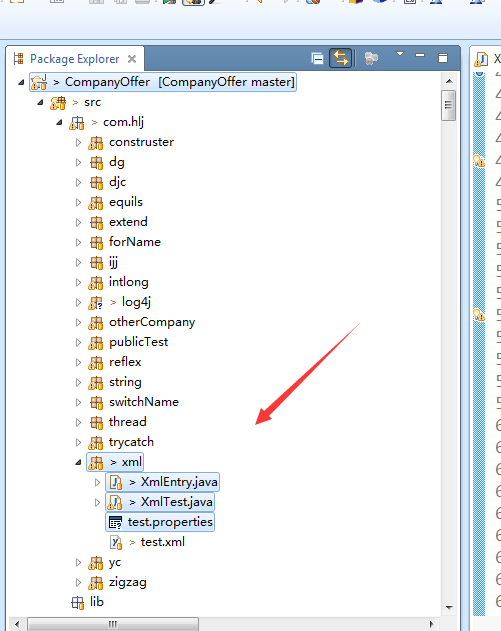
# 1、项目位置



# 2、先编写 properties文件，前面为xml文件字段，后面为实体类java字段

|  |
| --- |
| #explain \u524D\u9762\u662Fxml\u4E2D\uFF0C\u540E\u9762\u662F\u5B9E\u4F53\u7C7B\u4E2D\uFF0C\u5176\u4E2D\u6709\u7684\u6570\u636E\u6CA1\u6709\u7528\u5230\u4E0D\u7528\u7406\u4F1A  xuexiao=school  people=person  peoples=persons  sex=sex  persons=persons  name=name  age=age  banji=class  salary=salary  classPersonNum=classPersonNum |

# 3、编写实体类

|  |
| --- |
| **public** **class** XmlEntry {  **public** String person ;  **public** Integer age;  **public** Integer sex;  **public** String name ;  **public** **double** salary; //薪水  **public** Integer classPersonNum;  **public** XmlEntry() {  }  **public** String getPerson() {  **return** person;  } |

# 4、编写xml文件

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <xuexiao> <!-- firstElement -->  <sex>2</sex>  <people> <!-- twoElement -->  <name>张宇晋</name>  <age>23</age>  <salary>4992.2</salary>  </people>  <banji>  <classPersonNum>54</classPersonNum>  </banji>  </xuexiao> |

## 5、编写解析代码

## 解决思路，最重要的就是分析好xml的结构，并且利用好配置文件以及Field反射

|  |
| --- |
| //解析xml  **public** **class** XmlTest {  **static** Logger *logger* = Logger.*getLogger*(XmlEntry.**class**);    **public** **static** **void** main(String[] args) {  XmlTest xmlTest = **new** XmlTest();  xmlTest.paraseXml(**new** XmlEntry());  }  **public** String paraseXml(XmlEntry xmlEntry){  *logger*.info(getPropFileByBusCd().toString());  **try** {  SAXReader reader=**new** SAXReader();  String fileUrl = XmlTest.**class**.getResource("test.xml").getFile();  File fileXml = **new** File(fileUrl);  Document document = reader.read(fileXml);  //取得根节点  Element rootElement = document.getRootElement();    //反射取得字段名  Field[] fields = XmlEntry.**class**.getDeclaredFields();  //xml与实体类对应配置文件  Properties pop = getPropFileByBusCd();    Iterator<Element> firstElement=rootElement.elementIterator();  **while**(firstElement.hasNext()){ //第一层  Element twoElement = firstElement.next();  String tagele = twoElement.getName(); //取得标签的名字  String valueTxt = twoElement.getText(); //取得标签的内容    **if**(!twoElement.isTextOnly()){ //判断是不是仅仅是一个文本，比如sex仅仅是一个文本，则不执行下面的  Iterator<Element> twonowElement = twoElement.elementIterator();  **while**(twonowElement.hasNext()){  Element threeElement = twonowElement.next();  String threetagele = threeElement.getName(); //取得标签的名字  String threevalueTxt = threeElement.getText(); //取得标签的内容  //进行反射以及配置类文件的生效制作实体类  getXmlEntry(xmlEntry, fields, pop, threetagele, threevalueTxt);  }  }**else**{  getXmlEntry(xmlEntry, fields, pop, tagele, valueTxt);  }      }  System.*out*.println(xmlEntry.toString());    } **catch** (DocumentException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  **return** **null**;  }    //最重要的代码  **private** **void** getXmlEntry(XmlEntry xmlEntry, Field[] fields, Properties pop,  String tagele, String valueTxt) {  **for**(**int** i=0;i<fields.length;i++){  Field field=fields[i]; //取得反射的数组  String tageleEntry = pop.getProperty(tagele);  **if**(tageleEntry.toUpperCase().equals(field.getName().toUpperCase())){  *logger*.info(valueTxt);  setFieldValue(field, xmlEntry, valueTxt);  **break**;  }  }  }  /\*\*  \* 根据业务类型获得相应的属性文件  \* **@param** busCd  \* **@return**  \*/  **private** Properties getPropFileByBusCd(){    **try** {  File file=**new** File(XmlTest.**class**.getResource("test.properties").getFile());  InputStream is;  is = **new** FileInputStream(file);  Properties pro=**new** Properties();  pro.load(is);  **return** pro;    } **catch** (FileNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }    **return** **null**;  }  /\*\*  \* 为属性赋值  \* **@param** field 要赋值的属性  \* **@param** ins 属性所属对象  \* **@param** value 值  \* **@throws** IllegalAccessException  \* **@throws** IllegalArgumentException  \* **@throws** ParseException  \*/  **private** **void** setFieldValue(Field field,XmlEntry ins,String value){  **try**{  **if**(field==**null** || ins==**null** || value==**null** || "".equals(value)){  **return**;  }  field.setAccessible(**true**);    String fieldType=field.getGenericType().toString();  **if**("class java.lang.String".equals(fieldType)){  field.set(ins, value);  }  **if**("class java.lang.Integer".equals(fieldType)){  Integer val=Integer.*valueOf*(value);  field.set(ins, val);  }  **if**("class java.math.BigDecimal".equals(fieldType)){  BigDecimal bde=**new** BigDecimal(value);  field.set(ins, bde);  }  **if**("class java.util.Date".equals(fieldType)){  SimpleDateFormat df=**new** SimpleDateFormat("yyyyMMdd");  Date date=df.parse(value);  field.set(ins, date);  }  **if**("class java.lang.Double".equals(fieldType)){  SimpleDateFormat df=**new** SimpleDateFormat("yyyyMMdd");  Date date=df.parse(value);  field.set(ins, date);  }  }**catch**(Exception ex){  *logger*.error(ex);  }  }      } |