**·数组的定义与使用**

1. 数组的基本使用

·数组的动态初始化:

-声明并初始化对象:

数据类型 数组名称 [] = new 数据类型[数据长度]

数据类型 [] 数组名称 = new 数据类型[数据长度]

·数组的额静态初始化 : 在数组定义的时候就为其赋值

-简化格式 :

数据类型 数组名称 [] = {数据1，数据2 ，数据3.....，数据n}；

-完整格式:

数据类型 数组名称 [] = new 数据类型 []{数据1，数据2 ，数据3.....，数据n};（建议使用完整格式进行定义）

|  |
| --- |
| 范例:  public class ArrayDemo{  public static void main(String arg[]){  //使用动态数组初始化  int data [] = new int[3];  //静态初始化  int data [] = new int []{3,4,5,6,7,8}  }  } |

注: 数组长度可以用 数组名称.length 进行访问

1. **数组引用传递**

|  |
| --- |
| 范例:  public class ArrayDemo{  public static void main(String arg[]){  //使用动态数组初始化  int data [] = new int[] {1，2，4}; //静态初始化  int temp [] = data; //引用传递  temp[0] = 99;  f or(int i; i < data.length; i++ ){  System.out.println(data[x]);  }  }  } |

通过上述代码进行数组内存分析数组本身依然是属于引用数据类型: 那么既然是引用数据类型，就一定会发生引用传递引用传递应该还是按照传统的方式一样（与类与对象的传递一模一样）: 一个堆内存可以被多个栈内存所指向,上述代码执行后可发现: 当temp语句执行后，data所指向的 堆内存数组的值也改变了，所以他和我们正常的引用过程是一样的。由于数组属于引用类型，所以一定要为其开辟堆内存空间之后才能使用， 当我们没有为其指定堆内存使用数组时，会报一个 NullPointException (空指针异常)的异常，所以必须要提供实例化对象才可以使用数组的操作。

1. **foreach迭代输出**

|  |
| --- |
| 传统做法:  public class ArrayDemo{  public static void main(String arg[]) {  int data [] = new int []{1,2,3,4,5};  for(int i ; i < data.length ; i++){  System.out.println(data[i]);  }  } |

JDK1.5以后为了防止下标对程序的影响（如果下标处理不当会产生越界的异常） ，所以参考了NET中的设计引用了一个增强 for 循环（foreach），利用foreach可以自动获取数组中的每一个元素进行访问，避免下标访问操作出错

格式: for(数据类型 变量 : 数组 | 集合){}

最大的特点:**是将数组中的每一个元素取出保存在变量里面，这样就可以直接通过变量访问**

|  |
| --- |
| 范例:  public class ArrayDemo{  public static void main(String args[]){  int data [] = new int [] {1,2,3,4,5};  for(int temp : data){  System.out.println(temp);  }  }  } |

这样的最大的好处就是没有下标的使用;

1. 二维数组

对于二维数组可以使用以下语法定义:

·数组的动态定义:

-数据类型 数组名称 [][] = new 数据类型 [行个数][列个数];

·数组的静态初始化:

-数据类型 数组名称 [][] = new 数据类型[][] {{数据1,数据2,....},{数据1,数据2,....}..}

**那么如何使用foreach来进行二维数组遍历呢?**

|  |
| --- |
| 范例:  public class ArrayDemo{  public static void main(String args[]){  int data [] = new int [] { {1,2,3},{4,5,6},{1,2,3} };  for(int temp[] : data){  for(int num : temp){  System.out.println(temp);  }  }  }  } |

1. 数组的引用传递

|  |
| --- |
| 范例:  public class ArrayDemo{  int data [] = initArray(); //通过方法返回一个数组  printArry(data); //传递数组  public static int [] initArray(){  int arr [] = new int []{1,2,3,4,5,6};  return arr ;  }  //要求接收一个int型的数组  public static void printArray(int temp[]){  for(int i; i < temp.length; i++){  System.out.println(temp[i]);  }  }  } |

1. 数组的相关操作方法

即Java语言本身提供的相关支持原理;

1. 数组排序: java.util.Array.sort(数组名称)。

·方法的可变参数

如果说现在要求定义一个方法，这个方法可以实现任意多个整型数据相加处理。

public static int sum (int ... data){//变种参数}

ArrayUtil.sum(1,2,3); //实现多参传入

可变参数最大的作业在于，在以后进行一些程序类设计或者开发调用的时候，利用可变此种

形式就可以避免数组的传递操作。

但我们要清楚可变参数的本质依然是：数组

1. 对象数组

在之前所接触到的都是基本数据类型定义的数组，但是在Java程序本身各类数据类型都可以成为数组，

所以类也可以成为数组类型，对象数组的格式如下:

·动态初始化: 类 对象数组名称 [] = new 类 [长度]，每一个元素都是null；

·静态初始化: 类 对象数组名称 [] = new 类 [长度]，{实例化对象，实例化对象，...}；

使用动态初始化定义对象数组

Person per [] = new Person[3] //此数组中的对象都是null,因为未给其指定堆内存

per[0] = new Person("01",02);