层的测试依赖于 dao 层的实现
（要确保每个单元测试运行后不改变环境）

9. 引入 stub 类：UserDaoMap.java



```java
public class UserDaoMap implements IUserDao {
    private Map<String, User> map = new HashMap<String, User>();

    public void add(User user) {
        if (map.containsKey(user.getName())) {
            throw new RuntimeException("Already exist.");
        }

        map.put(user.getName(), user);
    }

    public User getUserByName(String name) {

        return map.get(name);
    }

    public void delete(User user) {
        map.remove(user.getName());
    }

}
```

10. 在 UserServiceTest 中使用 UserDaoMap



```java
public class UserServiceTest_UserDaoMap {

    private IUserDao userDao;
    private IUserService userService;
    private User baseUser;

    @Before
    public void setUp() {
        userDao = new UserDaoMap();

        baseUser = new User("admin", "Ningbo");
        userDao.add(baseUser);

        userService = new UserService(userDao);

    }

    @Test
    public void testGetUserByName() {

        User tu = userService.getUserByName("admin");
        EntitiesHelper.assertUser(tu, baseUser);
    }
```

第五部分 DbUnit

一、相关理论

1. DbUnit is a JUnit extension (also usable with Ant) targeted at database-driven projects that, among other things, puts your database into a known state between test runs. This is an excellent way to avoid the myriad of problems that can occur when one test case corrupts the database and causes subsequent tests to fail or exacerbate the damage.
官方网站：http://dbunit.sourceforge.net/

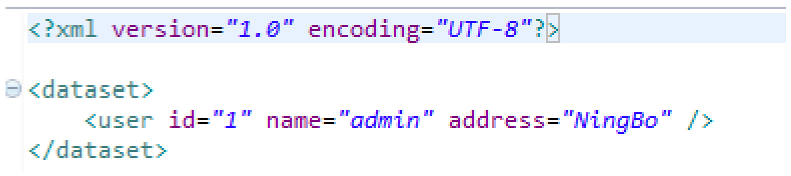
2. 其他也可以达到不影响现有数据库的方式：使用嵌入式数据库 HSQL

二、案例演示

1. 先下载 dbunit 的 jar 包，并导入到项目中

```xml
<dependency>
    <groupId>dbunit</groupId>
    <artifactId>dbunit</artifactId>
    <version>2.2</version>
</dependency>
```

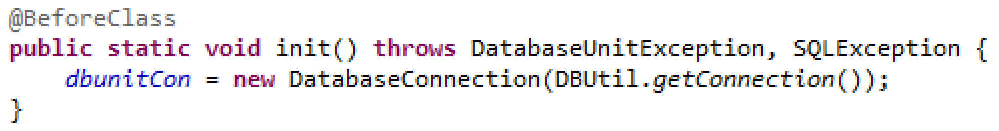
2. 创建 dbunit 的 xml 测试数据文件
放到 source folder 下面： user.xml



```xml
<?xml version="1.0" encoding="UTF-8"?>

<dataset>
    <user id="1" name="admin" address="NingBo" />
</dataset>
```

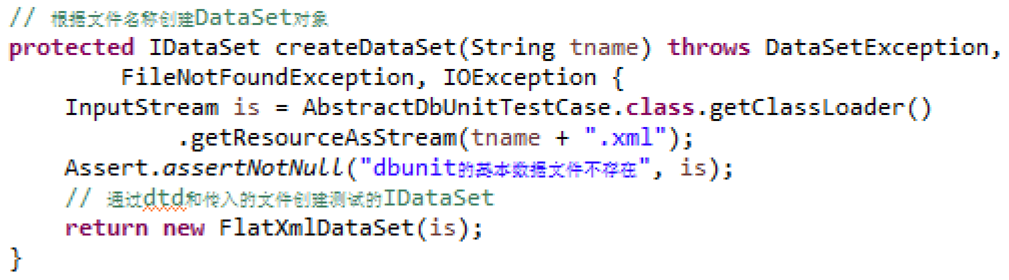
3. 引入 DbUtil.java、AbstractDbUnitTestCase.java 类
在 DbUtil.java 里设置好数据库连接信息

3. 创建 dbunit 的 connection
是用来对数据文件进行操作的，必须依赖于目前项目中所使用的 Connection



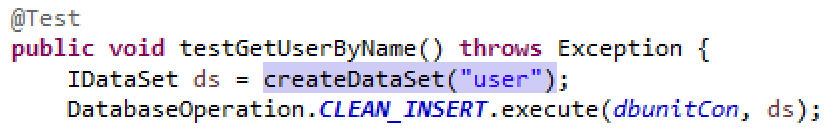
```java
@BeforeClass
public static void init() throws DatabaseUnitException, SQLException {
    dbunitCon = new DatabaseConnection(DBUtil.getConnection());
}
```

4. 创建 IDataSet，通过 Dataset 获取测试数据中的数据



```java
// 根据文件名称创建DataSet对象
protected IDataSet createDataSet(String tname) throws DataSetException,
        FileNotFoundException, IOException {
    InputStream is = AbstractDbUnitTestCase.class.getClassLoader()
            .getResourceAsStream(tname + ".xml");
    Assert.assertNotNull("dbunit的基本数据文件不存在", is);
    // 通过dtd和传入的文件创建测试的IDataSet
    return new FlatXmlDataSet(is);
}
```

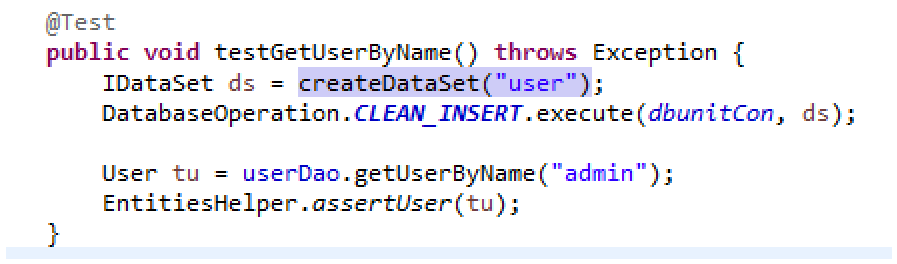
5. 每个单元测试方法开始时先初始化数据



```java
@Test
public void testGetUserByName() throws Exception {
    IDataSet ds = createDataSet("user");
    DatabaseOperation.CLEAN_INSERT.execute(dbunitCon, ds);
```

用的最多的是 CLEAN_INSERT 方法

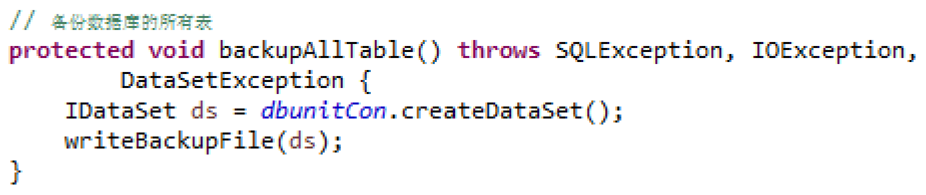
6.测试方法完整实现



```java
@Test
public void testGetUserByName() throws Exception {
    IDataSet ds = createDataSet("user");
    DatabaseOperation.CLEAN_INSERT.execute(dbunitCon, ds);

    User tu = userDao.getUserByName("admin");
    EntitiesHelper.assertUser(tu);
}
```

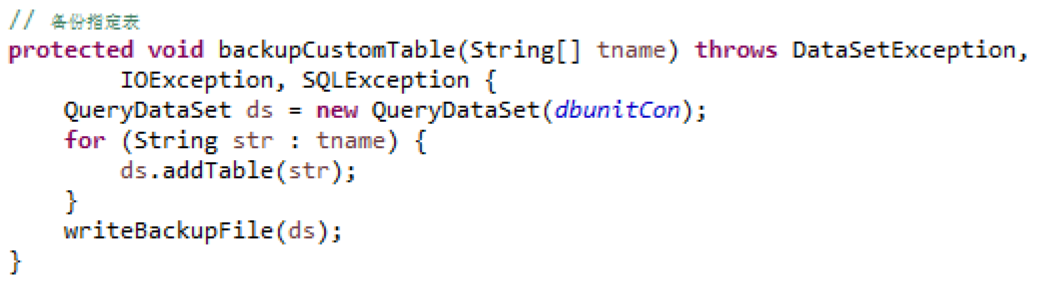
7. 运行单元测试，并查看数据库。发现数据库没有还原。

8. 备份数据
备份所有表：



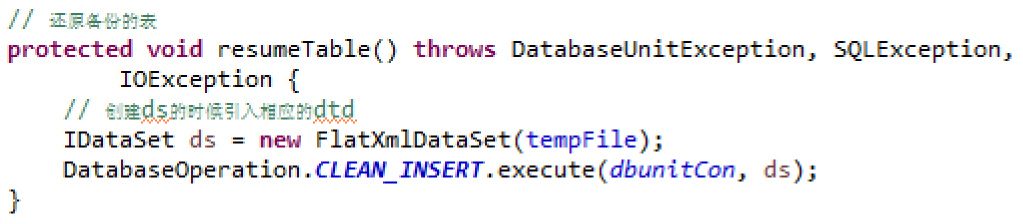
```
// 备份数据库的所有表
protected void backupAllTable() throws SQLException, IOException,
        DataSetException {
    IDataSet ds = dbunitCon.createDataSet();
    writeBackupFile(ds);
}
```

备份特定的表：



```
// 备份指定表
protected void backupCustomTable(String[] tname) throws DataSetException,
        IOException, SQLException {
    QueryDataSet ds = new QueryDataSet(dbunitCon);
    for (String str : tname) {
        ds.addTable(str);
    }
    writeBackupFile(ds);
}
```

9. 还原数据



```
// 还原备份的表
protected void resumeTable() throws DatabaseUnitException, SQLException,
        IOException {
    // 创建ds的时候引入相应的dtd
    IDataSet ds = new FlatXmlDataSet(tempFile);
    DatabaseOperation.CLEAN_INSERT.execute(dbunitCon, ds);
}
```

10. 在运行单元测试。测试完之后查看数据库，发现没有对数据库进行改变。

11. 上面的方式效率不高(比如 connection 不用每次都打开，而应该重用)，一般需要给 dbunit 写一个公共的基类 AbstractDBUnitTestCase

12. 修改单元测试类
在@BeforeClass 标注的方法里来打开 connection
在@AftercClass 标注的方法里关闭 connection

第六部分 EasyMock

一、相关理论

1. EasyMock has been the first dynamic Mock Object generator, relieving users of hand-writing Mock Objects, or generating code for them.
EasyMock provides Mock Objects by generating them on the fly using Java's proxy mechanism.

官方网站：http://easymock.org/

Getting Started: http://easymock.org/getting-started.html

2. mock 对象用来对一些未实现关联对象的类进行测试
mock 关注的是交互(主要解决对象之间的交互，诸如:service 就依赖于 DAO，如果 DAO 没有实现，可以通过 mock 来模拟 DAO 的实现)，stub 关注的是状态
EasyMock 就是实现 mock 对象的框架

3. mock 对象生命周期
三个阶段：
record（记录 mock 对象上的操作）：创建 mock 对象，并期望这个 mock 对象的方法被调用，同时给出我们希望这个方法返回的结果。
replay: 主要测试对象将被创建，之前在 record 阶段创建的相关依赖被关联到主要测试对象，然后执行被测试的方法，以模拟真实运行环境下主要测试对象的行为
verify: 验证测试的结果和交互行为

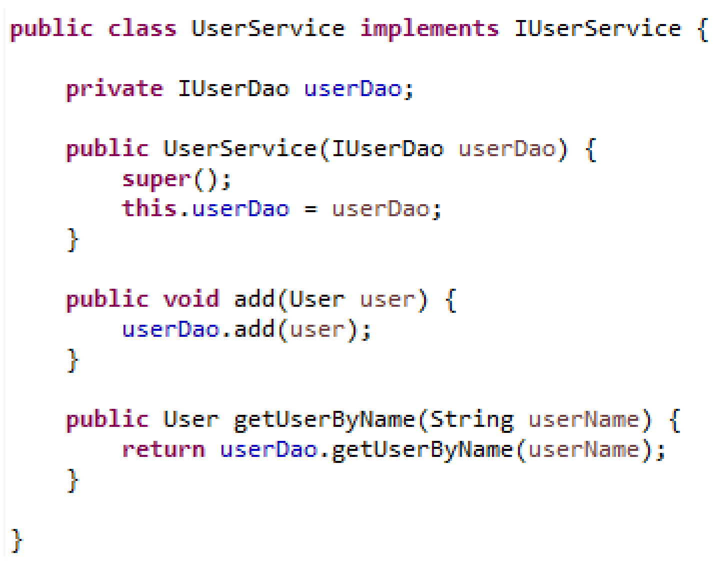
4. 应用场景举例：项目开发过程中，dao 层还没开发完成，但是 service 已经开发好，需要进行测试。这里就要引入 mock。

5. Maven 写法
```
<dependency>
    <groupId>org.easymock</groupId>
    <artifactId>easymock</artifactId>
    <version>3.1</version>
</dependency>
```

二、案例演示

基于上面的 User 项目代码。
已经创建了 IUserService, UserService, UserServiceTest

1. 确保 UserService 已经实现



```
public class UserService implements IUserService {

    private IUserDao userDao;

    public UserService(IUserDao userDao) {
        super();
        this.userDao = userDao;
    }

    public void add(User user) {
        userDao.add(user);
    }

    public User getUserByName(String userName) {
        return userDao.getUserByName(userName);
    }

}
```

2. 导入 EasyMock 的 jar 包

3. 开发 UserServiceTest 类

```
    @Before
    public void setUp() {
        // 1. 创建DAO的Mock对象，目前是record阶段
        userDao = createStrictMock(IUserDao.class);
        userService = new UserService(userDao);
        baseUser = new User("admin", "Beijing");
    }

    @Test
    public void testGetUserByName() {
        // 2. 记录ud可能会发生的操作的结果
        expect(userDao.getUserByName("admin")).andReturn(baseUser);

        // 3. 进入测试阶段,也就是replay阶段
        replay(userDao);

        User tu = userService.getUserByName("admin");
        EntitiesHelper.assertUser(tu, baseUser);
    }

    @After
    public void tearDown() {
        // 4. 验证交互关系
        verify(userDao);
    }
```

5. 被测试方法的交互有多步

（1）

```
    public User getUserByName(String userName) {
        userDao.getUserByName(userName);
        return userDao.getUserByName(userName);
    }
```

修改 EasyMock 代码

```
    expect(userDao.getUserByName("admin")).andReturn(baseUser).times(2);
```

（2）两次 load 的值不一样

```
        expect(userDao.getUserByName("admin")).andReturn(baseUser);
        expect(userDao.getUserByName("test")).andReturn(baseUser);
```

（3）调用了无返回值的方法：

```
    public User getUserByName(String userName) {
        User user = userDao.getUserByName(userName);
        userDao.delete(user);
        return userDao.getUserByName("test");
    }
```

使用 expectLastCall()方法：

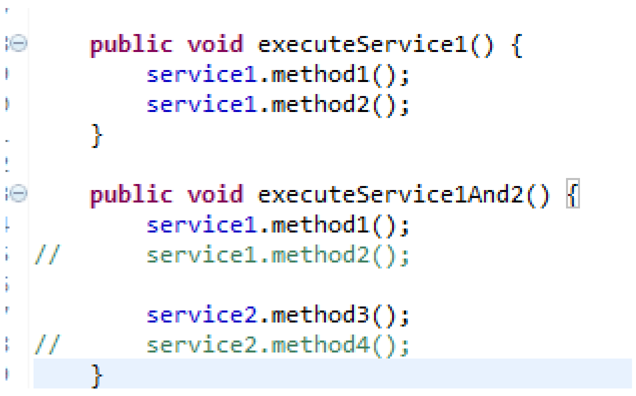
```
        expect(userDao.getUserByName("admin")).andReturn(baseUser);
        userDao.delete(baseUser);
        EasyMock.expectLastCall();
        expect(userDao.getUserByName("test")).andReturn(baseUser);
```

6. EasyMock.createMock

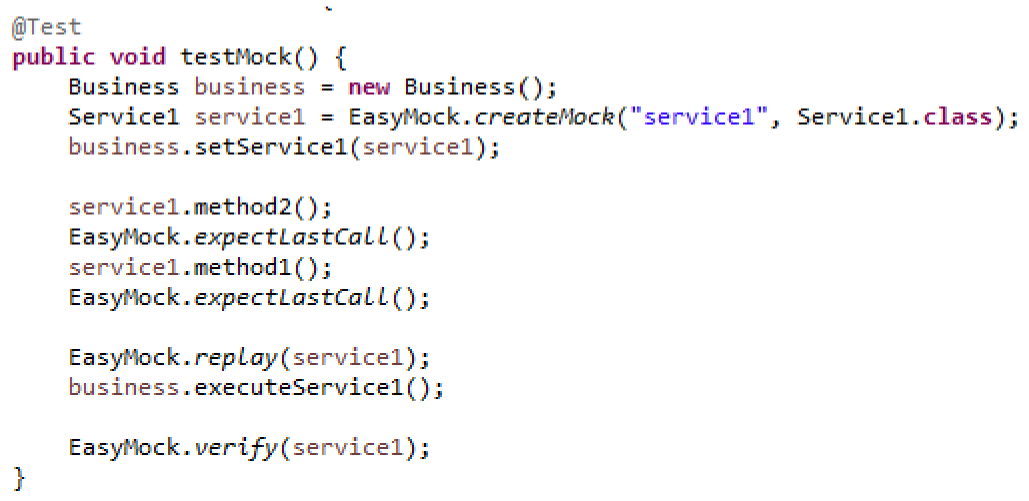
在进行 verify 的时候仅仅只是检查关联方法是否正常完成调用，如果完成次数一致，就认为测试通过。不考虑顺序问题。

EasyMock.createStrictMock

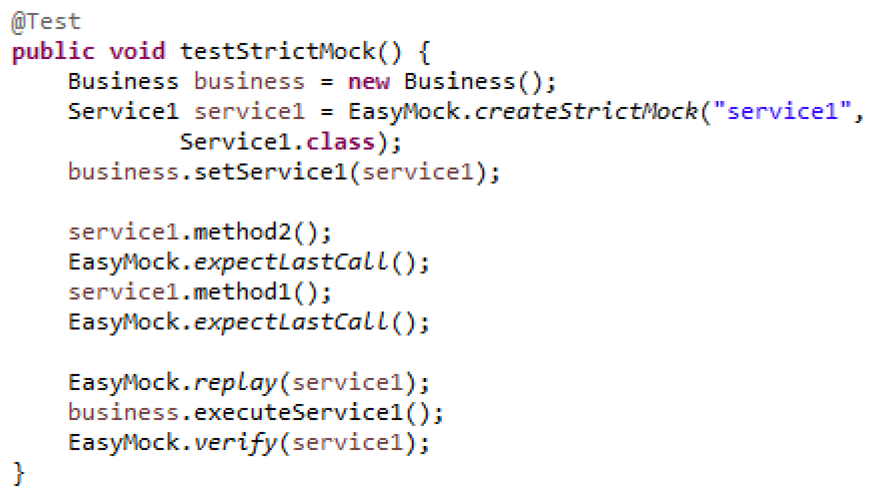
还要验证顺序



```
    public void executeService1() {
        service1.method1();
        service1.method2();
    }

    public void executeService1And2() {
        service1.method1();
//      service1.method2();

        service2.method3();
//      service2.method4();
    }
```



```java
@Test
public void testMock() {
    Business business = new Business();
    Service1 service1 = EasyMock.createMock("service1", Service1.class);
    business.setService1(service1);

    service1.method2();
    EasyMock.expectLastCall();
    service1.method1();
    EasyMock.expectLastCall();

    EasyMock.replay(service1);
    business.executeService1();

    EasyMock.verify(service1);
}
```



```java
@Test
public void testStrictMock() {
    Business business = new Business();
    Service1 service1 = EasyMock.createStrictMock("service1",
            Service1.class);
    business.setService1(service1);

    service1.method2();
    EasyMock.expectLastCall();
    service1.method1();
    EasyMock.expectLastCall();

    EasyMock.replay(service1);
    business.executeService1();
    EasyMock.verify(service1);
}
```

7. MocksControl

使用 mockControl 可以检查一组调用对象之间的关系

如果希望使用 Strict 方式，而且依赖了两个类以上，这两个依赖类应该通过 control 的方式创建

```
@Test
public void testWithStrictControlInWrongOrder() {
    Business business = new Business();
    IMocksControl mocksControl = EasyMock.createStrictControl();

    Service1 service1 = mocksControl.createMock("service1", Service1.class);
    // EasyMock.createStrictMock("service1", Service1.class);
    Service2 service2 = mocksControl.createMock("service2", Service2.class);
    // EasyMock.createStrictMock("service2", Service2.class);
    business.setService1(service1);
    business.setService2(service2);


    service2.method3();
    EasyMock.expectLastCall();
    service1.method1();
    EasyMock.expectLastCall();

    mocksControl.replay();
    business.executeService1And2();
    EasyMock.verify(service1, service2);
}
```