Java进阶2 第2天

**【学习目标】理解、了解、应用、记忆**

通过今天的学习，参训学员能够：（解释的时候说出二级目标的掌握程度）

1. **【理解】BeanUtils的概述及使用**
2. 【理解】阐述BeanUtils的的作用及用法
3. 【应用】独立编写代码使用BeanUtils封装javaBean类
4. 【应用】独立编写代码定义工具类使用BeanUtils类封装javaBean
5. **【应用】能够定义XML及约束**
6. 【理解】阐述XML的正作用
7. 【理解】阐述XML文档声明、元素声明、属性声明、注释及CDATA区书写格式
8. 【理解】阐述XML约束的作用
9. 【理解】阐述XML中DTA约束的作用使用格式
10. 【理解】阐述DTD约束中文档声明、元素声明、属性声明的语法格式
11. 【理解】阐述Schema约束的作用及使用格式
12. 【理解】阐述Schema的命名空间定义格式
13. 【应用】编写代码定义Schema命名空间
14. **【应用】编写代码解析XML文件**
15. 【理解】阐述DOM解析XML的原理
16. 【应用】编写代码实现使用dom4j解析XML文件

# BeanUtils的概述及使用

## BeanUtils的概述和简单使用

### BeanUtils概述

BeanUtils 是 Apache commons组件的成员之一，主要用于简化JavaBean封装数据的操作。它可以给JavaBean封装一个字符串数据，也可以将一个表单提交的所有数据封装到JavaBean中。

### 需要的jar包



BeanUtils工具常用工具类有两个：BeanUtils、ConvertUtils。BeanUtils用于封装数据，ConvertUtils用于处理类型转换

常用方法：

|  |  |
| --- | --- |
| setProperty(Object obj,String name,Object value) | 设置属性值，如果指定属性不存在，不抛异常 |
| getProperty(Object obj,String name) | 获得属性值，如果指定属性不存在，抛方法找不到异常 |

### 案例代码一

**使用BeanUtils把数据封装进User对象:**

|  |
| --- |
| **package** com.igeek\_01;  **import** org.apache.commons.beanutils.BeanUtils;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: BeanUtilsDemoTest  \* **@Description**: BeanUtils的测试类  \* **@date** 2018年2月7日 上午9:01:07  \* Company www.igeekhome.com  \*  \* BeanUtils工具常用工具类有两个：BeanUtils、ConvertUtils。  \* BeanUtils用于封装数据，ConvertUtils用于处理类型转换  \*  \* 常用方法：  \* setProperty(Object obj,String name,Object value)：  \* 设置属性值，如果指定属性不存在，不抛异常  \*  \* getProperty(Object obj,String name)：  \* 获得属性值，如果指定属性不存在，抛方法找不到异常  \*  \*/  **public** **class** BeanUtilsDemoTest {    /\*\*  \* **@Title**: testDemo01  \* **@Description**: 模拟向User对象封装数据  \*/  @Test  **public** **void** testDemo01(){  User user = **new** User();  user.setUid("u001");  user.setUserName("jack");  user.setPassword("1234");    System.***out***.println(user);  }    /\*\*  \* **@Title**: testDemo02  \* **@Description**: 使用BeanUtils工具类封装User对象数据  \* **@throws** Exception  \*/  @Test  **public** **void** testDemo02() **throws** Exception{  /\* 实际开发中，需要封装到JavaBean中的数据存放在配置文件  \* 内容：uid=u001 、userName=jack  \*  \* JavaBean中两个概念：字段Field、属性Property  \* 字段：就是成员变量。  \* 例如：private String userName;  \* 字段名称：userName  \*  \* 属性：通过getter或setter方法推敲获得  \* 获得方式：省略set或get前缀，首字母小写  \* 例如：getUserName -->UserName -->userName  \*  \* 注意：一般情况下，字段名称和属性名称一致的。  \*  \* 特殊情况  \* private String description; //字段名：description  \* public String getDesc(){ //属性名：desc  \* return description;  \* }  \*/    //1、BeanUtils工具类，提供setProperty给指定属性设置内容和getProperty获得指定属性的值  User user = **new** User();  System.***out***.println("封装前：" + user);    //封装数据来自配置文件 , 底层其实user.setUid("u002")  BeanUtils.*setProperty*(user, "uid", "u002");  BeanUtils.*setProperty*(user, "userName", "rose");    System.***out***.println("封装后：" + user);    //2、 获得属性值  String userName = BeanUtils.*getProperty*(user, "userName");  System.***out***.println(userName);    }  } |

**User类:**

|  |
| --- |
| **package** com.igeek\_01;  **import** java.util.Arrays;  /\*\*  \* **@ClassName**: User  \* **@Description**: 用户类  \* **@date** 2018年2月7日 上午9:02:30  \* Company www.igeekhome.com  \*  \*/  **public** **class** User {  /\*\*  \* **@Fields** uid : 用户id  \*/  **private** String uid;  /\*\*  \* **@Fields** userName : 用户名  \*/  **private** String userName;  /\*\*  \* **@Fields** password : 密码  \*/  **private** String password;  /\*\*  \* **@Fields** hobbies : 爱好  \*/  **private** String[] hobbies;  /\*\*  \* **@Fields** age : 年龄  \*/  **private** **int** age;  /\*\*  \* **@Title**: User  \*/  **public** User() {  **super**();  }  /\*\*  \* **@return** the uid  \*/  **public** String getUid() {  **return** uid;  }  /\*\*  \* **@param** uid the uid to set  \*/  **public** **void** setUid(String uid) {  **this**.uid = uid;  }  /\*\*  \* **@return** the userName  \*/  **public** String getUserName() {  **return** userName;  }  /\*\*  \* **@param** userName the userName to set  \*/  **public** **void** setUserName(String userName) {  **this**.userName = userName;  }  /\*\*  \* **@return** the password  \*/  **public** String getPassword() {  **return** password;  }  /\*\*  \* **@param** password the password to set  \*/  **public** **void** setPassword(String password) {  **this**.password = password;  }  /\*\*  \* **@return** the hobbies  \*/  **public** String[] getHobbies() {  **return** hobbies;  }  /\*\*  \* **@param** hobbies the hobbies to set  \*/  **public** **void** setHobbies(String[] hobbies) {  **this**.hobbies = hobbies;  }  /\*\*  \* **@return** the age  \*/  **public** **int** getAge() {  **return** age;  }  /\*\*  \* **@param** age the age to set  \*/  **public** **void** setAge(**int** age) {  **this**.age = age;  }  /\*\*  \* **@Title**: toString  \* **@Description**: User类的toString方法  \* **@return**  \* **@see** java.lang.Object#toString()  \*/  @Override  **public** String toString() {  **return** "User [uid=" + uid + ", userName=" + userName + ", password=" + password + ", hobbies="  + Arrays.*toString*(hobbies) + ", age=" + age + "]";  }  } |

## BeanUtils的populate方法的使用

### 方法说明

BeanUtils类的populate(Object bean, Map<String,String[]> properties)：

将Map数据封装到指定Javabean中，一般用于将表单的所有数据封装到javabean。

### 案例代码二

**使用populate方法将数据封装到User对象中:**

|  |
| --- |
| **package** com.igeek\_02;  **import** java.util.HashMap;  **import** java.util.Map;  **import** org.apache.commons.beanutils.BeanUtils;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: BeanUtilsDemoTest  \* **@Description**: BeanUtils的测试类  \* **@date** 2018年2月7日 上午9:16:02  \* Company www.igeekhome.com  \*  \* BeanUtils类的populate(Object bean, Map<String,String[]> properties)：  \* 将Map数据封装到指定Javabean中，一般用于将表单的所有数据封装到javabean。  \*/  **public** **class** BeanUtilsDemoTest {    /\*\*  \* **@Title**: testDemo01  \* **@Description**: 使用populate进行填充数据方法1  \* **@throws** Exception  \*/  @Test  **public** **void** testDemo01() **throws** Exception{  //1.模拟数据，创建一个Map，填充需要的数据  Map<String,String[]> map = **new** HashMap<String,String[]>();  map.put("uid", **new** String[]{"u001"});  map.put("userName", **new** String[]{"jack","rose"});  map.put("password", **new** String[]{"1234"});    //2.将使用populate进行填充  User user = **new** User();  BeanUtils.*populate*(user, map);    System.***out***.println(user);  }    /\*\*  \* **@Title**: testDemo02  \* **@Description**: 使用populate进行填充数据方法2  \* **@throws** Exception  \*/  @Test  **public** **void** testDemo02() **throws** Exception{  //1.模拟数据，创建一个Map，填充需要的数据  Map<String,String[]> map = **new** HashMap<String,String[]>();  map.put("uid", **new** String[]{"u001"});  map.put("userName", **new** String[]{"jack","rose"});  map.put("password", **new** String[]{"1234"});  map.put("hobbies", **new** String[]{"敲代码","早上敲代码","晚上敲代码"});  map.put("age", **new** String[]{"123"});    //2.将使用populate进行填充  // 可以向指定的属性，填充一组数据。需要类型必须是[]  // 如果属性不是数组，将使用map.value表示数组中的第一个。  // BeanUtils支持类型：基本类型和基本类型对应的包装类，自动将字符串转换基本类型。  User user = **new** User();  BeanUtils.*populate*(user, map);    System.***out***.println(user);  }  } |

## BeanUtils自定义工具类

### 自定义工具类的基本使用

### 案例代码三

**1.定义静态的一个工具类，定义方法populate，参数为Object 和Map<String,String[]>，返回值类型为void**

|  |
| --- |
| **package** com.igeek\_03;  **import** java.util.Map;  **import** org.apache.commons.beanutils.BeanUtils;  /\*\*  \* **@ClassName**: MyBeanUtils  \* **@Description**: 自定义的JavaBean工具类  \* **@date** 2018年2月7日 上午9:24:43  \* Company www.igeekhome.com  \*  \*/  **public** **class** MyBeanUtils {  /\*\*  \* **@Title**: populate  \* **@Description**: 自动填充  \* **@param** bean  \* **@param** properties  \*/  **public** **static** **void** populate(Object bean, Map<String,String[]> properties){  **try** {  BeanUtils.*populate*(bean, properties);  } **catch** (Exception e) {  //将编译时异常，转换成运行时，方便调用者(使用者不需要再次处理异常)  **throw** **new** RuntimeException(e);  }  }  } |

**测试代码**

|  |
| --- |
| **package** com.igeek\_03;  **import** java.util.HashMap;  **import** java.util.Map;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: MyBeanUtilsDemoTest  \* **@Description**: 自定义JavaBean工具类的测试类  \* **@date** 2018年2月7日 上午9:27:02  \* Company www.igeekhome.com  \*  \*/  **public** **class** MyBeanUtilsDemoTest {  /\*\*  \* **@Title**: testDemo01  \* **@Description**: 使用自定义的populate方法进行填充数据  \*/  @Test  **public** **void** testDemo01() {  //1.模拟数据，创建一个Map，填充需要的数据  Map<String,String[]>map = **new** HashMap<String,String[]>();  map.put("uid", **new** String[]{"u001"});  map.put("userName", **new** String[]{"jack","rose"});  map.put("password", **new** String[]{"1234"});    //2.将使用populate进行填充  User user = **new** User();  MyBeanUtils.*populate*(user, map);    System.***out***.println(user);    }    /\*\*  \* **@Title**: testDemo02  \* **@Description**: 使用自定义的populate方法进行填充数据  \*/  @Test  **public** **void** testDemo02() {  //1.模拟数据，创建一个Map，填充需要的数据  Map<String,String[]>map = **new** HashMap<String,String[]>();  map.put("uid", **new** String[]{"u001"});  map.put("userName", **new** String[]{"jack","rose"});  map.put("password", **new** String[]{"1234"});    //2.将使用populate进行填充  User user = (User) MyBeanUtils.*populate*(User.**class**, map);    System.***out***.println(user);    }  } |

**2.定义静态的一个工具类，定义方法populate，参数为Class和Map<String,String[]>，返回值类型为Object**

|  |
| --- |
| **package** com.igeek\_03;  **import** java.util.Map;  **import** org.apache.commons.beanutils.BeanUtils;  /\*\*  \* **@ClassName**: MyBeanUtils  \* **@Description**: 自定义的JavaBean工具类  \* **@date** 2018年2月7日 上午9:24:43  \* Company www.igeekhome.com  \*  \*/  **public** **class** MyBeanUtils {  /\*\*  \* **@Title**: populate  \* **@Description**: 自动填充  \* **@param** bean  \* **@param** properties  \*/  **public** **static** **void** populate(Object bean, Map<String,String[]> properties){  **try** {  BeanUtils.*populate*(bean, properties);  } **catch** (Exception e) {  //将编译时异常，转换成运行时，方便调用者(使用者不需要再次处理异常)  **throw** **new** RuntimeException(e);  }  }    /\*\*  \* **@Title**: populate  \* **@Description**: 自动填充，并返回bean对象  \* **@param** beanClass  \* **@param** properties  \* **@return**  \*/  **public** **static** Object populate(Class beanClass, Map<String,String[]> properties){  **try** {  //1 使用反射进行实例化  Object bean = beanClass.newInstance();  //2 填充数据  BeanUtils.*populate*(bean, properties);  //3 将填充好JavaBean实例返回  **return** bean;  } **catch** (Exception e) {  //将编译时异常，转换成运行时，方便调用者(使用者不需要再次处理异常)  **throw** **new** RuntimeException(e);  }  }  } |

**测试类：**

|  |
| --- |
| **package** com.igeek\_03;  **import** java.util.HashMap;  **import** java.util.Map;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: MyBeanUtilsDemo  \* **@Description**: 自定义JavaBean工具类的测试类  \* **@date** 2018年2月7日 上午9:27:02  \* Company www.igeekhome.com  \*  \*/  **public** **class** MyBeanUtilsDemo {  /\*\*  \* **@Title**: demo01  \* **@Description**: 使用自定义的populate方法进行填充数据  \*/  @Test  **public** **void** demo01() {  //1.模拟数据，创建一个Map，填充需要的数据  Map<String,String[]>map = **new** HashMap<String,String[]>();  map.put("uid", **new** String[]{"u001"});  map.put("userName", **new** String[]{"jack","rose"});  map.put("password", **new** String[]{"1234"});    //2.将使用populate进行填充  User user = **new** User();  MyBeanUtils.*populate*(user, map);    System.***out***.println(user);    }    /\*\*  \* **@Title**: demo02  \* **@Description**: 使用自定义的populate方法进行填充数据  \*/  @Test  **public** **void** demo02() {  //1.模拟数据，创建一个Map，填充需要的数据  Map<String,String[]>map = **new** HashMap<String,String[]>();  map.put("uid", **new** String[]{"u001"});  map.put("userName", **new** String[]{"jack","rose"});  map.put("password", **new** String[]{"1234"});    //2.将使用populate进行填充  User user = (User) MyBeanUtils.*populate*(User.**class**, map);    System.***out***.println(user);    }  } |

### 带泛型的自定义工具类

### 代码案例四

**1.定义静态的一个工具类，定义方法populate，参数为Class<T>和Map<String,String[]>，返回值类型为T**

|  |
| --- |
| **package** com.igeek\_03\_03;  **import** java.util.Map;  **import** org.apache.commons.beanutils.BeanUtils;  /\*\*  \* **@ClassName**: MyBeanUtils  \* **@Description**: 自定义的JavaBean工具类  \* **@date** 2018年2月7日 上午9:24:43  \* Company www.igeekhome.com  \*  \*/  **public** **class** MyBeanUtils {  /\*\*  \* **@Title**: populate  \* **@Description**: 自动填充  \* **@param** bean  \* **@param** properties  \*  \* Class<T>此处的T就是一个变量，在运行时，接收具体类型。例如：User  \* 变量必须先定义在使用  \* 泛型变量的定义方式：修饰符 <变量名> 返回值  \*/  **public** **static** <T> T populate(Class<T> beanClass, Map<String,String[]> properties){  **try** {  //1 使用反射进行实例化  T bean = beanClass.newInstance();  //2 填充数据  BeanUtils.*populate*(bean, properties);  //3 将填充好JavaBean实例返回  **return** bean;  } **catch** (Exception e) {  //将编译时异常，转换成运行时，方便调用者(使用者不需要再次处理异常)  **throw** **new** RuntimeException(e);  }  }  } |

**2.测试类**

|  |
| --- |
| **package** com.igeek\_03\_03;  **import** java.util.HashMap;  **import** java.util.Map;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: MyBeanUtilsDemoTest  \* **@Description**: 自定义JavaBean工具类的测试类  \* **@date** 2018年2月7日 上午9:27:02  \* Company www.igeekhome.com  \*  \*/  **public** **class** MyBeanUtilsDemoTest {    /\*\*  \* **@Title**: testDemo01  \* **@Description**: 使用自定义的populate方法进行填充数据  \*/  @Test  **public** **void** testDemo01() {  //1.模拟数据，创建一个Map，填充需要的数据  Map<String,String[]>map = **new** HashMap<String,String[]>();  map.put("uid", **new** String[]{"u001"});  map.put("userName", **new** String[]{"jack","rose"});  map.put("password", **new** String[]{"1234"});    //2.将使用populate进行填充  User user = MyBeanUtils.*populate*(User.**class**, map);    System.***out***.println(user);    }  } |

# XML简介及约束

## XML基本使用--XML概述

### XML概述

XML全称为Extensible Markup Language，意思是可扩展的标记语言。

W3C在1998年2月发布1.0版本，2004年2月又发布1.1版本，但因为1.1版本不能向下兼容1.0版本，所以1.1没有人用。同时，在2004年2月W3C又发布了1.0版本的第三版。我们要学习的还是1.0版本！！！

标记，就是标签。例如：<a></a>

### XML的作用

* 存放数据

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

<persons>

<person id="p001">

<name>张三</name>

</person>

<person id="p002">

<name>李四</name>

</person>

</persons>

类似于java代码

class Person{

String id;

String name;

}

@Test

public void test(){

HashSet<Person> persons = new HashSet<Person>();

persons.add( new Person("p001","张三") );

persons.add( new Person("p002","李四") );

}

* 配置文件

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

<beans>

<bean id="user1" className="com.igeek\_01.User">

<property name="id" value="u001"></property>

<property name="username" value="jack"></property>

<property name="pwd" value="1234"></property>

</bean>

<bean id="user2" className="com.igeek\_01.User">

<property name="id" value="u002"></property>

<property name="username" value="rose"></property>

<property name="pwd" value="5678"></property>

</bean>

</beans>

类似于java代码

class Bean{

private String id;

private String className;

private Properties prop;

//补全set\get方法

}

@Test

public void test(){

HashSet<Bean> beans = new HashSet<Bean>();

Bean b1 = new Bean();

b1.setId("user1");

b1.setClassName("com.igeek\_01.User");

Properties prop = new Properties();

prop.setProperties("id","u001");

prop.setProperties("username","jack");

prop.setProperties("pwd","1234");

b1.setProp( prop );

Bean b2 = new Bean();

b2.setId("user2");

b2.setClassName("com.igeek\_01.User");

Properties prop = new Properties();

prop.setProperties("id","u002");

prop.setProperties("username","rose");

prop.setProperties("pwd","5678");

b2.setProp( prop );

beans.add(b1);

beans.add(b2);

}

## XML基本语法--文档声明

### XML文件要求

* + 固定的文档声明
  + 合格的元素和属性
  + 正确的注释
  + 符合要求的特殊字符编写规则
  + 文件扩展名必须为xml

### 文档声明

* XML文档声明格式：

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

1. 文档声明必须为<?xml开头，以?>结束；
2. 文档声明必须从文档的0行0列位置开始；
3. 文档声明只有2个属性：
   1. versioin：指定XML文档版本。必须属性，因为我们不会选择1.1，只会选择1.0；
   2. encoding：指定当前文档的编码。可选属性，常用值是utf-8；

## XML基本语法--元素

### 元素的概述

元素 element

<bean></bean>

1. 元素是XML文档中最重要的组成部分，

2. 普通元素的结构开始标签、元素体、结束标签组成。例如：<hello>大家好</hello>

3. 元素体：元素体可以是元素，也可以是文本，例如：<b><a>你好</a></b>

4. 空元素：空元素只有开始标签，而没有结束标签，但元素必须自己闭合，例如：<c/>

5. 元素命名：

a) 区分大小写

b) 不能使用空格，不能使用冒号:

c) 不建议以XML、xml、Xml开头

6. 格式化良好的XML文档，必须只有一个根元素。

### 案例代码五

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <人>  <男人></男人>  <女人></女人>  <女汉子></女汉子>  </人> |

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans>  <bean>  <property></property>  </bean>  <bean></bean>  </beans> |

## XML基本语法—属性

### 属性的概述

* 属性attribute

<bean id="" className="">

1. 属性是元素的一部分，它必须出现在元素的开始标签中
2. 属性的定义格式：属性名=属性值，其中属性值必须使用单引或双引
3. 一个元素可以有0~N个属性，但一个元素中不能出现同名属性
4. 属性名不能使用空格、冒号等特殊字符，且必须以字母开头

### 案例代码六

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans>  <bean id=*"xx"* className=*"xx"*>  <property name=*"xx"* value=*"xx"*></property>  </bean>  <bean></bean>  </beans> |

## XML基本语法--注释-CDATA区

### 注释

XML的注释，以“<!--”开始，以“-->”结束。注释内容会被XML解析器忽略！

### 转义字符

* 转义字符

因为很多符号已经被XML文档结构所使用，所以在元素体或属性值中想使用这些符号就必须使用转义字符，例如：“<”、“>”、“’”、“””、“&”。



### CDATA

* CDATA区

<![CDATA[

任意内容

]]>

当大量的转义字符出现在xml文档中时，会使xml文档的可读性大幅度降低。这时如果使用CDATA段就会好一些。

在CDATA段中出现的“<”、“>”、“””、“’”、“&”，都无需使用转义字符。这可以提高xml文档的可读性。

在CDATA段中不能包含“]]>”，即CDATA段的结束定界符。

### 案例代码七

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans>  <!-- 以下是配置xxx内容 -->  <!-- xxxx -->  <bean id=*"xx"* className=*"xx"*>  <property name=*"xx"* value=*"xx"*>  1 &lt; 3 &gt; &amp; &quot; &apos;  </property>  </bean>  <bean>  <![CDATA[  <bean id="xx" className="xx">  <property name="xx" value="xx" >  1 &lt; 3 &gt; &amp; &quot; &apos;  </property>  </bean>  ]]>    <![CDATA[.....]]>  </bean>  </beans> |

## XML约束--概述

在XML技术里，可以编写一个文档来约束一个XML文档的书写规范，这称之为XML约束。

常见的xml约束：DTD、Schema

目的：通过约束文档，可以编写出xml文档（具有固定格式，规定的格式）。

## XML约束--DTD概述和使用

DTD（Document Type Definition），文档类型定义，用来约束XML文档。规定XML文档中元素的名称，子元素的名称及顺序，元素的属性等。

### DTD重点要求

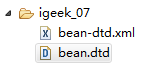
开发中，我们很少自己编写DTD约束文档，通常情况我们都是通过框架提供的DTD约束文档，编写对应的XML文档。常见框架使用DTD约束有：struts2、hibernate等。

通过提供的DTD "bean.dtd" 编写XML

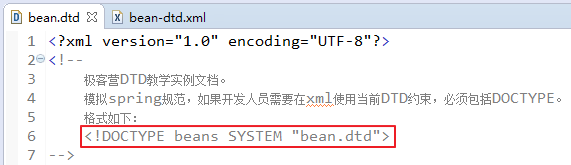
|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!--  极客营DTD教学实例文档。  模拟spring规范，如果开发人员需要在xml使用当前DTD约束，必须包括DOCTYPE。  格式如下：  <!DOCTYPE beans SYSTEM "bean.dtd">  -->  <!ELEMENT beans (bean\*,import\*)>  <!ELEMENT bean (property\*)>  <!ELEMENT property (#PCDATA)>  <!ELEMENT import (#PCDATA)>  <!ATTLIST bean id CDATA #REQUIRED  className CDATA #REQUIRED  >  <!ATTLIST property name CDATA #REQUIRED  value CDATA #REQUIRED  >  <!ATTLIST import resource CDATA #REQUIRED> |

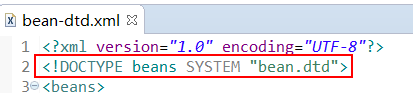
### 执行流程

* 步骤1：创建bean-dtd.xml文档，并将“bean.dtd”拷贝相同目录下。



* 步骤2：从DTD文档开始处，拷贝需要的“文档声明”





* 步骤3：完成xml内容编写
  + 需要掌握基本符号
    - , 表示两个元素之间关系，为顺序
    - | 表示两个元素之间关系，为选中
    - \* 表示当前修饰出现次数，为任意
    - + 表示当前修饰出现次数，为至少有一个（>=0）
    - ? 表示当前修饰出现次数，为0或1

### 案例代码八

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE beans SYSTEM "bean.dtd">  <beans>  <bean id=*"b001"* className=*""*>  <property name=*""* value=*""*></property>  </bean>  <bean id=*""* className=*""*></bean>  <bean id=*""* className=*""*></bean>  <bean id=*""* className=*""*></bean>    <import resource=*""*></import>  <import resource=*""*></import>  <import resource=*""*></import>  <import resource=*""*></import>  </beans> |

## XML约束--DTD语法--文档声明

1. **内部DTD，在XML文档内部嵌入DTD，只对当前XML有效。**

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

<!DOCTYPE beans [

... //具体的语法

]>

<beans>

</beans>

1. **外部DTD—本地DTD，DTD文档在本地系统上，公司内部自己项目使用。**

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

<!DOCTYPE beans **SYSTEM** "bean.dtd">

<beans>

</beans>

1. **外部DTD—公共DTD，DTD文档在网络上，一般都有框架提供。**

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

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

"http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>

</beans>

## XML约束--DTD语法--元素声明

定义元素语法：<!ELEMENT 元素名 元素描述>

元素名：自定义

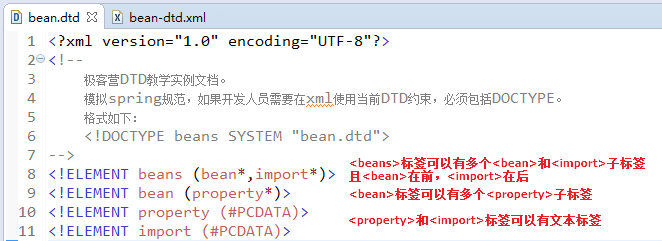
元素描述包括：符号和数据类型

常见符号：? \* + () | ,

常见类型：#PCDATA 表示内容是文本，不能是子标签



* 实例



## XML约束--DTD语法--属性声明

### 属性的语法：（attribute）

<!ATTLIST 元素名

属性名 属性类型 约束

属性名 属性类型 约束

...

>

元素名：属性必须是给元素添加，所有必须先确定元素名

属性名：自定义

属性类型：ID、CDATA、枚举 …

ID : ID类型的属性用来标识元素的唯一性

CDATA：文本类型

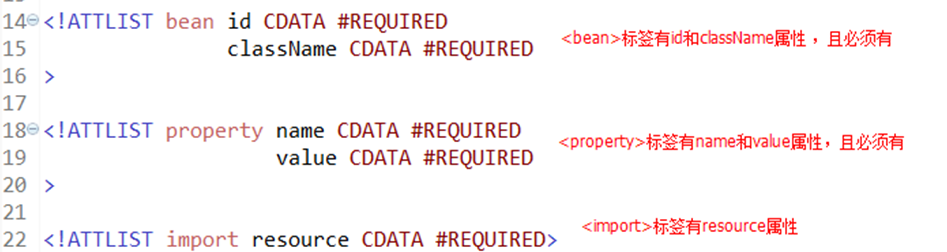
枚举：(e1 | e2 | ...) 多选一

约束：

#REQUIRED：说明属性是必须的；required

#IMPLIED：说明属性是可选的；implied

### 实例



### 案例代码九

DTD约束：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!--  极客营DTD教学实例文档。  模拟spring规范，如果开发人员需要在xml使用当前DTD约束，必须包括DOCTYPE。  格式如下：  <!DOCTYPE beans SYSTEM "bean.dtd">  -->  <!ELEMENT beans (bean\*,import\*)>  <!ELEMENT bean (property\*)>  <!ELEMENT property (#PCDATA)>  <!ELEMENT import (#PCDATA)>  <!ATTLIST bean id CDATA #REQUIRED  className CDATA #REQUIRED  >  <!ATTLIST property name CDATA #REQUIRED  value CDATA #REQUIRED  >  <!ATTLIST import resource CDATA #REQUIRED> |

XML文件：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE beans SYSTEM "bean.dtd">  <beans>  <bean id=*"b001"* className=*""*>  <property name=*""* value=*""*></property>  <property name=*""* value=*""*></property>  <property name=*""* value=*""*></property>  <property name=*""* value=*""*></property>  </bean>  <bean id=*"b002"* className=*""*></bean>  <bean id=*"b003"* className=*""*></bean>  <bean id=*"b004"* className=*""*></bean>    <import resource=*""*></import>  <import resource=*""*></import>  <import resource=*""*></import>  <import resource=*""*></import>  </beans> |

## XML约束—Schema概述

### Schema概述

Schema是新的XML文档约束；

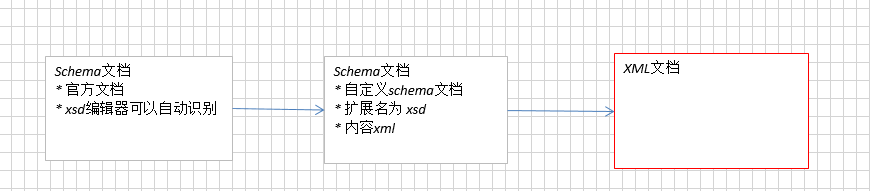
Schema要比DTD强大很多，是DTD 替代者；

Schema本身也是XML文档，但Schema文档的扩展名为xsd，而不是xml。

Schema 功能更强大，数据类型更完善

Schema 支持名称空间

### 图解



## XML约束—Schema使用

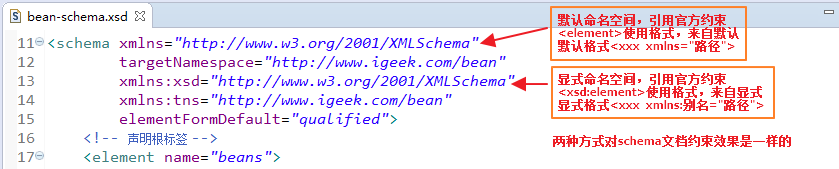
### 使用概述

与DTD一样，要求可以通过schema约束文档编写xml文档。常见框架使用schema的有：Spring等

通过提供 "bean-schema.xsd" 编写xml文档

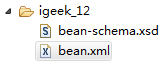
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!--  极客营Schema教学实例文档。  模拟spring规范，如果开发人员需要在xml使用当前Schema约束，必须包括指定命名空间。  格式如下：  <beans xmlns="http://www.igeek.com/bean"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.igeek.com/bean bean-schema.xsd"  >  -->  <schema xmlns=*"http://www.w3.org/2001/XMLSchema"*  targetNamespace=*"http://www.igeek.com/bean"*  xmlns:xsd=*"http://www.w3.org/2001/XMLSchema"*  xmlns:tns=*"http://www.igeek.com/bean"*  elementFormDefault=*"qualified"*>  <!-- 声明根标签 -->  <element name=*"beans"*>  <complexType>  <choice minOccurs=*"0"* maxOccurs=*"unbounded"*>  <element name=*"bean"*>  <complexType>  <sequence minOccurs=*"0"* maxOccurs=*"unbounded"*>  <element name=*"property"*>  <complexType>  <attribute name=*"name"* use=*"required"*></attribute>  <attribute name=*"value"* use=*"required"*></attribute>  </complexType>  </element>  </sequence>  <attribute name=*"id"* use=*"required"*></attribute>  <attribute name=*"className"* use=*"required"*></attribute>  </complexType>  </element>  <element name=*"import"*>  <complexType>  <attribute name=*"resource"* use=*"required"*></attribute>  </complexType>  </element>  </choice>  </complexType>  </element>  </schema> |

案例文档中同一个“命名空间”分别使用“默认命名空间”和“显示命名空间”进行引入，所以文档中<schema>和<xsd:schema>作用一样。

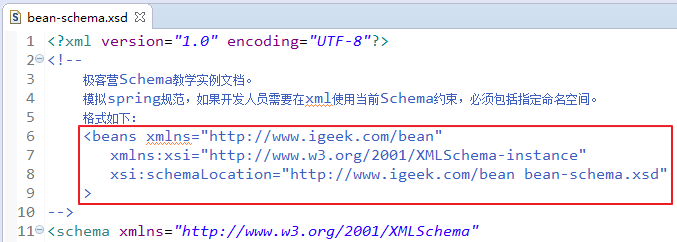


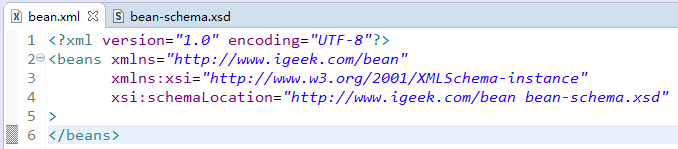
### 案例实现

步骤1：创建bean.xml，并将“bean-schema.xsd”拷贝到同级目录



步骤2：从xsd文档中拷贝需要的“命名空间”





步骤3：完成xml内容编写

### 案例代码十

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.igeek.com/bean"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.igeek.com/bean bean-schema.xsd"*  >  <import resource=*""*/>    <bean id=*""* className=*""*>  <property name=*""* value=*""*/>  </bean>    <bean id=*""* className=*""*></bean>    <import resource=*""*/>  </beans> |

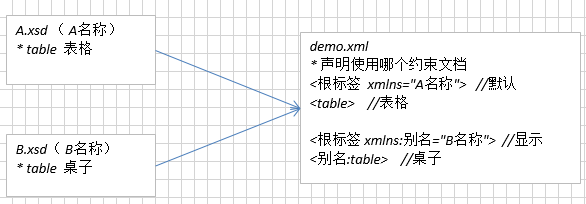
## XML约束--Schema命名空间

### 命名空间的概述

如果一个XML文档中使用多个Schema文件，而这些Schema文件中定义了相同名称的元素时就会出现名字冲突。这就像一个Java文件中使用了import java.util.\*和import java.sql.\*时，在使用Date类时，那么就不明确Date是哪个包下的Date了。

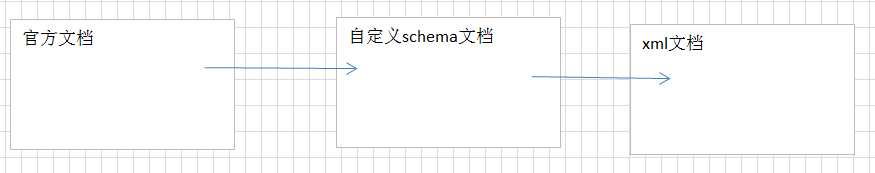
总之名称空间就是用来处理元素和属性的名称冲突问题，与Java中的包是同一用途。如果每个元素和属性都有自己的名称空间，那么就不会出现名字冲突问题，就像是每个类都有自己所在的包一样，那么类名就不会出现冲突。

### 关系图



### 约束文档和XML关系

当W3C提出Schema约束规范时，就提供“官方约束文档”。我们通过官方文档，必须“自定义schema 约束文档”，开发中“自定义文档”由框架编写者提供。我们提供“自定义文档”限定，编写出自己的xml文档。



### 声明命名空间

默认命名空间：<xxx xmlns=””> ，使用<标签>

显式命名空间：<xxx xmlns:别名=””> ， 使用<别名:标签>

### 案例代码十一

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <my:beans xmlns:my=*"http://www.igeek.com/bean"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.igeek.com/bean bean-schema.xsd"*  >  <!--  1.schema文档，必须有一个名称(namespace命名空间)  名称要求：全球唯一。开发中常用URL (http://www.igeek.com/web/v1.0.xsd)  例如：bean-schema.xsd  targetNamespace="http://www.igeek.com/bean"  2.引用指定的命名空间  默认引用  <beans xmlns="http://www.igeek.com/bean">  之后使用时<标签名>  显式引用  <beans xmlns:my="http://www.igeek.com/bean">  之后使用时<my:标签名>  3.必须确定xsd文档位置  <根标签 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="名称位置"  >    官方文档，xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  例如：  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.igeek.com/bean bean-schema.xsd"  -->  <my:bean className=*""* id=*""*></my:bean>  <my:import resource=*""*/>  </my:beans> |

**命名空间高级使用**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <my:beans xmlns:my=*"http://www.igeek.com/bean"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.igeek.com/bean bean-schema.xsd"*  >  <!--  1.一个xml文档中，只能出现一次“默认引用”  <beans xmlns="名称"  2.一个xml文档中，可以出现多次“显示引用”  <beans xmlns:a="名称a"  xmlns:b="名称b"  3.所有引用(默认+显式)，都必须确定xsd文档位置  <beans xmlns="默认名称"  xmlns:a="名称a"  xmlns:b="名称b"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="默认名称默认位置  名称a a位置  名称b b位置"  -->  </my:beans> |

## XML约束--Schema基本语法

### 案例代码十二

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!--  极客营Schema教学实例文档。  模拟spring规范，如果开发人员需要在xml使用当前Schema约束，必须包括指定命名空间。  格式如下：  <beans xmlns="http://www.igeek.com/bean"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.igeek.com/bean bean-schema.xsd"  >  -->  <schema xmlns=*"http://www.w3.org/2001/XMLSchema"*  targetNamespace=*"http://www.igeek.com/bean"*  xmlns:xsd=*"http://www.w3.org/2001/XMLSchema"*  xmlns:tns=*"http://www.igeek.com/bean"*  elementFormDefault=*"qualified"*>  <!--声明根标签  1. <element>声明元素(标签)  2. 每一个元素必须确定类型：  complexType 复杂类型  simpleType 简单类型，一般不用，大部分都是复杂类型  3. 需要继续明确子标签出差顺序  <choice>选择， ()  <sequence>顺序，  <all>任意  minOccurs最少出现次数  maxOccurs最大出现次数，unbounded不限制(没有边)  4. <attribute>用于给元素声明属性的  use设置属性使用， optional可选、required必选    -->  <element name=*"beans"*>  <complexType>  <choice minOccurs=*"0"* maxOccurs=*"unbounded"*>  <element name=*"bean"*>  <complexType>  <sequence minOccurs=*"0"* maxOccurs=*"unbounded"*>  <element name=*"property"*>  <complexType>  <attribute name=*"name"* use=*"required"*></attribute>  <attribute name=*"value"* use=*"required"*></attribute>  </complexType>  </element>  </sequence>  <attribute name=*"id"* use=*"required"*></attribute>  <attribute name=*"className"* use=*"required"*></attribute>  </complexType>  </element>  <element name=*"import"*>  <complexType>  <attribute name=*"resource"* use=*"required"*></attribute>  </complexType>  </element>  </choice>  </complexType>  </element>  </schema> |

# XML解析

## xml解析--概述和解析方式

### xml解析概述

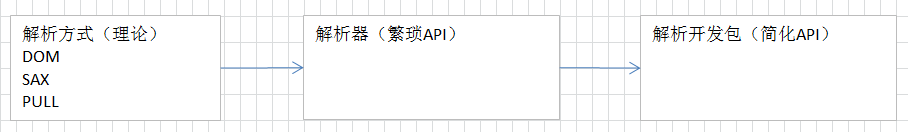
当将数据存储在XML后，我们就希望通过程序获得XML的内容。如果我们使用Java基础所学习的IO知识是可以完成的，不过你需要非常繁琐的操作才可以完成，且开发中会遇到不同问题（只读、读写）。人们为不同问题提供不同的解析方式，并提供对应的解析器，方便开发人员操作XML。

### 解析方式和解析器

开发中比较常见的解析方式有三种，如下：

1. DOM：要求解析器把整个XML文档装载到内存，并解析成一个Document对象。
   1. 优点：元素与元素之间保留结构关系，故可以进行增删改查操作。
   2. 缺点：XML文档过大，可能出现内存溢出显现。
2. SAX：是一种速度更快，更有效的方法。它逐行扫描文档，一边扫描一边解析。并以事件驱动的方式进行具体解析，每执行一行，都将触发对应的事件。（了解）
   1. 优点：处理速度快，可以处理大文件
   2. 缺点：只能读，逐行后将释放资源。
3. PULL：Android内置的XML解析方式，类似SAX。（了解）

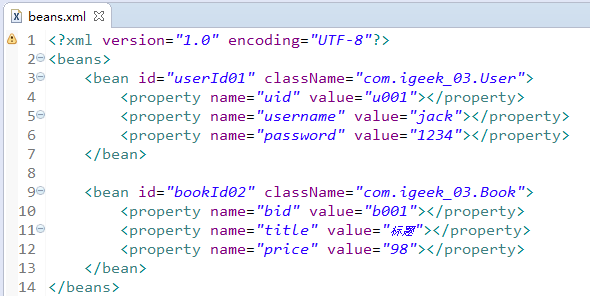
解析器：就是根据不同的解析方式提供的具体实现。有的解析器操作过于繁琐，为了方便开发人员，有提供易于操作的解析开发包。

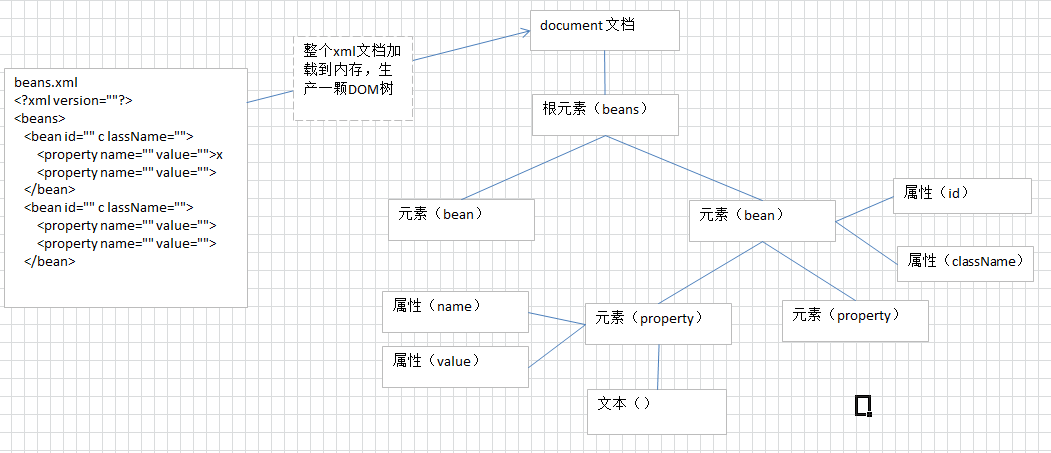


* 常见的解析开发包：
  + JAXP：sun公司提供支持DOM和SAX开发包
  + JDom：dom4j兄弟
  + jsoup：一种处理HTML特定解析开发包
  + dom4j：比较常用的解析开发包，hibernate底层采用。

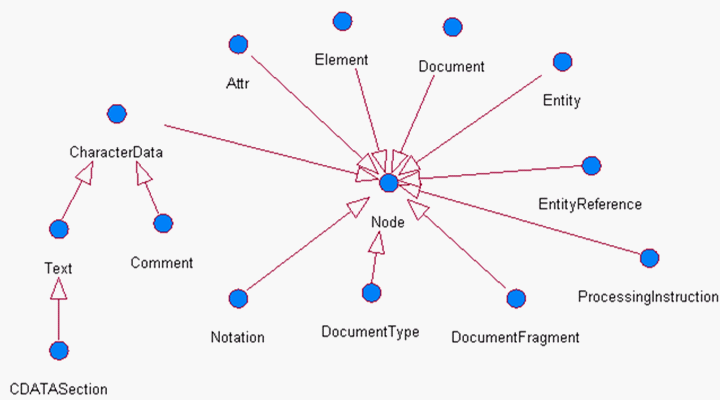
## xml解析--dom解析原理

XML DOM 将 整个XML文档加载到内存，生成一个DOM树，并获得一个Document对象，通过Document对象就可以对DOM进行操作





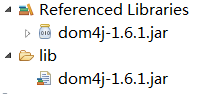
DOM中的核心概念就是节点，在XML文档中的元素、属性、文本等，在DOM中都是节点！



## xml解析--dom4j解析实现

### 准备工作

导入jar包。



### 常用API

常用API如下：

1. SaxReader对象
   1. read(…) 加载执行xml文档
2. Document对象
   1. getRootElement() 获得根元素
3. Element对象
   1. elements(…) 获得指定名称的所有子元素。可以不指定名称
   2. element(…) 获得指定名称第一个子元素。可以不指定名称
   3. getName() 获得当前元素的元素名
   4. attributeValue(…) 获得指定属性名的属性值
   5. elementText(…) 获得指定名称子元素的文本值
   6. getText() 获得当前元素的文本内容

### 案例代码十二

**Dom4j解析类**

|  |
| --- |
| **package** com.igeek\_03;  **import** java.util.List;  **import** org.dom4j.Document;  **import** org.dom4j.DocumentException;  **import** org.dom4j.Element;  **import** org.dom4j.io.SAXReader;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: Dom4jDemoTest  \* **@Description**: 使用Dom4j工具包解析xml文件  \* **@date** 2018年2月7日 下午12:33:39  \* Company www.igeekhome.com  \*  \* 解析步骤：  \* 1.核心类；  \* 2.获得Document（整个xml）；  \* 3.获得根元素 beans；  \* 4.所有子元素 bean；  \* 4.1 bean属性  \*/  **public** **class** Dom4jDemoTest {    /\*\*  \* **@Title**: testParse  \* **@Description**: 解析beans.xml文件的测试方法  \* **@throws** DocumentException  \*/  @Test  **public** **void** testParse() **throws** DocumentException{    //1.核心类  SAXReader saxReader = **new** SAXReader();    //2.获得Document（整个xml）  Document document = saxReader.read("beans.xml");    //3.获得根元素 beans  Element rootElement = document.getRootElement();    //4.所有子元素 bean  List<Element> allBeanElement = rootElement.elements("bean");  **for** (Element beanEle : allBeanElement) {  //4.1 bean属性  String id = beanEle.attributeValue("id");  String className = beanEle.attributeValue("className");  System.***out***.println("bean属性：" + id + " , " + className);    //5.子元素property  List<Element> allPropElement = beanEle.elements("property");  **for** (Element propEle : allPropElement) {  String name = propEle.attributeValue("name");  String value = propEle.attributeValue("value");  System.***out***.println("property属性：" + name + " , " + value);  }    System.***out***.println();  }  }  } |

**被解析的beans.xml**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans>  <bean id=*"userId01"* className=*"com.igeek\_03.User"*>  <property name=*"uid"* value=*"u001"*></property>  <property name=*"username"* value=*"jack"*></property>  <property name=*"password"* value=*"1234"*></property>  </bean>    <bean id=*"bookId02"* className=*"com.igeek\_03.Book"*>  <property name=*"bid"* value=*"b001"*></property>  <property name=*"title"* value=*"标题"*></property>  <property name=*"price"* value=*"98"*></property>  </bean>  </beans> |

重点和总结

1、BeanUtils的概述及使用

2、XML简介及约束

4、DTD概述和使用

5、Schema概述和使用

6、XML解析概述及解析方式

7、XML解析原理及使用dom4j工具包完成解析