# MyEclipse

配置project属性-->右键工程-->propertise

Java build path :指定当前工程引入类库

source中指定.java 文件 和.class文件 存放位置

librialies 指定当前工程引用类库

Java Compiler :指定编译器版本 JRE的版本必须高于编译器的版本.

对已经存在工程进行分组 working set 通过分组隐藏不需要使用工程

# XML

XML是一种标记语言,常用来存储数据和传输数据,xml语言与任何编程语言无关,可用于 php,ava, .net 任何编程语言,这也是XML技术流行的原因

XML语法

1. 文档声明, 任何xml必须具备文档声明

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

2. CDATA块

该块内的代码是不被解析的代码

<![CDATA[

<name></name>

]]>

1. 转义字符

&gt; &…; 别忘记分号.

1. 必须结束标签只有一个根标签
2. 空格和换行符当作内容处理
3. 区分发小写,元素和属性不能以数字开头
4. <!-- 这是注释 -->

属性定义语法 <元素名称 属性名称="属性值" 属性名称="属性值" ....>单引号或双引号

XML 技术应用

1. 传输和存储数据 ----- 程序员为了简化开发，不对xml使用约束技术

\* 不使用xml约束技术，xml 格式不一定正确

\* 所有xml 数据内容生成,解析过程,都是通过程序来完成的

2. 使用xml 作为配置文件

Android SSH ----- 程序员为了保证xml元素准确性,通常xml文档使用约束技术的

3. 为什么xml使用约束？ 使用约束xml文档 --- 元素内容格式更加规范

如果企业中xml没有使用约束 ------ 通过程序生成xml和解析xml时,不使用约束,代码更加简单

xml文档约束的两种方式

DTD约束(Document Type Definition)

Schema约束

# XML文件的解析

1. DOM解析

DOM (Document Object Model)文档对象模型,官方开发

DOM思想： 将整个xml 加载内存中，形成文档对象，所有对xml操作都对内存中文档对象进行

1. SAX和STAX解析. 基于事件驱动,回调.

SAX (Simple API for XML)XML简API,开源的框架

当xml 文档非常大，不可能将xml所有数据加载到内存

SAX 思想：一边解析,一边处理,一边释放内存资源

STAX (The Stream API for XML)是一种pull模式解析方式,SAX是一种push模式解析方式

推push模式：由服务器为主导，向客户端主动发送数据

拉pull模式: 由客户端为主导，主动向服务器申请数据

1. DOM、SAX、STAX 在实际开发中选择？

在javaee日常开发中 ,优先使用DOM （编程简单）

当xml 文档数据非常多 ,不可能使用DOM ,会造成内存溢出. 优先使用STAX

移动开发 使用 STAX Android XML PULL

1. 解析开发包

开发工具包 JAXP 、DOM4j 、XML PULL

**//DOM解析及简单演示,比较灵活,方式多!**

**public** **class** DOMTest {

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

//1. 工厂类对象-->解析器对象装载要解析的文档-->解析加载

DocumentBuilderFactory dbf = DocumentBuilderFactory

.*newInstance*();

DocumentBuilder builder = dbf.newDocumentBuilder();

Document dm = builder.parse("books.xml");

//2. Document通过 getElementsByTagName 获得 节点集合 NodeList

NodeList list = dm.getElementsByTagName("author");

//3. 通过 NodeList 提供 getLength 和 item 遍历 节点集合

System.*out*.println("name节点数量"+list.getLength());

**for** (**int** i = 0; i<list.getLength() ; i++) {

//4. 将每个节点Node强转成Element,并调用getAttribute()getTextContent获取内容

Element element = (Element) list.item(i);

System.*out*.println("属性:"+element.getAttribute("country"));

System.*out*.println("内容:"+element.getTextContent());

}

}

}

# SAX解析 通过编写一个类(解析器)结成DefaultHandler并重写 :

startDocument() 文档开始回调

startElemen() 元素开始回调

characters() 文本元素回调

endElement() 元素结束回调

endDocument() 文档结束回调

# PULL解析

java使用pull需要导入jar包 :导入方法,将jar文件导入lib目录,并在工程属性的java build path中导入 .导入成功工程下会有referenced libraries文件.

**/\***

**一个节点一个节点的读，包括换行（两个字符/n/t）和空格！！！**

**事件静态常量：** *START\_DOCUMENT ,START\_TAG ,END\_TAG, END\_DOCUMENT*

**解析器的方法:** setInput() 设置源文件.

getEventType() 获取解析事件

getName() 获取节点名称

next() 解析下一个节点事件

**\*/**

**private** **static** **void** xmlpuu() **throws** Exception{

XmlPullParserFactory f = XmlPullParserFactory.*newInstance*();

XmlPullParser p = f.newPullParser();

p.setInput(**new** FileInputStream("books.xml"),"utf-8");

//开始解析某个节点,并记录节点,判断取出.

**int** eventType = p.getEventType();

**while**(eventType!=XmlPullParser.*END\_DOCUMENT*){

//判断事件

**switch** (eventType) {

**case** XmlPullParser.*START\_TAG*:

**if** ("name".equals(p.getName())) {

//nextText()将节点移动到下一个结束节点

System.*out*.println(p.nextText());

}**else** **if** ("id".equals(p.getName())) {

System.*out*.println(p.nextText());

}**else** **if** ("price".equals(p.getName())) {

System.*out*.println(p.nextText());

}**else** **if** ("author".equals(p.getName())) {

System.*out*.println(p.nextText());

}

**break**;

**case** XmlPullParser.*END\_TAG*:

**break**;

**default**:

**break**;

}

eventType= p.next();

}

}