**一个提高项目开发效率的方法**

-基于增强枚举的方式

**问题提出：**

在实际的java web项目中，经常会遇到对于不同的选择分支，采取不同的处理方法，常用的解决手段是通过if或者switch判断不同的选择分支，然后进行相应的处理方法。这种方式，在只有一两个分支时，能够较好的处理业务的逻辑，当分支有5、6个或者以上时，代码的清晰与逻辑变的复杂了。

**给出的一个解决方法：**

**1、普通枚举用法：**

Java1.5中新增了枚举类型，但是在很多项目中，枚举一般只是被简单的用来替代int枚举常量，比如：

在接口Constant.java中定义：

public interface Constant{

public static final int APP\_FUJI = 0;

}

用枚举替代ConstantType.java：

public enum ConstantType{

APP\_FUJI(0)；

private final int value;

private ConstatType(int value){

this.value = value;

}

public int getValue(){

return value;

}

}

上面的用法是很多项目中枚举最常用的方法。

**2、增强枚举用法：**

基于普通枚举的用法基础之上，在其中声明抽象的方法，不同的枚举对象需要用具体的方法覆盖该方法。

**举个栗子：**

**public** **enum** Operation {

***PLUS***("+") {

@Override

**double** apply(**double** x, **double** y) {

**return** x + y;

}

},

***MINUS***("-") {

@Override

**double** apply(**double** x, **double** y) {

**return** x - y;

}

};

**private** **final** String symbol;

**private** Operation(String symbol) {

**this**.symbol = symbol;

}

**abstract** **double** apply(**double** x, **double** y);

}

上述的增强枚举类型已经能够非常清楚的表示出每个枚举类型对应的方法，而且抽象方法的个数可以是多个，重要的一点是，每增加一个抽象方法，每个枚举中就会要强制要求去用具体的方法（避免遗漏）。

**3、更加通用的增强枚举用法：**

上面的增强枚举方法虽然已经很清晰了，但是在某些情况下，项目中获取的类型往往不是枚举类似，而是类型“+”或“-”这样的字符串，因此需要对该中类型处理：

**public** **enum** Operation {

***PLUS***("+") {

@Override

**double** apply(**double** x, **double** y) {

**return** x + y;

}

},

***MINUS***("-") {

@Override

**double** apply(**double** x, **double** y) {

**return** x - y;

}

};

**private** **final** String symbol;

**private** **static** **final** Map<String, Operation> ***stringToEnum*** = **new** HashMap<String, Operation>();

**static** {

**for** (Operation op : *values*()) {

***stringToEnum***.put(op.toString(), op);

}

}

**public** **static** Operation fromString(String symbo) {

**return** ***stringToEnum***.get(symbo);

};

**private** Operation(String symbol) {

**this**.symbol = symbol;

}

**abstract** **double** apply(**double** x, **double** y);

}

上面的底色黄色代码中通过将Operation常量从静态代码中放入到stringToEnum的map中。

具体用法如下：

public static void main(){

String oper =”+”;

int a = 123;int b = 34;

int sum = Operation.fromString(oper).apply(a, b);

}

**4、进一步优化增强枚举用法（策略枚举类）：**

在上一个的增强枚举用法中仍然存在一些不足之处，那就是如果每个枚举类型对于不同的抽象方法一定进行具体化，而如果有几个枚举类型的方法是一样的，则导致了代码重复和冗余。改进方式如下：

**举个栗子：**

**public** **enum** Operation {

***PLUS***("+") {

@Override

**double** apply(**double** x, **double** y) {

**return** x + y;

}

},

***PLUSADD***("++"){

@Override

**double** apply(**double** x, **double** y) {

**return** x + y;

}

},

***MINUS***("-") {

@Override

**double** apply(**double** x, **double** y) {

**return** x - y;

}

};

….

}

上面的枚举类型中PLUS和PLUSADD对于apply方法是完全一致的。

**public** **enum** Operation2 {

***PLUS***(OperType.***ADD***),

***PLUSADD***(OperType.***ADD***),

***MINUS***(OperType.***SUB***);

**private** **final** OperType operType;

**private** Operation2(OperType operType){

**this**.operType = operType;

}

**public** **int** apply(**int** num1, **int** num2){

**return** operType.apply(num1, num2);

}

**private** **enum** OperType{

***ADD*** {

@Override

**int** apply(**int** num1, **int** num2) {

**return** num1 + num2;

}

},

***SUB*** {

@Override

**int** apply(**int** num1, **int** num2) {

**return** num1 - num2;

}

};

**abstract** **int** apply(**int** num1, **int** num2);

}

}

**调用方法如下：**

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

System.***err***.println(Operation2.***PLUS***.apply(1, 2));

System.***err***.println(Operation2.***PLUSADD***.apply(1, 2));

System.***err***.println(Operation2.***MINUS***.apply(1, 2));

}

PLUS和PLUSADD的方法共用一个子枚举类型，把这个枚举类型叫做***策略枚举***（strategy enum）

进一步，针对传入某个字符串类型，要调用相应的枚举的，则需要更改如下：

**最终推荐的写法如下：**

**public** **enum** Operation2 {

***PLUS***("+", OperType.***ADD***),

***PLUSADD***("++", OperType.***ADD***),

***MINUS***("-", OperType.***SUB***);

**private** **final** String operVal;

**private** **final** OperType operType;

**public** String getOperVal(){

**return** operVal;

}

**public** OperType getOperType(){

**return** operType;

}

**private** **static** **final** Map<String, Operation2> ***stringToEnum*** = **new** HashMap<String, Operation2>();

**static** {

**for** (Operation2 op : *values*()) {

***stringToEnum***.put(op.getOperVal(), op);

}

}

**public** **static** Operation2 fromString(String symbo) {

**return** ***stringToEnum***.get(symbo);

};

**private** Operation2(String operVal, OperType operType){

**this**.operVal = operVal;

**this**.operType = operType;

}

**public** **int** apply(**int** num1, **int** num2){

**return** operType.apply(num1, num2);

}

**private** **enum** OperType{

***ADD*** {

@Override

**int** apply(**int** num1, **int** num2) {

**return** num1 + num2;

}

},

***SUB*** {

@Override

**int** apply(**int** num1, **int** num2) {

**return** num1 - num2;

}

};

**abstract** **int** apply(**int** num1, **int** num2);

}

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

String oper = "+";

String oper2 = "++";

String oper3 = "-";

System.***err***.println(Operation2.*fromString*(oper).apply(1, 2));

System.***err***.println(Operation2.*fromString*(oper2).apply(1, 2));

System.***err***.println(Operation2.*fromString*(oper3).apply(1, 2));

System.***err***.println(Operation2.***PLUS***.apply(1, 2));

System.***err***.println(Operation2.***PLUSADD***.apply(1, 2));

System.***err***.println(Operation2.***MINUS***.apply(1, 2));

}

}

**【参考文献】**

1 、Effective Java 中文版 P129-P136