Java数组

今天我们学习一个Java中非常重要的技术——数组。

认识数组

先来认识一下什么数组

什么数组

数组是具有相同数据类型且按一定次序排列的一组变量的集合体。即用一个变量名表示一批数据。Java为数组在内存中分配一段连续的空间，这段空间中存储数据的个数是固定的，数组就是一个容器，用来存一批同种类型的数据的。

比如：想要存储 20,10,80,60,90 这些数据。 我们可以把代码写成这样

```
int[] array = {20,10,80,60,90}; // 5 0, 1, 2, 3, 4
```

比如：想要存储 “牛二“,“西门“,“全蛋“ 这些数据。我们可以把代码写成这样

```
String[] names = {"牛二", "西门", "全蛋"}; // 3  0, 1, 2
```

- 数组元素：构成数组的每一个元素
- 数组下标：下标就是数组元素在数组中的位置。下标从0开始，依次累加1，也称为索引。
- 数组大小：数组中元素的个数，也称为数组的长度。

数组的应用场景

有变量，为什么还要有数组呢？ 比如，我们要做一个点名器

如果用变量来做的话，代码是这样子的

```
String name1 = "张三";
String name2 = "李四";
String name3 = "王五";
```

```
String name4 = "赵六";
String name5 = "田七";
// ...
String name35 = "张十五";
String name36 = "张十六";
String name37 = "张十七";



Random rand = new Random();
int number = rand.nextInt(37) + 1; // 1-37
switch (number) {
    case 1:
        System.out.println(name1);
        break;
    case 2:
        System.out.println(name2);
        break;
    // ...
}
```

- 代码繁琐：大量变量的定义。
- 实现需求繁琐。

如果用数组来做的话，代码是这样子的

```
String[] names = {"张三", "李四", "王五", "赵六", "田七",.., "张十五", "张十六", "张十七"}; // 37 0-36
Random rand = new Random();
int number = rand.nextInt(37) + 1;
System.out.println(names[number - 1] + "出来回答问题");
```

- 代码简洁
- 逻辑清晰

一对比我们发现数组的写法比变量的写法更加简洁，所以我们可以得出一个结论

结论：遇到批量数据的存储和操作时，数组比变量更适合

数组的定义和访问

我们已经知道数组是用来干什么的。那么如何使用Java语言写一个数组呢？

定义数组

Java中定义数组有两种语法格式：`数据类型 数组名[ ];`或`数据类型[ ] 数组名;`

例如：`String names[]` 或 `String[] names` 推荐第二种方式

语法解析：

- 数组是什么数据类型，数组的元素就是什么数据类型
- 数组的特征是[ ]
- 数组是引用类型

数组有两种初始化的方式，一种是静态初始化、一种是动态初始化。我们先用静态初始化来学习数组的操作。

静态初始化数组

定义数组、为数组元素分配内存、数组元素初始化，这三步可以合并在一起写，例如:

`int[] scores = new int[]{12,56,34,78};`

或

`int[] scores = {12,56,34,78};`

在定义数组时直接给数组中的数据赋值这种方式称为静态初始化。标准格式是

```
数据类型[] 变量名 = new 数据类型[]{元素1,元素2,元素3};

简化为:
数据类型[] 变量名 = {元素1,元素2,元素3};

String[] names = {"金文涛", "李瑶瑶", "小樊同学"...};
System.out.println(names[1]);
```

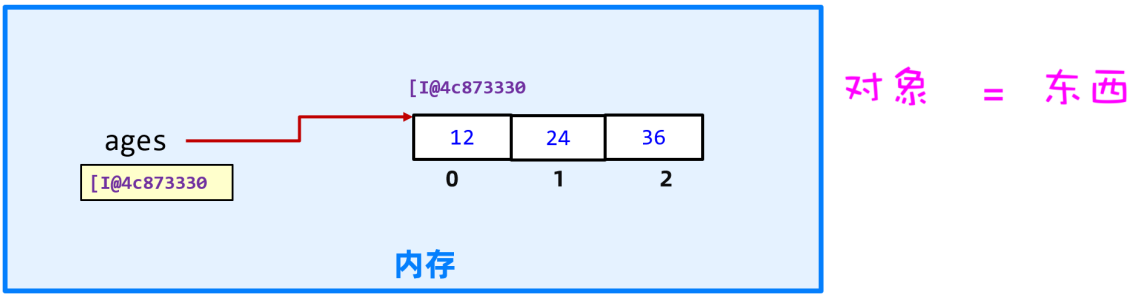
数组在计算机中的基本原理

我们知道数组是怎么定义的之后，那么接下来看一下数组在计算机中的基本原理。

我们以`int[] ages = {12,24,36};`这句话为例，看一下这句话到底在计算机中做了那些事情。

- 首先，左边`int[] ages`表示定义了一个数组类型的变量，变量名叫ages
- 其次，右边`{12,24,36}`表示创建一个数组对象，你完全可以把它理解成一个能装数据的东西。这个对象在内存中会有一个地址值`[I@4c873330`，每次创建一个数组对象都会有不用的地址值。
- 然后，把右边的地址值`[I@4c873330`赋值给左边的ages变量
- 所以，ages变量就可以通过地址值，找到数组这个东西。

```
int[] ages = {12, 24, 36};
```



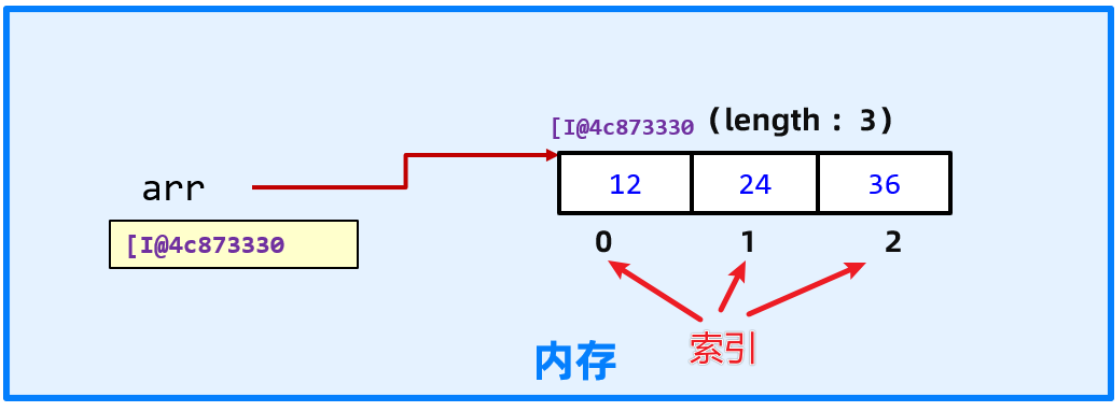
注意：数组变量名中存储的是数组在内存中的地址，数组是一种引用数据类型。

数组的元素访问

各位同学，通过刚才的学习，我们知道数组是用来存储数据的。那么数组中存储的数据又如何访问呢？这里所说的访问，意思就是获取中数组中数据的值、或者给数组中的数据赋值。

这里先给大家统一几个概念，数组中存储的数据我们叫做元素；而且数组中的每一个元素都有一个编号与之对应，我们把这个编号叫做索引，这个索引是从0依次递增的整数。如下图所示

```
int[] arr = {12, 24, 36};
```



要想访问数组中的元素，格式如下

```
// 数组名可以找到数组对象的地址，再通过索引就可以定位到具体的元素了
数组名[索引]    // 索引 0 --> 长度 - 1
```

接下来用代码来演示一下

```
// 索引:      0   1   2
int[] arr = {12, 24, 36}; // 静态初始化
// 1、访问数组的全部数据
System.out.println(arr[0]); // 12
System.out.println(arr[1]); // 24
System.out.println(arr[2]); // 36
// 下面代码没有3索引，会出现ArrayIndexOutOfBoundsException 索引越界异常
// System.out.println(arr[3]);

// 2、修改数组中的数据
arr[0] = 66;
arr[2] = 100;
System.out.println(arr[0]); //66
System.out.println(arr[1]); 0
System.out.println(arr[2]); //100
```

除了访问数组中的元素，我们可以获取数组中元素的个数，后面我们统称为数组的长度。

```
// 3、访问数组的元素个数：数组名.length
System.out.println(arr.length);

// 技巧：获取数组的最大索引：arr.length - 1(前提是数组中存在数据)
System.out.println(arr.length - 1);

int[] arr2 = {};
// arr2[arr2.length - 1]
System.out.println(arr2.length - 1);
```

数组的遍历

各位同学，接下来我们学习一个对数组最最最常见的操作——数组遍历。所谓遍历意思就是将数组中的元素一个一个的取出来。

我们刚才学习了数组中元素的访问，访问元素必须用到索引，如下列代码。

```java
int[] ages = {12, 24, 36};
System.out.println(ages[0]);
System.out.println(ages[1]);
System.out.println(ages[2]);
```

但是，如果数组中有很多很多元素，索引靠自己一个一个数肯定是不行的！我们可以使用for循环从0开始一直遍历到长度-1的位置，就可以获取所有的索引了。

当你获取到每一个索引，那么每一个元素不就获取到了吗？上代码吧

```java
int[] ages = {12, 24, 36};
for (int i = 0; i < ages.length; i++) {
    // i的取值 = 0, 1, 2
    System.out.println(ages[i]);
}
```

数组静态初始化案例

学习完数组的静态初始化之后，接下来我们做一个练习题来巩固一下。

```
需求：某部门5名员工的销售额分别是：16、26、36、6、100，请计算出他们部门的总销售额。

需求分析：
    1.看到有16、26、36、6、100这5个数据数据，而且数据值很明确；
        1)想到,可以使用数组静态初始化把这5个数据存起来

    2.请计算出他们部门的总销售额（这不就是求数组中数据的和吗？）
        2)必须先将数组中所有的元素遍历出来
        3)想要求和，得先有一个求和变量sum
        4)再将每一个元素和求和变量sum进行累加（求和思想）
```

按照分析的思路来写代码

```java
// 1、定义一个数组存储5名员工的销售额
//索引          0   1    2  3   4
int[] money = {16, 26, 36, 6, 100};

// 3、定义一个变量用于累加求和
int sum = 0;

// 2、遍历这个数组中的每个数据。
for (int i = 0; i < money.length; i++) {
    // i = 0  1  2  3  4
    sum += money[i];
}
System.out.println("员工的销售总额: " + sum);
```

数组的动态初始化

刚才我们初始化数组时，都是直接将元素写出来。但是还有另一个初始化数组的方式叫 **动态初始化**。

动态初始化不需要我们写出具体的元素，而是指定元素类型和长度就行。格式如下

数据类型[] 数组名;

`数组名 = new 数据类型[数组长度];`

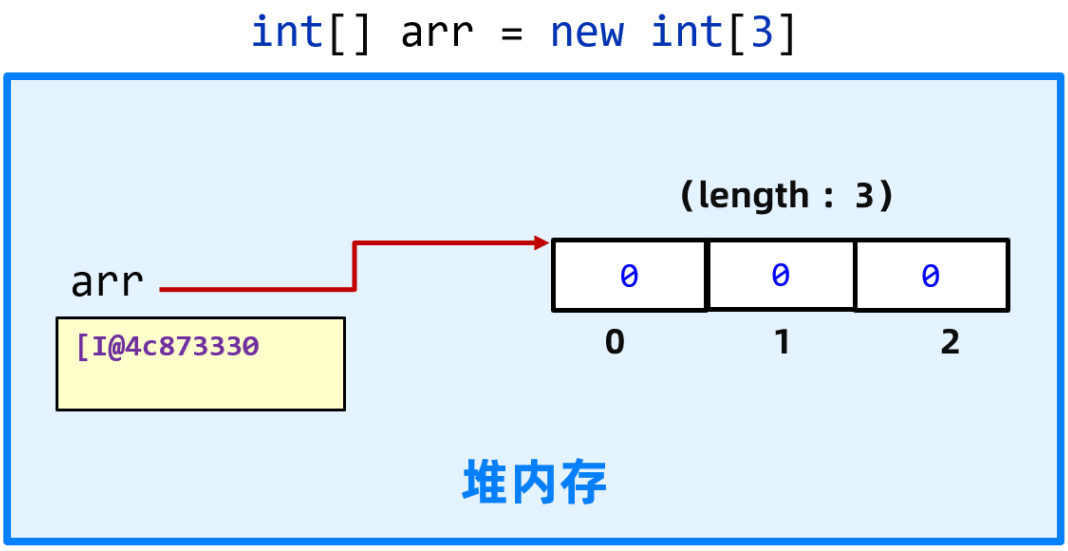
例如：`names = new String[5];`

定义数组和为数组元素分配内存，这两步可以合并在一起写，例如：

`String[] names = new String[5];`

```java
// 数据类型[]  数组名 = new 数据类型[长度];
int[] arr = new int[3];
```

下面是动态初始化数组的原理图。我们发现`int[] arr`其实就是一个变量，它记录了数组对象的地址值，而且数组中的元素默认值是0。



注意：

使用动态初始化定义数组时，根据元素类型不同，默认值也有所不同。

数组元素类型	默认初始值
byte，short，int，long	0
float，double	0.0
char	'\u0000'（空字符）
boolean	false
引用数据类型	null

数组动态初始化案例

各位同学，接下来我们做一个数组动态初始化的案例。

```
案例需求：
    某歌唱比赛，需要开发一个系统：可以录入6名评委的打分，录入完毕后立即输出平均分做
    选手得分

需求分析：
    1.需要录入6名评委的分数，可以用一个数组来保存。
        因为在评委没有录入分数之前，还不确定数组中应该存哪些数据。
        所以可以使用数组的动态初始化
    2.遍历数组中的每一个位置，并录入分数，将分数存入数组中
    3.遍历数组中的每一个元素，对元素求和
public static void main(String[] args) {
```

```
    // 某歌唱比赛，需要开发一个系统：可以录入6名评委的打分，录入完毕后立即输出平均分做选手得分
    double[] scores = new double[6];
    Scanner sc = new Scanner(System.in);
    for (int i = 1; i <= 6; i ++) {
        System.out.println("请输入第" + i + "个评委的成绩：");
        double score = sc.nextDouble();
        scores[i - 1] = score;
    }
    double sum = 0;
    double avg = 0;
    for (int i = 0; i < scores.length; i++) {
        System.out.println(scores[i]);
        sum += scores[i];
    }
    avg = sum / scores.length;
    System.out.println("选手的成绩是" + avg);
}
```

使用数组实现斐波拉切数列

```
public static void main(String[] args) {
    // 输入 n 输出前 n 个数 存储到数组 再输出
    Scanner sc = new Scanner(System.in);
    System.out.println("请输入一个数字");
    int number = sc.nextInt();
    int[] feibo = new int[number];
    // int prev = 1; // 第一个值
    // int next = 1; // 第二个值
    // feibo[0] = prev;
    // feibo[1] = next;
    if (number == 1) {
        feibo[0] = 1;
    } else if (number == 2) {
        feibo[0] = 1;
        feibo[1] = 1;
    } else {
        // feibo : 1 1 2 3
        feibo[0] = 1;
        feibo[1] = 1;    // 1 1 x
        for (int i = 3; i <= number; i++) { // i = 3 i = 4
            feibo[i - 1] = feibo[i - 3] + feibo[i - 2];
            // int curr = prev + next;
            // feibo[i - 1] = curr;
            // prev = next;
            // next = curr;
```

```
            }
        }
        for (int i = 0; i < feibo.length; i++) {
            System.out.println(feibo[i]);
        }
    }
```