# 一．自动化测试环境配置步骤：

## 1. 安装 Eclipse补丁包：

该补丁包解决eclipse classpath过长的bug，补丁包在uatlib下，用补丁包替代eclipse安装目录下plugins目录下的同名文件：

org.eclipse.jdt.launching\_3.5.200.v20110105\_r362.jar -----对应eclipse 3.6 版本

org.eclipse.jdt.launching\_3.6.1.v20111006\_r372.jar -----对应eclipse 3.7 版本

## 2. TestNG Plug-in的安装

将plugin文件夹下的org.testng.eclipse\_6.4.0.20120308\_1617文件夹拷贝到eclipse安装目录的plugins目录下，启动eclipse，在类文件上右键，菜单有TestNG选项，即完成TestNG插件的安装。也可以通过TestNG官方网站进行安装（<http://testng.org/doc/index.html>）。

## 3. 运行环境说明

### 3.1测试开发：

进行测试开发时需要以下环境：使用Eclipse 3.6.\*或Eclipse 3.7.\*，UAT包需要依赖NC Module.

### 3.2回归测试：

执行回归测试时需要以下环境：Windows操作系统，Oracle 数据库，NC\_Home.

# 二．初始测试环境：

## 1.安装uat测试框架

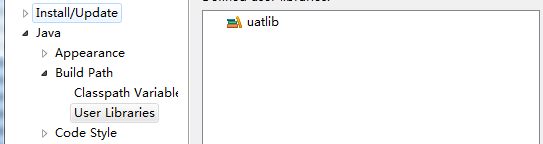
### 1.1 导入uat文件

在项目开始使用UAT测试框架前，首先要在项目所在工程中进行初始测试环境，操作如下：

(//langpf11/uat测试框架)将uatlib文件夹,extlib文件夹和uat\plugin文件夹下的com.yonyou.uat.plugin\_x.x.x.xxxxxx… 文件,将它们拷贝到Eclipse安装文件夹的dropins文件夹下。

### 1.2新建uatlib

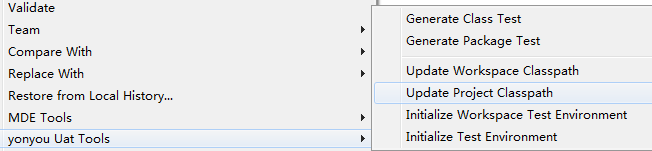
步骤是单击Window🡪Preferences，弹出Preferences对话框，在Java🡪Build Path🡪User Libraries，单击”New…”按钮，新建User Library，命名为uatlib，导入uatlib和extlib文件夹里的jar文件：



### 1.3 导入uatlib

将uat依赖的jar包导入到当前项目的classpath下，步骤是：

在Eclipse某个项目文件夹上右键:

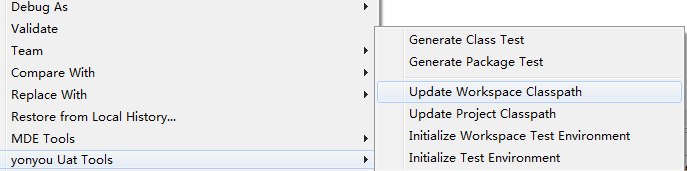


成功后显示：



ps: 如果想将uat依赖的jar包导入到当前工作区间所有项目的classpath下，可以按如下步骤：

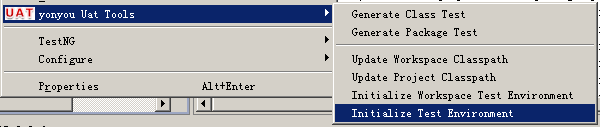
在Eclipse项目项目文件夹上右键



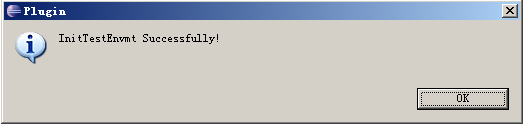
## 2．uat环境初始化

### 2.1 初始化测试环境：

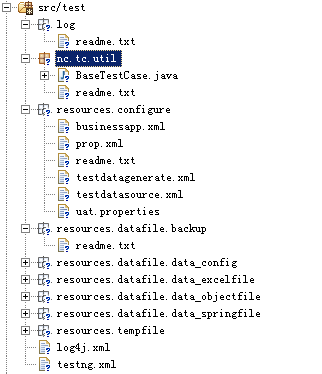
1> 初始化测试环境会在项目源码(src)目录下按照测试目录组织结构在项目SRC/目录下生成的Test目录，具体步骤是，在Eclipse项目文件夹上右键：



初始化成功后，界面显示成功提示：



2> 在项目上按F5刷新，效果如下：

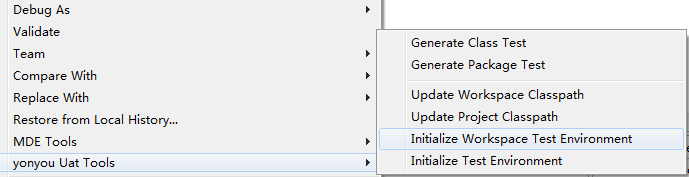


目录说明(可以先不看)：

* Log 目录是保存测试过程中参生日志信息
* Resources.configure 目录保存与uat框架运行相关的配置信息
* uat.properties uat 框架运行配置文件
* BussinessApp.xml 如果需要启动中间件或者其他服务在这里对服务进行配置。
* Prop.xml 针对NC中间件提供的数据源配置信息，该文件提供给默认的BusinessApp NC中间件服务使用。
* Testdatagenerate.xml
* testDataSource.xml
* resources.datafile.backup 目录提供给测试数据备份使用
* resources.datafile.data\_configure 目录保存测试数据与测试代码对应关系文件
* resources.datafile.data\_excelfile 目录保存excel文件格式测试数据
* resources.datafile.data\_objectfile 目录保存序列化的对象数据文件
* resources.datafile.data\_springfile 目录保存SpringIOC 方法构造数据对象的配置文件
* resources.tempfile 目录保存测试过程中参生的临时文件信息。
* testing.xml 文件是testng 运行配置文件
* log4j.xml 日志配置信息。
* nc.tc.util 包是测试时需要使用的一些通用测试基础类存放位置
* nc.tc.util.BaseTestCase 是测试项目的基础类，生成的测试代码都集成该类，测试人员可根据需要对基础类中内容进行调整，来满足测试需要。

ps: 如果想在工作区间下所有的项目上生成测试代码目录：

在Eclipse项目文件夹右键：

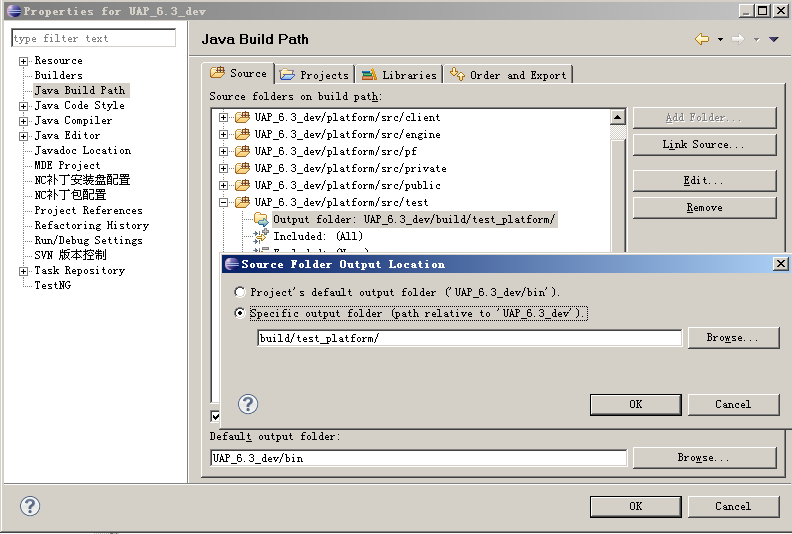


### 2.2 手工配置环境：

初始完测试环境后需要做些手工配置：

* 设置测试模块编译目录

在多个模块项目中建议在build目录为每个模块建立测试编译输出目录：



* 修改resources.configure包下的uat.properties配置文件：

#project information

app\_name=platform //项目名称

app\_version=V6.0 //项目版本号

#Common Configure

nc\_home=D:/nc\_home/uap\_nchome/kongxla\_NC\_UAP\_MODULES6.3\_2\_int/NC6\_UAP\_VOB/NC\_UAP\_MODULES //nc home 目录

log\_file=D:/nc\_home/uap\_nchome/kongxla\_NC\_UAP\_MODULES6.3\_2\_int/NC6\_UAP\_VOB/NC\_UAP\_MODULES/nclogs/testlog1.log //nc log或者其他需要监控的日志文件

uatLogSwitch=2

#BusinessServer configure

serverStart=on //测试时是否启动中间件或服务

server\_spring\_file=businessapp.xml //服务注入配置文件

nc\_config\_file=resources/configure/prop.xml //启动nc中间件使用的配置文件

#TestData Configure Section

testdata\_config\_dir=resources/datafile/data\_config/ //数据配置文件目录

testdata\_Springfile\_dir=resources/datafile/data\_springfile/ //Spring注入配置文件

testdata\_Objectfile\_dir=resources/datafile/data\_objectfile/ //对象文件

testdata\_excelfile\_dir=resources/datafile/data\_excelfile/ //excel文件

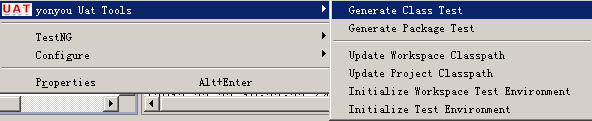
Ttestdata\_backup\_dir=resources/datafile\_backup/ //数据备份目录

# 三．生成测试代码

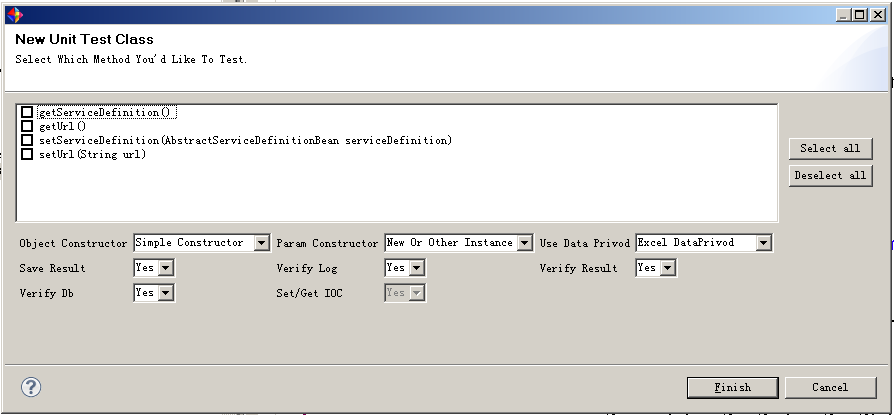
## 1. 按类生成测试代码

1> 按单个类来生成相应的测试代码（也可以按包生成测试代码，见后面）：

选择单个的类文件并鼠标右键选择：



2> 点Generate Class Test 弹出配置界面，界面中列出了所有方法（具体可参考下一条“代码生成配置说明”），可以点Select All 按钮对全部方法进行选择：



## 2. 代码生成配置说明

### 2.1 Object Constructor

Object Constructor选项提供下列三种对象构造方法：

* Simple Constructor 选择参数最少的类构造方法，如果参数相同随机选择，生成构造语句。

例子：

generalDMO = new GeneralDMO();

* Complex Constructor选择参数最多的类构造方法，如果参数相同随机选择，生成构造语句。

例子：

Int id=0;

String name = “a”;

generalDMO = new GeneralDMO(id,name);

* Lookup Constructor被测试对象的构造，通过Lookup方式来获取对象实例（针对nc接口测试），生成构造语句。

例子：

iPFSplitBill=NCLocator.getInstance().lookup(IPFSplitBill.class);

### 2.2 Param Constructor

Param Constructor 方法参数对象的构造方法：

* New or Other Instance 使用new 或者getInstance方法构造参数对象，生成构造语句。

例子

XXVO xxVo = new XXVO();

Or

XXVO xxVo = XXVO().getInstance();

generalDMO = new GeneralDMO(xxVo);

* Spring IOC使用Spring IOC的方式构造参数对象，生成构造语句。

例子：

ChangeVOAdjustContext context =(ChangeVOAdjustContext) super.getSpringObj("changevoadjustcontext");

* Object File使用 对象文件生成参数对象，生成构造语句。

例子：

//Construct method parameter

MobileConfig mc=(MobileConfig)getExpectResultObject("");

//Invoke tested method

void retObj=WirelessManager.saveMobileConfig(mc);

* Mock使用Mock方式构造参数对象，生成构造语句。

//Construct method parameter

SysMessageParam smp=mock(SysMessageParam.class);

when(smp.methodName()).thenReturn("expectedValue");

boolean retObj=WirelessManager.saveSysMsgParam(smp);

* DataBase Constructor 使用 数据库方式获得参数对象，生成构造语句。

//Construct method parameter

ArrayList<String> paramConds=new ArrayList<String>();

paramConds.add("sid");

paramConds.add("where condition");

MobileConfig mc=(MobileConfig)super.getDbObject(MobileConfig.class,paramConds);

### 2.3 Use Data Privoder

DataPrivoder 是testng 提供的一种测试方法参数化方法。DataPrivoder提供二维数组对象，方法每次调用获取一行数据，通过get方法获取行字段值。可以选择是否使用DataPrivoder，DataPrivoder提供两组数据保存方式：excel，database：

* Excel DataPrivod使用Excel文件来保存参数，生成个DataPrivoder语句。

@DataProvider(name="dp")

public Iterator<Object[]> dataForTestMethod( Method method) throws IOException, BiffException {

return new ExcelDataProvider(this.getClass().getName(),method.getName());

}

* DataBase DataPrivod 使用数据库表来保存参数，生成个DataPrivoder语句。

@DataProvider(name="dp")

public Iterator<Object[]> dataForTestMethod( Method method) throws IOException, BiffException {

String dataSourceName="";

String tableName="";

Map<String,Object> paraMap=new HashMap<String,Object>();

paraMap.put("","");

DBDataProvider dBDataProvider=new DBDataProvider(dataSourceName,tableName,paraMap);

return dBDataProvider;

}

* No 不使用DataPrivoder 功能，不生成个DataPrivoder语句。

### 2.4 Save Result

Save Result：是否在测试代码中生成保存中间对象数据到对象文件的语句：

Yes 使用结果对象保存方法，生成语句

No 不使用结果对象保存方法，不生成语句

### 2.5 Verify Log

Verify Log：是否生成Log 验证代码：

Yes 生成验证日志的语句

No 不生成验证日志的语句

### 2.6 Verfiy Result

Verfiy Result：是否生成基本的结果验证语句：

Yes 生成基本的验证语句

No 不生成基本验证语句

### 2.7 Verify Db

Verify Db：是否生成数据库验证语句：

Yes 生成数据结果验证语句

No 不生生成数据结果验证语句

### 2.8 Set/Get IOC

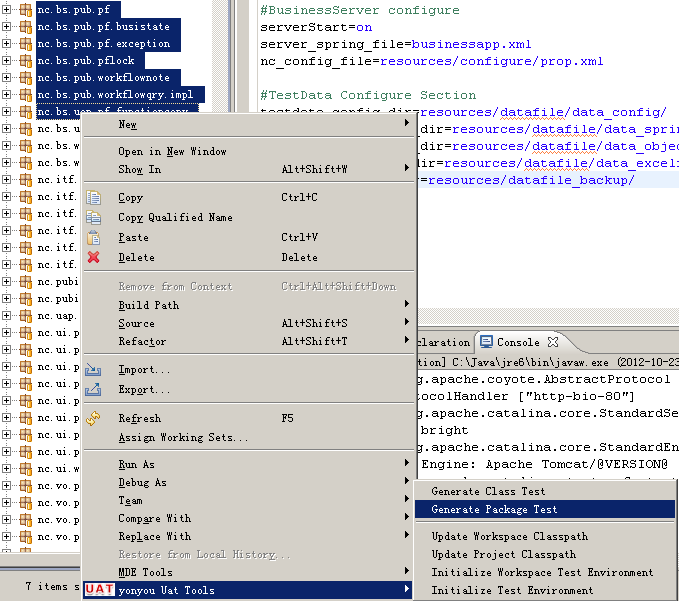
该配置项与参数构造配置中Spring IOC构造方法相关，如果选择SpringIOC方法并且Set/Get IOC 配置

Yes 生成Spring注入配置文件时生成get,set 方法属性值配置。

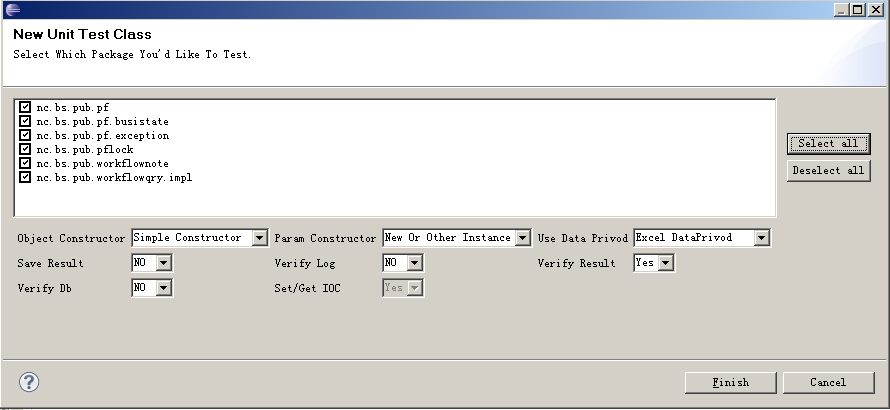
No 不生成Spring注入配置文件时生成get,set 方法属性值配置。

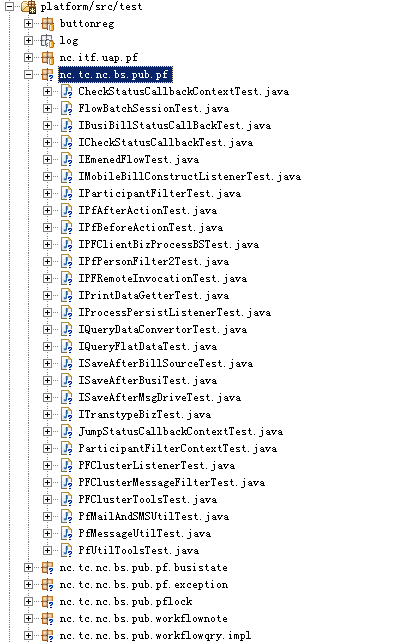
## 3. 按包生成测试代码

1> 在Eclipse项目源码目录中选择多个包或者多个类，鼠标右键：



2> 弹出配置界面

3> 点finish按钮，生成测试代码，选择src/test 目录按F5 刷新目录。



## 4. 自动生成代码完整样例

可以在生成的样例的基础上就行修改。

public class BillMappingConvertorTest extends BaseTestCase {

BillMappingConvertor billMappingConvertor=null; //定义一个空的被测试对象变量；

Object retObj; //定义通用返回对象变量

@BeforeClass //在所有测试方法执行前调用，完成被测试对象的实例化；

public void BeforeClass(){

billMappingConvertor=new BillMappingConvertor();

}

@AfterClass //在所有测试方法执行完后调用，类级资源回收；

public void AfterClass(){

}

@BeforeMethod //在每个测试方法执行前调用，测试方法前初始操作

public void BeforeMethod(){

}

@AfterMethod //在每个测试方法执行后调用，对测试方法级资源回收

public void AfterMethod(){

}

@DataProvider(name="dp") //构造DataProvider，构造测试参数数据

public Iterator<Object[]> dataForTestMethod( Method method) throws IOException, BiffException {

return new ExcelDataProvider(this.getClass().getName(),method.getName());

}

//测试方法，每一个被测试方法对应一个测试方法。

@Test(description="",dependsOnMethods={},groups="",timeOut=100000,dataProvider="dp")

public void setBuziDate( Map<String,ArrayList<String>> dpdata){

//Construct method parameter

String date="a";

//Invoke tested method

billMappingConvertor.setBuziDate(date);

//Verify result is Ok

//Base Verify Method

Assert.assertEquals("actual","expected");

//Verify DB result is ok //生成数据库验证语句段

ArrayList<String> verifyConds=new ArrayList<String>();

verifyConds.add("sid"); //数据库实例名称

verifyConds.add("where condition"); //数据查询where条件语句

Object actualObject=super.getDbObject(VOName.class,verifyConds);

Assert.assertEquals("(VOName)actualObject.getxxx()",dpdata.get("colName"));//验证对象属性

//Verify whether NC Log have exception information

super.verifyLog("Error key word");//验证制定的日志文件中是否有错误信息

//Verify Return or middle Object == expect Object(from object file)

Object expectedObj=super.getExpectResultObject("caseName"); //获取对应的序列化对象

if (expectedObj != null) { //如果不为空，用获取的序列化对象与实际返回对象对比

Assert.assertEquals(retObj,expectedObj);

}

else {//如果为空，把返回对象序列化保存到对象文件中

super.saveResultObject((Serializable)retObj,"caseName");

}

}

}

## 5.测试代码组织

### 5.1 目录结构

有一个readme 文档描述如何搭建测试代码编译、部署、运行过程文档。

### 5.2 包结构

在UAP项目中源码目录结构如下SRC/PUBLIC,SRC/PRIVATE,SRC/CLIENT ,SRC/TEST,单元测试在SRC/TEST 目录下建立测试代码包，包结构与开发代码结构相同，与开发代码有呼应关系：

例如

BapPorject

-src/public

-src/private

-src/client

-src/test/

-src/test/resource

-src/test/log

-src/test/testng.xml

-build.xml

-build.properites

如果项目中SRC目录下不在分源码目录 ，建议在SRC平级目录中建立Test目录，单元测试在Test目录下建立测试代码包，包结构与开发代码结构相同：

例如

BapPorject

-src/

-src/test/

-src/test/resource

-src/test/log

-test/testng.xml

-build.xml

-build.properites

如果项目中SRC目录下再建立源码目录，那么在项目源码目录同级建立Test目录，单元测试在Test目录下建立测试代码包，包结构与开发代码结构相同。

例如

OtherPorject

-src/main/java

-src/main/resources

-src/test/

-src/test/resources

-src/test/log

-src/test/testng.xml

-build.xml

-build.properites

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **文件名** | **功能** | **路径** |
| 1 | 单元测试代码 | 用于测试每个类方法或者测试接口方法 | Test/ |
| 2 | 测试基类和支持代码 | 用于支持单元测试的基础类 | Test/ util包 |
| 3 | 测试相关的配置文件 | 测试相关的配置文件 | test.resources.configure |
| 4 | 测试数据 | 测试数据 | test.resources.datafile |

### 5.3 配置文件

*[配置文件部分主要是构建单元测试代码的开发环境所需要的配置文件。]*

测试基于UAT框架时需要配置UAT测试相关属性文件configure,如果测试数据使用了Spring IOC方法，还需要配置Spring配置文件，例如：

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **配置文件名** | **功能** | **路径** |
| 1 | Uat.properties | uat框架属性文件，配置项目相关及NC中间件相关的配置信息 | test.resources.configure.uat |
| 2 | Prop.xml | NC中间件属性文件 | test.resources.configure.uat |
| 3 | Testdatagenerate.xml | 生成测试数据模块目录配置 | test.resources.configure.uat |

### 5.4 数据文件

*[数据文件部分主要是构建单元测试代码的开发环境所需要的测试数据。]*

测试代码与数据尽量做到分离，数据可以保存在database,Excel中，例如：

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **数据文件名** | **功能** | **路径** |
| 1 | {className}.xls | 测试用例数据 | test.resources.datafile.Excel |
| 2 | {ObjectName}.xml | 对象数据序列化文件 | test.resources.data file.ObjectFile |

## 6. 单元测试实践建议

### 6.1单元测试的目标

单元测试的目标是测试最小独立单元的功能和代码逻辑实现。在开始单元测试前需要对单元测试项目进行了解：

* 首先并对初始出来的测试配置文件进行配置；
* 其次提取一些通用的数据或操作作为测试工具类；
* 再次根据测试项目特点修订BaseTestCase来满足项目测试需要；

### 6.2单元测试的原则

测试代码生成过程中尽量遵循以下原则：

* 单元测试和接口测试尽量调用真实对象，对于很难构造或构造复杂可使用Mock技术进行隔离。调用真实对象时，被测试对象使用New or getInstance 方式生成。测试参数构造根据实际情况选择SpringIOC，DataBase，DataFile 或new 方式，尽量避免使用Mock 。

使用SpringIOC的方式多是针对多层次对象构造属性；

使用DataFile 的方式针对能够使用其它工具抓取参数对象并保存成对象文件的方式；

使用DataBase的方式针对参数对象保存在数据库表中，通过SQL查询并生成对象文件；

* 如果测试的关联模块不具备被调用条件或者搭建环境比较复杂，可采用Mock技术进行模块隔离。

### 6.3测试代码命名规范

1> 测试类名

被测试类名后加入Test。

例如：

被测试代码名称：GetoptDemo

测试代码名称为：GetoptDemoTest

2> 测试方法名

单元测试中测试方法尽量与被测试方法一致，如果被测试方法存在重载的情况可使用数字结尾表示，并在测试描述中给与说明。