Java面向对象 第4天

**【学习目标】理解、了解、应用、记忆**

通过今天的学习，参训学员能够：（解释的时候说出二级目标的掌握程度）

1. **【理解】面向过程和面向对象的思想**
2. 【理解】过程的概念
3. 【理解】面向过程解决问题的步骤
4. 【理解】对象的概念
5. 【理解】面向对象解决问题的步骤
6. 【理解】举例解释面向对象的思想
7. 【理解】面向过程和面向对象的联系和区别
8. **【理解】能够理解类和对象的区别**
9. 【理解】java中类的作用
10. 【应用】java中对象的作用
11. 【理解】java中类和对象的关系
12. **【应用】能够独立使用成员变量和局部变量**
13. 【理解】成员变量和局部变量的概念
14. 【应用】成员变量和局部变量的用法
15. 【理解】成员变量和局部变量的区别
16. **【应用】能够独立使用封装**
17. 【理解】封装的概念
18. 【理解】java中封装的体现
19. 【理解】使用封装的好处
20. 【理解】private关键字的作用
21. 【应用】封装的代码实例
22. **【应用】能够独立使用构造函数**
23. 【理解】构造函数的作用
24. 【应用】构造函数的写法
25. 【应用】构造函数的调用方式
26. **【应用】能够独立使用this关键字**
27. 【理解】this关键字的作用
28. 【应用】this关键字的用法
29. **【应用】API概述**
30. 【了解】能够阐述API的作用
31. 【应用】能够根据类名在API中快速找到该类
32. **【应用】Scanner类与String类**
33. 【应用】能够独立使用Scanner完成键盘录入字符串
34. 【应用】能够熟练写出String类的常用构造方法
35. 【理解】能够阐述创建字符串对象两种方式的区别
36. 【应用】能够独立写出String判断功能与获取功能相关案例
37. 【应用】能够独立写出String转换功能与其它功能相关案例
38. **【应用】StringBuilder类**
39. 【理解】能够阐述StringBuilder拼接字符串高效的原因
40. 【应用】能够完成String与StringBuilder相互转换的案例
41. 【应用】能够利用StringBuilder把数组拼接成一个字符串案例
42. 【应用】能够利用StringBuilder完成字符串反转案例
43. 【应用】能够利用StringBuilder完成判断一个字符串是否是对称字符串案例

# 面向对象基础

## 编程思想概述

### 面向过程思想概述

我们回想一下，这几天我们完成一个需求的步骤：首先是搞清楚我们要做什么，然后在分析怎么做，最后我们再代码体现。一步一步去实现，而具体的每一步都需要我们去实现和操作。这些步骤相互调用和协作，完成我们的需求。

在上面的每一个具体步骤中我们都是参与者，并且需要面对具体的每一个步骤和过程，这就是面向过程最直接的体现。

那么什么是面向过程开发呢? 面向过程开发，其实就是面向着具体的每一个步骤和过程，把每一个步骤和过程完成，然后由这些功能方法相互调用，完成需求。

面向过程的代表语言：C语言

### 面向对象思想概述

当需求单一，或者简单时，我们一步一步去操作没问题，并且效率也挺高。可随着需求的更改，功能的增多，发现需要面对每一个步骤很麻烦了，这时就开始思索，能不能把这些步骤和功能在进行封装，封装时根据不同的功能，进行不同的封装，功能类似的封装在一起。这样结构就清晰了很多。用的时候，找到对应的类就可以了。这就是面向对象的思想。

## 面向对象思想特点及举例

### 面向对象思想特点

面向对象是基于面向过程的编程思想；

面向过程：强调的是每一个功能的步骤；

面向对象：强调的是对象，然后由对象去调用功能；

A：是一种更符合我们思考习惯的思想

B：可以将复杂的事情简单化

C：将我们从执行者变成了指挥者

角色发生了转换

### 面向对象思想举例

买电脑：

面向过程：我要买电脑--我要明确买电脑的意义--上网查对应的参数信息--去电脑城买电脑--讨价还价--买回电脑

面向对象：我要买电脑--让采购去给我买电脑--买回电脑

洗衣服：

面向过程：把衣服脱下来--找一个盆--放点洗衣粉--加点水--浸泡10分钟--揉一揉--清洗衣服--拧干--晾起来

面向对象：把衣服脱下来--扔给全自动洗衣机--晾起来

## 类与对象的关系

我们学习编程语言，就是为了模拟现实世界的事物，实现信息化。比如：去超市买东西的计费系统，去银行办业务的系统。

我们如何表示一个现实世界事物呢：

A：属性 就是该事物的描述信息

B：行为 就是该事物能够做什么

举例：学生事物

我们学习的Java语言最基本单位是类，所以，我们就应该把事物用一个类来体现。

类：是一组相关的属性和行为的集合

对象：是该类事物的具体体现

举例：

类 学生

对象 班长就是一个对象

# 类和对象

## 类的定义

### 现实世界的事物

类是用来描述现实世界的事物的。

事物：

属性 事物的描述信息

行为 事物能做什么

### Java中用class描述事物也是如此

成员变量 就是事物的属性

成员方法 就是事物的行为

### 定义类其实就是定义类的成员(成员变量和成员方法)

1:成员变量 和以前定义变量是一样的。

只不过位置发生了改变：在类中，方法外。

初始化值：不需要给初始化值。

2:成员方法 和以前定义方法是一样的。

只不过把static去掉，面向对象进阶在详细讲解static的作用。

## 类与对象的案例

### 学生类(讲解)

#### 如何定义

按照事物到类的过程一步步分析

#### 案例代码一

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: Demo  \* **@Description**: 学生类的分析  \* **@date** 2017年11月3日 下午4:17:21  \* Company www.igeekhome.com  \*  \* 需求：写一个学生类  \*  \* 学生事物：  \* 属性：姓名，年龄...  \* 行为：学习，吃饭...  \*  \* 学生类：  \* 成员变量：姓名，年龄  \* 成员方法：学习，吃饭  \*  \* 1:成员变量 和以前定义变量是一样的。  \* 只不过位置发生了改变：在类中，方法外。  \* 初始化值：不需要给初始化值。  \* 2:成员方法 和以前定义方法是一样的。  \* 只不过把static去掉，面向对象进阶在详细讲解static的作用。  \*  \*/  **public** **class** Student {  //成员变量    /\*\*  \* **@Fields** name : 姓名  \*/  String name;    /\*\*  \* **@Fields** age : 年龄  \*/  **int** age;    //成员方法  /\*\*  \* **@Title**: study  \* **@Description**: 学习的方法  \*/  **public** **void** study(){  System.***out***.println("好好学习，天天向上！");  }    /\*\*  \* **@Title**: eat  \* **@Description**: 吃饭的方法  \*/  **public** **void** eat(){  System.***out***.println("学习饿了，要吃饭。");  }  } |

#### 如何使用

使用一个类，其实就是使用该类的成员。(成员变量和成员方法)

而我们要想使用一个类的成员，就必须首先拥有该类的对象。

我们该如何拥有一个类的对象呢？

创建对象就可以了？

我们如何创建对象呢？

创建对象的格式：

类名 对象名 = new 类名();

对象如何访问成员呢？

成员变量：对象名.成员变量

成员方法：对象名.成员方法

#### 代码案例二

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: StudentDemo  \* **@Description**: Student类的使用  \* **@date** 2017年11月10日 上午11:03:26  \* Company www.igeekhome.com  \*  \* Student类是一个学生事物描述类，所以main方法不适合放在它里面  \* 使用一个类，其实就是使用该类的成员。(成员变量和成员方法)  \* 而我们要想使用一个类的成员，就必须首先拥有该类的对象。  \*  \* 我们该如何拥有一个类的对象呢？  \* 创建对象就可以了？  \*  \* 我们如何创建对象呢？  \* 创建对象的格式：  \* 类名 对象名 = new 类名();  \* 对象如何访问成员呢？  \* 成员变量：对象名.成员变量  \* 成员方法：对象名.成员方法  \*/  **public** **class** StudentDemo {  **public** **static** **void** main(String[] args) {  //格式：类名 对象名=new 类名();  Student s=**new** Student();  System.***out***.println("s:"+s);//s:com.igeek\_02.Student@feb48    //直接输出成员变量值  System.***out***.println("姓名："+s.name);//null  System.***out***.println("年龄："+s.age);//0  System.***out***.println("---------------");    //给成员变量赋值  s.name="吴京";  s.age=35;    //再次输出成员变量的值  System.***out***.println("姓名："+s.name);//吴京  System.***out***.println("年龄："+s.age);//35  System.***out***.println("---------------");    //调用成员方法  s.study();  s.eat();  }  } |

### 手机类练习

#### 代码案例三

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: Phone  \* **@Description**: 手机类  \* **@date** 2017年11月10日 下午4:10:51  \* Company www.igeekhome.com  \*  \* 手机类：  \* 成员变量：品牌、价格、颜色...  \* 成员方法：打电话、发短信...  \*/  **public** **class** Phone {  /\*\*  \* **@Fields** brand : 品牌  \*/  String brand;  /\*\*  \* **@Fields** price : 价格  \*/  **int** price;  /\*\*  \* **@Fields** color : 颜色  \*/  String color;    /\*\*  \* **@Title**: call  \* **@Description**: 打电话  \* **@param** name  \*/  **public** **void** call(String name){  System.***out***.println("给"+name+"打电话");  }    /\*\*  \* **@Title**: sendMessage  \* **@Description**: 发短信  \*/  **public** **void** sendMessage(){  System.***out***.println("群发短信");  }  } |

#### 代码案例四

手机类的使用

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: PhoneDemo  \* **@Description**: 手机类的测试类  \* **@date** 2017年11月10日 下午4:19:28  \* Company www.igeekhome.com  \*  \*/  **public** **class** PhoneDemo {  **public** **static** **void** main(String[] args) {  //创建对象  Phone p=**new** Phone();    //输出成员变量  System.***out***.println("品牌："+p.brand);//null  System.***out***.println("价格："+p.price);//0  System.***out***.println("颜色："+p.color);//null  System.***out***.println("-------------------");    //给成员变量赋值  p.brand="华为Mate10";  p.price=3999;  p.color="黑色";    //再次输出成员变量  System.***out***.println("品牌："+p.brand);//华为Mate10  System.***out***.println("价格："+p.price);//3999  System.***out***.println("颜色："+p.color);//黑色  System.***out***.println("-------------------");    //调用成员方法  p.call("吴京");  p.sendMessage();  }  } |

## 对象内存图

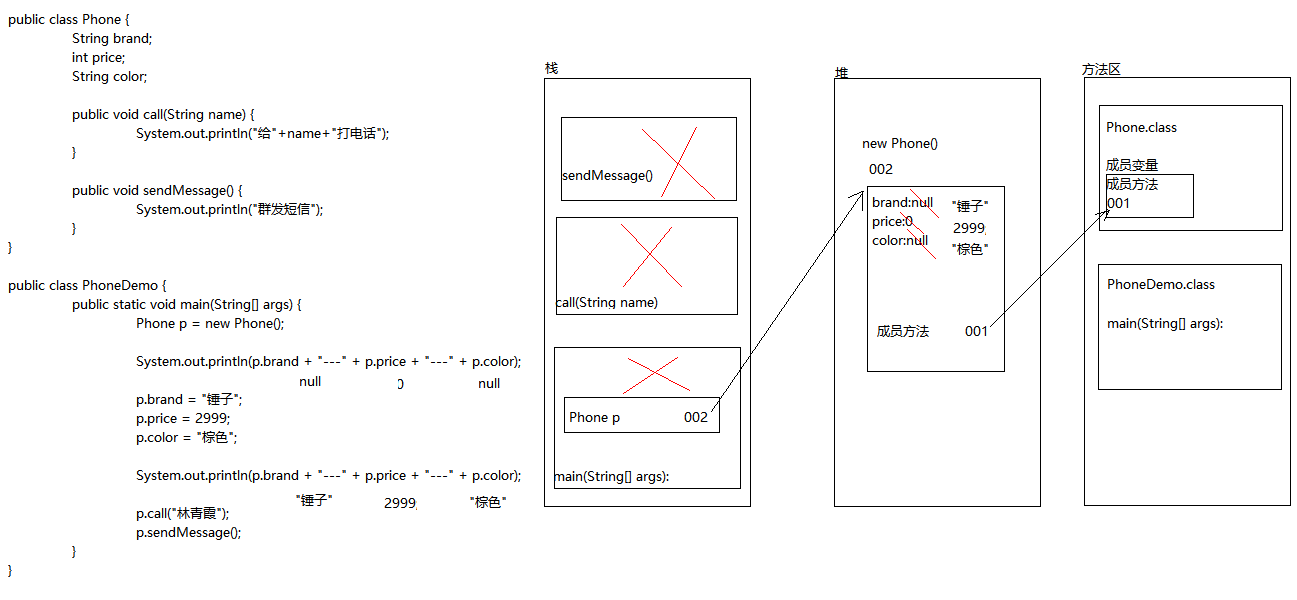
### 一个对象的情况

#### 代码案例五

|  |
| --- |
| **package** com.igeek\_03;  /\*\*  \* **@ClassName**: Phone  \* **@Description**: 手机类  \* **@date** 2017年11月10日 下午4:10:51  \* Company www.igeekhome.com  \*  \* 手机类：  \* 成员变量：品牌、价格、颜色...  \* 成员方法：打电话、发短信...  \*/  **public** **class** Phone {  /\*\*  \* **@Fields** brand : 品牌  \*/  String brand;  /\*\*  \* **@Fields** price : 价格  \*/  **int** price;  /\*\*  \* **@Fields** color : 颜色  \*/  String color;    /\*\*  \* **@Title**: call  \* **@Description**: 打电话  \* **@param** name  \*/  **public** **void** call(String name){  System.***out***.println("给"+name+"打电话");  }    /\*\*  \* **@Title**: sendMessage  \* **@Description**: 发短信  \*/  **public** **void** sendMessage(){  System.***out***.println("群发短信");  }  } |

|  |
| --- |
| **package** com.igeek\_03;  /\*\*  \* **@ClassName**: PhoneDemo  \* **@Description**: 手机类的测试类  \* **@date** 2017年11月10日 下午4:19:28  \* Company www.igeekhome.com  \*  \*/  **public** **class** PhoneDemo {  **public** **static** **void** main(String[] args) {  //创建对象  Phone p=**new** Phone();    //输出成员变量  System.***out***.println(p.brand+"---"+p.price+"---"+p.color);    //给成员变量赋值  p.brand="锤子";  p.price=2999;  p.color="棕色";    //再次输出成员变量  System.***out***.println(p.brand+"---"+p.price+"---"+p.color);    //调用成员方法  p.call("林青霞");  p.sendMessage();  }  } |

#### 1个对象的内存图

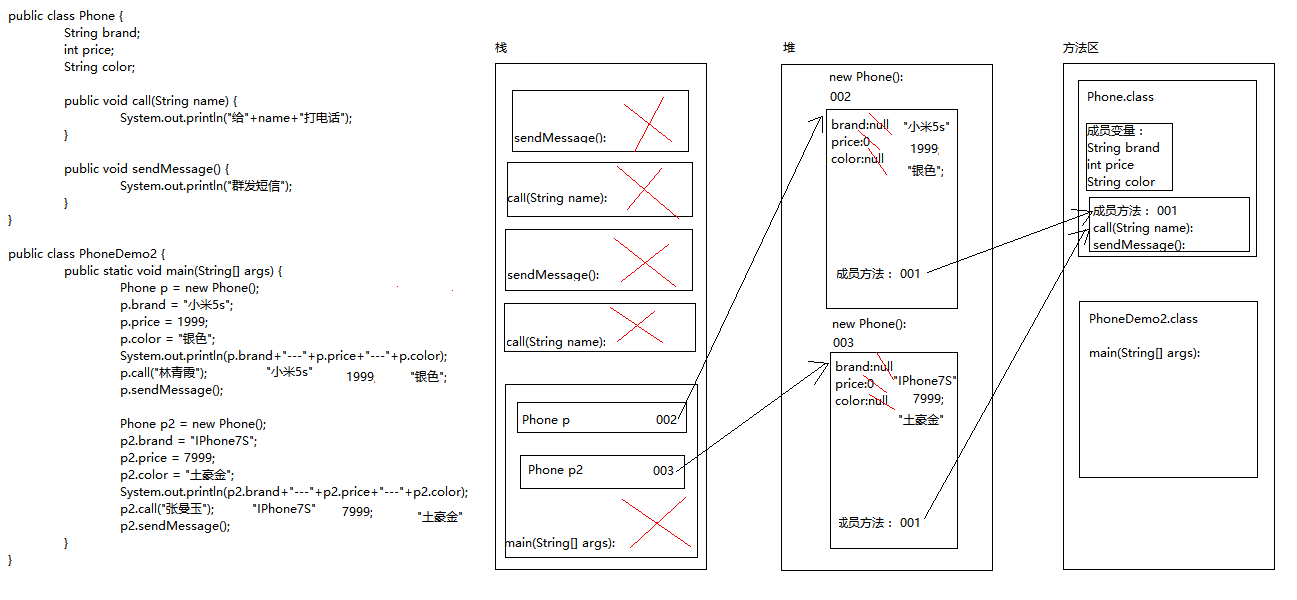
一个对象的基本初始化过程

### 两个对象的情况一

#### 代码案例六

|  |
| --- |
| **package** com.igeek\_03;  /\*\*  \* **@ClassName**: PhoneDemo2  \* **@Description**: 手机类的测试类  \* **@date** 2017年11月10日 下午4:19:28 Company www.igeekhome.com  \*  \*/  **public** **class** PhoneDemo2 {  **public** **static** **void** main(String[] args) {  // 创建对象  Phone p = **new** Phone();  // 给成员变量赋值  p.brand = "小米5s";  p.price = 1999;  p.color = "银色";  // 输出成员变量  System.***out***.println(p.brand + "---" + p.price + "---" + p.color);  // 调用成员方法  p.call("林青霞");  p.sendMessage();  // 创建对象  Phone p2 = **new** Phone();  // 给成员变量赋值  p2.brand = "IPhone7S";  p2.price = 7999;  p2.color = "土豪金";  // 输出成员变量  System.***out***.println(p2.brand + "---" + p2.price + "---" + p2.color);  // 调用成员方法  p2.call("张曼玉");  p2.sendMessage();  }  } |

#### 2个对象的内存图



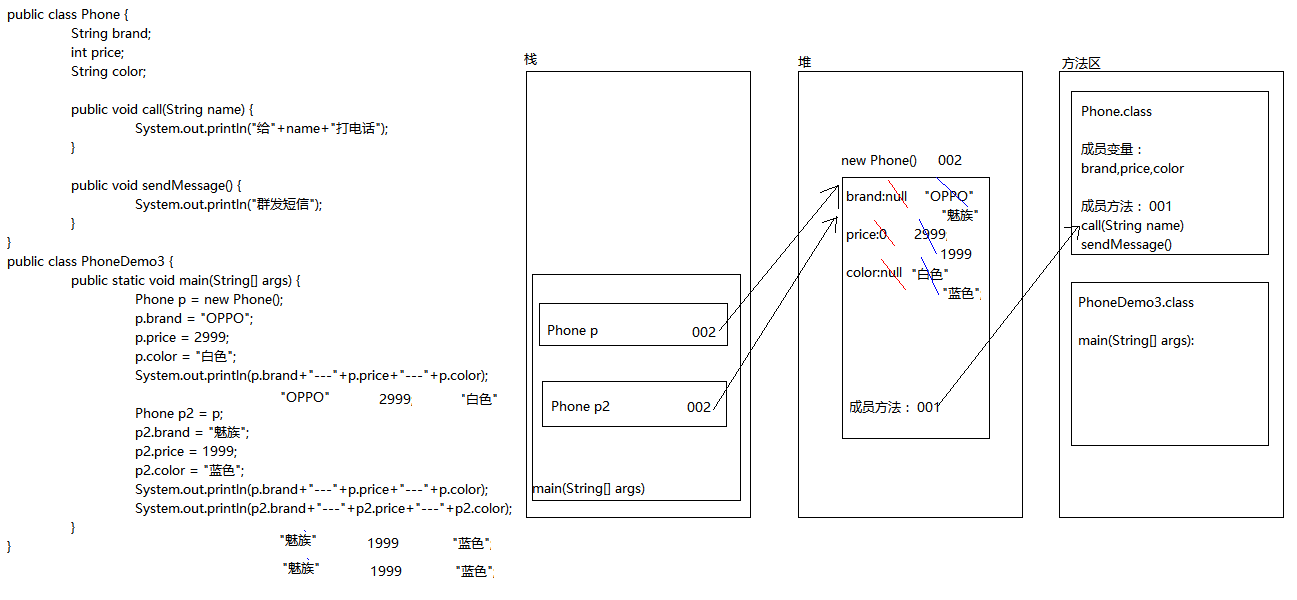
方法的共用

### 两个对象的情况二

#### 代码案例七

|  |
| --- |
| **package** com.igeek\_03;  /\*\*  \* **@ClassName**: PhoneDemo3  \* **@Description**: 手机类的测试类  \* **@date** 2017年11月10日 下午4:19:28 Company www.igeekhome.com  \*  \*/  **public** **class** PhoneDemo3 {  **public** **static** **void** main(String[] args) {  // 创建对象  Phone p = **new** Phone();  // 给成员变量赋值  p.brand = "OPPO";  p.price = 2999;  p.color = "白色";  // 输出成员变量  System.***out***.println(p.brand + "---" + p.price + "---" + p.color);  // 把p对象赋值给p2  Phone p2 = p;  // 给成员变量赋值  p2.brand = "魅族";  p2.price = 1999;  p2.color = "蓝色";  // 再次输出成员变量  System.***out***.println(p.brand + "---" + p.price + "---" + p.color);  System.***out***.println(p2.brand + "---" + p2.price + "---" + p2.color);  }  } |

#### 2个对象的内存图



两个引用指向同一个对象

## 成员变量和局部变量的区别

### 在类中的位置不同

成员变量： 类中方法外

局部变量： 方法内或者方法声明上(形式参数)

### 在内存中的位置不同

成员变量： 堆内存

局部变量： 栈内存

### 生命周期不同

成员变量： 随着对象的存在而存在，随着对象的消失而消失

局部变量： 随着方法的调用而存在，随着方法的调用完毕而消失

### 初始化值不同

成员变量： 有默认的初始化值

局部变量： 没有默认的初始化值，必须先定义，赋值，才能使用。

### 代码案例八

|  |
| --- |
| **package** com.igeek\_04;  /\*\*  \* **@ClassName**: Variable  \* **@Description**: 成员变量和局部变量的区别：  \* **@date** 2017年11月10日 下午5:03:05  \* Company www.igeekhome.com  \* 成员变量和局部变量的区别  \* A：在类中的位置不同  \* 成员变量： 类中方法外  \* 局部变量： 方法内或者方法声明上(形式参数)  \* B：在内存中的位置不同  \* 成员变量： 堆内存  \* 局部变量： 栈内存  \* C：生命周期不同  \* 成员变量： 随着对象的存在而存在，随着对象的消失而消失  \* 局部变量： 随着方法的调用而存在，随着方法的调用完毕而消失  \* D：初始化值不同  \* 成员变量： 有默认的初始化值  \* 局部变量： 没有默认的初始化值，必须先定义，赋值，才能使用。  \*/  **public** **class** Variable {  /\*\*  \* **@Fields** x : 成员变量x  \*/  **int** x;    /\*\*  \* **@Title**: show  \* **@Description**: 显示变量  \*/  **public** **void** show(){  //局部变量  **int** y=0;    System.***out***.println(x);  System.***out***.println(y);  }  } |

## private关键字

### private关键字：

是一个权限修饰符。

可以修饰成员(成员变量和成员方法)

被private修饰的成员只在本类中才能访问。

### 代码案例九

|  |
| --- |
| **package** com.igeek\_05;  /\*\*  \* **@ClassName**: Student  \* **@Description**: 学生类  \* **@date** 2017年11月10日 下午5:14:25  \* Company www.igeekhome.com  \*  \* 通过对象直接访问成员变量，会存在数据安全问题  \* 这个时候，我们就想能不能不让外界的对象直接访问成员变量呢？  \* 能！  \* 如何实现呢？  \* private关键字  \*  \* private：  \* 是一个权限修饰符。  \* 可以修饰成员(成员变量和成员方法)  \* 被private修饰的成员只能在本类中才能访问。  \*  \* 针对private修饰的成员变量，我们会相应的提供getXxx()和setXxx()用于获取和设置成员变量的值，方法用public修饰  \*/  **public** **class** Student {  /\*\*  \* **@Fields** name : 姓名  \*/  String name;  /\*\*  \* **@Fields** age : 年龄  \*/  //int age;  **private** **int** age;  /\*\*  \* **@Title**: setAge  \* **@Description**: 设置年龄  \* **@param** a  \*/  **public** **void** setAge(**int** a){  **if**(a<0||a>200){  System.***out***.println("你给的年龄有误");  }**else**{  age=a;  }  }    /\*\*  \* **@Title**: getAge  \* **@Description**: 获取年龄  \* **@return**  \*/  **public** **int** getAge(){  **return** age;  }    /\*\*  \* **@Title**: show  \* **@Description**: 显示学生信息  \*/  **public** **void** show(){  System.***out***.println("姓名是："+name+"，年龄是："+age);  }  } |

|  |
| --- |
| **package** com.igeek\_05;  /\*\*  \* **@ClassName**: StudentDemo  \* **@Description**: 学生测试类  \* **@date** 2017年11月10日 下午5:17:34  \* Company www.igeekhome.com  \*  \*/  **public** **class** StudentDemo {  **public** **static** **void** main(String[] args) {  //创建学生对象  Student s=**new** Student();  s.show();    s.name="林青霞";  //s.age=28;  //s.age=-28;  //s.setAge(-28);  s.setAge(28);  s.show();  }  } |

### private最常见的应用：

把成员变量用private修饰

提供对应的getXxx()/setXxx()方法

一个标准的案例的使用

### 代码案例十

|  |
| --- |
| **package** com.igeek\_05\_03;  /\*\*  \* **@ClassName**: Student  \* **@Description**: 学生类  \* **@date** 2017年11月10日 下午5:14:25  \* Company www.igeekhome.com  \*  \*/  **public** **class** Student {  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;  /\*\*  \* **@Fields** age : 年龄  \*/  **private** **int** age;    /\*\*  \* **@Title**: setName  \* **@Description**: 设置姓名  \* **@param** n  \*/  **public** **void** setName(String n){  name=n;  }    /\*\*  \* **@Title**: getName  \* **@Description**: 获取姓名  \* **@return**  \*/  **public** String getName(){  **return** name;  }    /\*\*  \* **@Title**: setAge  \* **@Description**: 设置年龄  \* **@param** a  \*/  **public** **void** setAge(**int** a){  age=a;  }    /\*\*  \* **@Title**: getAge  \* **@Description**: 获取年龄  \* **@return**  \*/  **public** **int** getAge(){  **return** age;  }    } |

|  |
| --- |
| **package** com.igeek\_05\_03;  /\*\*  \* **@ClassName**: StudentDemo  \* **@Description**: 学生类的测试类  \* **@date** 2017年11月10日 下午5:17:34  \* Company www.igeekhome.com  \*  \*/  **public** **class** StudentDemo {  **public** **static** **void** main(String[] args) {  //创建学生对象  Student s=**new** Student();  System.***out***.println(s.getName()+"---"+s.getAge());    s.setName("吴京");  s.setAge(35);  System.***out***.println(s.getName()+"---"+s.getAge());  }  } |

# 封装

## 什么是封装

### 封装概述

是面向对象三大特征之一

是面向对象编程语言对客观世界的模拟，客观世界里成员变量都是隐藏在对象内部的，外界无法直接操作和修改。就像上章说的年龄。

### 封装原则：

将不需要对外提供的内容都隐藏起来。

把属性隐藏，提供公共方法对其访问。

成员变量private，提供对应的getXxx()/setXxx()方法

### 好处：

通过方法来控制成员变量的操作，提高了代码的安全性

把代码用方法进行封装，提高了代码的复用性

## this关键字

this: 代表所在类的对象引用

记住： 方法被哪个对象调用，this就代表那个对象

什么时候使用this呢? 局部变量隐藏成员变量

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: Student  \* **@Description**: 学生类  \* **@date** 2017年11月10日 下午5:14:25  \* Company www.igeekhome.com  \*  \* 起名字我们要求做到见名知意。  \* 而我们现在的代码中的n和a就没有做到见名知意，所以我们要改进。  \*  \* 如果有局部变量名和成员变量名相同，在局部使用的时候，采用的是就近原则。  \*  \* 我们有没有办法把局部变量的name赋值给成员变量的name呢？有  \*  \* 什么办法呢？  \* 用this关键字就可以解决这个问题。  \*  \* this: 代表所在类的对象引用  \* 方法被哪个对象调用，this就代表那个对象  \*  \* 使用场景： 局部变量隐藏成员变量  \*/  **public** **class** Student {  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;  /\*\*  \* **@Fields** age : 年龄  \*/  **private** **int** age;    /\*\*  \* **@Title**: setName  \* **@Description**: 设置姓名  \* **@param** n  \*/  **public** **void** setName(String name){//吴京  //name=name;  **this**.name=name;  }    /\*\*  \* **@Title**: getName  \* **@Description**: 获取姓名  \* **@return**  \*/  **public** String getName(){  **return** name;  }    /\*\*  \* **@Title**: setAge  \* **@Description**: 设置年龄  \* **@param** a  \*/  **public** **void** setAge(**int** age){  //age=age;  **this**.age=age;  }    /\*\*  \* **@Title**: getAge  \* **@Description**: 获取年龄  \* **@return**  \*/  **public** **int** getAge(){  **return** age;  }    } |

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: StudentDemo  \* **@Description**: 学生类的测试类  \* **@date** 2017年11月10日 下午5:17:34  \* Company www.igeekhome.com  \*  \*/  **public** **class** StudentDemo {  **public** **static** **void** main(String[] args) {  //创建学生对象  Student s=**new** Student();  System.***out***.println(s.getName()+"---"+s.getAge());    s.setName("吴京");  s.setAge(35);  System.***out***.println(s.getName()+"---"+s.getAge());  }  } |

## 构造方法

### 构造方法作用概述

给对象的数据进行初始化

### 构造方法格式

方法名与类名相同

没有返回值类型，连void都没有

没有具体的返回值

#### 代码案例十一

|  |
| --- |
| **package** com.igeek\_03;  /\*\*  \* **@ClassName**: Student  \* **@Description**: 学生类  \* **@date** 2017年11月11日 上午9:23:37  \* Company www.igeekhome.com  \*  \* 构造方法：  \* 给对象的数据进行初始化  \*  \* 格式：  \* 方法名与类名相同  \* 没有返回值类型，连void都没有  \* 没有具体的返回值  \*  \*/  **public** **class** Student {  /\*\*  \* **@Title**: Student类的无参构造方法  \*/  **public** Student(){  System.***out***.println("这是构造方法");  }  } |

|  |
| --- |
| **package** com.igeek\_03;  /\*\*  \* **@ClassName**: StudentDemo  \* **@Description**: 学生类的测试类  \* **@date** 2017年11月11日 上午9:26:15  \* Company www.igeekhome.com  \*  \*/  **public** **class** StudentDemo {  **public** **static** **void** main(String[] args) {  //如何调用构造方法呢？  //通过new关键字调用  //格式：类名 对象名 = new 构造方法(...);  //也就是创建对象的格式  Student s=**new** Student();  }  } |

### 构造方法注意事项

如果你不提供构造方法，系统会给出默认构造方法

如果你提供了构造方法，系统将不再提供

构造方法也是可以重载的

#### 代码案例十二

|  |
| --- |
| **package** com.igeek\_03\_03;  /\*\*  \* **@ClassName**: Student  \* **@Description**: 学生类  \* **@date** 2017年11月11日 上午9:23:37  \* Company www.igeekhome.com  \*  \* 构造方法：  \* 给对象的数据进行初始化  \*  \* 格式：  \* 方法名与类名相同  \* 没有返回值类型，连void都没有  \* 没有具体的返回值  \*  \* 构造方法注意事项:  \* A:如果你不提供构造方法，系统会给出一个默认无参构造方法  \* B:如果你提供了构造方法，系统将不再提供默认无参构造方法  \* 这个时候，如果想使用无参构造方法，就必须自己提供  \* 推荐：自己给出无参构造方法  \* C:构造方法也是可以重载的  \*  \* 成员变量赋值：  \* A：setXxx()方法  \* B：带参数构造方法  \*/  **public** **class** Student {  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;  /\*\*  \* **@Fields** age : 年龄  \*/  **private** **int** age;    /\*\*  \* **@Title**: Student类的无参构造方法  \*/  /\*  public Student(){  System.out.println("这是构造方法");  }\*/    **public** Student(){}    /\*\*  \* **@Title**: Student  \* **@param** name  \*/  **public** Student(String name){  **this**.name=name;  }  /\*\*  \* **@Title**: Student  \* **@param** age  \*/  **public** Student(**int** age){  **this**.age=age;  }    /\*\*  \* **@Title**: Student  \* **@param** name  \* **@param** age  \*/  **public** Student(String name,**int** age){  **this**.name=name;  **this**.age=age;  }    /\*\*  \* **@Title**: show  \* **@Description**: 显示学生信息方法  \*/  **public** **void** show(){  System.***out***.println(name+"---"+age);  }  } |

|  |
| --- |
| **package** com.igeek\_03\_03;  /\*\*  \* **@ClassName**: StudentDemo  \* **@Description**: 学生类的测试类  \* **@date** 2017年11月11日 上午9:26:15  \* Company www.igeekhome.com  \*  \*/  **public** **class** StudentDemo {  **public** **static** **void** main(String[] args) {  //如何调用构造方法呢？  //通过new关键字调用  //格式：类名 对象名 = new 构造方法(...);  //也就是创建对象的格式  //public Student()  Student s=**new** Student();  s.show();    //public Student(String name)  Student s2=**new** Student("吴京");  s2.show();    //public Student(int age)  Student s3=**new** Student(35);  s3.show();    //public Student(String name,int age)  Student s4=**new** Student("吴京", 35);  s4.show();  }  } |

## 标准类的代码写法和测试

### 类

- 成员变量

- 构造方法

* 无参构造方法
* 带参构造方法

- 成员方法

* + - getXxx()
    - setXxx()

### 给成员变量赋值的方式

无参构造方法+setXxx()

带参构造方法

### 面向对象标准案例

#### 学生类(老师讲)

#### 代码案例十三

|  |
| --- |
| **package** com.igeek\_04;  /\*\*  \* **@ClassName**: Student  \* **@Description**: 学生类  \* **@date** 2017年11月11日 上午9:23:37  \* Company www.igeekhome.com  \*  \*/  **public** **class** Student {  //成员变量  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;  /\*\*  \* **@Fields** age : 年龄  \*/  **private** **int** age;    //构造方法  /\*\*  \* **@Title**: Student  \*/  **public** Student(){}    /\*\*  \* **@Title**: Student  \* **@param** name  \* **@param** age  \*/  **public** Student(String name,**int** age){  **this**.name=name;  **this**.age=age;  }    //成员方法  /\*\*  \* **@Title**: setName  \* **@Description**: 设置姓名  \* **@param** name  \*/  **public** **void** setName(String name){  **this**.name=name;  }    /\*\*  \* **@Title**: getName  \* **@Description**: 获取姓名  \* **@return**  \*/  **public** String getName(){  **return** name;  }    /\*\*  \* **@Title**: setAge  \* **@Description**: 设置年龄  \* **@param** age  \*/  **public** **void** setAge(**int** age){  **this**.age=age;  }    /\*\*  \* **@Title**: getAge  \* **@Description**: 获取年龄  \* **@return**  \*/  **public** **int** getAge(){  **return** age;  }    } |

|  |
| --- |
| **package** com.igeek\_04;  /\*\*  \* **@ClassName**: StudentDemo  \* **@Description**: 学生类的测试类  \* **@date** 2017年11月11日 上午9:26:15  \* Company www.igeekhome.com  \*  \*/  **public** **class** StudentDemo {  **public** **static** **void** main(String[] args) {  //无参构造方法+setXxx()方法  Student s=**new** Student();  s.setName("吴京");  s.setAge(35);  System.***out***.println(s.getName()+"---"+s.getAge());    //带参构造方法  Student s2=**new** Student("吴京", 35);  System.***out***.println(s2.getName()+"---"+s2.getAge());  }  } |

手机类(学生练)

汽车类(学生练)

## 类名作为形式参数和返回值

### 类名作为形式参数案例

要的其实是该类的对象

#### 代码案例十四

|  |
| --- |
| **package** com.igeek\_05\_01;  /\*\*  \* **@ClassName**: Student  \* **@Description**: 学生类  \* **@date** 2017年11月11日 上午10:20:28  \* Company www.igeekhome.com  \*  \*/  **public** **class** Student {  /\*\*  \* **@Title**: study  \* **@Description**: 学习方法  \*/  **public** **void** study(){  System.***out***.println("好好学习，天天向上");  }  } |

|  |
| --- |
| **package** com.igeek\_05\_01;  /\*\*  \* **@ClassName**: Teacher  \* **@Description**: 老师类  \* **@date** 2017年11月11日 上午10:21:13  \* Company www.igeekhome.com  \*  \*/  **public** **class** Teacher {  /\*\*  \* **@Title**: test  \* **@Description**: 测试学生学习的方法  \* **@param** s  \*/  **public** **void** test(Student s){  s.study();  }  } |

|  |
| --- |
| **package** com.igeek\_05\_01;  /\*\*  \* **@ClassName**: Test  \* **@Description**: 调用Teacher的test()方法  \* **@date** 2017年11月11日 上午10:22:21  \* Company www.igeekhome.com  \*  \* 类名作为形式参数时：其实这里实际需要传递的是该类的对象  \*/  **public** **class** Test {  **public** **static** **void** main(String[] args) {  Teacher t=**new** Teacher();  Student s=**new** Student();  t.test(s);  }  } |

### 类名作为返回值案例

返回的其实是该类的对象

#### 代码案例十五

|  |
| --- |
| **package** com.igeek\_05\_02;  /\*\*  \* **@ClassName**: Student  \* **@Description**: 学生类  \* **@date** 2017年11月11日 上午10:20:28  \* Company www.igeekhome.com  \*  \*/  **public** **class** Student {  /\*\*  \* **@Title**: study  \* **@Description**: 学习方法  \*/  **public** **void** study(){  System.***out***.println("好好学习，天天向上");  }  } |

|  |
| --- |
| **package** com.igeek\_05\_02;  /\*\*  \* **@ClassName**: Teacher  \* **@Description**: 老师类  \* **@date** 2017年11月11日 上午10:21:13  \* Company www.igeekhome.com  \*  \*/  **public** **class** Teacher {  /\*\*  \* **@Title**: getStudent  \* **@Description**: 获取学生对象  \* **@return**  \*/  **public** Student getStudent(){  Student s=**new** Student();  **return** s;  }  } |

|  |
| --- |
| **package** com.igeek\_05\_02;  /\*\*  \* **@ClassName**: Test  \* **@Description**: 通过Teacher对象得到Student对象，然后调用Student对象的方法  \* **@date** 2017年11月11日 上午10:22:21  \* Company www.igeekhome.com  \*  \* 如果方法的返回值是类名：其实返回的是该类的对象  \*/  **public** **class** Test {  **public** **static** **void** main(String[] args) {  Teacher t=**new** Teacher();  Student s=t.getStudent();  s.study();  }  } |

# API概述

## API概念

API(Application Programming Interface) : 应用程序编程接口

编写一个机器人程序去控制机器人踢足球，程序就需要向机器人发出向前跑、向后跑、射门、抢球等各种命令，没有编过程序的人很难想象这样的程序如何编写。但是对于有经验的开发人员来说，知道机器人厂商一定会提供一些用于控制机器人的Java类，这些类中定义好了操作机器人各种动作的方法。其实，这些Java类就是机器人厂商提供给应用程序编程的接口，大家把这些类称为API。本章涉及的Java API指的就是JDK中提供的各种功能的Java类

## 快速使用API步骤:

A:打开帮助文档

B:点击显示，找到索引，看到输入框

C:你要学习什么内容，你就在框框里面输入什么内容

举例：Random

D:看包

java.lang包下的类在使用的时候是不需要导包的

E:看类的描述

Random类是用于生成随机数的类

F:看构造方法

Random():无参构造方法

Random r = new Random();

G:看成员方法

public int nextInt(int n):产生的是一个[0,n)范围内的随机数

调用方法：

看返回值类型：人家返回什么类型，你就用什么类型接收

看方法名：名字不要写错了

看形式参数：人家要几个参数，你就给几个，人家要什么数据类型的，你就给什么数据类型的

int number = r.nextInt(100);

# Scanner类与String类

## Scanner类

### Scanner类作用

用Scanner类的方法可以完成接收键盘录入的数据

### Scanner类接受键盘录入的字符串

#### 案例代码十六

|  |
| --- |
| **package** com.igeek\_01;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: ScannerDemo  \* **@Description**: Scanner的作用  \* **@date** 2017年11月11日 上午10:41:35  \* Company www.igeekhome.com  \*  \* Scanner:用于获取键盘录入的数据。(基本数据类型，字符串数据)  \* public String nextLine():获取键盘录入的字符串数据  \*/  **public** **class** ScannerDemo {  **public** **static** **void** main(String[] args) {  //创建键盘录入对象  Scanner sc = **new** Scanner(System.***in***);  //接收数据  System.***out***.println("请输入一个字符串数据：");  String s = sc.nextLine();    //输出结果  System.***out***.println("s:"+s);  }  } |

## String类

### String类概述

通过JDK提供的API，查看String类的说明

A:"abc"是String类的一个实例,或者成为String类的一个对象

B:字符串字面值"abc"也可以看成是一个字符串对象

C:字符串是常量，一旦被赋值，就不能被改变

D:字符串本质是一个字符数组

### String类的构造方法

String(String original):把字符串数据封装成字符串对象

String(char[] value):把字符数组的数据封装成字符串对象

String(char[] value, int index, int count):把字符数组中的一部分数据封装成字符串对象

#### 常用构造方法演示

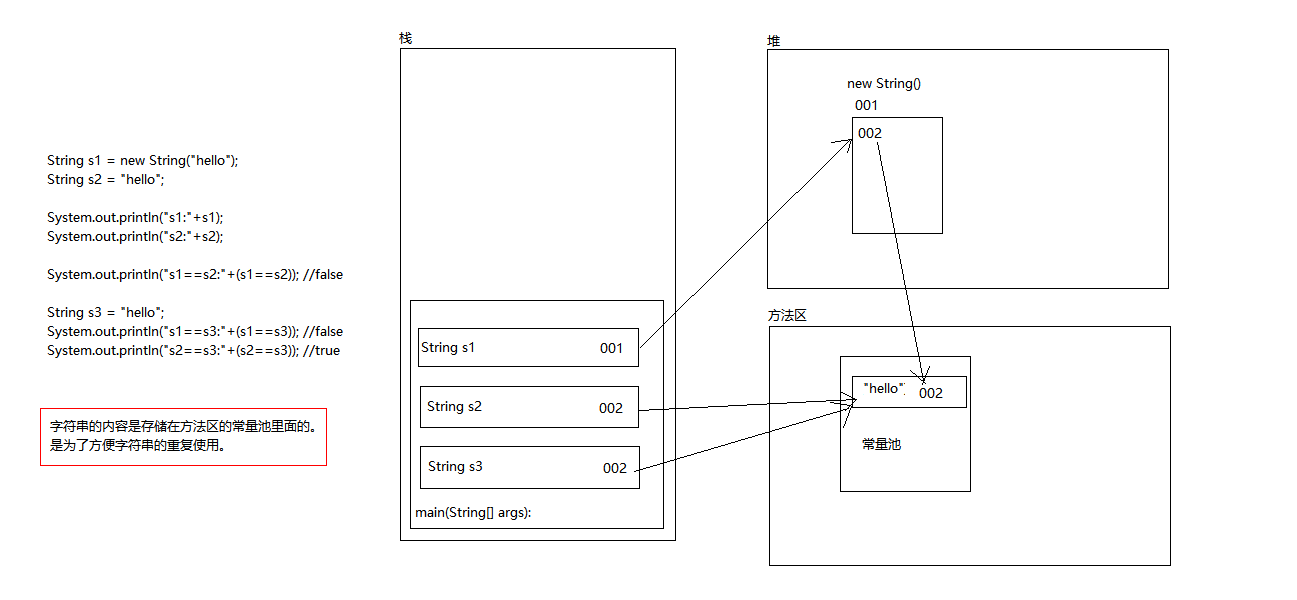
##### 案例代码十七

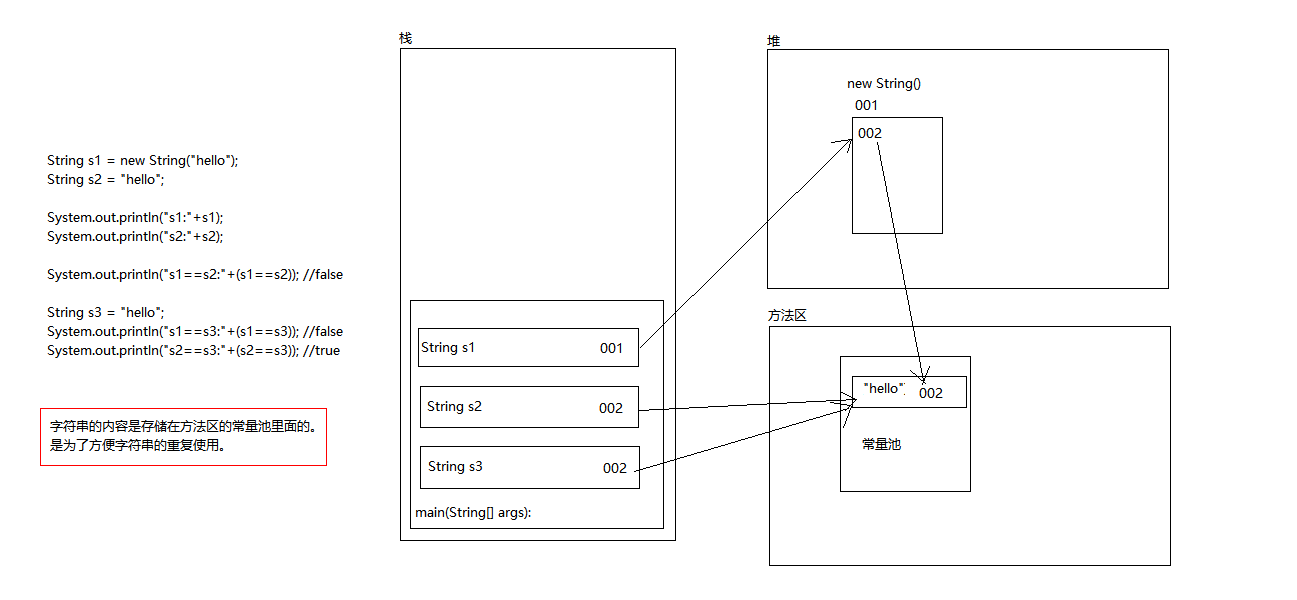
|  |
| --- |
| **package** com.igeek\_02\_02;  /\*\*  \* **@ClassName**: StringDemo  \* **@Description**: 字符串常用构造方法演示  \* **@date** 2017年11月11日 上午10:44:21  \* Company www.igeekhome.com  \*  \* String:字符串类  \* 由多个字符组成的一串数据  \* 字符串其本质是一个字符数组  \*  \* 构造方法：  \* String(String original):把字符串数据封装成字符串对象  \* String(char[] value):把字符数组的数据封装成字符串对象  \* String(char[] value, int index, int count):把字符数组中的一部分数据封装成字符串对象  \*  \* 注意：字符串是一种比较特殊的引用数据类型，直接输出字符串对象输出的是该对象中的数据。  \*  \*/  **public** **class** StringDemo {  **public** **static** **void** main(String[] args) {  //方式1  //String(String original):把字符串数据封装成字符串对象  String s1 = **new** String("hello");  System.***out***.println("s1:"+s1);  System.***out***.println("---------");    //方式2  //String(char[] value):把字符数组的数据封装成字符串对象  **char**[] chs = {'h','e','l','l','o'};  String s2 = **new** String(chs);  System.***out***.println("s2:"+s2);  System.***out***.println("---------");    //方式3  //String(char[] value, int index, int count):把字符数组中的一部分数据封装成字符串对象  //String s3 = new String(chs,0,chs.length);  String s3 = **new** String(chs,1,3);  System.***out***.println("s3:"+s3);  System.***out***.println("---------");    //方式4  String s4 = "hello";  System.***out***.println("s4:"+s4);  }  } |

#### 创建字符串对象两种方式的区别

##### 案例代码十八

|  |
| --- |
| **package** com.igeek\_02\_02;  /\*\*  \* **@ClassName**: StringDemo2  \* **@Description**: 创建字符串对象两种方式的区别  \* **@date** 2017年11月11日 上午10:47:56  \* Company www.igeekhome.com  \*  \* 通过构造方法创建的字符串对象和直接赋值方式创建的字符串对象有什么区别呢?  \* 通过构造方法创建字符串对象是在堆内存。  \* 直接赋值方式创建对象是在方法区的常量池。  \*  \* ==:  \* 基本数据类型：比较的是基本数据类型的值是否相同  \* 引用数据类型：比较的是引用数据类型的地址值是否相同  \*  \*/  **public** **class** StringDemo2 {  **public** **static** **void** main(String[] args) {  String s1 = **new** String("hello");  String s2 = "hello";    System.***out***.println("s1:"+s1);  System.***out***.println("s2:"+s2);    System.***out***.println("s1==s2:"+(s1==s2)); //false    String s3 = "hello";  System.***out***.println("s1==s3:"+(s1==s3)); //false  System.***out***.println("s2==s3:"+(s2==s3)); //true  }  } |





### String类的判断功能

boolean equals(Object obj):比较字符串的内容是否相同

boolean equalsIgnoreCase(String str):比较字符串的内容是否相同,忽略大小写

boolean startsWith(String str):判断字符串对象是否以指定的str开头

boolean endsWith(String str):判断字符串对象是否以指定的str结尾

#### 判断方法演示

##### 案例代码十九

|  |
| --- |
| **package** com.igeek\_02\_03;  /\*\*  \* **@ClassName**: StringDemo  \* **@Description**: String判断方法演示  \* **@date** 2017年11月11日 上午10:57:15  \* Company www.igeekhome.com  \*  \* Object:是类层次结构中的根类，所有的类都直接或者间接的继承自该类。  \* 如果一个方法的形式参数是Object，那么这里我们就可以传递它的任意的子类对象。  \*  \* String类的判断功能：  \* boolean equals(Object obj):比较字符串的内容是否相同  \* boolean equalsIgnoreCase(String str):比较字符串的内容是否相同,忽略大小写  \* boolean startsWith(String str):判断字符串对象是否以指定的str开头  \* boolean endsWith(String str):判断字符串对象是否以指定的str结尾  \*/  **public** **class** StringDemo {  **public** **static** **void** main(String[] args) {  //创建字符串对象  String s1 = "hello";  String s2 = "hello";  String s3 = "Hello";    //boolean equals(Object obj):比较字符串的内容是否相同  System.***out***.println(s1.equals(s2));  System.***out***.println(s1.equals(s3));  System.***out***.println("-----------");    //boolean equalsIgnoreCase(String str):比较字符串的内容是否相同,忽略大小写  System.***out***.println(s1.equalsIgnoreCase(s2));  System.***out***.println(s1.equalsIgnoreCase(s3));  System.***out***.println("-----------");    //boolean startsWith(String str):判断字符串对象是否以指定的str开头  System.***out***.println(s1.startsWith("he"));  System.***out***.println(s1.startsWith("ll"));  }  } |

#### 判断功能案例

需求:模拟登录,给三次机会,并提示还有几次

##### 案例代码二十

|  |
| --- |
| **package** com.igeek\_02\_03;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: StringTest  \* **@Description**: 需求:模拟登录,给三次机会,并提示还有几次  \* **@date** 2017年11月11日 上午10:59:52  \* Company www.igeekhome.com  \*  \* 分析：  \* A:定义两个字符串对象，用于存储已经存在的用户名和密码  \* B:键盘录入用户名和密码  \* C:拿键盘录入的用户名和密码和已经存在的用户名和密码进行比较  \* 如果内容相同，提示登录成功  \* 如果内容不同，提示登录失败，并提示还有几次机会  \*/  **public** **class** StringTest {  **public** **static** **void** main(String[] args) {  //定义两个字符串对象，用于存储已经存在的用户名和密码  String username = "admin";  String password = "admin";    //给三次机会，用for循环实现  **for**(**int** x=0; x<3; x++) {  //键盘录入用户名和密码  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入用户名：");  String name = sc.nextLine();  System.***out***.println("请输入密码：");  String pwd = sc.nextLine();    //拿键盘录入的用户名和密码和已经存在的用户名和密码进行比较  **if**(username.equals(name) && password.equals(pwd)) {  System.***out***.println("登录成功");  **break**;  }**else** {  **if**(x==2) {  System.***out***.println("用户名和密码被锁定,请与管理员联系");  }**else** {  System.***out***.println("登录失败,你还有"+(2-x)+"次机会"); //2,1,0  }  }  }  }  } |

### String类的获取功能

#### 获取方法演示

##### 代码案例二十一

|  |
| --- |
| **package** com.igeek\_02\_04;  /\*\*  \* **@ClassName**: StringDemo  \* **@Description**: String类的获取功能  \* **@date** 2017年11月11日 上午11:03:08  \* Company www.igeekhome.com  \*  \* String类的获取功能：  \* int length():获取字符串的长度，其实也就是字符个数  \* char charAt(int index):获取指定索引处的字符  \* int indexOf(String str):获取str在字符串对象中第一次出现的索引  \* String substring(int start):从start开始截取字符串  \* String substring(int start,int end):从start开始，到end结束截取字符串。包括start，不包括end  \*/  **public** **class** StringDemo {  **public** **static** **void** main(String[] args) {  //创建字符串对象  String s = "helloworld";    //int length():获取字符串的长度，其实也就是字符个数  System.***out***.println(s.length());  System.***out***.println("--------");    //char charAt(int index):获取指定索引处的字符  System.***out***.println(s.charAt(0));  System.***out***.println(s.charAt(1));  System.***out***.println("--------");    //int indexOf(String str):获取str在字符串对象中第一次出现的索引  System.***out***.println(s.indexOf("l"));  System.***out***.println(s.indexOf("owo"));  System.***out***.println(s.indexOf("ak"));  System.***out***.println("--------");    //String substring(int start):从start开始截取字符串  System.***out***.println(s.substring(0));  System.***out***.println(s.substring(5));  System.***out***.println("--------");    //String substring(int start,int end):从start开始，到end结束截取字符串  System.***out***.println(s.substring(0, s.length()));  System.***out***.println(s.substring(3,8));  }  } |

#### 获取功能案例

##### 案例代码二十二

|  |
| --- |
| **package** com.igeek\_02\_04;  /\*\*  \* **@ClassName**: StringTest  \* **@Description**: 遍历字符串(获取字符串中的每一个字符)  \* **@date** 2017年11月11日 上午11:09:15  \* Company www.igeekhome.com  \*  \*/  **public** **class** StringTest {  **public** **static** **void** main(String[] args) {  //创建一个字符串对象  String s = "abcde";    //原始做法  System.***out***.println(s.charAt(0));  System.***out***.println(s.charAt(1));  System.***out***.println(s.charAt(2));  System.***out***.println(s.charAt(3));  System.***out***.println(s.charAt(4));  System.***out***.println("---------");    //用for循环改进  **for**(**int** x=0; x<5; x++) {  System.***out***.println(s.charAt(x));  }  System.***out***.println("---------");    //用length()方法获取字符串的长度  **for**(**int** x=0; x<s.length(); x++) {  System.***out***.println(s.charAt(x));  }  }  } |

##### 案例代码二十三

|  |
| --- |
| **package** com.igeek\_02\_04;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: StringTest2  \* **@Description**: 统计一个字符串中大写字母字符，小写字母字符，数字字符出现的次数。(不考虑其他字符)  \* **@date** 2017年11月11日 上午11:10:44  \* Company www.igeekhome.com  \*  \* 分析：  \* A:键盘录入一个字符串数据  \* B:定义三个统计变量，初始化值都是0  \* C:遍历字符串，得到每一个字符  \* D:拿字符进行判断  \* 假如ch是一个字符。  \* 大写：ch>='A' && ch<='Z'  \* 小写：ch>='a' && ch<='z'  \* 数字：ch>='0' && ch<='9'  \* E:输出结果  \*/  **public** **class** StringTest2 {  **public** **static** **void** main(String[] args) {  //键盘录入一个字符串数据  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入一个字符串数据：");  String s = sc.nextLine();    //定义三个统计变量，初始化值都是0  **int** bigCount = 0;  **int** smallCount = 0;  **int** numberCount = 0;    //遍历字符串，得到每一个字符  **for**(**int** x=0; x<s.length(); x++) {  **char** ch = s.charAt(x);  //拿字符进行判断  **if**(ch>='A' && ch<='Z') {  bigCount++;  }**else** **if**(ch>='a' && ch<='z') {  smallCount++;  }**else** **if**(ch>='0' && ch<='9') {  numberCount++;  }**else** {  System.***out***.println("该字符"+ch+"非法");  }  }    //输出结果  System.***out***.println("大写字符："+bigCount+"个");  System.***out***.println("小写字符："+smallCount+"个");  System.***out***.println("数字字符："+numberCount+"个");  }  } |

### String类的转换功能

#### 转换方法演示

char[] toCharArray():把字符串转换为字符数组

String toLowerCase():把字符串转换为小写字符串

String toUpperCase():把字符串转换为大写字符串

##### 案例代码二十四

|  |
| --- |
| **package** com.igeek\_02\_05;  /\*\*  \* **@ClassName**: StringDemo  \* **@Description**: String类的转换功能  \* **@date** 2017年11月11日 上午11:13:33  \* Company www.igeekhome.com  \*  \* String类的转换功能：  \* char[] toCharArray():把字符串转换为字符数组  \* String toLowerCase():把字符串转换为小写字符串  \* String toUpperCase():把字符串转换为大写字符串  \*  \* 字符串的遍历：  \* A:length()加上charAt()  \* B:把字符串转换为字符数组，然后遍历数组  \*/  **public** **class** StringDemo {  **public** **static** **void** main(String[] args) {  //创建字符串对象  String s = "abcde";    //char[] toCharArray():把字符串转换为字符数组  **char**[] chs = s.toCharArray();  **for**(**int** x=0; x<chs.length; x++) {  System.***out***.println(chs[x]);  }  System.***out***.println("-----------");    //String toLowerCase():把字符串转换为小写字符串  System.***out***.println("HelloWorld".toLowerCase());  //String toUpperCase():把字符串转换为大写字符串  System.***out***.println("HelloWorld".toUpperCase());  }  } |

#### 转换功能案例

##### 案例代码二十五

|  |
| --- |
| **package** com.igeek\_02\_05;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: StringTest  \* **@Description**: 键盘录入一个字符串，把该字符串的首字母转成大写，其余为小写。(只考虑英文大小写字母字符)  \* **@date** 2017年11月11日 上午11:16:20  \* Company www.igeekhome.com  \*  \* 分析：  \* A:键盘录入一个字符串  \* B:截取首字母  \* C:截取除了首字母以外的字符串  \* D:B转大写+C转小写  \* E:输出即可  \*/  **public** **class** StringTest {  **public** **static** **void** main(String[] args) {  //键盘录入一个字符串  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入一个字符串：");  String s = sc.nextLine();    //截取首字母  String s1 = s.substring(0, 1);    //截取除了首字母以外的字符串  String s2 = s.substring(1);    //B转大写+C转小写  String s3 = s1.toUpperCase()+s2.toLowerCase();    //输出即可  System.***out***.println("s3:"+s3);  }  } |

### String类的其它功能

#### 其它方法演示

##### 案例代码二十六

|  |
| --- |
| **package** com.igeek\_02\_06;  /\*\*  \* **@ClassName**: StringDemo  \* **@Description**: String类的其它功能  \* **@date** 2017年11月11日 上午11:18:31  \* Company www.igeekhome.com  \*  \* 去除字符串两端空格  \* String trim()  \* 按照指定符号分割字符串  \* String[] split(String str)  \*/  **public** **class** StringDemo {  **public** **static** **void** main(String[] args) {  //创建字符串对象  String s1 = "helloworld";  String s2 = " helloworld ";  String s3 = " hello world ";  System.***out***.println("---"+s1+"---");  System.***out***.println("---"+s1.trim()+"---");  System.***out***.println("---"+s2+"---");  System.***out***.println("---"+s2.trim()+"---");  System.***out***.println("---"+s3+"---");  System.***out***.println("---"+s3.trim()+"---");  System.***out***.println("-------------------");  //String[] split(String str)  //创建字符串对象  String s4 = "aa,bb,cc";  String[] strArray = s4.split(",");  **for**(**int** x=0; x<strArray.length; x++) {  System.***out***.println(strArray[x]);  }  }  } |

#### String类的其它案例

##### 案例代码二十七

|  |
| --- |
| **package** com.igeek\_02\_06;  /\*\*  \* **@ClassName**: StringTest  \* **@Description**: 把数组中的数据按照指定个格式拼接成一个字符串  \* **@date** 2017年11月11日 上午11:20:04  \* Company www.igeekhome.com  \*  \* 举例：int[] arr = {1,2,3};  \* 输出结果：[1, 2, 3]  \*  \* 分析：  \* A:定义一个int类型的数组  \* B:写方法实现把数组中的元素按照指定的格式拼接成一个字符串  \* C:调用方法  \* D:输出结果  \*/  **public** **class** StringTest {  **public** **static** **void** main(String[] args) {  //定义一个int类型的数组  **int**[] arr = {1,2,3};    //写方法实现把数组中的元素按照指定的格式拼接成一个字符串    //调用方法  String s = *arrayToString*(arr);    //输出结果  System.***out***.println("s:"+s);  }    /\*  \* 两个明确：  \* 返回值类型：String  \* 参数列表：int[] arr  \*/  /\*\*  \* **@Title**: arrayToString  \* **@Description**: 数组转成字符串  \* **@param** arr  \* **@return**  \*/  **public** **static** String arrayToString(**int**[] arr) {  String s = "";    //[1, 2, 3]  s += "[";  **for**(**int** x=0; x<arr.length; x++) {  **if**(x==arr.length-1) {  s += arr[x];  }**else** {  s += arr[x];  s += ", ";  }  }  s += "]";  **return** s;  }  } |

##### 案例代码二十八

|  |
| --- |
| **package** com.igeek\_02\_06;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: StringTest2  \* **@Description**: 字符串反转  \* **@date** 2017年11月11日 上午11:23:19  \* Company www.igeekhome.com  \*  \* 举例：键盘录入”abc”  \* 输出结果：”cba”  \*  \* 分析：  \* A:键盘录入一个字符串  \* B:写方法实现字符串的反转  \* a:把字符串倒着遍历，得到的每一个字符拼接成字符串。  \* b:把字符串转换为字符数组，然后对字符数组进行反转，最后在把字符数组转换为字符串  \* C:调用方法  \* D:输出结果  \*/  **public** **class** StringTest2 {  **public** **static** **void** main(String[] args) {  //键盘录入一个字符串  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入一个字符串：");  String s = sc.nextLine();    //写方法实现字符串的反转    //调用方法  String result = *reverse*(s);    //输出结果  System.***out***.println("result:"+result);  }    /\*  \* 把字符串倒着遍历，得到的每一个字符拼接成字符串。  \*  \* 两个明确：  \* 返回值类型：String  \* 参数列表：String s  \*/  /\*  public static String reverse(String s) {  String ss = "";    for(int x=s.length()-1; x>=0; x--) {  ss += s.charAt(x);  }    return ss;  }  \*/    /\*\*  \* **@Title**: reverse  \* **@Description**: 把字符串转换为字符数组，然后对字符数组进行反转，最后在把字符数组转换为字符串  \* **@param** s  \* **@return**  \*/  **public** **static** String reverse(String s) {  //把字符串转换为字符数组  **char**[] chs = s.toCharArray();    //对字符数组进行反转  **for**(**int** start=0,end=chs.length-1; start<=end; start++,end--) {  **char** temp = chs[start];  chs[start] = chs[end];  chs[end] = temp;  }    //最后在把字符数组转换为字符串  String ss = **new** String(chs);  **return** ss;  }  } |

# StringBuilder类

## StringBuilder类概述

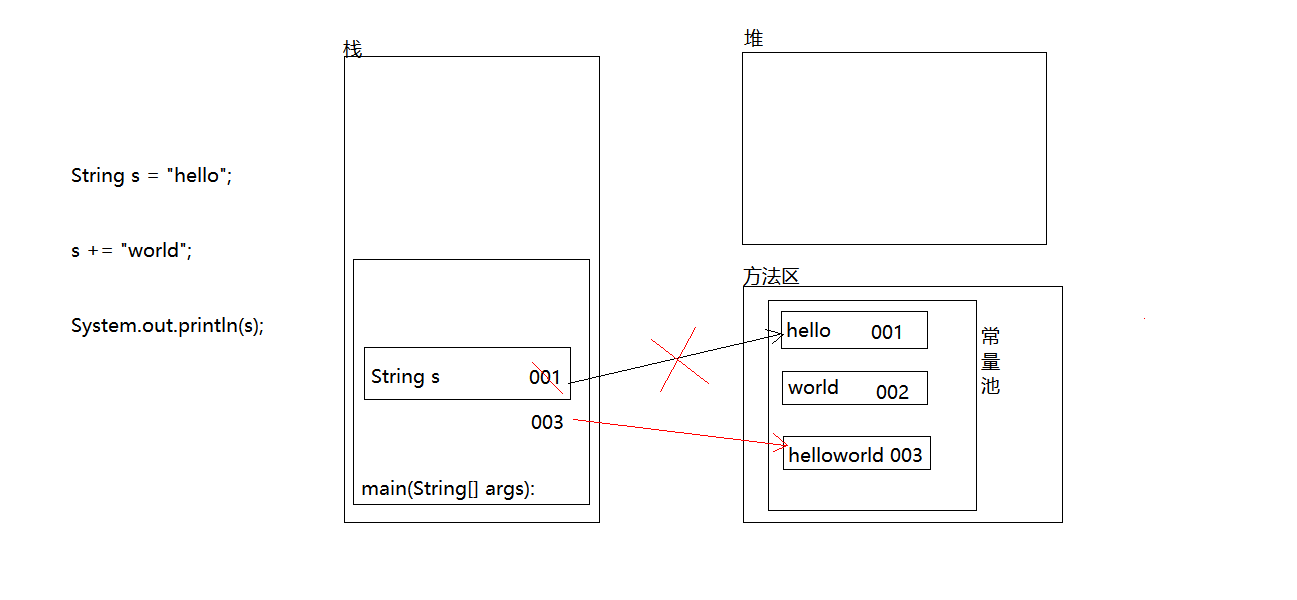
StringBuilder：是一个可变的字符串。字符串缓冲区类。

String和StringBuilder的区别：

String的内容是固定的

StringBuilder的内容是可变的

### +=拼接字符串耗费内存原因:

每次拼接都会产生新的字符串对象，而利用StringBuilder来拼接字符串自始至终用的都是同一个StringBuilder容器

## StringBuilder类的常用方法

A:构造方法:

StringBuilder()

B:成员方法:

public int capacity():返回当前容量 (理论值)

public int length():返回长度(已经存储的字符个数)

public StringBuilder append(任意类型):添加数据，并返回自身对象

public StringBuilder reverse():反转功能

### 案例代码二十九

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: StringBuilderDemo  \* **@Description**: StringBuilder:是一个可变的字符串。字符串缓冲区类。  \* **@date** 2017年11月11日 上午11:27:33  \* Company www.igeekhome.com  \*  \* String和StringBuilder的区别：  \* String的内容是固定的。  \* StringBuilder的内容是可变的。  \*  \* 构造方法：  \* StringBuilder()  \*  \* 成员方法：  \* public int capacity():返回当前容量  \* public int length():返回长度（字符数）  \*  \* 容量：理论值  \* 长度：实际值  \*/  **public** **class** StringBuilderDemo {  **public** **static** **void** main(String[] args) {  //创建对象  StringBuilder sb = **new** StringBuilder();  System.***out***.println("sb:"+sb);  System.***out***.println("sb.capacity():"+sb.capacity());  System.***out***.println("sb.length():"+sb.length());  }  } |

### 案例代码三十

|  |
| --- |
| **package** com.igeek\_02;  /\*\*  \* **@ClassName**: StringBuilderDemo2  \* **@Description**: StringBuilder其他功能  \* **@date** 2017年11月11日 上午11:28:55  \* Company www.igeekhome.com  \*  \* 添加功能  \* public StringBuilder append(任意类型):添加数据，并返回自身对象  \* 反转功能  \* public StringBuilder reverse()  \*/  **public** **class** StringBuilderDemo2 {  **public** **static** **void** main(String[] args) {  //创建对象  StringBuilder sb = **new** StringBuilder();    //public StringBuilder append(任意类型)  //StringBuilder sb2 = sb.append("hello");    /\*  System.out.println("sb:"+sb);  System.out.println("sb2:"+sb2);  System.out.println(sb == sb2); //true  \*/    /\*  sb.append("hello");  sb.append("world");  sb.append(true);  sb.append(100);  \*/    //链式编程  sb.append("hello").append("world").append(**true**).append(100);    System.***out***.println("sb:"+sb);    //public StringBuilder reverse()  sb.reverse();  System.***out***.println("sb:"+sb);    }  } |

## StringBuilder案例

### 案例一需求:

StringBuilder和String通过方法完成相互转换

#### 案例代码三十一

|  |
| --- |
| **package** com.igeek\_03;  /\*\*  \* **@ClassName**: StringBuilderTest  \* **@Description**: StringBuilder和String通过方法完成相互转换  \* **@date** 2017年11月11日 上午11:30:56  \* Company www.igeekhome.com  \*  \* StringBuilder -- String  \* public String toString():通过toString()就可以实现把StringBuilder转成String  \*  \* String -- StringBuilder  \* StringBuilder(String str):通过构造方法就可以实现把String转成StringBuilder  \*/  **public** **class** StringBuilderTest {  **public** **static** **void** main(String[] args) {  //StringBuilder -- String  /\*  StringBuilder sb = new StringBuilder();  sb.append("hello").append("world");    String s = sb.toString();  System.out.println(s);  \*/    //String -- StringBuilder  String s = "helloworld";  StringBuilder sb = **new** StringBuilder(s);  System.***out***.println(sb);  }  } |

### 案例二需求:

利用StringBuilder把数组拼接成一个字符串

举例：

int[] arr = {1,2,3};

结果：

[1, 2, 3]

#### 案例代码三十二

|  |
| --- |
| **package** com.igeek\_03;  /\*\*  \* **@ClassName**: StringBuilderTest2  \* **@Description**: 利用StringBuilder把数组拼接成一个字符串  \* **@date** 2017年11月11日 上午11:32:01  \* Company www.igeekhome.com  \*  \* 把数组拼接成一个字符串  \* 举例：  \* int[] arr = {1,2,3};  \* 结果：  \* [1, 2, 3]  \*/  **public** **class** StringBuilderTest2 {  **public** **static** **void** main(String[] args) {  //定义一个数组  **int**[] arr = {1,2,3};    //写方法实现拼接    //调用方法  String s = *arrayToString*(arr);    //输出结果  System.***out***.println("s:"+s);  }    /\*  \* 两个明确：  \* 返回值类型：String  \* 参数列表：int[] arr  \*/  /\*\*  \* **@Title**: arrayToString  \* **@Description**: 把数组拼接成字符串  \* **@param** arr  \* **@return**  \*/  **public** **static** String arrayToString(**int**[] arr) {  StringBuilder sb = **new** StringBuilder();  //[1, 2, 3]  sb.append("[");  **for**(**int** x=0; x<arr.length; x++) {  **if**(x==arr.length-1) {  sb.append(arr[x]);  }**else** {  sb.append(arr[x]).append(", ");  }  }  sb.append("]");    String result = sb.toString();    **return** result;  }  } |

### 案例三需求:

利用StringBuilder完成字符串反转

#### 案例代码三十三

|  |
| --- |
| **package** com.igeek\_03;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: StringBuilderTest3  \* **@Description**: 利用StringBuilder完成字符串反转  \* **@date** 2017年11月11日 上午11:34:03  \* Company www.igeekhome.com  \*  \* 分析：  \* A:键盘录入一个字符串  \* B:写方法实现反转  \* String -- StringBuilder -- reverse() -- String  \* C:调用方法  \* D:输出结果  \*/  **public** **class** StringBuilderTest3 {  **public** **static** **void** main(String[] args) {  //键盘录入一个字符串  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入一个字符串：");  String s = sc.nextLine();    //写方法实现反转    //调用方法  String result = *myReverse*(s);    //输出结果  System.***out***.println("result:"+result);  }    /\*  \* 两个明确：  \* 返回值类型：String  \* 参数列表：String  \*/  /\*\*  \* **@Title**: myReverse  \* **@Description**: 字符串反转  \* **@param** s  \* **@return**  \*/  **public** **static** String myReverse(String s) {  //String -- StringBuilder -- reverse() -- String  StringBuilder sb = **new** StringBuilder(s);  sb.reverse();  String result = sb.toString();  **return** result;  }  } |

### 案例四需求:

判断一个字符串是否是对称字符串

例如"abc"不是对称字符串，"aba"、"abba"、"aaa"、"mnanm"是对称字符串

#### 案例代码三十四

|  |
| --- |
| **package** com.igeek\_03;  **import** java.util.Scanner;  /\*\*  \* **@ClassName**: StringBuilderTest4  \* **@Description**: 判断一个字符串是否是对称字符串  \* **@date** 2017年11月11日 上午11:35:44  \* Company www.igeekhome.com  \*  \* 例如"abc"不是对称字符串，"aba"、"abba"、"aaa"、"mnanm"是对称字符串  \*  \* 分析：  \* A:键盘录入一个字符串  \* B:写方法实现判断一个字符串是否是对称字符串  \* 把字符串反转，和反转前的字符串进行比较，如果内容相同，就说明是对称字符串  \* C:调用方法  \* D:输出结果  \*/  **public** **class** StringBuilderTest4 {  **public** **static** **void** main(String[] args) {  //键盘录入一个字符串  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("请输入一个字符串：");  String s = sc.nextLine();    //写方法实现判断一个字符串是否是对称字符串    //调用方法  **boolean** b = *isSymmetry*(s);    //输出结果  System.***out***.println("b:"+b);  }    /\*  \* 两个明确：  \* 返回值类型：boolean  \* 参数列表：String s  \*/  /\*\*  \* **@Title**: isSymmetry  \* **@Description**: 判断一个字符串是否是对称字符串  \* **@param** s  \* **@return**  \*/  **public** **static** **boolean** isSymmetry(String s) {  //把字符串反转，和反转前的字符串进行比较，如果内容相同，就说明是对称字符串  StringBuilder sb = **new** StringBuilder(s);  sb.reverse();  String result = sb.toString();    **return** result.equals(s);  }  } |

重点和总结

1、面向对象的基本知识

2、类和对象的关系

3、什么是封装

4、API的使用方法

5、Scanner类与String类的区别及使用

6、StringBuilder类的使用