# Junit单元测试

### Junit单元测试框架的基本使用

一、搭建环境：

导入junit.jar包（junit4）

二、写测试类：

0，一般一个类对应一个测试类。

1，测试类与被测试类最好是放到同一个包中（可以是不同的源文件夹）

2，测试类的名字为被测试类的名字加Test后缀。

三：写测试方法：

0，一般一个方法对应一个单元测试方法。

1，测试方法的名字为test前缀加被测试方法的名字，如testAddPerson()。

2，单元测试方法上面要加上@Test注解（org.junit.Test）！

3，单元测试方法不能有参数，也不能有返回值（返回void）！测试的方法不能是静态的方法。

四、测试方法的基本使用：

1，可以单独执行一个测试方法，也可以一次执行所有的、一个包的、一个类中所有的测试方法。

2，执行完后，显示绿色表示测试成功；显示红色表示测试失败（抛异常后会测试失败）。

### Assert断言工具类

其中有一些静态的工具方法（不符合期望就抛异常）：

assertTrue(...) 参数的值应是true

assertFalse(...) 参数的值应是false

assertNull(...) 应是null值

assertNotNull(...) 应是非null的值

assertSame(...) 使用==比较的结果为true（表示同一个对象）

AssertNotSame(...) 使用==比较的结果为false

assertEquals(...) 两个对象equals()方法比较结果为true

### 用于准备环境、清理环境的方法

@Test

表示单元测试方法。

@Before

所修饰的方法应是非static的（且没有参数，返回值为void）。

表示这个方法会在本类中的每个单元测试方法之前都执行一次。

@After

所修饰的方法应是非static的（且没有参数，返回值为void）。

表示这个方法会在本类中的每个单元测试方法之后都执行一次。

@BeforeClass

所修饰的方法应是static的（且没有参数，返回值为void）。

表示这个方法会在本类中的所有单元测试方法之前执行，只执行一次。

@AfterClass

所修饰的方法应是static的（且没有参数，返回值为void）。

表示这个方法会在本类中的所有单元测试方法之后执行，只执行一次。

# 内省(Introspector)

# 为什么要学内省？

开发框架时，经常需要使用java对象的属性来封装程序的数据，每次都使用反射技术完成此类操作过于麻烦，所以sun公司开发了一套API，专门用于操作java对象的属性。

内省是用于操作java对象的属性的，那么以下问题我们必须要清楚。

**问题一：** 什么是Java对象的属性和属性的读写方法？

问题二: 如何通过内省访问到javaBean的属性 ?

1. 通过PropertyDescriptor类操作Bean的属性.

|  |
| --- |
| **public** **static** **void** testPropertyDescriptor() **throws** Exception{  Person p = **new** Person();  PropertyDescriptor propertyDescriptor = **new** PropertyDescriptor("id",Person.**class**);  //获取属性的写的方法。  Method writeMethod = propertyDescriptor.getWriteMethod();  Method readMethod = propertyDescriptor.getReadMethod();  propertyDescriptor.getReadMethod();  writeMethod.invoke(p, 12);  System.*out*.println(readMethod.invoke(p, **null**));  } |

2. 通过Introspector类获得Bean对象的 BeanInfo，然后通过 BeanInfo 来获取属性的描述器（ PropertyDescriptor ），通过这个属性描述器就可以获取某个属性对应的 getter/setter 方法，然后通过反射机制来调用这些方法。

|  |
| --- |
| **public** **static** **void** testIntrospector() **throws** Exception{  BeanInfo beanInfo = Introspector.*getBeanInfo*(Person.**class**);  PropertyDescriptor[] descriptor = beanInfo.getPropertyDescriptors();  **for**(PropertyDescriptor itemProperty : descriptor){  System.*out*.println(itemProperty.getReadMethod().getName());  }  } |

存在的问题： sun公司的内省API过于繁琐，所以Apache组织结合很多实际开发中的应用场景开发了一套简单、易用的API操作Bean的属性——BeanUtils。

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception {  Person p = **new** Person();  ConvertUtils.*register*(**new** Converter() {    @Override  **public** Object convert(Class type, Object value) {  **try** {  **if**(value!=**null**){    SimpleDateFormat dateFormat = **new** SimpleDateFormat("yyyy MM dd");  Date d = dateFormat.parse((String) value);  **return** d;  }  } **catch** (ParseException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }    **return** **null**;  }  }, Date.**class**);    BeanUtils.*setProperty*(p,"id","110");  BeanUtils.*setProperty*(p,"name","狗娃");  BeanUtils.*setProperty*(p, "birthDay","1992 12 12");  System.*out*.println(p.getId() +"=="+ p.getName()+"======"+p.getBirthDay());  } |

# Properties类与配置文件

### Properties配置文件说明

Properties类对应.properties文件。文件内容是键值对，键值对之间使用"="或空格隔开。开头是"#"的表示注释

Properties类在加载.properties文件时使用的iso8859-1的编码。所以这个文件中的中文要特殊处理：如果这个配置文件中有中文就必须要进行转义，使用native2ascii.exe命令操作:

**native2ascii d:/my.properties d:/my2.properties**

使用Properties类中的load(InputStream) 方法可以加载配置文件，使用其中的store(OutputStream) 方法可以保存配置到指定文件。

更多的信息可以看Properties类的API文档。

### 加载配置文件

**public** **static** **void** testLoadProperties() **throws** Exception {

Properties properties = **new** Properties();

InputStream in = **new** FileInputStream("E:/itcast/config.properties");

properties.load(in); // 加载

in.close();

System.*out*.println(properties);

}

### 写配置文件

**public** **static** **void** testStoreProperties() **throws** Exception {

// 准备配置信息

Properties properties = **new** Properties();

properties.setProperty("name", "李四");

properties.setProperty("age", "20");

// 准备

OutputStream out = **new** FileOutputStream("d:/my.properties");

String comments = "这是我的配置文件";

// 写出去

properties.store(out, comments);

out.close();

}

作业：使用properties读取配置文件，读取数据库的用户名、密码。并且打包成jar包。

### 使用Properties类

|  |
| --- |
| **public** **class** DBUtil {    **static** Properties *properties* = **new** Properties();    **static**{  **try** {  Class clazz = DBUtil.**class**;  InputStreamReader fileReader =  **new** InputStreamReader(clazz.getResourceAsStream("/db.properties"));  *properties*.load(fileReader);  } **catch** (IOException e) {  e.printStackTrace();  }  }  **public** **static** String getUserName(){  String userName =*properties*.getProperty("userName");  **return** userName;  }    **public** **static** String getPassword(){  **return** *properties*.getProperty("password");  }  **public** **static** **void** main(String[] args) {  System.*out*.println("用户名："+ *getUserName*());  System.*out*.println("密码: "+ *getPassword*());  }  } |

# 文件路径

### 绝对路径

以根目录或某盘符开头的路径（或者说完整的路径）

例如：

* c:/a.txt （Windows操作系统中）
* c:/xxx/a.txt （Windows操作系统中）
* /var/xx/aa.txt （Linux操作系统中）

绝对路径的问题: 比如C:\abc\a.properties文件路径，该路径在windows上执行没有 问题，但是如果把该项目移动到linux上面执行 ，该路径就会出现问题了，因为在linux上面没有c盘的，只有根目录\。

### 相对路径

相对于当前路径的一个路径。例如当前文件夹为c:/abc时：相对路径a.txt表示c:/abc/a.txt，相对路径xx/a.txt = c:/abc/xx/a.txt

* . 表示当前文件夹
* .. 表示上级文件夹

相对路径存在的问题:相对路径是相对于目前执行class文件的时候，控制台所在的路径，这样子也会导致出现问题。

### Java程序中的相对路径

**测试代码：**

**public** **class** PathTest {

**public** **static** **void** main(String[] args) **throws** Exception {

System.*out*.println(**new** File("a.txt").getAbsolutePath());

}

}

在命令行中使用cd命令切换到不同的路径下试试，可以看到以上所说的效果。

**在Eclipse中，当前路径是工程的根目录。**

### classpath路径

#### classpath路径说明

在Java程序中，一般情况下使用绝对路径还是相对路径都不太合适，因为Java程序的jar包所放的位置不确定，执行java程序时当前的路径也不确定，所以不合适。一般在Java程序中我们会把资源放到classpath中，然后使用classpath路径查找资源。

Classpath路径：就是使用classpath目前的路径。

#### 获取classpath中的资源（InputStream）

**public** **static** **void** main(String[] args) **throws** Exception {

Class clazz = **new** ClassPathTest().getClass();

// 开头的'/'表示classpath的根目录，这个是表示从classpath的根目录中开始查找资源

InputStream in = clazz.getResourceAsStream("/cn/itcast/my.properties");

// 如果开头没有'/'，表示从当前这个class所在的包中开始查找

InputStream in2 = clazz.getResourceAsStream("my.properties");

}

资源路径在java工程和web工程的区别：

\* 在java项目下，classpath的根目录从bin目录开始

\* 在web项目下，classpath的根目录从WEB-INF/classes目录开始

在java项目下，. java命令的运行目录从项目的根目录开始

\* 在web项目下， . java命令的而运行目录从tomcat/bin目录开始

**在Java程序中使用File时写相对路径，是指相对于执行java命令时当前所在的文件夹。**

### web工程下，加载资源文件

web工程下，路径问题是区分大小写的。

所以windows养成的坏习惯得改掉啊，敲命令行，区分大小写。

获取工程目录下的src中的资源文件。

因为web工程中，src的文件都会发布到服务器的web-inf/classes目录下

有三个方法：

1. 使用类加载器

code:

ClassName.class.getClassLoader().getResourceAsStream(“资源”);

该方式的得到的路径是 工程目录/WEB-INF/classes

1. 在Servlet中使用servletContext对象获取

this.getServletContext().getResourceAsStream(“WEB-INF/classes/资源名”);

该方式的得到的路径是在工程目下

3.使用 new FileInputStream（不推荐，脱裤子放屁的感觉）

code:

String path=className.class.getClassLoader().getResource().getPath();

//这个path路径是：工程目录/web-inf/classes

new FileInputStream(path+”资源路径”);

### 文件夹位置

web工程中：

1. 在WebContent中新建文件夹file，发布到服务器，该文件夹在工程目录下。
2. 在工程下创建文件夹file，发布到服务器，找不到该文件夹