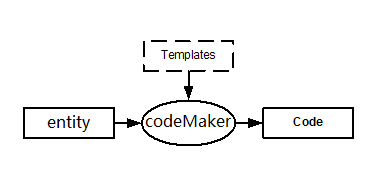
# CodeMaker V1.0

# 使用说明

2013.8.29

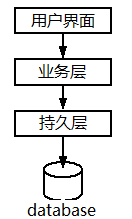
#### 简介

系统开发中往往涉及到很多相似或重复的工作，比如学生管理、教师管理，都存在增删改查的操作，CodeMaker将这种共同的部分（如增删改查）称为模板（Template），而学生和教师被看做为实体或模型（Entity或Model），针对于每一个实体，可以根据实体的定义和增删改查的模板自动生成各自的增删改查页面。原理如下：



*图1 代码生成过程*

JavaWeb开发中一般设计到如下几个层次的开发：



用户界面中采用诸如Extjs、JQueryUI、YUI、DWZ等技术。业务逻辑的处理上可以使用Servlet、Struts、SpringMVC。持久层有Hibernate、ibatis等技术。CodeMaker并不依赖于这些层次关系，这些层次的划分都是通过你的具体配置实现的。CodeMaker可以很快速的生成以上各层的代码。主要包括：

* 添加、修改、查看、列表页面JSP页面
* Action或Controller代码
* Service层和Dao层代码
* JavaBean定义代码
* 基于JPA的ORM代码
* Hibernate Xml形式的ORM代码
* 其他与实体相关的文件

基于以上的功能，对于某一个业务对象，你只需要做好业务对象的定义即可，CodeMaker帮你实现一下功能：

* + 增删改
  + 日期选择
  + 分页
  + 条件查询
  + 字段验证（必填项、数字、长度）
  + 复杂对象关系管理（一对一，多对一，多对多）
  + 支持多种数据类型，字符串、整数、小数、复杂类型、枚举类型

#### 特性

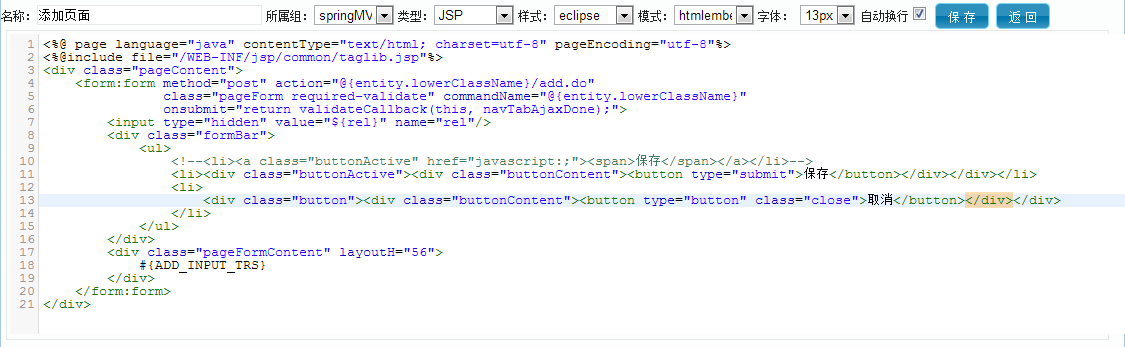
1、基于模板技术，搜集整理你的最佳实践，避免重复造轮子；

2、最佳实践搜集越多，越有利于反过来对系统进行扩展，系统功能越强大；

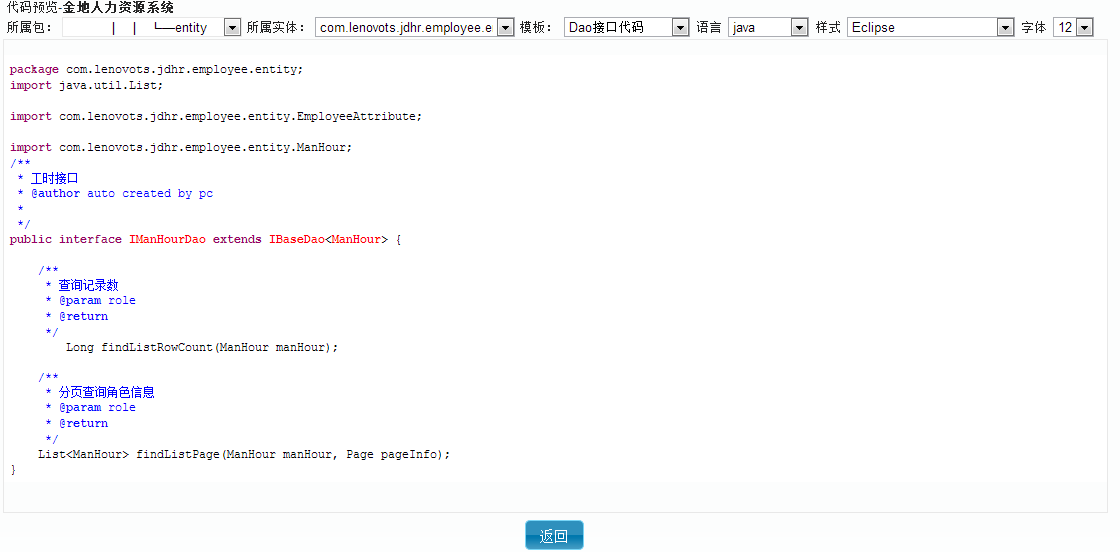
3、内置模板中已经支持增删改查页面、实体对象、ORM、数据库文件的自动生成、文件上传下载、Excel文件导入导出、一对多关联关系生成、简单业务逻辑生成；

4、对象管理关系支持；

5、在线代码编辑；



6、目标代码在线预览；



7、灵活的映射机制，可以选则对部分实体生成对应的目标代码；



#### 特别说明

CodeMaker v1.0目前仅支持系统部分代码生成，不支持整站的生成，不要希望CodeMaker能够把系统的方方面面的代码都生成完。你必须把基础的项目结构搭建好，然后让CodeMaker为你生成具有重复性质的代码，如增删改查页面和代码。CodeMaker适合于你有一套很完善的最佳实践，你可以将这些最佳实践都抽象成一堆模板，有需求的时候，你只需要做好实体定义，CodeMaker根据模板帮你自动生成这些最佳实践的代码。

CodeMaker并不依赖于开发语言、数据库和开发框架。无论是Java，还是PHP，还是Python，还是别的任何语言，只要是属于模型驱动的问题分析范畴，CodeModer都能帮你生成对应的语言的代码。

未来的CodeMaker还将支持工作流、统计报表等功能。CodeMaker的模板整理过程，就是对最佳实践总结的过程，随着CodeMaker的使用，你会发现，越来越多的最佳实践被积累了下来，你可以节省大量的重复劳动时间，大大地提高开发效率。

#### 开始使用

###### Step1: 创建项目

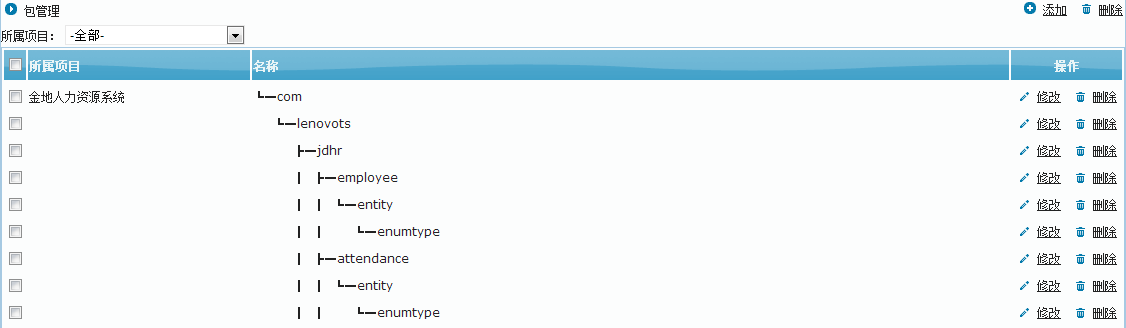
点击菜单，进入项目管理页面，添加一个项目：



*图2 项目管理界面*

###### Step2：管理包结构

点击进入，进入包管理页面，像如图所示那样建立好你的包结构。



*图3 包管理*

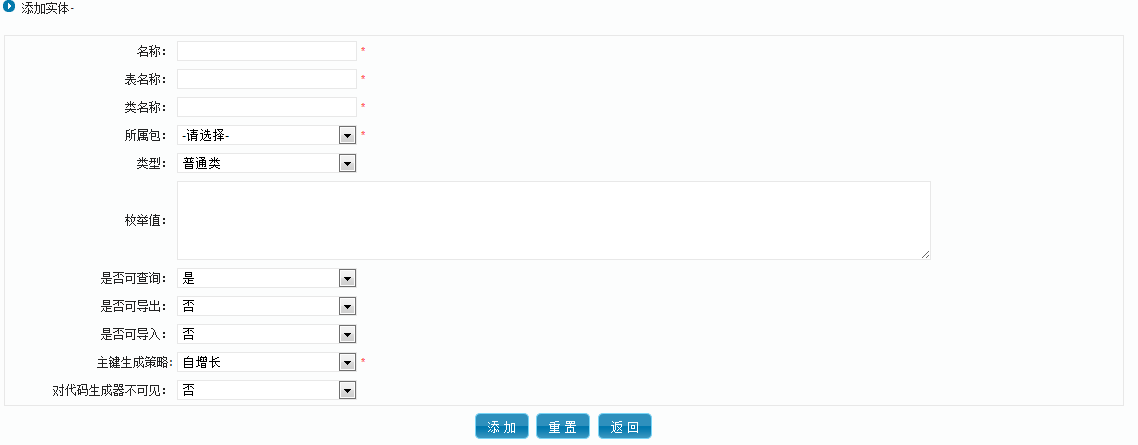
###### Step3：实体管理

接下来你需要在上一步中创建的包中创建实体（Entity），点击进入实体管理界面。



*图4 实体管理*

点击连接，进入实体添加页面：



*图5 实体定义*

CodeMaker支持批量导入，点击进入Excel导入页面：



*图6实体批量导入*

###### Step4：字段管理

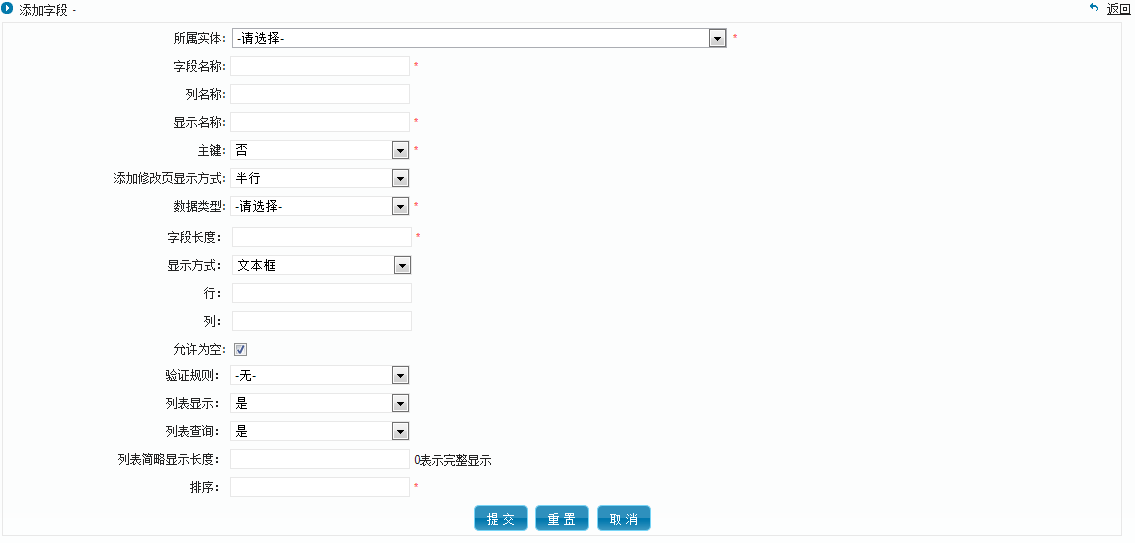
实体创建完毕之后，就需要为各个实体定义好字段了，这是一个比较关键和繁琐的步骤，你需要有点耐心。字段管理页面如下：



*图7 字段管理*

和实体一样，除了增删改查页面，也支持批量导入功能。

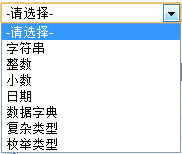
这里要特别说明的字段定义界面：



*图9 字段定义*

字段名称就是Java中类的属性名称，列名称是指该属性映射到数据库的列的名称，不填的话就是字段名称。显示名称是指字段的意义名称，如“姓名”、“性别”。

目前系统支持的数据类型如下：



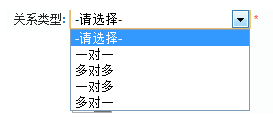
*图10 系统支持的数据类型*

当选择日期的时候，需要输入日期的格式：



*图11 日期类型格式*

当选择复杂类型的时候，需要选择关联的实体以及与关联实体之间的关联关系：



*图12 复杂类型关联关系*

当选择一对一、多对一关联关系的时候，需要填写value path值，该值表示关联关系对方的显示值，比如，学生多对一于教师，那么value path中应该填写“name”，表示在学生列表的教师列中显示的是教师的“name”字段；

当选择多对多的时候，需要填写中间表表名称；

当选择一对多的时候，关联关系对方关系外键id需要填写“关联关系对方关系外键id”。并且将“关系字段”设置为“是”。

###### Step5：标签管理

标签管理页面如下：



*图13 标签管理*

标签分为内置标签和自定义标签，分别用不同的图标标注。需要说明的是，内置标签就是实体、字段的相关属性。标签管理中的内置标签仅仅算做是一个文档说明，在以后的版本中，内置标签会自动生成。

自定义标签是需要用户手动添加进去的，它其实是模板的另外一种表现形式，自定义标签可以被重复的在模板中使用。

可以自定义标签的标签值和模板的内容中使用JSTL标签和EL表达式。作为Java开发工具，我们认为你已经了解这方面的内容了。

###### Step6 ：定义模板组和模板

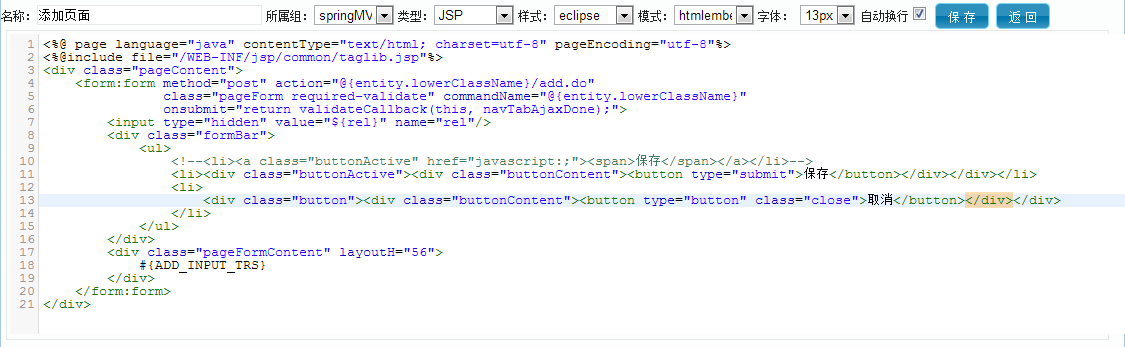
点击进入模板组管理页面，可以对模板组进行增删改操作。

点击进入模板管理界面：



*图14 模板管理*

添加页面如下：



*图15 模板定义*

模板内容是由不变量与变量组成的，变量都是标签中的内容（标签管理详见下节）。标签分为内置标签和自定义标签，模板类型目前分为JSP和Java两种。在这两种模板中访问内置标签和自定义标签使用的是不同的前缀：

|  |  |  |
| --- | --- | --- |
| **标签类型** | **Java** | **JSP** |
| 内置 | $ | @ |
| 自定义 | # | # |

*表1 访问内置标签的前缀符号*

###### Step7：映射规则管理

实体和模板都有了，根据不同的项目，需要定义根据模板生成的文件的输出位置，这就是映射规则了：



*图16 映射规则管理*

修改页面如下：



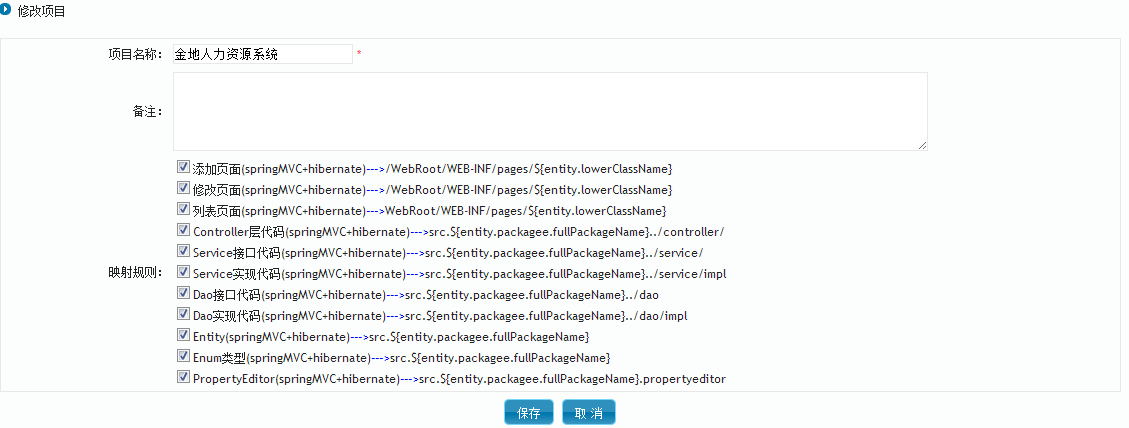
*图17 映射规则定义*

输入模板中选择一个之前定义好的模板。输出位置中填写相对于项目根路径的一个路径地址，你可以在这个路径值中使用EL表达式，访问实体的相关属性。输出文件名称同理，表示的是输出到输出位置的文件的名称。另外，输出位置中还可以使用../来表示上一级目录，当然，你可以使用多个../符号来表示更多的上一级。如上图所属，表示输出位置为实体包所在目录的上一级目录的service目录下的impl目录。文件名称是实体的Class名称+ServiceImpl.java。

查询条件其实是对Entity的一个过滤条件，如上图所示表示的是这条规则仅适用于Entity的type属性为0的记录（type为0表示Entity为“Class”，type为1表示Entity为“枚举”）。目前查询条件仅支持多个“与”关系的条件，如果有多个条件，需要用分号（;）连接。在以后的版本中将会支持更多的过滤条件（如分组、或者、否定等）。

###### Step8：代码预览、生成和下载

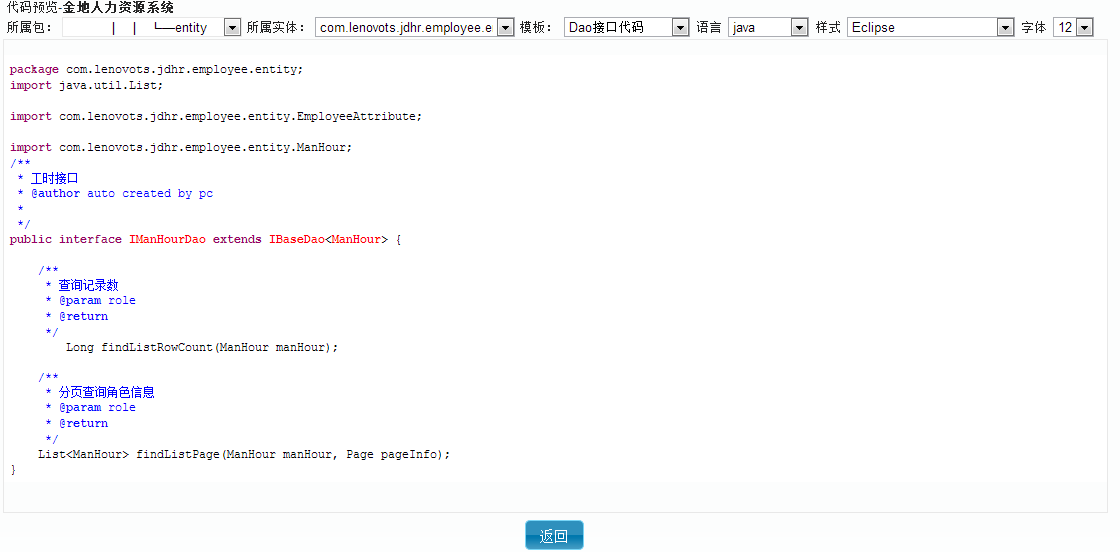
终于可以看到生成的代码了。进入项目管理，点击，勾选需要的映射规则：



*图18 配置项目映射规则*

这表示当前这个项目，需要使用到这些映射规则，即每一个实体会按照所选的映射规则生成一个文件。

点击保存，回到项目管理列表，然后点击，进入代码预览界面，选择要预览的实体和模板，预览代码如下：



*图19 项目代码预览*

从新回到项目管理，点击，进入代码文件查看页面：



*图20 项目代码查看*

点击，系统开始根据映射规则生成代码：

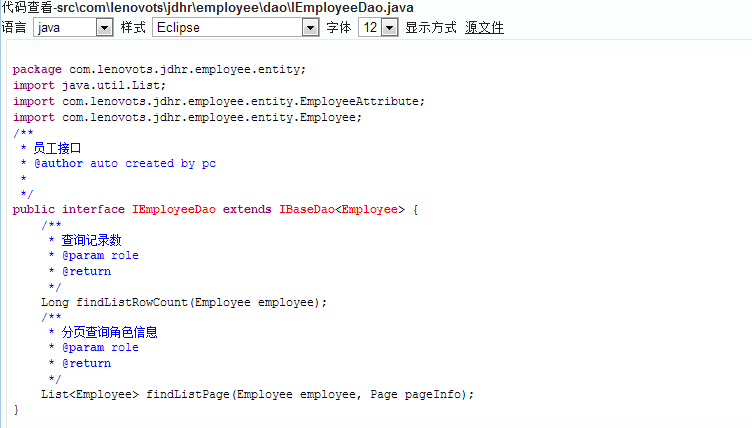


*图22 代码生成*

代码生成完毕之后，你可以在线查看生成结果：

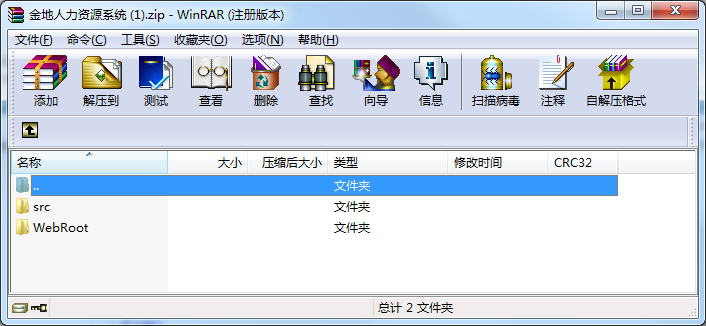


*图23 代码生成结果*



*图24 生成结果在线查看*

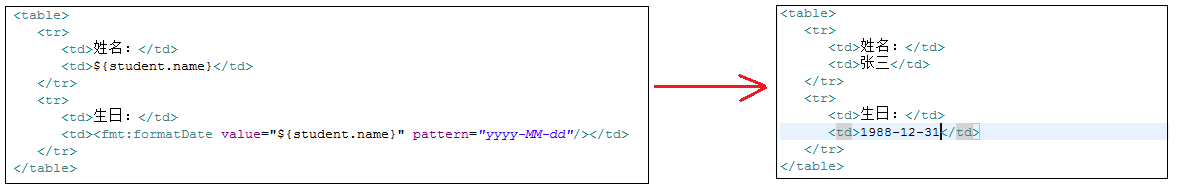
点击文件列表下方的下载代码，系统自动将生成的文件压缩之后进行下载：



*图25 生成结果打包下载*

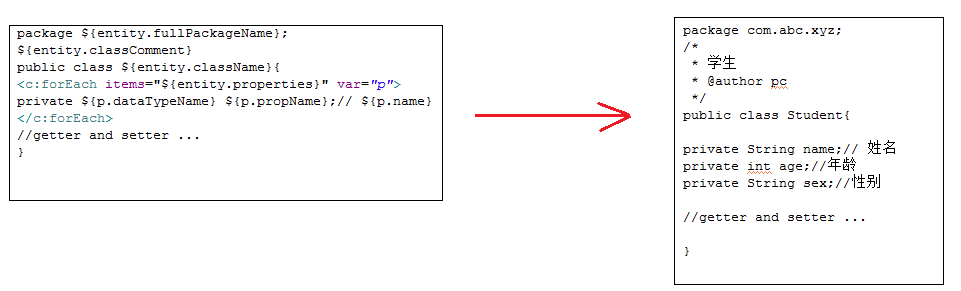
#### 额外说明

使用系统的关键在于你是否能够很好的定义出模板和自定义标签。在此进行额外的补充说明。CodeMaker的生成原理依靠的是JSP的解释技术。你可能已经很熟悉了下面的解释过程：



*图26 JSP解释过程*

那你就应该能理解下面的解释过程了：



*图27 Java类模板解释过程*

这就是CodeMaker的基本原理。应该注意的是，模板或者标签中的JSTL或者EL表达式都是在生成源码的过程中被解析的。因此如果你需要在模板或者标签中录入需要输出结果文件中的JSTL或者EL表达式时，你需要用到转义字符。

**$** 转义为：\& 或者[\_D]

**<** 转义为： [\_LT]

**>** 转义为： [\_GT]

例如，想在结果文件中得到标签，你可以写成[\_LT]fmt:formatDate/>。道理很简单，把可能被JSP解释器翻译掉的特殊符号“破坏”掉，使其“安全”通过JSP解释，解释完毕之后，CodeMaker会帮你把这些特殊符号在转换成原来的$、<、>。

（注：CodeMaker目前还不支持对转义标记的转义）