

java_8_9_10

内容：

封装和继承

Java 封装

java 三大特性：封装、继承、多态

简介

封装是**高内聚低耦合**（如电视机（或是电脑）主机包括线子全部装一起，我们只会看到电视机（电脑）壳子，使用时，会用插电开机即可）[把编写复杂的细节包装起来，便于安全和可控让自己内部进行处理，外部调用者只需要关注如何方便调用即可]。

封装（Encapsulation）是面向对象方法的重要原则，就是把对象的属性和操作（或服务）结合为一个独立的整体，并尽可能隐藏对象的内部实现细节。

封装是把**过程和数据包围**起来，对数据的访问**只能通过**已定义的方法或接口[重要]。面向对象计算始于这个基本概念，即现实世界可以被描绘成一系列完全自治、封装的对象，这些对象通过一个**受保护**的方法或接口访问其他对象。封装是一种信息**隐藏**技术，在java中通过关键字 **private** (指属性[成员变量]私有)和 **public** (get, set公共，这两者结合使用) 实现类的封装。封装就是把对象的所有组成部分组合在一起，封装使用方法将类的数据隐藏起来，设置对外的公共方法[set,get]来引用对象的数据，封装实际上就是控制用户对类的**设置**[set]和**访问**[get]数据的操作控制程度。适当的封装可以让程式代码更容易理解和维护，加强程式代码的**安全性**。

封装的优点

- 1). 提高代码安全性(如年龄随意输出负或是过大，不让外部可以任意修改数据)
- 2). 提高代码复用性(重构也是复用性)
- 3). 高内聚:封装代码细节，便于修改内部代码，提高代码可维护性
- 4). 低耦合:简化外部调用，便于调用者方便使用，利于扩展及协作

没有封装前代码：

```
/**
 * 类属性变量不封装时（随意设置写入年龄）
 * @author djy
 * @version
 */
public class Person {
    String name;
    int age;
```

```

    public static void main(String[] args) {
        Person person = new Person();
        person.name = "张三";
        person.age = -30; //或是为负或为超过200岁
        System.out.println(person.name + ", 年龄: " + person.age);
    }
}

```

封装使用访问控制符

java使用“访问控制符”对那些细节代码使用封装，而那些细节代码却是需要暴露（外部调用）。java 有4种“访问控制符”，四个关键字为：private（私有），default（默认），protected（受保护），public（公共）。说明了面向对象的封装性，利用这个控制符尽可能的让访问权限降至最低，从而提高代码的安全性。

访问控制符	同一个类	同一个包	子类	不同包	所有类
private	*				
default	*	*	*		
protected	*	*	*		
public	*	*	*	*	*

企业开发中：属性 private，set, get公共的，普通方法公共的（把流程方法统一写在接口类中）

- 1). private: 私有，只有自己类能访问，出去本类不可以（修饰属性字段，不使用方法[写了方法给别人使用]）
- 2). default: 没有修饰符修饰（什么都不写默认为 default），只有同一个包的类能访问
- 3). protected: 可以被同一个包的类及其它包的子类访问（同包下）
- 4). public: 该项目中所有包的所有类都能访问到（最宽泛，范围最大）

注：从上至下范围越来越大，从下至上范围越来越小

注：default 与 protected 区别，protected 不能修饰在为类，default 可以修饰在类上

```

/**
 * 封装测试
 * @author Administrator
 * @date 2019年7月7日
 * @version
 */
public class EncapsulationTest {

    public static void main(String[] args) {
        Boy boy = new Boy();
        //boy.name = "张三";
        //不加修饰时可用，一旦加了private私有修饰符，就不可以赋值，因此需要有类中提供给调用者使用的
        public方法
        boy.setName();

        boy.age = 20; //不加修饰符， 同一个包可用，出了不可以
    }
}

```

```

    }

}

class Boy{
    /*String name;*/
    int age;
    private String name;//封装name属性,自己可见,调用者没法看到,出了本类则无法看到

    void setName(){
        name = "张三";
        System.out.println(name);
    }
}

class Son extends Boy{
    void showName(){
        //依然无法使用boy.name
        //System.out.println(name);
        //那怕继承父类,子因也无法拥有它的私有属性//如同旧的粮票以前可以使用,现在那怕继承拿到也无法使用

        Boy boy = new Boy();
        boy.age = 20;
        System.out.println(boy.age);
    }
}

```

不同包下封装测试：

```

package com.fy.test;
import com.fy.model.Boy;
/**
 * 测试调用封装类
 * @author Administrator
 * @date 2019年7月7日
 * @version
 */
public class Test {

    public static void main(String[] args) {
        Boy boy = new Boy();
        //则是无法调用,EncapsulationTest可以是因为是public,因此如果类是默认时,只能是本包本类,要使用则修改成Public修饰符
        //为方便使用单独创建一个Boy的类为public(不同包可以访问)
        boy.setName();//如果需要访问调用的都要修改成public,因为出了类出了包范围
        //boy.address//受保护不同包之下无法访问
    }
}

```

```

package com.fy.model;
public class Boy {
    int age; //默认本包本类
    private String name; //封装name属性,自己可见,调用者没法看到,出了本类则无法看到

    protected String address; //protected受保护,作用范围:本包本类子类

    public void setName() { //public修饰符是公共的,事个项目之下都可以访问
        name = "张三";
        System.out.println(name);
    }
}

```

protected 测试（本包本类子类）：

```

public class EncapsulationTest {

    public static void main(String[] args) {
        Boy boy = new Boy();
        //boy.name = "张三";
        //不加修饰时可用,一旦加了private私有修饰符,就不可以赋值,因此需要有类中提供给调用者使用的
        public方法
        boy.setName();

        boy.age = 20; //不加修饰符,同一个包可用,出了包呢? (不可以)
        boy.address = "苏州"; //protected本包本类都可访问
    }
}

```

注：开发当中，几乎全是使用 private（属性字段全设置成私有），set 写, get 读设置成 public 公共。

封装的步骤

(1).修改属性(属性变量名称,如: name, age...)的可见性来限制对属性的访问（使用 private 定义），如：

```

/**
 * 封装: Person
 * @author Administrator
 * @date 2019年1月13日
 */
public class Person {
    private String name;
    private int age;
    private String address; //booble类型属性
    //booble类型属性
}

```

将 **name** 和 **age** 属性设置为私有的，只有 Person 类才能访问，其他类都访问不了，这样就对信息进行了隐藏保护，安全性能提高了。

(2).对每个值属性提供对外的公共方法访问，创建一对**赋取**(写读)值方法，用于对私有属性的访问，如：

```
package com.fengyun.model;

/**
 * 封装类：赋取值
 * @author djy
 * @date 2019年1月17日
 */
public class Person {

    private String name;
    private int age;

    //取name值
    public String getName() {
        return name;
    }
    //赋name值
    public void setName(String name) {
        this.name = name; //name是最近的局部变量，this.name是类Person下的name属性
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

    public Person() {
    }

}
```

采用 **this** 关键字是为了解决实例变量 (private String name) 和局部变量 (setName(String name)中的name 变量) 之间发生的同名的冲突。

封装设置年龄(可控)[**高内聚**设置]：

```
public void setAge(int age) { //高内聚
    if (age >= 1 && age <= 120) {
        this.age = age;
    } else {
        System.out.println("输入年龄不合法，请重新输入");
    }
}
```

测试 [**低耦合**调用]：

```

//程序方入口
public static void main(String[] args) {
    Person person = new Person();
    //低耦合：调用
    person.setName("张三");
    person.setAge(-20); //输出“输入年龄不合法，请重新输入”
    //setAge使用修饰符做对外访问控制，起到保护作用，调用者并不需要知道具体如何编写细节

    //get获取年龄值
    System.out.println(person.getAge()); //前面因为不合法，因此输出是初始化的值0
}

```

提快捷方式：set, get快捷键（右击 source 查找对应设置）及构造方法快捷，除了 boolean 生成不是 get, set 而是如 isSign(), 开发当中使用快捷方式生成方便快捷开发，提高效率。

提供属性私有，set, get 公有，这样的类称之为 **JavaBean** (封装类)(简单普通类：没有复杂逻辑功能的类)。

其它的方法功能（除了属性之外，所有的 set, get 全设置成 Public），几乎全设置成 public，如之前的 eating() 或 sleeping() 等...(除非像 jdk 工具类，访问设置则不同考虑性能也不同，会考虑那些是可调用那些是不可，工具类面向人群不同，访问级别考虑也不同)。

注：创建 JavaBean 类，一般放置的包名称(也称之为实体)为：entity, **model**, bean, javabean, **pojo**, vo

创建封装类对象

实例或是叫对象，都指同一个。

封装类也叫 **JavaBean**

1) 封装 House（房子）代码：

```

package com.fy.model;
/**
 * 封装类：房
 * @author Administrator
 * @date 2019年7月12日
 * @version
 */
public class House {
    private int id; //编号
    private String address; //地址
    private String type; //房子类型:别墅,洋房,大平层,商铺,公寓,复式
    private double price; //价格
    private double area; //面积大小

    //source, 生成set/get
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}

```

```

    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
    public double getArea() {
        return area;
    }
    public void setArea(double area) {
        this.area = area;
    }
}

```

- 1).实例中 public 方法 (如 : public String getName()) 是外部类访问该类成员变量的入口。
- 2).这些方法被称为 getter 和 setter 方法。
- b).任何要访问类中私有成员变量的类都要通过这些 getter 和 setter 方法。

2).访问 House 类的变量赋取值

```

/**
 * 测试封装
 * @author djy
 * @date 2019年7月12日
 * @version
 */
public class HouseTest {

    //程序主入口
    public static void main(String[] args){
        //创建对象
        House house = new House();
        house.setId(1); //赋值从1编号
        house.setAddress("北京七环");
        house.setPrice(2000000000.0);
        //别墅,洋房,大平层,商铺,公寓,复式
        house.setType("别墅");
        house.setArea(150);

        //取数据
    }
}

```

```

        System.out.println("编号:"+house.getId()
                            +",地址:"+house.getAddress()
                            +",价格:" +house.getPrice()
                            +",房型:" +house.getType()
                            +",面积:" +house.getArea()

        );
    }

}

```

3).带参数构造方法的赋取值，也可添加 Person 类到 House 类定义赋取值

```

//House类设置Person类
private Person person;//引入Person类 //set,get设置
public Person getPerson() {
    return person;
}
public void setPerson(Person person) {
    this.person = person;
}

```

4).测试代码：

```

/**
 * 测试封装
 * @author djy
 * @date 2019年7月12日
 * @version
 */
public class HouseTest {

    //程序主入口
    public static void main(String[] args){
        //创建对象
        House house = new House();
        house.setId(1);//赋值从1编号
        house.setAddress("北京七环");
        house.setPrice(2000000000.0);
        //别墅,洋房,大平层,商铺,公寓,复式
        house.setType("别墅");
        house.setArea(150);

        //房子属于Person类的谁家的
        Person person = new Person();
        person.setName("爱新觉罗.王凯");
        person.setAge(18);
        house.setPerson(person);
    }
}

```



```
//取数据
System.out.println("编号:"+house.getId()
                    +",地址:"+house.getAddress()
                    +",价格:" +house.getPrice()
                    +",房型:" +house.getType()
                    +",面积:" +house.getArea()
                    +",姓名:" +house.getPerson().getName()
                    +",年龄:" +house.getPerson().getAge()
                );
    }

}
```

Java 继承

简介

Java 继承是面向对象的最显著的一个特征。继承是从已有的类中派生出新的类，新的类能吸收已有类的数据属性和行为，并能扩展新的能力（主要特性是做扩展，如父亲，父亲下面有儿子或女儿，儿子下面又有儿子...）。

类和类之间的有继承关系，其中父类又叫超类或基类，子类又叫派生类。父类是子类的一般化，子类是父类的特化。（一般只说：父类[或超类，基类]，子类[或派生类]）

JAVA 不支持多继承，**单继承**使JAVA的继承关系很简单，一个类只能有一个父类，易于管理程序，同时一个类可以实现多个接口，从而克服单继承的缺点。（真正开发当中使用面向接口方式，变成多继承实现方式，而类只有单继承）

继承就是子类继承父类的特征和行为，使得子类对象（或叫实例化）具有父类的实例域和方法，或子类从父类继承方法，使得子类具有父类相同的行为。（如：子继承父方式，俗语：子承父业）

注：只要继承关系存在，子就拥有父的一切（除了私有属性之外,如同前面介绍的，父亲留给的粮票，现在这年代还是能用吗？）Java 中最高的父类是 **Object**，所有的 java 类都是默认继承自 Object。

使用关键字：**extends** (中文译：扩展)。

继承格式

```
public class 父类 { }
public class 子类 extends 父类 { }
```

extends：类继承为**单一继承**，一个子类只能拥有一个父类，因此extends 只能继承一个类。

继承类（父子关系）

1).子类

```

/**
 * 子类
 * @author Administrator
 * @date 2019年1月13日
 */
public class Son{
    private String name;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Son() {
    }

    public Son(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void eat(){
        System.out.println(name + "正在吃饭");
    }

    public void watchTV(){
        System.out.println(name + "正在看电视");
    }
}

```

2).女(儿)类

```

/**
 * 女(儿)类
 * @author Administrator
 * @date 2019年1月13日
 */
public class Daughter {
    private String name;
    private int age;
}

```

```

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Daughter() {
    }

    public Daughter(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void eat(){
        System.out.println(name + "正在吃饭");
    }

    public void watchTV(){
        System.out.println(name + "正在看电视");
    }
}

```

上述两个类的代码可以看出来，代码存在重复了，导致产生太多重复性代码且冗余，维护性变得不高(维护性主要是后期需要修改的时候，就需要修改很多的代码，容易出错)，**继承**则从根本上解决重复性代码，，将两段代码中相同的部分提取出来（抽出相近的代码，也可以叫抽象）组成一个父类 Father 类，Son 子类，Daughter 子类，只要具有继承关系，子类的代码就会变得简洁。

3)父类：

```

/**
 * 父类：做继承使用
 * @author Administrator
 * @date 2019年1月13日
 */
public class Father { /*extends Object*/ //Ctrl+t查看Father是否默认就在Object类下，层级关系明显

    private String name;
    private int age;

    public String getName() {
        return name;
    }
}

```

```

    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Father() {
    }

    public Father(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void eat(){
        System.out.println(name + "正在吃饭");
    }

    public void watchTV(){
        System.out.println(name + "正在看电视");
    }

}

```

Father 类就可以作为一个父类，然后 Son 类, Daughter 类继承这个父类，就具有父类当中的属性和方法，子类就不会存在重复性的代码，维护性也提高，代码变少更加简洁，提高代码的复用性（复用性主要是指可以多次使用，不用多次写同样的代码）继承之后的代码：

4).Son 子类, Daughter 子类继承父 Father 类

```

/**
 * 子类
 * @author Administrator
 * @date 2019年1月13日
 */
public class Son extends Father{

    //继承父类拥有的属性及方法
    public Son(String name, int age) {
        super(name, age);
    }

    public Son() {
    }

}

```

```

/**
 * 女类
 * @author Administrator
 * @date 2019年1月13日
 */
public class Daughter extends Father {

    //继承父类拥有的属性及方法
    public Daughter(String name, int age){
        super(name, age);
    }
}

```

5).测试代码

```

public class Test {
    public static void main(String[] args) {
        // 父
        Father father = new Father("李渊", 40);

        //instanceof 类型比较 比较属性什么类型
        //System.out.println(father instanceof Father);//还是自身类:true
        //System.out.println(father instanceof Son);//向下比较子类:false
        //System.out.println(father instanceof Object);//最高类Object:true

        father.eat();
        father.watchTV();

        // 儿子
        Son son = new Son("李世民", 20);
        //System.out.println(son instanceof Father);//向上比较:true
        son.eat();
        son.watchTV();

        // 女儿
        Daughter daughter = new Daughter("李秀宁", 25);
        daughter.eat();
        daughter.watchTV();
    }
}

```

6).运行结果：

```

李渊正在吃饭
李渊正在看电视
李世民正在吃饭
李世民正在看电视
李秀宁正在吃饭
李秀宁正在看电视

```

注：子类继承父类的一切（可以不断的扩展），从而代码**重用**。

重写 Override

前面的父类的 eat 和 watchTV 两个方法，如果子类认为不能满足自己想要的功能实现方式，子类可以**重写父类的方法**，输出自己指定的需求功能（一句话：就是重写父类方法）。

方法重写要点：

- 1). 方法名，形参（就是传入参数及类型及位置顺序）列表一样
- 2). 返回值类型和声明异常类型，子类小于等于父类
- 3). 访问权限，子类则大于等父类

如 Son 子类重写 eat 方法：

```
public void eat(){//子类方法覆盖父类的方法
    System.out.println("肚子饿了，点个外卖过来");
}
```

当调用测试 son 的 eat 方法时，输出就是自己编写的代码。

注：重写也有快捷方式

Father 有一个方法返回类型：

```
public Father getFath(){
    return new Father();
}
```

相当于 Father fater = new Father () ；

子类重写此方法：

```
@Override
public Son getFath() { //返回类型可以是Father或Father下的子类
    return new Son();
}
```

访问权限:

```
//访问权限（子类则大于等父类）
Son son2 = new Son(); //子类返回可以是子类，也可以是父类
son.watchTV(); //子类可以访问到父类的方法
son.sleep(); //子类编写一个专属自己的方法，自己可以访问
Father father2 = new Father("李渊", 40);
father2.sleep(); //父类只能访问自己的方法，不可以访问Son子类的sleep()休息的方法

//Son son4 = new Father(); //父类返回只能是父类
```

Object 类

object 所有类的最高父类或叫根类。

位于 java.lang 包下，了解查看更多信息（源码或 api 方式）：

```
5 package java.lang;
7
8 /**
9  * Class {@code Object} is the root of the class hierarchy.
10  * Every class has {@code Object} as a superclass. All objects,
11  * including arrays, implement the methods of this class.
12  *
13  * @author unascribed
14  * @see java.lang.Class
15  * @since JDK1.0
16  */
17 public class Object {
18
19     private static native void registerNatives();
20     static {
21         registerNatives();
22     }
23 }
```

native 则是本地实现，如 hashCode 方法...

编写查看方法：

```
/**
 * 测试Object类方法
 * @author Administrator
 * @date 2019年7月7日
 * @version
 */
public class ObjectTest {

    public static void main(String[] args) {
        ObjectTest ot = new ObjectTest();
        System.out.println(ot); //控制台输出：com.fy.test.ObjectTest@2a139a55
        //调用对象输出，实际上是调用了对象的toString()方法
        System.out.println(ot.toString()); //输出与前面一致
        //ot能直接调用toString()方法，是因为继承于Object根类，父类有的方法，子类都有，可以直接使用
    }
}
```

查看 toString() 源码:

```

/*
 * @return a string representation of the object.
 */
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
    //返回：包+类名+@+hashCode码（内存地址）
    //对应：com.fy.test.ObjectTest@2a139a55

    //hashCode转换成16进制数字,点击toHexString查看便知或是鼠标移到方法上也可查看
}

```

也可以自己重写 toString() 方法：

感

```

//重写toString(), 重写方法名要一致, 如果toString()则不是重写父类方法, 而是新建方法, 因此大写敏感
public String toString() {
    return "重写toString";
}

```

重新启动执行 main 发现，此时控制台的输出为“重写toString”。

选择前面的任意类重写 toString(), 看一下变化是什么：

```

@Override
public String toString() {
    return "Father [name=" + name + ", age=" + age + "]";
}

```

==和equals比较：

- 1) .==：双方两边的值是否相同，基础类型则表示值相等，引用类型则表示地址相等，指向的是同一个对象
- 2) .equals：对象内容[值]均相等，是Object类中的方法(需要重写此方法equals())

注：比较数字类型使用==, 字符串使用equals

注：开发中equals很常用，用于登录账号名（真正比对的是账号类里的id），或比较身份证号是否相同等等...

源码：

```

/*
 * Note that it is generally necessary to override the {@code hashCode}
 * method whenever this method is overridden, so as to maintain the
 * general contract for the {@code hashCode} method, which states
 * that equal objects must have equal hash codes.
 */
public boolean equals(Object obj) {
    return (this == obj); //比较hashCode是否相同（是否为同一个对象）
}

```

比较1：


```

// 字符串比较(定义一个String变量,然后再+[追加])
public static void runStringCompana6() {
    String s1 = "12"; //方法区
    String s2 = s1 + "3"; //+追加,在堆中生成一个内存地址,这时已经是个对象,相当于new之后
    String s3 = "123"; //在方法区

    // ==与equals
    System.out.println("==号比较:" + (s2 == s3));
    System.out.println("*****equals*****");
    System.out.println("[String变量之后+]equals比较," + s2 + "与" + s3 + "字符串:" +
(s2.equals(s3))); // 比较输出值是否一样
}

// 字符串比较(new方式,还有String a=55)
public static void runStringCompana5() {
    String s1 = new String("张三"); //new 在堆中产生一个对象,产生一个内存地址
    String s2 = "张三"; //字符串在方法区

    // ==与equals
    System.out.println("==号比较:" + (s1 == s2)); //比较内存地址[@11222]是否一致,然后才比较
值
    System.out.println("*****equals*****");
    System.out.println("String类型new之后 (与一个String直接赋值)的equals比较," + s1 +
"与" + s2 + "字符串:" + (s1.equals(s2))); // 比较输出值是否一样
}

// 字符串比较(new方式)
public static void runStringCompana4() {
    String h = new String("张三"); //new 在堆中产生一个对象,产生一个内存地址
    String i = new String("张三"); //在堆中生成一个新的对象

    // ==与equals
    System.out.println("==号比较:" + (h == i)); //比较内存地址[@11222]是否一致,然后才比较值
    System.out.println("*****equals*****");
    System.out.println("String类型new之后的equals比较," + h + "与" + i + "字符串:" +
(h.equals(i))); // 比较输出值是否一样
}

// 字符串比较
public static void runStringCompana3() {
    String e = "12";
    int f = Integer.valueOf(e)+3; //Integer.valueOf(e) = 把"12"转换成
int//Integer.valueOf产生内存地址,堆中
    String g = "15";

    // ==与equals
    System.out.println("equals比较," + f + "和" + g + "数字:" + (g.equals(f))); // 比
较输出值是否一样
}

//字符串比较
public static void runStringCompana2(){
    String c = "111"; //在方法区

```

样

```
String d = "111";//在方法区

//==与equals
System.out.println("==比较,"+c+"和"+d+"数字:"+ (c==d));
System.out.println("*****equals比较*****");
System.out.println("equals比较,"+c+"和"+d+"数字:"+ (c.equals(d)));//比较输出值是否一
}

public static void runNum1(){
    int a = 55;//在方法区
    int b = 85;//在方法区
    //boolean result = (a==b);
    boolean result = (a==b);
    System.out.println(a+"和"+b+"的数字型比较:"+result);//没有内存地址比较
}
```

比较2：

址

```
public static void main(String[] args) {
    String a = "123";
    String b = "12"+"3";
    String c = "12"+3;
    //上面的定义全是在方法区的
    System.out.println(a==b);//此时，a输出123,b打印输出123
    System.out.println(a==c);//此时，a输出123,c打印输出123

    //new 之后是在堆中生成@地址
    String sPwd = new String("z111111");
    String sRePwd = new String("z111111");
    System.out.println(sPwd==sRePwd);//==号比较地址是否一样，当new之后就是生成一个堆对象地址

    b = "z111111";//方法区显示
    System.out.println(sPwd == b);//前面是对象地址在堆区，b是还是在方法区

    //runCompara();
}

public static void runCompara(){
    int pwd = 111111;
    int rePwd = 222222;
    //==判定使用在数字型
    /*if(pwd == rePwd){
        System.out.println("密码一致");
    }else{
        System.out.println("两个密码不一致");
    }*/

    //equals只能使用在String类型中比较
    /*String sPwd = "z111111";//在方法区
```

```

String sRePwd = "1111111"; */

//对象地址不一样,寻址地址不一样 ,堆方法区图()
String sPwd = new String("z111111"); //在堆区中产生
String sRePwd = new String("z111111");

// == 先比较地址不一样
//只要是字符串String就不要使用 == 做比较,出现两个输入值是不一致,判定会是不一致
if(sPwd == sRePwd){ //new String("z111111")    new String("z111111")
    System.out.println("一致==");
}else{
    System.out.println("不一致...");
}

//equals比较两个值是否一样
if(sPwd.equals(sRePwd)){ //z111111    z111111
    System.out.println("equals比较:String两者密码一致");
}else{
    System.out.println("equals比较:String两者密码不一致");
}
}

```

比较3 :

```

/**
 * 测试equals方法
 * @author Administrator
 * @date 2019年7月7日
 * @version
 */
public class EqualsTest {

    public static void main(String[] args) {
        //创建一个User的javabean(有idCard, name两个属性)
        User u1 = new User("421234199022345678", "张三");
        User u2 = new User("421234199022345678", "李四");
        System.out.println(u1 == u2); //false
        System.out.println(u1.equals(u2)); //false
    }
}

```

User 添加 equals 之后:

```

@Override
public boolean equals(Object obj) {
    if (this == obj) //传入obj与this相等,则为同一对象,返回true
        return true;
    if (obj == null) //传入对象为空
        return false;
    if (getClass() != obj.getClass()) //类 ( Father, Car ) 不一样,也无需比较

```

```

        return false;
    }
    User other = (User) obj; //强制转型
    if (idCard == null) { //身份证号为空
        if (other.idCard != null)
            return false;
    } else if (!idCard.equals(other.idCard)) //两边idCard不相等
        return false;
    return true;
}

```

```

public static void main(String[] args) {
    //创建一个User的javabean(有idCard, name两个属性)
    User u1 = new User("421234199022345678", "张三");
    User u2 = new User("421234199022345678", "李四");
    System.out.println(u1 == u2); //false
    System.out.println(u1.equals(u2)); //true, 变化这是因为重写equals, 开发中根据id进行比较, 如id相同则为同一个人
}

```

String 类 equals 源码：

```

public boolean equals(Object anObject) {
    if (this == anObject) { //传入anObject与this相等
        return true;
    }
    if (anObject instanceof String) { //也为String类型
        String anotherString = (String)anObject; //类型强制转型
        int n = value.length;
        if (n == anotherString.value.length) { //字符长度是否相同, 按每个字符进行逐一比较
            char v1[] = value;
            char v2[] = anotherString.value;
            int i = 0;
            while (n-- != 0) {
                if (v1[i] != v2[i])
                    return false;
                i++;
            }
            return true; //全相同返回true
        }
    }
    return false;
}

```

测试：

```

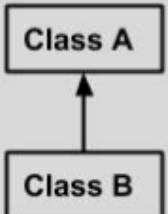
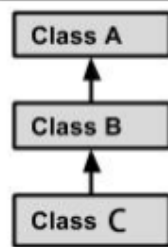
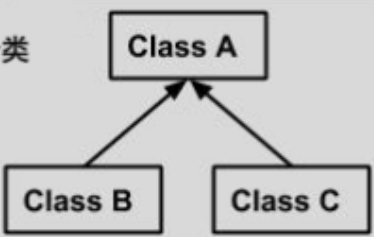
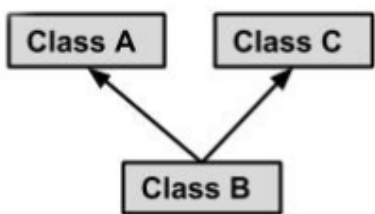
String name = new String("张三");
String name2 = new String("张三");
System.out.println(name == name2); //false : ==比较对象hashCode地址是否一致

System.out.println(name.equals(name2)); //true

```

继承类型说明

Java 不支持多继承（单继承），但支持多重继承。

单继承	 <pre>graph BT; B[Class B] --> A[Class A]</pre>	<pre>public class A { } public class B extends A { }</pre>
多重继承	 <pre>graph BT; C[Class C] --> B[Class B]; B --> A[Class A]</pre>	<pre>public class A {} public class B extends A {} public class C extends B {}</pre>
不同类继承同一个类	 <pre>graph BT; B[Class B] --> A[Class A]; C[Class C] --> A</pre>	<pre>public class A {} public class B extends A {} public class C extends A {}</pre>
多继承（不支持）	 <pre>graph BT; B[Class B] --> A[Class A]; B --> C[Class C]</pre>	<pre>public class A {} public class B {} public class C extends A,B { } // Java 不支持多继承</pre>

继承的特性

- 1). 子类拥有父类非 `private` 的属性、方法。（父亲有的，儿子全可以使用）
- 2). 子类可以拥有自己的属性和方法，即子类可以对父类进行扩展。
- 3). 子类可以用自己的方式实现父类的方法。
- 4). Java 的继承是单继承，但是可以多重继承，单继承就是一个子类只能继承一个父类，多重继承就是，例如 A 类继承 B 类，B 类继承 C 类，所以按照关系就是 C 类是 B 类的父类，B 类是 A 类的父类，这是 Java 继承区别于 C++ 继承的一个特性。
- 5). 提高类之间的耦合性（继承的缺点，耦合度高就会造成代码之间的联系越紧密，代码独立性越差）。

继承关键字

继承使用 `extends` 和 `implements`（接口类关键字）这两个关键字来实现继承。

*: class 类继承使用 extends 关键字，implements 是继承接口实现方式（接口后面会学习）

super

1).super 关键字

通过 super 关键字来实现对父类的**属性及方法**的访问，用来引用**当前对象**的父类。

构造方法的**第一句**总是 supper()（不写也是默认调用父类构造器，不写，编译器也会自动加上），如果使用自动生成构造器，则会自动创建 supper()，作用则是调用父类的构造方法。

Son 添加：

```
public class Son extends Father{

    //父类
    public Son(String name, int age) {
        super(name, age);
    }

    public void eat(){//子类方法覆盖父类的方法
        super.eat();//调用父类原来的eat()普通方法，查看其原来是什么
        System.out.println("原来父类姓名："+super.getName());//访问原来父类属性
        System.out.println("肚子饿了，点个外卖过来");
    }
    //子类通过super关键字能访问父类的属性和方法
    .....
}
```

```
/**
 * super测试
 * @author Administrator
 * @date 2019年7月7日
 * @version
 */
public class SupperTest {

    public static void main(String[] args) {
        System.out.println("执行main...");
        new Boy();//先输出父类构造器，后子类构造器(从上至下输出，而查看时从下到上追溯，一直到Object)
    }

}

class Person{
    public Person(){
        System.out.println("Person...");
    }
}

class Boy extends Person{
    public Boy(){
        //super();//默认第一句//不加编译器自动添加，最高Object
    }
}
```

```
        System.out.println("Boy...");
    }
}
```

2).this 关键字：指向自己的对象引用（对象和类中有介绍 this）。

构造器

子类是不继承父类的构造器（构造方法或者构造函数）的，它只是调用（隐式或显式）。如果父类的构造器带有参数，则必须在子类的构造器中显式地通过 super 关键字调用父类的构造器并配以适当的参数列表。

如果父类构造器没有参数，则在子类的构造器中不需要使用 super 关键字调用父类构造器，系统会自动调用父类的无参构造器。

匿名对象

简称为没有名字的对象。

```
package test_test;

public class Car {
    String color;//颜色
    int num;//轮胎数量

    public void run() {
        System.out.println(color + "==" + num);
    }
}
```

```
public class CarTest {

    public static void main(String[] args) {
        // 创建有名字的对象
        Car c1 = new Car();
        //c1.run();

        //匿名对象调用方法
        //new Car().run();
        //匿名对象只适合对方法的一次调用,因为调用多次就会产生多个对象,不如用有名字的对象

        //匿名对象可以调用属性,但是没有意义,因为调用后就变成垃圾,如果需要赋值还是使用有名字对象更好
        new Car().color = "red";
        new Car().num = 4;
        //new Car().run();

    }

    //类中定义类, class Car{...}则为内部类
}
```

