Java 基础（9）：可变参数列表介绍

一、可变参数简介

在不确定参数的个数时，可以使用可变的参数列表。

1. 语法

参数类型...（三个点）
例如： void printArray（Object...）
注意： 每个方法最多只有一个可变参数，因为：可变参数必须是方法的最后一个参数

2. 可变参数的类型

可变参数可以设置为任意类型：**引用类型，基本类型**；当然也会进行类型检查的；

3. 参数的个数

- 0 个参数
- 1 个参数： 如果是数组，那么就直接将这个数组作为参数传进方法里面，不再填充新的数组；
- 多个参数： 参数可以是数组，也可以是单个变量、常量；但是这时候会，将这些参数填充进新的数组里面，再将这个数组，传进方法里面；

4. 可变参数的使用

可变参数完全可以当作一个数组来使用，或者说，本质上可变参数就是一个数组（下面详细介绍）。所以，数组拥有的方法、属性，可变参数一样拥有。

```
public void varArgMethod(int b,int... arr) {
    //和数组一样，拥有属性 length
    int lenth = arr.length;
    //索引遍历
    for(int i=0;i<arr.length;i++) {
        System.out.println(arr[i]);
    }
    //forEach 循环遍历
```

```
    for(int ele:arr) {
        System.out.println(ele);
    }
}
```

上面的例子中，可变参数的使用跟数组的使用是完全一样，也就是说，**可变参数是可以等价成数组的。**

5. 可变参数的方法重载

可变参数列表的方法的重载不同于普通方法：**无法仅通过改变可变参数的类型，来重载方法。**

如：varArray(int... a)、varArray(Object... a)，这两个方法在调用时会出错，方法重载失败。

二、 深入分析可变参数的原理

前面已经很详细地介绍了可变参数的各个方面。这一小节将深入去了解可变参数的实现原理，特别是为什么可变参数的使用与数组是一样的。

看下面一个简单的例子：

```
public class MyTest{
   public static void main(String[] args) {
       int a = 100;
       varArgMethod(5, 7,8,9,10,a);
 }


   public static void varArgMethod(int b,int... arr) {
       //索引遍历
       for(int i=0;i<arr.length;i++) {
           System.out.println(arr[i]);
       }
   }
}
```

例子很简单，为了了解编译器是怎么处理的，我们用 jad 对上面例子的 class 文件进行反编译：

```
public class MyTest {
    public static void main(String args[]) {
        int a = 100;
```

```
        varArgMethod(5, new int[]{7, 8, 9, 10, a});//参数列表被编译器处理成了一个int[]数组
    }
    public static transient void varArgMethod(int b, int arr[]) { //形参被编译器处理成数组
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }
}
```

从反编译的结果可以看出，编译器不仅将可变参数处理成数组 varArgMethod(int b, int arr[])，还处理了调用可变参数方法处的参数列表，把参数列表封装进一个数组 varArgMethod(5, new int[]{7, 8, 9, 10, a})。

现在看来，可变参数列表并没有多神奇，只不过是将程序员做的工作简化了，交给了编译器来处理。最后，**可变参数的使用和数组一样也就不出奇了，因为可变参数最后还是被编译器处理成了数组，可变参数就是数组。**

1) JDBC 中，用于表示数据库连接的对象是:　B
   A.Statement
   B.Connection
   C.DriverManager
   D.PreparedStatement
2) 初始化了一个没有 run()方法的线程类,是否会出错？
   答案：不会。

- **第一种方法：直接继承 Thread 类。**

```
public class Test{
   public static void main(String[] args){
        ThreadClass t = new ThreadClass();
        t.start();
        System.out.println("end");//输出“end”
   }
}

class ThreadClass extends Thread  //Thread 类已经实现了空的 run()方法。
{}
```

- **第二种方法：实现 Runnable 接口**

```
public class Test{
    public static void main(String[] args){
        ThreadClass t = new ThreadClass();
        Thread thread = new Thread(t);
```

```
        thread.start();
        System.out.println("end");
    }
}

class ThreadClass implements Runnable{
   public void run() { //必须有此方法否则编译报错。它是 Runnable 接口中的抽象方法。
        System.out.println("Threads");
   }
}
```