单元测试(JUNIT)

单元:在JAVA中单元,指的是独立的一个 类

单元测试:指对JAVA程序的类进行测试

专业工具:JUNIT

使用步骤:

1、编写目标类（独立单元）

2、导包步骤：Properties ——>Java Build Path ——> Libraries ——> Add Library ——>Junit ——>Junit4

3、编写测试类：在测试类中写一个测试方法，在方法上加一个@Test注解

4、在方法中创建目标类对象，并调用需要测试的目标方法，使用断言同时可以对运行结果进行测试

代码如下：

```
@Test
public void addTest(){
    Cale cale = new Cale();
    int result = cale.add(1,1);
    //还可使用断言对运行结果进行测试
    Assert.assertEquals(2, result);//参数：第一个是预期值，第二个是实际运行结果
```

5、开始测试：右键 ——> Run As ——>Junit Test

6、测试或断言成功，运行通过并输出结果（有输出的情况下），如果失败则会给出红色警告并提示

注解

是一种代码执行的标记，给虚拟机解释程序如何执行

注解（Annotation），也叫元数据。一种代码级别的说明。JDK1.5以及以后版本引入的一个特性，与类、接口、枚举是在同一个层次。可以声明在包、类、字段、方法、局部变量、方法参数等的

前面，用来对这些元素进行说明，注释。注解是以‘@注解名’在代码中存在的。

作用：

●编写文档：通过代码里标识的元数据生成文档（生成文档doc）

●代码分析：通过代码里标识的元数据对代码进行分析（使用发射）

●编译检查：通过代码里标识的元数据让编译器能够实现基本的编译检查（Override）

常见三个场景:方法的重写 ，抛异常 ， @Test(JUnit单元测试)

xml

xml是一种格式严谨的可扩展标记语言

xml的主要的功能是存储数据,（不同语言不同操作系统之间的数据交互）语言之间的数据交互（不是显示数据）,软件配置文件（以描述程序模块之间的关系）。

作用：数据存储交互、配置文件

xml的版本有1.0 、1.1 ， 一般使用的是1.0版本。

xml文件格式被广泛的应用在移动开发的网络传输的数据格式

标记型语言：HTML是标记型语言，是使用标签进行操作，XML里面的操作也是使用标签进行操作。

注意：可扩展：html里面的标签，每个标签有自己特定的含义，比如
 <hr/>，在xml中标签自己定义的，比如 <aa> <猫>。

xml的语法

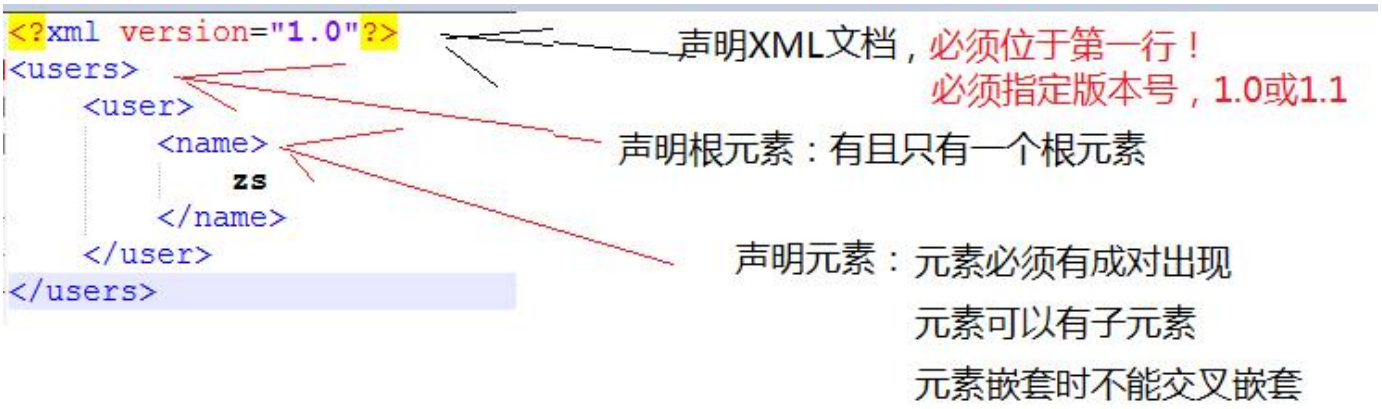
在编写XML文档时，需要先使用文档声明来声明XML文档。

写法： <? xml version="1.0" encoding="utf-8"?>

属性	说明
version	必须要有，xml的版本，一般使用1.0

encoding	可选的，xml的编码方式
standalone	可选的，xml是否依赖其他文件 yes no

注意：文档声明必须要放在xml文件的第一行第一列



xml的元素的定义

(1) xml中的标签成对出现，有开始标签也要有结束标签<a>。

(2) 有的标签没有内容（没有结束标签，类似于html
），需要在标签内结束 <aa/>。

(3) xml中的标签可以嵌套，但是必须合理嵌套，<a>这样写是不对的，正确写法：<a>。

(4) 一个XML文档必须有且仅有一个根标签，其他标签都是这个根标签的子标签或孙标签。

命名规范

一个XML元素可以包含字母、数字以及其它一些可见字符。

* xml区分大小写，例如，<P>和<p>是两个不同的标记。

* xml的标签不能以数字或"_" (下划线)开头。 <1a> <_bb>这样写不对的。

* xml的标签不能以xml(或XML、或Xml 等)开头。

* 在xml的标签里面不能包含空格。<a b>这样写不对的。

* xml的标签名称中间不能包含冒号（:）。<bb:cc>这样写不对的。

xml的属性的定义 用ID来定义属性，主要用于区分相同数据内容的子标签

(1)在xml的标签里面可以有多个属性，但是属性的名称不能重复。

(2)属性和属性值之间需要使用=隔开，属性值需要使用引号包起来（双引号和单引号）。<aa id1="abcd" id2='qqq'></aa>

(3)属性名称的命名规范与元素的命名规范相同。

xml的注释

语法：<!--这是注释-->

注意：XML声明之前不能有注释，注释不能嵌套。

xml的解析（解析就是获取XML中包含的数据）

xml解析有三种方式：dom解析，sax解析，pull解析

dom解析原理：

在java中dom解析是将整个XML文档做为Document对象，将xml文件全部载入，组装成一颗dom树，然后通过节点以及节点之间的关系来解析xml文件

dom解析xml的优点：

因为分配了一个树形结构，很方便的实现增加、修改、删除的操作。

dom解析xml的缺点：

如果要解析的文件过大，一次性在内存中分配一个树形结构，造成内存的溢出。

涉及三个类：DocumentBuilderFactory

DocumentBuilder

Document

解析方式：

文档对象Document ：代表整个文档

元素对象Element：代表元素对象（标签）

属性对象：标签中的属性

文本对象：标签体中的文本

Node节点对象：是上面对象的父对象

1、通过DocumentBuilderFactory抽象类的newInstance()方法获得解析工厂

2、通过工厂对象调用newDocumentBuilder()方法获取解析器，返

回一个DocumentBuilder抽象类

3、通过解析器对象调用parse("bin/x.xml")方法获取需要解析的文档对象，返回一个Document接口

4、通过文档对象调用getElementsByTagName("xml的根节点下一级子节点标签")方法，获取XML文档节点集合，返回一个NodeList集合。

5、通过普通for　　和　　集合对象.getLength()方法遍历文档所有子节点.

6、通过 集合对象.item(i)方法获得每一个子节点的节点。

7、通过子节点对象.getChildNodes()方法获取子节点集合，再次返回一个NodeList集合。

8、通过普通for　　和　　子节点集合对象.getLength()方法遍历文档所有子节点下所有子节点.

9、通过 子节点集合对象.item(j)方法获得每一个子节点的节点。

10、通过子节点对象.getNodeName()方法获取每一个子节点的节点名称。

11、通过节点名称判断，通过子节点对象.getFirstChild().getNodeValue()获取相应节点下的数据.

注意：如果要将XML文档中的数据添加到对象中通过集合输出再控制台，那么在开始解析前需要创建一个集合对象，集合中保存的数据就是对象类

在第4步完成之后需要创建一个对象的延迟加载，进入到第5步里面创建对象并添加集合元素。

DOM解析代码

```java
public class Dom_Xml {
    @Test
    public void test() throws Exception{
        //将per类加入集合
        ArrayList<per> al = new ArrayList<per>();
    //拿到工厂
    DocumentBuilderFactory f = DocumentBuilderFactory.newInstance();
    //获取解析器
    DocumentBuilder d = f.newDocumentBuilder();
    //获取文档对象
    Document p = d.parse("bin/x.xml");
    //解析数组,通过拿到子节点遍历获取根节点中下一级的子节点
    per p1 = null;//建立per对象延迟加载
    NodeList o = p.getElementsByTagName("o1");
        for (int i = 0; i <o.getLength(); i++) {
            p1 = new per();//获取到根节点的时候创建对象
            al.add(p1);//添加元素
            Node o1 = o.item(i);
            //通过拿到子节点遍历获取子节点中的子节点元素
            NodeList t = o1.getChildNodes();


            for (int j = 0; j <t.getLength(); j++) {
                Node it = t.item(j);
            //通过获取子节点中所有元素进行判断,获取元素中的数组
                String no = it.getNodeName();
                if("name".equals(no)){
                p1.setName(it.getFirstChild().getNodeValue().trim());
                }
                if("sex".equals(no)){
                p1.setSex(it.getFirstChild().getNodeValue().trim());
                }
                if("age".equals(no)){
                p1.setAge(Integer.parseInt(it.getFirstChild().getNodeValue().trim()));
                }
            }
        }
        System.out.println(al);
    }
}
```

Dom_Xml.java | x.xml | per.java

```xml
<?xml version="1.0" ?>
<o>
    <o1 id = "001">
        <name>
        你大爷
        </name>
        <sex>
        不知道
        </sex>
        <age>
        25
        </age>
    </o1>
    <o1 id = "002">
        <name>
        大爷
        </name>
        <sex>
        知道
        </sex>
        <age>
        2
        </age>
    </o1>
```

Dom_Xml.java | x.xml | per.java

```java
package other.test;

public class per {
    private String name;
    private String sex;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getSex() {
        return sex;
    }
    public void setSex(String sex) {
        this.sex = sex;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    @Override
    public String toString() {
```

```
public  class  MyHandler  extends  DefaultHandler{
    重写startDocument（）方法－－－>开始文档
    重写startElement（） 方法－－－>开始标签
    重写characters（） 方法－－－>获取数据
        获取数据只需要在重写的方法里加
        String  str  =  new  String(ch,  start,  length).trim();
        System.out.println(str);
    重写endElement（） 方法－－－>结束标签
```

重写 endDocument（）方法－－－>结束文档

```
}
public class Sax_Xml {

    @Test
    public void saxxml() throws Exception{
        SAXParserFactory n = SAXParserFactory.newInstance();
        SAXParser ns = n.newSAXParser();
        ns.parse("bin/x.xml", new MyHandler());
    }



public class MyHandler extends DefaultHandler{

    @Override
    public void startDocument() throws SAXException {

        super.startDocument();
    }

    @Override
    public void startElement(String uri, String localName, String qName,
            Attributes attributes) throws SAXException {

        super.startElement(uri, localName, qName, attributes);
    }

    @Override
    public void characters(char[] ch, int start, int length)
            throws SAXException {

        super.characters(ch, start, length);
        String str = new String(ch, start, length).trim();
        System.out.print(str+"  ");
    }



    @Override
    public void endElement(String uri, String localName, String qName)
            throws SAXException {

        super.endElement(uri, localName, qName);
    }

    @Override
    public void endDocument() throws SAXException {

        super.endDocument();
    }
```

pull解析原理：

Pull解析器的运行方式与 SAX 解析器相似。它提供了类似的事件，如：开始元素和结束元素事件，使用parser.next()可以进入下一个元素并触发相应事件。跟SAX不同的是， Pull解析器产生的事件是一个数字，而非方法，因此可以使用一个switch对感兴趣的事件进行处理。当元素开始解析时，调用parser.nextText()方法可以获取下一个Text类型节点的值。

是Android内建的xml解析方式，依赖于两个包：kxml2-2.3.0

xmlpull_1_1_3_4c

涉及三个类： XmlPullParser

File流

XmlPullParserFactory 解析器工厂

解析xml步骤：

首先，导入pull的jar包(两个jar包)，原理其实就是sax解析。

* 1、创建解析器工厂
* 2、根据解析器工厂创建解析器
* 3、把要操作的文件放到解析器里面

1、通过XmlPullParserFactory类调用newInstance()方法，创建解析器工厂，返回一个XmlPullParserFactory对象

2、通过工厂对象.newPullParser()方法，获取解析器，返回一个XmlPullParser对象。

3、通过解析器对象.setInput（输入流对象，“字符编码”）方法，将要操作的文件放入解析器中。（注意：xml文件需要通过内建一个：右键 ——> NEW ——> other ——> XML(Basic Templates) ——>选择存放位置 ——> 打开XML文件 ——>编写文件内容 如果是字符流对象，不用写字符编码）

4、通过解析器对象.getEventType()方法，获取元素事件，返回一个int数。

5、通过while(eventType!=parser.END_DOCUMENT)循环判断这个元

素事件不是文档结束。

6、在循环中通过if（eventType==parser.START_TAG）判断这个元素事件是标签开始。

7、在if中继续通过if（"根节点下的子节点标签名".equals(parser.getName())）判断根节点下的子节点标签名与解析器对象.getName()方法获取的节点名称是否相同（注意：如果需要通过集合输出对象储存那么在这个判断加入创建对象，添加集合元素）。然后继续if（"name".equals(parser.getName())）判断子节点下其它节点与获取节点名是否相同，在通过解析器对象.nextText()方法获取数据。

8、标签判断结束之后eventType = parser.next();获取下一个元素事件

文件解析的定义

```
public class Pull_Xml {
    @Test
    public void jiexi() throws Exception{
        //创建集合对象
        ArrayList<Book> list = new ArrayList<Book>();
        //获取解析器工厂
        XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
        //获取解析器
        XmlPullParser parser = factory.newPullParser();
        //将要解析的XML文件输入解析器
        parser.setInput(new FileInputStream("bin/Newfile.xml"), "UTF-8");
        //定义文件解析元素事件
        int eventType = parser.getEventType();
        //创建对象延迟加载
        Book book = null;
        //通过元素事件，开始循环捕捉文件数据，只要元素事件不是文档结束
        while(eventType!=parser.END_DOCUMENT){
            //判断标签开始
            if(eventType==parser.START_TAG){
                //如果标签是根节点下的子节点
                if("o1".equals(parser.getName())){
                book = new Book();
                list.add(book);
                }
```

```
                if("name".equals(parser.getName())){
                book.setName(parser.nextText().trim());
                }if("sex".equals(parser.getName())){
                book.setSex(parser.nextText().trim());
                }if("year".equals(parser.getName())){
                book.setYear(Integer.parseInt(parser.nextText().trim()));
                }if("age".equals(parser.getName())){
                book.setAge(Integer.parseInt(parser.nextText().trim()));
                }
            }
            eventType = parser.next();

        }
        System.out.println(list);
    }
}
```

Book类的定义

```
package other.test;

public class Book {
    private String name;
    private String sex;
    private int year;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getSex() {
        return sex;
    }
    public void setSex(String sex) {
        this.sex = sex;
    }
    public int getYear() {
        return year;
    }
    public void setYear(int year) {
        this.year = year;
    }


    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    @Override
    public String toString() {
        return "name:" + name + ", sex:" + sex + ", year:" + year
                + ", age:" + age ;
    }
    public Book() {
        super();

    }
    public Book(String name, String sex, int year, int age) {
        super();
        this.name = name;
        this.sex = sex;
        this.year = year;
        this.age = age;
    }
```

XML文件的定义

Pull_Xml.java | Book.java | NewFile.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<o>
    <o1>
        <name>
        小明
        </name>
        <sex>
        男
        </sex>
        <year>
        1999
        </year>
        <age>
        25
        </age>
    </o1>
    <o1>
        <name>
        大明
        </name>
        <sex>
        女
        </sex>
        <year>
        1879
        </year>
```

JSON简介

JSON(JavaScript Object Notation)，JavaScript对象表示法。JSON是一种简单的数据的格式，是一种轻量级别的数据交互格式。

是基于JS脚本的一种数据交互格式，是目前互联网最常用的简单数据交互格式

JSON的规则：

(1) 映射用冒号（":"）表示。名称:值

(2) 并列的数据之间用逗号（","）分隔。名称1:值1,名称2:值2

(3) 映射的集合（对象）用大括号（"{}"）表示。{名称1:值1,名称2:值2}

(4) 并列数据的集合（数组）用方括号("[]")表示。

```
[
    {名称1:值,名称2:值2},
    {名称1:值,名称2:值2}
]
```

(5) 元素值可具有的类型：string, number, object, array, true, false, null。

格式：

简单的字符串：String j = "{名称1:'值1',名称2:'值2',名称3:'值3',名称4:'值4'}"

用　：作为分割一组数据的映射，值必须用‘’单引，一个大括号{}　代表一条JSON数据

简单的数组：String j1 = ″[{名称1:’值1’},{名称2:’值2’},{名称3:’值3’}]″

```
//简单的json
String json = "{name:'张三',sex:'男',age:18}";
String json1 = "{'name':'张三','sex':'男','age':18}";
//json数组
String jsonArr = "[{name:'张三',sex:'男',age:18},{name:'如花',sex:'女',age:19}]";
//2. 导入JSON解析的包 org.json.jar
```

复杂**JSON**的格式

复杂的格式就是两种json格式相互组合起来形成的，开发中也是非常常见的。

{"person":[{"name":"zhangsan","addr":"beijing"},{"name":"lisi","addr":"tianjin"}] }

JSON的解析

JSONObject这个类里有一个已经定义好的HashMap键值对方式存取，所以可以通过键拿到对应的值。

1、导包，独立使用org.json第三方框架解析Json。将将包导入选中右键——>Bulid path

2、使用JSONObject类将JSON字符串变为JSON对象

3、定义一条JSON字符串

4、定义一个方法，创建JSONObject类的对象构造中传入JSON数据。

5、通过对象的get（）方法获取JSON数据。

```
//使用 JSONObject类将JSON字符串变成JSON对象
JSONObject obj = new JSONObject(json1);
System.out.println("姓名：" + obj.get("name"));
System.out.println("性别：" + obj.get("sex"));
int temp = obj.getInt("age");
System.out.println("年纪：" + obj.get("age"));
System.out.println("年纪：" + temp);

System.out.println("----------------------------");
//使用JSONArray将字符串变成JSON数组对象
JSONArray arr = new JSONArray(jsonArr);
for(int i = 0 ; i<arr.length() ; i++){
    JSONObject objtemp = arr.getJSONObject(i);
    System.out.println("姓名：" + objtemp.get("name"));
    System.out.println("性别：" + objtemp.get("sex"));
    System.out.println("年纪：" + objtemp.get("age"));
}
```

```
1  package lianxi;
2
3  import org.json.JSONObject;
4  import org.junit.Test;
5
6  public class jsons {
7      String json = "{name:'张三',age:'25'}";
8      @Test
9      public void js() throws Exception{
10         JSONObject j = new JSONObject(json);
11         Object o = j.get("name");
12         int int1 = j.getInt("age");
13         System.out.println(""+o+int1);
14     }
15 }
16
```