# 第8章 Java新特性

知道有什么新特性。

！！！必须要清楚新特性主要能解决什么问题。

能明白新特性的语法与使用方法。

## 8.1 可变参数

例子1：设计一个方法，功能是实现2个整型变量的加法操作。

**package** com.yootk.demo;

**public** **class** Demo\_8\_1 {

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

System.***out***.println(*add*(10, 20));// 传递2个整型变量

}

**public** **static** **int** add(**int** a, **int** b) {

**int** sum = 0;

sum = a + b;

**return** sum;

}

}

可以看到在主方法中调用add(**int** a, **int** b)方法时，必须按照该方法定义的参数类型、参数个数进行参数传递才能调用。

**问题：有时并不确定要传递的参数个数（参数类型是确定的）**

例子2：设计一个方法，功能是实现若干整型变量的加法操作。

之前解决方法：将多个参数封装为数组。

**package** com.yootk.demo;

**public** **class** Demo\_8\_1 {

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

System.***out***.println(*add*(**new** **int**[] {1, 2, 3}));// 传递3个整型数据

System.***out***.println(*add*(**new** **int**[] {10, 20}));// 传递2个整型变量

}

/\*\*

\* 实现任意多个整型数据的相加操作

\* **@param** data 由于要接收多个整型数据，所以使用数组完成接收

\* **@return** 多个整型数据的累加结果

\*/

**public** **static** **int** add(**int**[] data) {

**int** sum = 0;

**for** (**int** i = 0; i < data.length; i++) {

sum += data[i];

}

**return** sum;

}

}

但是严格来讲这样实现并不标准。

开发要求是传递“若干整型变量”，理想的调用形式应该是：“*add*(1, 2, 3)、*add*(10, 20)”。

而不是传递“一个整型数组”，调用形式变成：“*add*(**new** **int**[] {1, 2, 3})”。

例子3：引入可变参数。

**package** com.yootk.demo;

**public** **class** Demo\_8\_1 {

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

// 可变参数支持接收数组

System.***out***.println(*add*(**new** **int**[] {1, 2, 3}));// 传递3个整型数据

System.***out***.println(*add*(**new** **int**[] {10, 20}));// 传递2个整型变量

// 或者使用 “,” 区分不同的参数，接收的时候还是数组

System.***out***.println(*add*(1, 2, 3));// 传递3个整型数据

System.***out***.println(*add*(10, 20));// 传递2个整型变量

System.***out***.println(*add*());// 传递0个参数

}

/\*\*

\* 实现任意多个整型数据的相加操作

\* **@param** data 由于要接收多个整型数据，所以使用数组完成接收

\* **@return** 多个整型数据的累加结果

\*/

**public** **static** **int** add(**int**... data) {

**int** sum = 0;

**for** (**int** i = 0; i < data.length; i++) {

sum += data[i];

}

**return** sum;

}

}

其实本质还是使用了数组，这样“例子2”的代码得到很好的兼容，不用修改也可以运行。

**一定要去看一下Math类中的加法操作源码是怎么实现的。**

总结：

1. 在设计一个类的时候可变参数绝对不是优先的选择。要尽可能让方法的参数个数是确定的，除了一些系统级别、抽象程度很高的的方法，如例子所说的计算器程序；
2. **可变参数就属于数组的变形应用。**

## 8.1 foreach循环

本次预计讲解的知识点：

增强型for循环的使用。

总结：

foreach循环支持数组的直接访问，避免索引访问带来的 麻烦。

解决问题：

简化数组或集合数据的输出操作

语法（for循环的定义从JDK1.5开始有了以下新形式）：

**for** (数据类型 变量 : 数组 | 集合) {

// 每一次循环会自动依次将数组或集合的内容设置给变量

}

分析新特性的用法（思路：需求场景的变化-产生的问题-用新特性怎么解决的）