程序员为什么累？

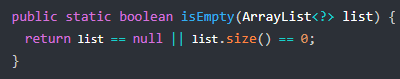
常用工具类和枚举类

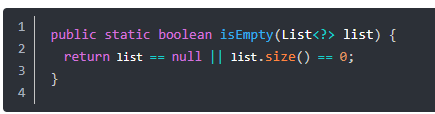
# 为什么要编写工具类

一个项目不可能没有工具类，工具类的初衷是良好的，代码重用，但到了后面工具类越来越乱，有些项目工具类有几十个，看的眼花缭乱，还有不少重复。如何编写出好的工具类，我有几点建议：

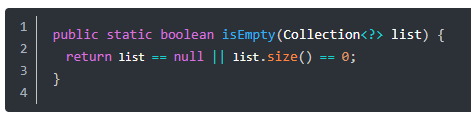
## 使用父类和接口

举例，假设我们写了一个判断arraylist是否为空的函数，一开始是这样的

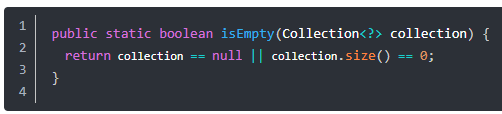


这个时候，我们需要思考一下参数的类型能不能使用父类。我们看到我们只用了size方法，我们可以知道size方法再list接口上有，于是我们修改成这样。

后面发现，size方法再list的父类/接口Collection上也有，那么我们可以修改为最终这样



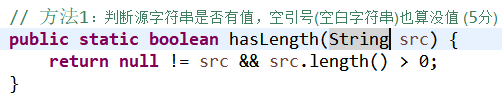
到了这一步，Collection没有父类/接口有size方法了，修改就结束了。最后我们需要把参数名字改一下，不要再使用list。改完后，所有实现了Collection都对象都可以用，最终版本如下：



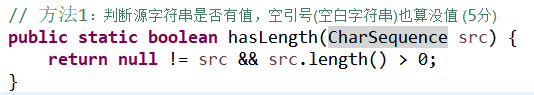
是不是看上去通用多了 ，看上去也专业多了？

## 问题来了:

## 如果要判断一个字符串是否为空该怎么写呢？



应该为：



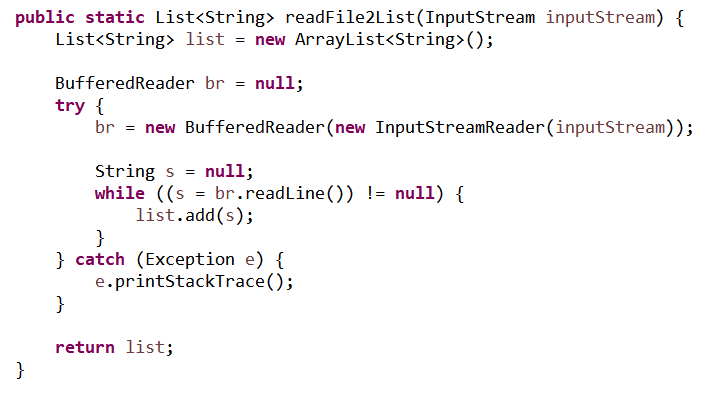
总结

思路和方法很简单，但效果很好，写出来的工具类也显得很专业！总结一下，思路是**抽象**的思想，主要是**修改参数类型**，方法就是往上找父类/接口，一直找到顶为止，记得修改参数名。

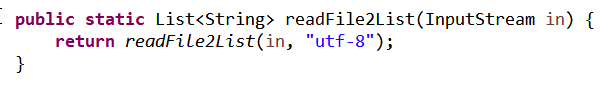
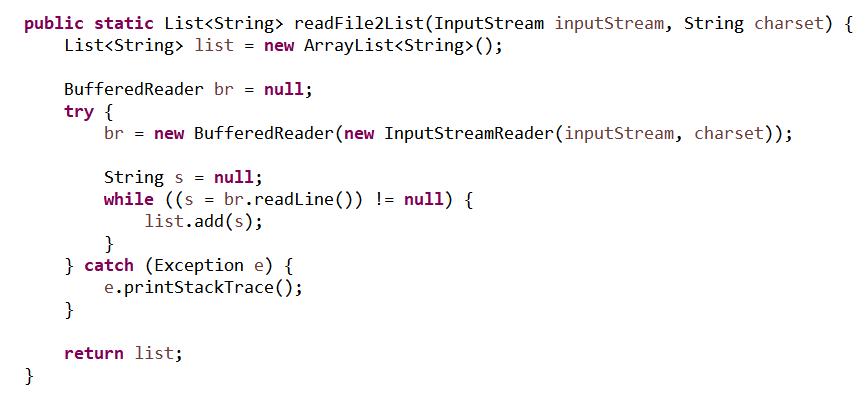
## 使用重载衍生函数组

有点开法经验的人都知道，有一些工具库，有一堆的重载函数，调用起来非常方便，经常能直接调用，不需要做参数转换。这些是怎么样编写出来的呢？我们举例说明。

现在需要编写一个方法，输入是一个utf-8格式的文件的文件名，把里面内容输出到一个list< String>。我们刚刚开始编写的时候，是这个样子的



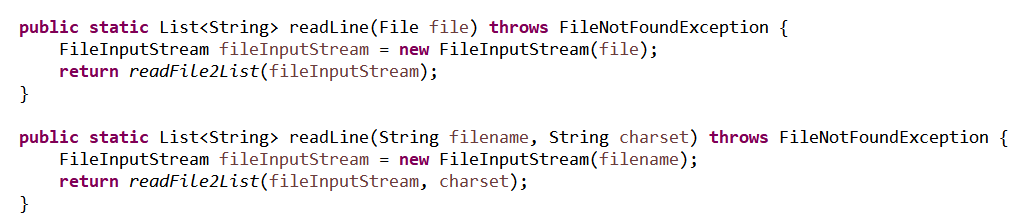
我们先实现，实现完之后我们做第一个修改，很明显，utf-8格式是很可能要改的，所以我们先把它做为参数提取出去，方法一拆为二，就变成这样。

多了一个方法，直接调用之前的方法主体，主要的代码还是只有一份，之前的调用地方不需要做任何修改！可以放心修改。

**思考？**

当前该方法传入的是inputstream类型的，如果我们为调用者提供更多选择传入类型，是不是更友好呢？



怎么样？4个方法，实际上代码主体只有一份，但提供各种类型的入参，调用起来很方便。开发组长编写的时候，多费一点点时间，就能写来看上去很专业调用起来很方便的代码。如果开发组长不写好，开发人员发现现有的方法只能传String，他传的是InputStream，他又不敢改原来的代码，就会copy一份然后修改一下，就多了一份重复代码。代码就是这样烂下去了。。。

## 物理上独立存放

习惯把和业务无关的代码放到独立的工程或者目录，在物理上要分开，专人维护。不是所有人都有能力写工具类，独立存放专门维护，专门的权限控制有助于保证代码的纯洁和质量。这样普通的开发人员就不会随意修改

# 常见的工具类

## 日期工具类

## 随机数

## 流工具类

## 字符串工具类

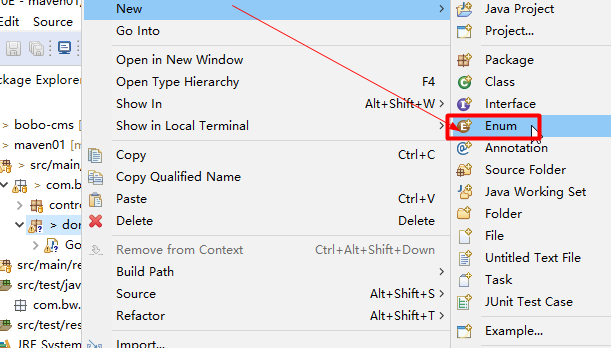
## 集合工具类

# 枚举类的使用

Java 1.5 引入的新的类型--枚举，在java中算是一个小的功能，但是

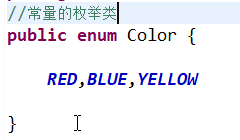
却给我们带来不小的方便，主要使用场景，总结如下：

## 如何创建枚举类

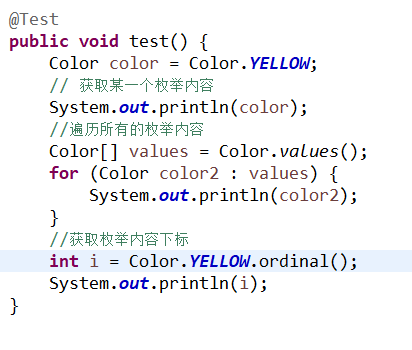


## 常量使用

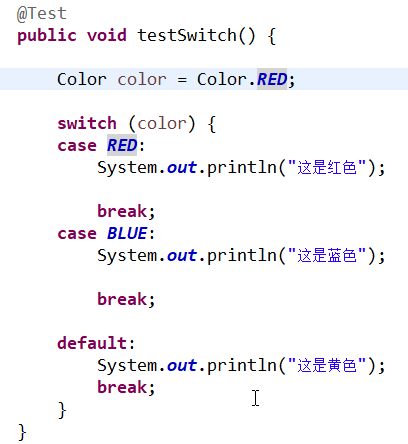
### 枚举定义



### 枚举的调用



## Switch

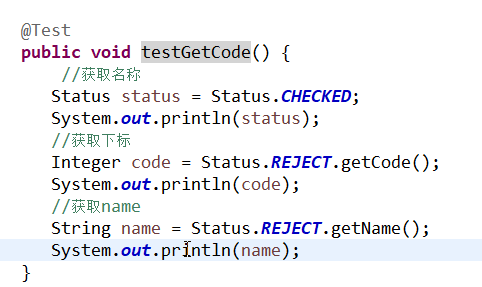


## 向枚举中添加新方法

### 创建

|  |
| --- |
| **package** com.bw.domain;  **public** **enum** Status {  //定义方式一:  //下标依次为 0 1 2  //未审,已审,驳回  //==================  //定义方式二（下标根据业务去定义）：  //下标依次为 0 1 9  ***UNCHECKED***(0,"未审"),***CHECKED***(1,"已审"),***REJECT***(9,"驳回");  **private** Status(**int** code,String name) {  **this**.code =code;  **this**.name= name;  }  **private** Integer code;  **private** String name;  **public** Integer getCode() {  **return** code;  }  **public** **void** setCode(Integer code) {  **this**.code = code;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }    } |

### 调用

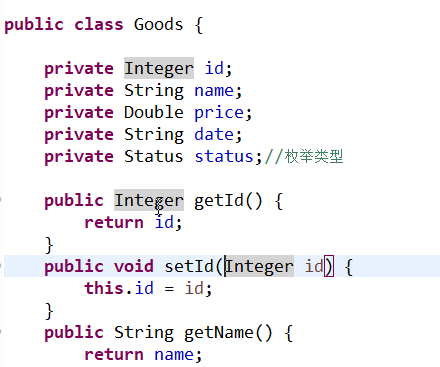


## 案例

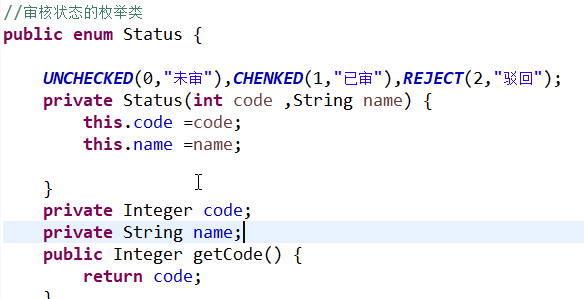
网上商城，商品有一个审核状态，一共有：未审（0）、已审（1）、驳回（9）。

要求：使用枚举类定义审核状态，数据库存储的是int（枚举的下标）

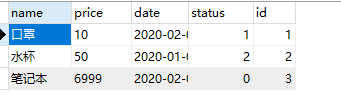
### 实体类



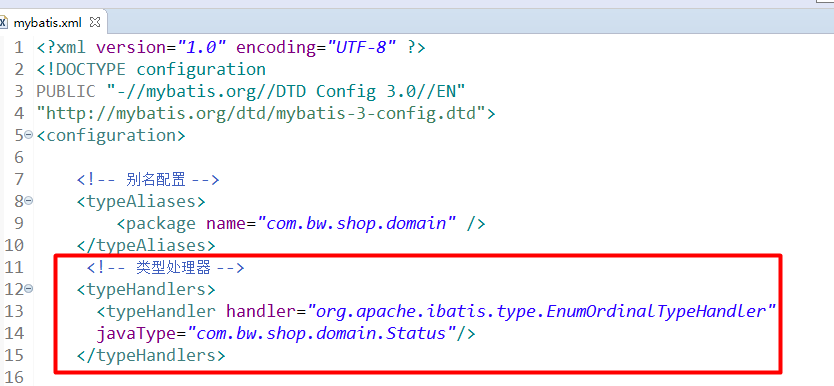
### 枚举类



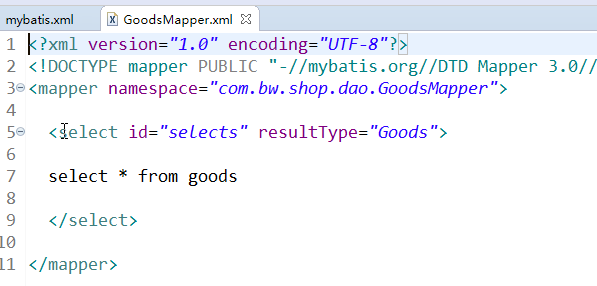
### 数据库



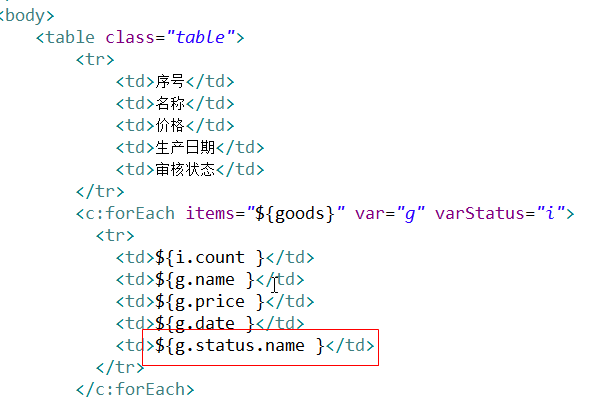
### Mybatis配置文件(重点)



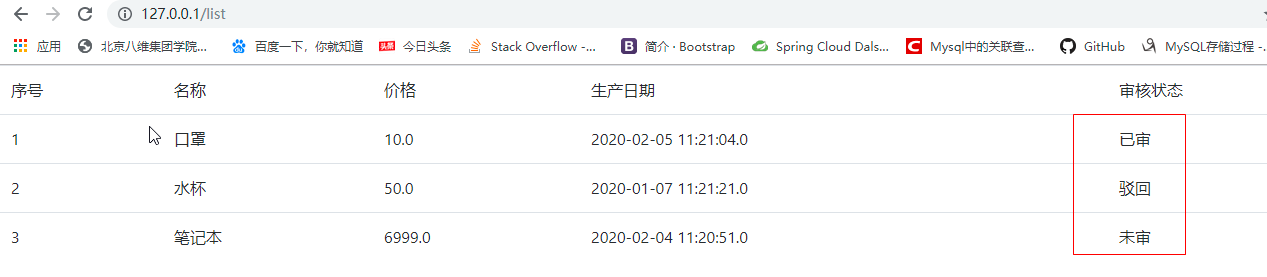
### Daomapper



### Jsp



### 效果



### 问题？

数据库到底能不能存指定的枚举下标？

答案是肯定的。

*org.apache.ibatis.type.EnumOrdinalTypeHandler*

*这个类存储是枚举的下标*

*org.apache.ibatis.type.EnumTypeHandler*

*这个类存储的是枚举的name*

*我们可以仿照.EnumTypeHandler定义一个自己的处理类如下：*

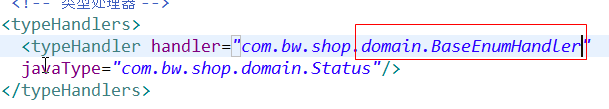
* *第一步：创建处理类*

|  |  |
| --- | --- |
| |  | | --- | | **package** com.bw.shop.domain;  **import** java.lang.reflect.InvocationTargetException;  **import** java.lang.reflect.Method;  **import** java.sql.CallableStatement;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** org.apache.ibatis.type.BaseTypeHandler;  **import** org.apache.ibatis.type.JdbcType;  **public** **class** BaseEnumHandler<E **extends** Enum<E>> **extends** BaseTypeHandler<E> {  **public** BaseEnumHandler() {}    **public** BaseEnumHandler(Class<E> type) {  **this**.type = type;  }  @Override  **public** **void** setNonNullParameter(PreparedStatement ps, **int** i, E parameter, JdbcType jdbcType) **throws** SQLException {  **if** (jdbcType == **null**) {  ps.setString(i, parameter.toString());  } **else** {  ps.setObject(i, parameter.name(), jdbcType.TYPE\_CODE);  }  }  @Override  **public** E getNullableResult(ResultSet rs, String columnName) **throws** SQLException {  **return** get(rs.getString(columnName));  }  @Override  **public** E getNullableResult(ResultSet rs, **int** columnIndex) **throws** SQLException {  **return** get(rs.getString(columnIndex));  }  **private** Class<E> type;  **private** E get(String v) {  **if** (v == **null**) **return** **null**;  **return** get(type, Integer.*parseInt*(v));  }    **private** <E **extends** Enum<E>> E get(Class<E> type, **int** v) {  Method method = **null**;  E result = **null**;  **try** {  method = type.getMethod("get", **int**.**class**);  result = (E)method.invoke(type, v);  } **catch** (NoSuchMethodException e) {  result = Enum.*valueOf*(type, String.*valueOf*(v));  e.printStackTrace();  } **catch** (IllegalAccessException e) {  e.printStackTrace();  } **catch** (InvocationTargetException e) {  e.printStackTrace();  }  **return** result;  }  @Override  **public** E getNullableResult(CallableStatement cs, **int** columnIndex) **throws** SQLException {  **return** **null**;  }    } | |

* *第二步：枚举类增加如下*

|  |
| --- |
| **package** com.bw.shop.domain;  //审核状态的枚举类  **public** **enum** Status {    ***UNCHECKED***(0,"未审"),***CHENKED***(1,"已审"),***REJECT***(9,"驳回");  **private** Status(**int** code ,String name) {  **this**.code =code;  **this**.name =name;    }  **private** Integer code;  **private** String name;  **public** Integer getCode() {  **return** code;  }  **public** **void** setCode(Integer code) {  **this**.code = code;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }    **public** **static** Status get(**int** v) {    **for** (Status e : *values*()) {  **if**(e.getCode()==v) {  **return** e;  }  }  **return** **null**;  }      } |

* *第三步：*



以上三步就搞定了

# 作业：

|  |
| --- |
| 1.完善工具类，学生的工具类工程中要包含  1.1 日期工具类  1.2 字符串处理工具类  判断字符串是否为空，判断字符串是否为邮箱，判断字符串是否是手机号  1.3 集合处理工具类  判断集合是否为空，  1.4.流处理工具类  读取传入的文件或文件地址到集合中（支持至少4中方式。方法重载的使用）  1.5随机数工具类  返回某个时间段的数据（如返回年龄在1-120之间）  返回4位验证码，可包含数字和字母  随机返回一段文章（至少500个文字）  2.以上工具类安装到本地仓库，并上传到github  3.枚举类的使用  定义一个枚举类，数据库存储枚举类的下标，  列表中显示枚举类的名称 |