面向对象进阶

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

目录

Contents

接口的定义与特点

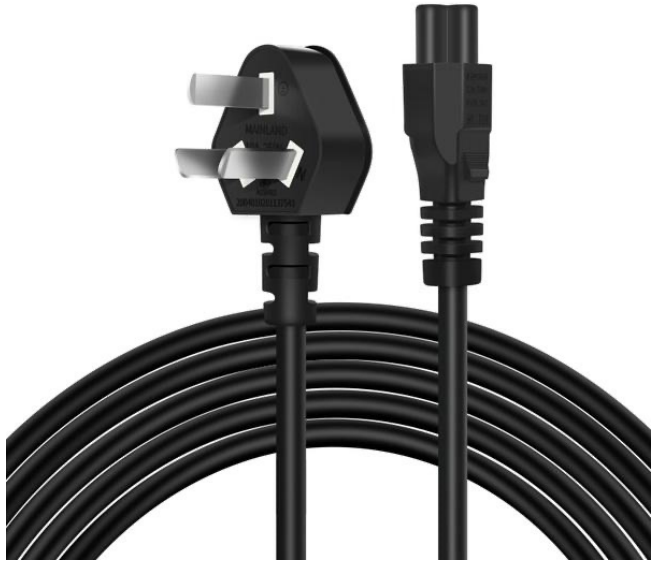
- 接口的格式如下：

```
接口用关键字interface来定义
public interface 接口名 {
    // 常量
    // 抽象方法
}
```

- JDK8之前接口中只能是抽象方法和常量，没有其他成分了。
- 接口不能实例化。
- 接口中的成员都是public修饰的，写不写都是，因为规范的目的是为了公开化。

什么是接口

- 接口也是一种规范。



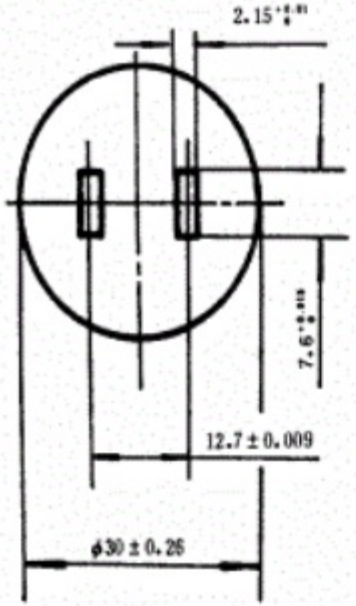
中华人民共和国国家标准

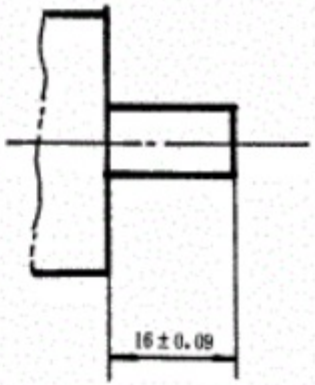
家用和类似用途单相插头插座 型式、基本参数和尺寸

GB 1002—1996

代替 GB 1002—80

Single phase plugs and socket-outlets for household and similar purposes Types, basic parameters and dimensions





1 范围

本标准规定了家用和类似用途单相插头插座的型式、基本参数和尺寸。

本标准适用于一般家庭和类似家庭环境的场合使用的、交流频率为 50 Hz、额定电压为 250 V、额定电流不超过 16 A 的单相插头和固定或可移动插座。

2 引用标准

下列标准所包含的条文，通过在本标准中引用而构成为本标准的条文。本标准出版时，所示版本均为有效。所有标准都会被修订，使用本标准的各方应探讨使用下列标准最新版本的可能性。

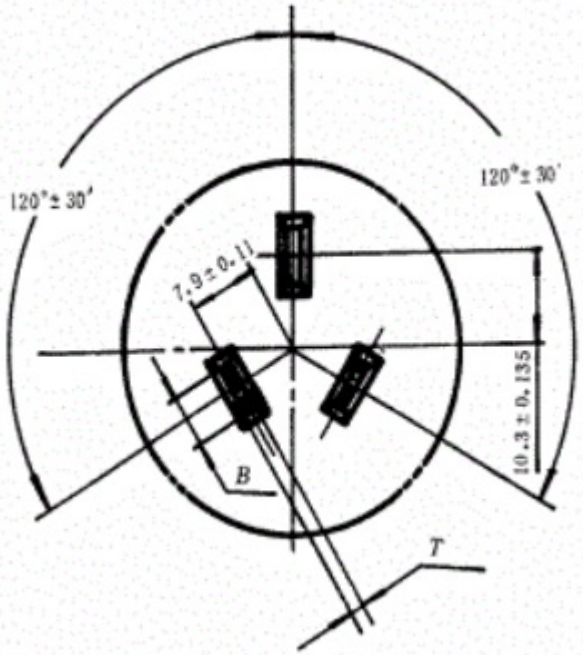
GB 10964—89 电器附件、控制器和保护器 术语

GB 2099.1—1996 家用和类似用途插头插座 第一部分：通用要求

GB 1184—80 形状和位置公差 未注公差的规定

3 术语

GB 10964 和 GB 2099.1 中的术语适用于本标准。



目录

Contents

接口的用法：

- 接口是用来被类实现（implements）的，实现接口的类称为实现类。实现类可以理解成所谓的子类。

```
修饰符 class 实现类 implements 接口1, 接口2, 接口3 , ... {
}
实现的关键字：implements
```

- 从上面可以看出，接口可以被类单实现，也可以被类多实现。

接口实现的注意事项：

- 一个类实现接口，必须重写完全部接口的全部抽象方法，否则这个类需要定义成抽象类。

目录

Contents

目录

Contents

基本小结

- 类和类的关系：单继承。
- 类和接口的关系：多实现。
- 接口和接口的关系：多继承，一个接口可以同时继承多个接口。

接口多继承的作用

- 规范合并，整合多个接口为同一个接口，便于子类实现。

目录

Contents

第一种：默认方法

- 类似之前写的普通实例方法：必须用default修饰
- 默认会public修饰。需要用接口的实现类的对象来调用

```
default void run(){
    System.out.println("--开始跑--");
}
```

第二种：静态方法

- 默认会public修饰，必须static修饰。
- **注意：接口的静态方法必须用本身的接口名来调用。**

```
static void inAddr(){
    System.out.println("我们都在黑马培训中心快乐的学习Java!");
}
```

第三种：私有方法

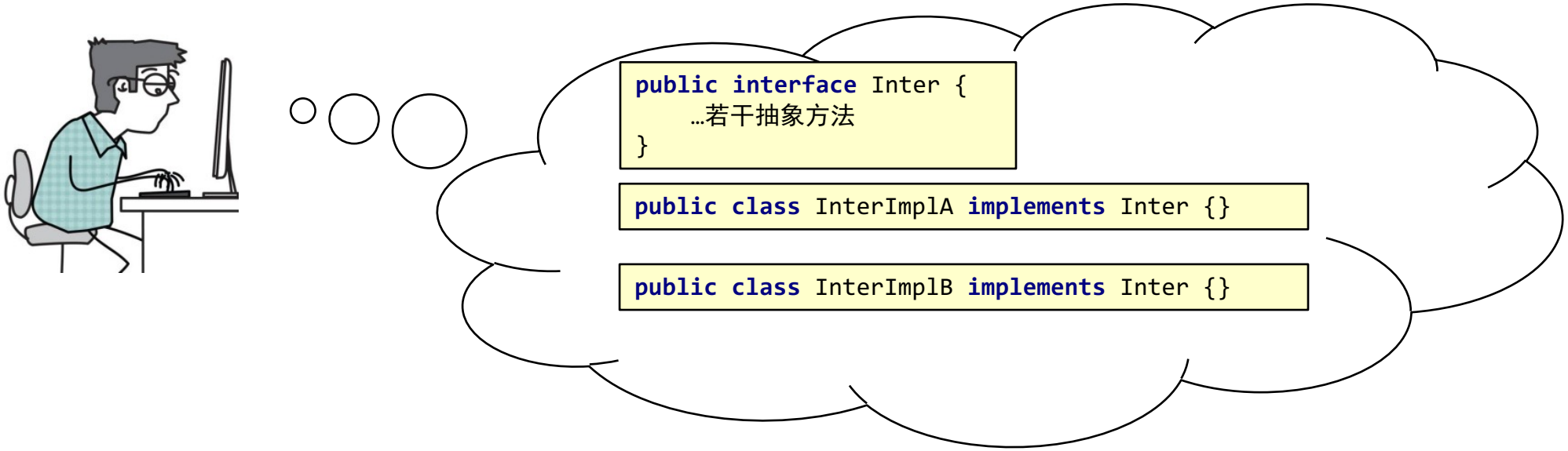
- 就是私有的实例方法:，必须使用private修饰，从**JDK 1.9才开始有的**。
- 只能在本类中被其他的默认方法或者私有方法访问。

```
private void go(){
    System.out.println("--准备--");
}
```

JDK8版本开始后，Java只对接口的成员方法进行了新增

原因如下

项目Version1.0 成功上线没有问题



项目Version2.0需要对Inter接口丰富，加入10个新的抽象方法，此时改了接口就要所有实现类实现这些方法。

项目Version2.0 需要扩展功能

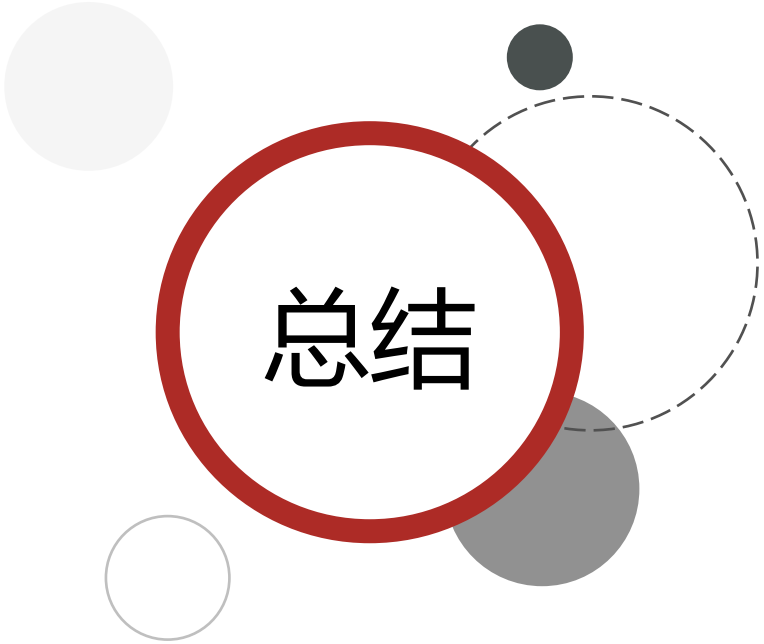
```
public interface Inter {
    …若干抽象方法
}
```

```
public class InterImplA implements Inter {}
```

```
public class InterImplB implements Inter {}
```

如何能在丰富接口功能的同时
又不对子类代码进行更改呢？

允许接口中直接定义带有方法体的方法



1、JDK8开始后新增了那些方法?

- **默认方法：default修饰，实现类对象调用。**
- **静态方法：static修饰，必须用当前接口名调用**
- **私有方法：private修饰，jdk9开始才有的，只能在接口内部被调用。**
- **他们都会默认被public修饰。**

注意：JDK8新增的3种方法我们自己在开发中很少使用，通常是Java源码涉及到的，我们需要理解、识别语法、明白调用关系即可。

目录

Contents

接口的注意事项

1、接口不能创建对象

2、一个类实现多个接口，多个接口的规范不能冲突

2、一个类实现多个接口，多个接口中有同样的静态方法不冲突。

3、一个类继承了父类，同时又实现了接口，父类中和接口中有同名方法，默认用父类的。

4、一个类实现了多个接口，多个接口中存在同名的默认方法，可以不冲突，这个类重写该方法即可。

5、一个接口继承多个接口，是没有问题的，如果多个接口中存在规范冲突则不能多继承。

目录

Contents

内部类

- 内部类就是定义在一个类里面的类，里面的类可以理解成（寄生），外部类可以理解成（宿主）。

```
public class People{
    // 内部类
    public class Heart{
    }
}
```

内部类的使用场景

- 场景：当一个事物的内部，还有一个部分需要一个完整的结构进行描述时。

基本作用

- 内部类通常可以方便访问外部类的成员，包括私有的成员。
- 内部类提供了更好的封装性，内部类本身就可以用private ，protectecd等修饰，封装性可以做更多控制。

内部类的分类

- 静态内部类[了解]
- 成员内部类（非静态内部类）[了解]
- 局部内部类[了解]
- 匿名内部类（重点）

目录

Contents

什么是静态内部类？

- 有static修饰，属于外部类本身。
- 它的特点和使用与普通类是完全一样的，类有的成分它都有，只是位置在别人里面而已。

```
public class Outer{
        // 静态成员内部类
        public static class Inner{
        }
}
```

静态内部类创建对象的格式：

格式：外部类名.内部类名 对象名 = new 外部类名.内部类构造器

;

范例：Outer.Inner in = new Outer.Inner();

静态内部类的访问拓展：

1、静态内部类中是否可以直接访问外部类的静态成员？

- **可以，外部类的静态成员只有一份可以被共享访问。**

2、静态内部类中是否可以直接访问外部类的实例成员？

- **不可以的，外部类的实例成员必须用外部类对象访问。**

总结

1. 静态内部类的使用场景、特点、访问总结。

- **如果一个类中包含了一个完整的成分，如汽车类中的发动机类。**
- **特点、使用与普通类是一样的，类有的成分它都有，只是位置在别人里面而已。**
- **访问总结：可以直接访问外部类的静态成员，不能直接访问外部类的实例成员。**
- **注意：开发中实际上用的还是比较少。**

目录

Contents

- 接口
- 内部类
  - 内部类概述
  - 内部类之一：静态内部类[了解]
  - 内部类之二：成员内部类[了解]
  - 内部类之三：局部内部类[了解]
  - 内部类之四：匿名内部类概述[重点]
  - 匿名内部类常见使用形式
  - 匿名内部类真实使用场景演示
- 常用API

什么是成员内部类？

- 无static修饰，属于外部类的对象。
- JDK16之前，成员内部类中不能定义静态成员，JDK 16开始也可以定义静态成员了。

```
public class Outer {
    // 成员内部类
    public class Inner {

    }
}
```

成员内部类创建对象的格式：

格式：外部类名.内部类名 对象名 = new 外部类构造器.new 内部类构造器();

范例：Outer.Inner in = new Outer().new Inner();

成员内部类的访问拓展：

1、成员内部类中是否可以直接访问外部类的静态成员？

- **可以，外部类的静态成员只有一份可以被共享访问。**

2、成员内部类的实例方法中是否可以直接访问外部类的实例成员？

- **可以的，因为必须先有外部类对象，才能有成员内部类对象，所以可以直接访问外部类对象的实例成员**

。

总结

1. 成员内部类是什么样的、有什么特点？
   - **无static修饰，属于外部类的对象。**
   - **可以直接访问外部类的静态成员，实例方法中可以直接访问外部类的实例成员。**
2. 成员内部类如何创建对象？
   - **外部类名.内部类名 对象名 = new 外部类构造器.new 内部类构造器();**

案例 成员内部类-面试笔试题

- 请观察如下代码，写出合适的代码对应其注释要求输出的结果。

```
class People{
    private int heartbeat = 150;
    public class Heart{
        private int heartbeat = 110;
        public void show(){
            int heartbeat = 78;
            System.out.println(??); // 78
            System.out.println(??); // 110
            System.out.println(??); // 150
        }
    }
}
```

注意：在成员内部类中访问所在外部类对象 ，格式：外部类名.this。

目录

Contents

局部内部类（鸡肋语法，了解即可）

- 局部内部类放在方法、代码块、构造器等执行体中。
- 局部内部类的类文件名为： 外部类$N内部类.class。

目录

Contents

- 接口
- 内部类
  - 内部类概述
  - 内部类之一：静态内部类[了解]
  - 内部类之二：成员内部类[了解]
  - 内部类之三：局部内部类[了解]
  - 内部类之四：匿名内部类概述[重点]
  - 匿名内部类常见使用形式
  - 匿名内部类真实使用场景演示[拓展]
- 常用API

匿名内部类：

- 本质上是一个没有名字的局部内部类。
- **作用**：方便创建子类对象，最终目的是为了简化代码编写。

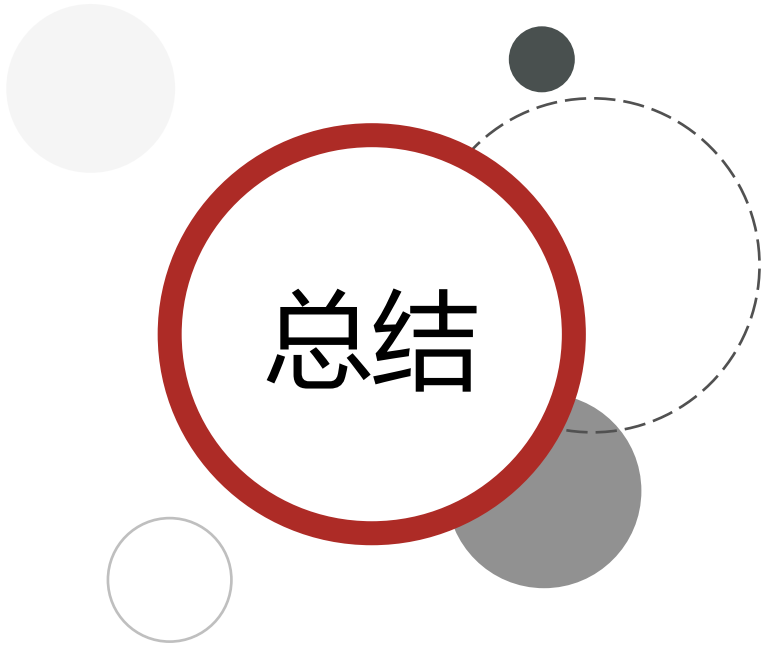
格式：

```
new 类|抽象类名|或者接口名() {
    重写方法;
};
```

```
Employee a = new Employee() {
    public void work() {
    }
};
a. work();
```

特点总结：

- 匿名内部类是一个没有名字的内部类，同时也代表一个对象。
- 匿名内部类产生的对象类型，相当于是当前new的那个的类型的子类类型。



1. 匿名内部类的作用？

- **方便创建子类对象，最终目的为了简化代码编写。**

2. 匿名内部类的格式？

```
Animal a = new Employee() {
    public void run() {
    }
};
a. run();
```

3. 匿名内部类的特点？

- **匿名内部类是一个没有名字的内部类，同时也代表一个对象。**
- **匿名内部类的对象类型，相当于是当前new的那个类型的子类类型。**

目录

Contents

- 接口
- 内部类
  - 内部类概述
  - 内部类之一：静态内部类[了解]
  - 内部类之二：成员内部类[了解]
  - 内部类之三：局部内部类[了解]
  - 内部类之四：匿名内部类概述[重点]
  - 匿名内部类常见使用形式
  - 匿名内部类真实使用场景演示
- 常用API

匿名内部类在开发中的使用形式了解

- 某个学校需要让老师，学生，运动员一起参加游泳比赛

```
/*游泳接口*/
public interface Swimming {
    void swim();
}
```

```
/* 测试类*/
public class JumppingDemo {
    public static void main(String[] args) {
        //需求：goSwimming方法
    }

    // 定义一个方法让所有角色进来一起比赛
    public static void goSwimming(Swimming swimming) {
        swimming.swim();
    }

}
```

使用总结

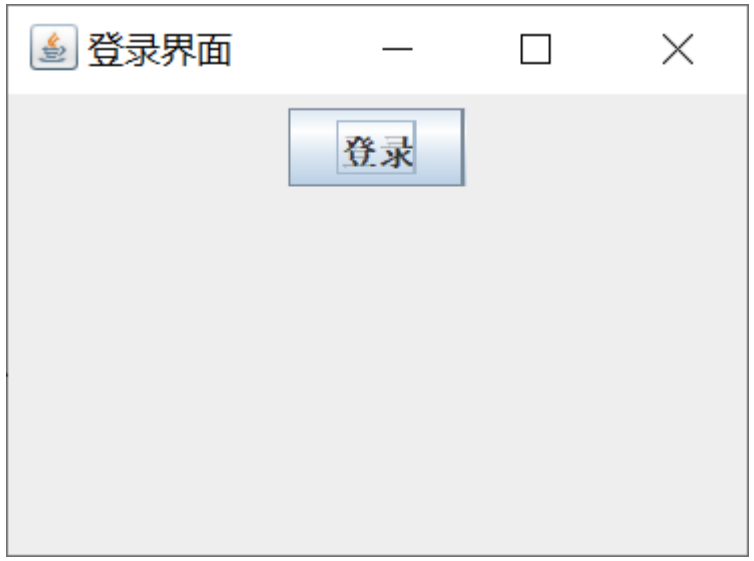
匿名内部类可以作为一个对象，直接传输给方法。

目录

Contents

匿名内部类在开发中的真实使用场景演示

- 给按钮绑定点击事件




```
// 为按钮绑定点击事件监听器。
btn.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("登录一下~~");
    }
});
// 简化
// btn.addActionListener(e -> System.out.println("登录一下~~"));
```

使用总结

匿名内部类通常是在开发中调用别人的方法时，别人需要我们写的时候才会定义出来使用。

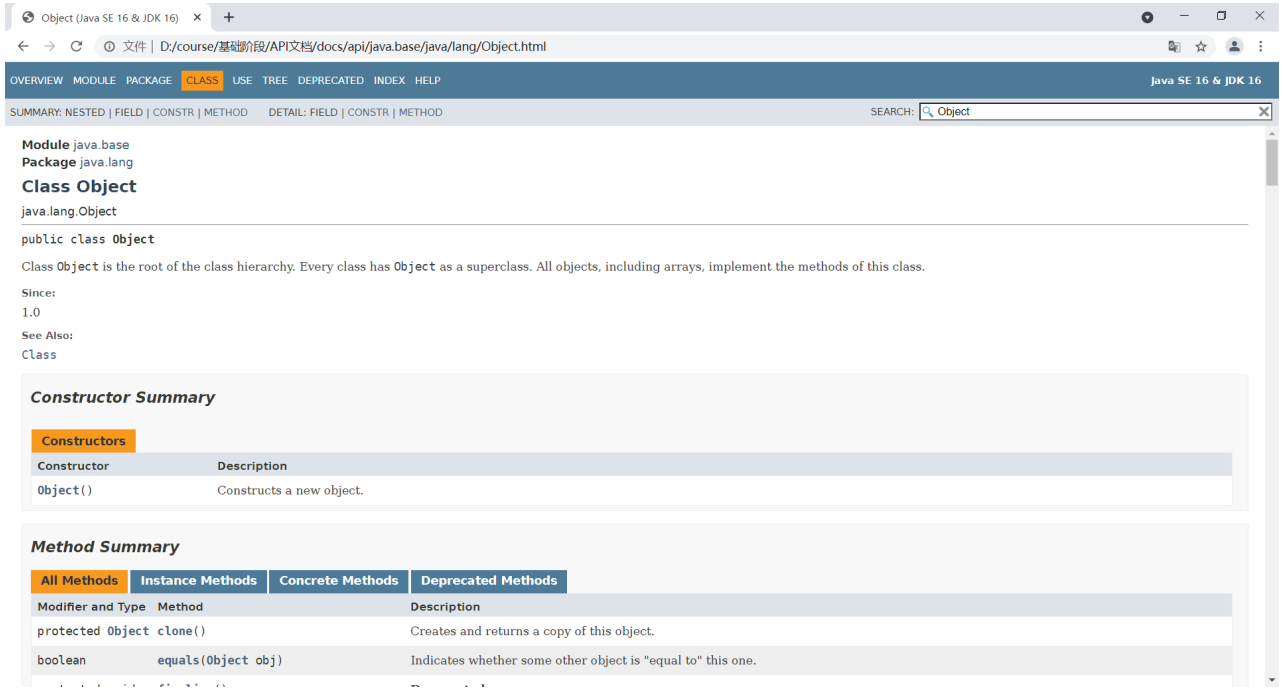
将来：匿名内部类还可以实现进一步的简化代码（后面其他技术会讲）

目录

Contents

什么是API?

- API(Application Programming interface) 应用程序编程接口。
- **简单来说：就是Java帮我们已经写好的一些方法，我们直接拿过来用就可以了。**



目录

Contents

Object类的作用：

- 一个类要么默认继承了Object类，要么间接继承了Object类，Object类是Java中的祖宗类。
- Object作为所有类的父类，提供了很多常用的方法给每个子类对象拿来使用。

Object类的常用方法：

方法名	说明
public String toString()	默认是返回当前对象在堆内存中的地址信息:类的全限名@内存地址
public boolean equals(Object o)	默认是比较当前对象与另一个对象的地址是否相同，相同返回true，不同返回false

Object的toString方法：

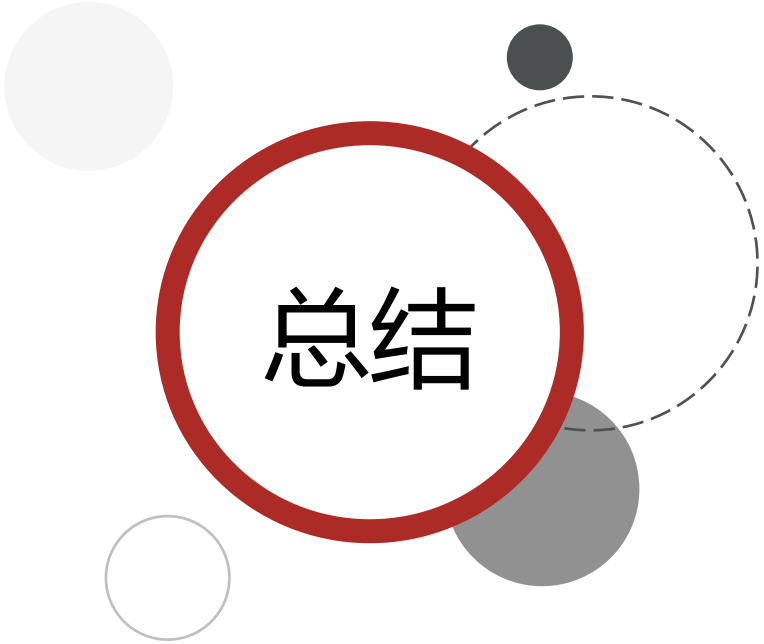
方法名	说明
public String toString()	默认是返回当前对象在堆内存中的地址信息:类的全限名@内存地址

问题引出

- 开发中直接输出对象，默认输出对象的地址其实是毫无意义的。
- 开发中输出对象变量，更多的时候是希望看到对象的内容数据而不是对象的地址信息。

toString存在的意义

- 父类toString()方法存在的意义就是为了被子类重写，以便返回对象的内容信息，而不是地址信息！！



1. Object的toString方法的基本作用是什么，存在的意义是什么？

- **基本作用：给子类继承，子类对象调用可以返回自己的地址**

  。

- **意义：让子类重写，以便返回子类对象的内容。**

目录

Contents

Object的equals方法：

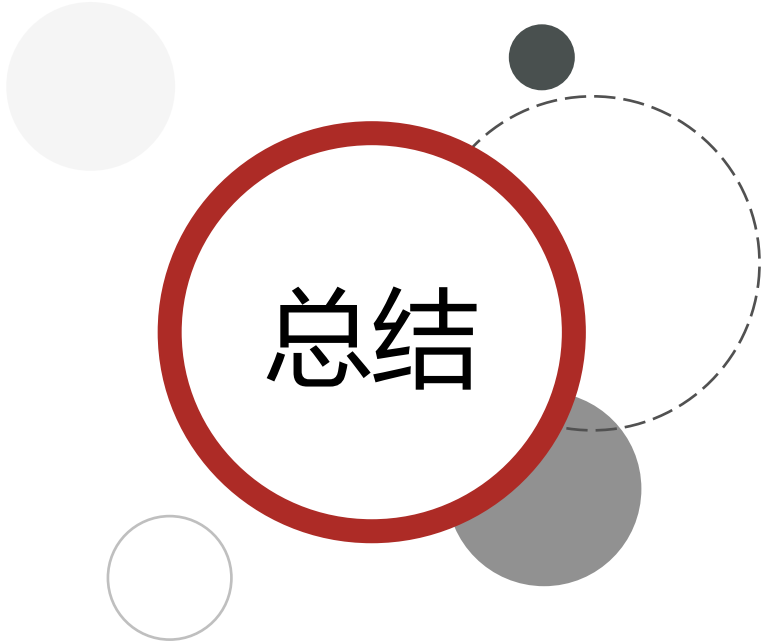
方法名	说明
public boolean equals(Object o)	默认是比较当前对象与另一个对象的地址是否相同，相同返回true，不同返回false

问题思考

- 直接比较两个对象的地址是否相同完全可以用“==”替代equals。
- 同时，开发中很多业务情况下，更想判断2个对象的内容是否一样。

equals存在的意义

- 为了被子类重写，以便子类自己来定制比较规则（比如比较对象内容）。



1. Object的equals方法的基本作用，存在的意义是什么？

- **基本作用：默认是与另一个对象比较地址是否一样**
- **存在的意义：让子类重写，以便比较对象的内容是否相同。**

目录

Contents

Objects概述

- Objects是一个工具类，提供了一些方法去完成一些功能。

官方在进行字符串比较时，没有用字符串对象的的equals方法，而是选择了Objects的equals方法来比较。

```
@Override
public boolean equals(Object o) {
    // 1、判断是否是同一个对象比较，如果是返回true。
    if (this == o) return true;
    // 2、如果o是null返回false  如果o不是学生类型返回false  ...Student !=  ..Pig
    if (o == null || this.getClass() != o.getClass()) return false;
    // 3、说明o一定是学生类型而且不为null
    Student student = (Student) o;
    return sex == student.sex && age == student.age && Objects.equals(name, student.name);
}
```

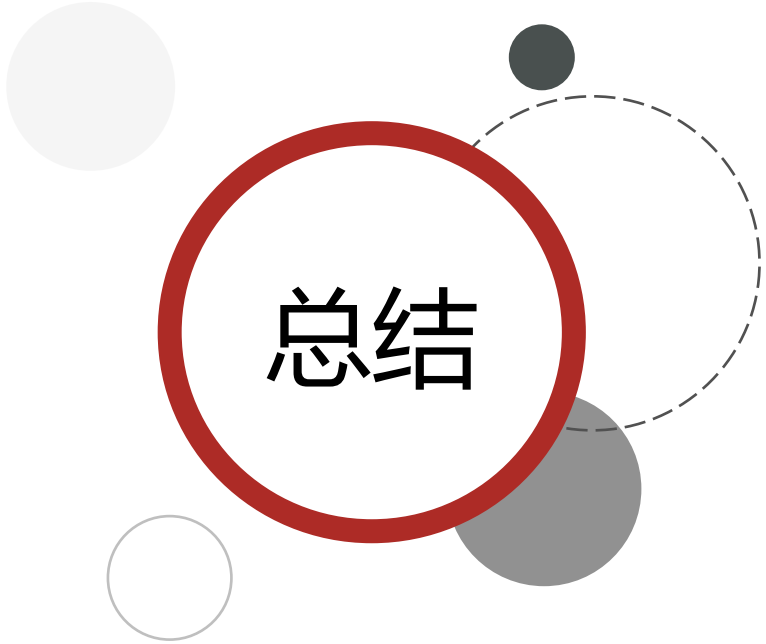
使用Objects的equals方法在进行对象的比较会更安全。

Objects的常见方法：

方法名	说明
`public static boolean equals(Object a, Object b)`	比较两个对象的，底层会先进行非空判断，从而可以避免空指针异常。再进行equals比较
`public static boolean isNull(Object obj)`	判断变量是否为null ,为null返回true ,反之

源码分析

```
public static boolean equals(Object a, Object b) {
    return (a == b) || (a != null && a.equals(b));
}
```



1. 对象进行内容比较的时候建议使用什么？为什么？

- **建议使用Objects提供的equals方法。**
- **比较的结果是一样的，但是更安全。**

目录

Contents

StringBuilder概述

- StringBuilder是一个可变的字符串的操作类，我们可以把它看成是一个对象容器。
- 使用StringBuilder的核心作用：操作字符串的性能比String要更高（如拼接、修改等）。

String类拼接字符串原理图

```
public class Test {
    public static void main(String[] args) {
        String s1 = "a";
        String s2 = s1 + "b";
        String s3 = s2 + "c";
        System.out.println(s3);
    }
}
```

栈内存

```
方法：main

String s1 = "a";
String s2 = s1 + "b";
String s3 = s2 + "c";

System.out.println(s3);
```

一个加号，堆内存中俩对象

堆内存

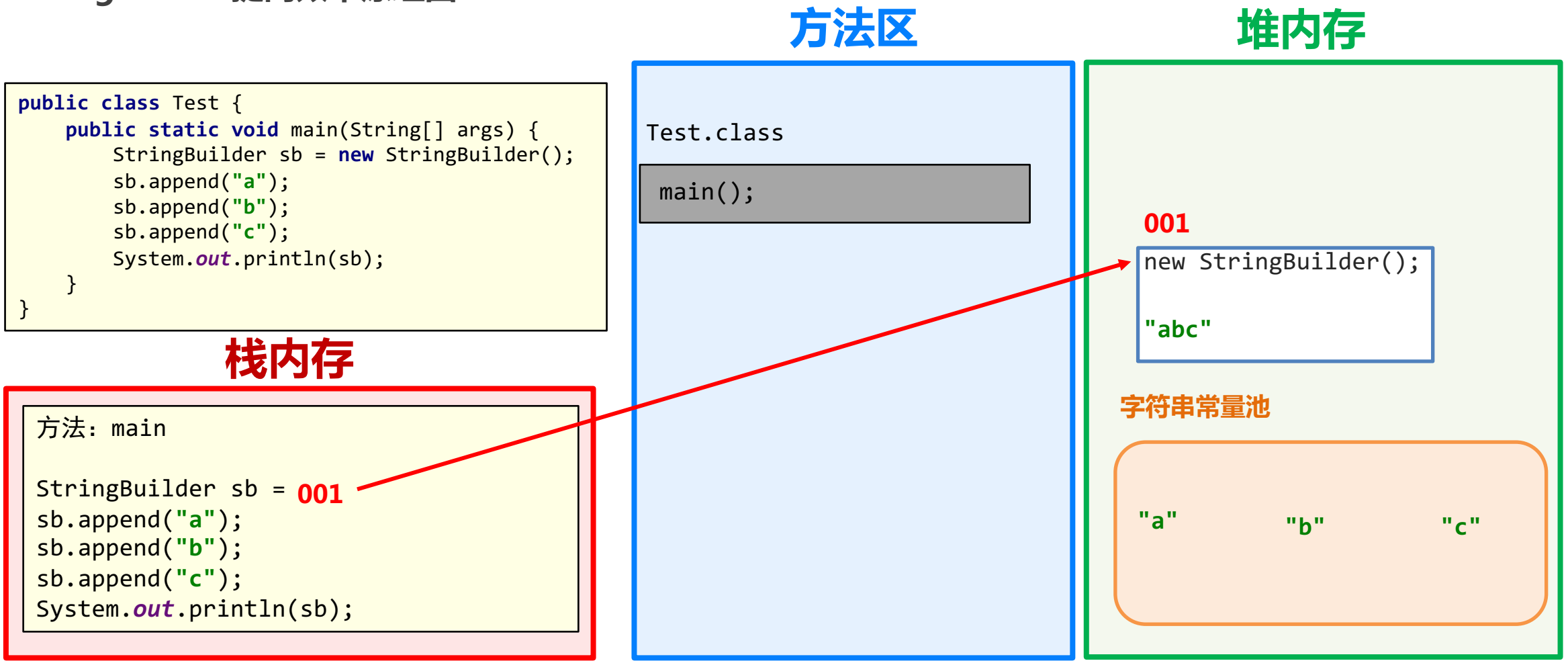
001 `new StringBuilder();` `"ab"` toString → 002 String `"ab"`

003 `new StringBuilder();` `"abc"` toString → 004 String `"abc"`

字符串常量池 `"a"` `"b"` `"c"`

StringBuilder提高效率原理图

```
public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        sb.append("a");
        sb.append("b");
        sb.append("c");
        System.out.println(sb);
    }
}
```



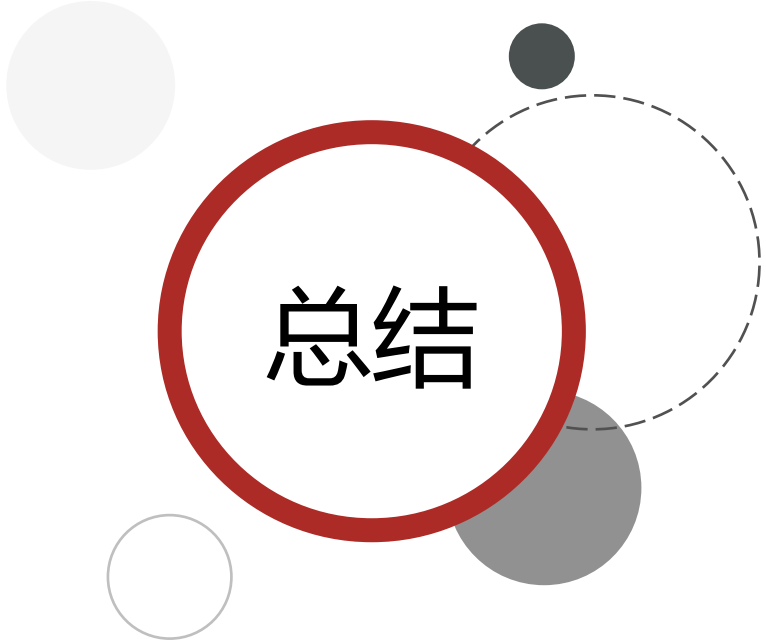
结论：当需要进行字符串操作的时候，应该选择StringBuilder来完成，性能更好

StringBuilder 构造器

名称	说明
public StringBuilder()	创建一个空白的可变的字符串对象，不包含任何内容
public StringBuilder(String str)	创建一个指定字符串内容的可变字符串对象

StringBuilder常用方法

方法名称	说明
public StringBuilder append(任意类型)	添加数据并返回StringBuilder对象本身
public StringBuilder reverse()	将对象的内容反转
public int length()	返回对象内容长度
public String toString()	通过toString()就可以实现把StringBuilder转换为String



1、为什么拼接、反转字符串建议使用StringBuilder?

- **StringBuilder：内容是可变的、拼接字符串性能好、代码优雅。**
- **String ：内容是不可变的、拼接字符串性能差。**

- **定义字符串使用String**
- **拼接、修改等操作字符串使用StringBuilder**

案例

打印整型数组内容

需求：

- 设计一个方法用于输出任意整型数组的内容，要求输出成如下格式：

  “该数组内容为：[11，22，33，44，55]”

分析：

1、定义一个方法，要求该方法能够接收数组，并输出数组内容。 ---> 需要参数吗？需要返回值类型申明吗？

2、定义一个静态初始化的数组，调用该方法，并传入该数组。

传智教育旗下高端IT教育品牌