链滴

Java 面向对象

作者：Giles
原文链接：https://ld246.com/article/1592405267090
来源网站：链滴

<h4 id="面向对象概念">面向对象概念</h4>
1 使用对象的关系来描述事物之间的联系，这种思想就是面向对象。
2 面向对象的特点主要为继承性、封装性和多态性。
3 对象就是存在的具体实体，具有明确定义的属性和方法。
4 类是具有相同属性和共同方法的一组对象的集合。
5 类是创建对象的模板，类声明的变量就是对象。通过 new 关键字来实例化对象。
6·使用操作符“.”来访问对象的属性和方法。
<h4 id="继承">继承</h4>
Java 中使用 extends 关键字来表示继承关系；
java.lang.Object 类是所有 Java 类的父类
类只支持单继承，不允许多重继承。（支持多层继承）
多个类可以继承一个父类。
在 Java 中，多层继承是可以的，即一个类的父类可以再去继承另外的父类。
关键字 extends 继承
super 超类，父类，基类
使用 super 关键字调用父类的成员变量和方法，
使用 super 关键字调用父类的构造方法的时候，super()必须要写在方法体的第一行，只能出现次。
只能在构造方法中使用 super 或者 this 调用其他的构造方法，不能在成员方法中使用。
class Person {
public Person() {
<code>}</code>
}
class Man extends Person {
public Man() {
<code>}</code>
}类 Man 继承于 Person 类，这样一来的话，Person 类称为父类（基类），Man 类称为子类（出类）。如果两个类存在继承关系，则子类会自动继承父类的方法和变量，在子类中可以调用父类的法和变量。在 java 中，只允许单继承，也就是说 一个类最多只能显示地继承于一个父类。但是一个却可以被多个类继承，也就是说一个父类可以拥有多个子类。
<h4 id="封装">封装</h4>
通俗点来讲就是汽车是我们私有的物品（不对外开放的 private），你朋友想要借你的车开，但你不想借给他，这个时候你就要提出一个方法（对外开放的 public），你必须满足我的条件（get 和 et 方法中的逻辑）你才能把我的车开走（实现调用）
使用封装隐藏类的实现细节，让调用者只能通过规定的方法访问数据；
封装可以被认为是一种能够保护代码和数据被定义在类外的其它代码任意访问的屏障。访问数据代码由一个接口严格控制。
封装的主要好处是修改我们实现的代码而又不会破坏其他人使用我们的代码。封装的这个特性使们的代码具有可维护性、灵活性以及扩展性。
public class Employee{
private Integer id;
private String lastName;
private String email;
<pre><code class="highlight-chroma">public Integer getId() {
    return id;
}
public void setId(I teger id) {
    this.id = id;
}
public String getL stName() {
    return lastName

}
public void setLas
Name(String lastName) {
    this.lastName =
astName;
}
public String get
mail() {
    return email;
}
public void setEm
il(String email) {
    this.email = em
il;
}
</code></pre>
}公有方法是从类外访问到类内字段的入口。通常情况下，这些方法被定义为 getters 和 setters 因此想要访问类内变量的任何其他类要使用 getters 和 setters 方法。
如下可以创建一个 test 访问如下：
public class EmployeeTest{
public static void main(String args[]){
Employee employee = new Employee();
employee.setId(1);
employee.setLastName("张三");
employee.setEmail("123@qq.com");
System.out.print("lastName : " + employee.getLastName()+
" email : "+ employee.getEmail());
}
}输出如下：
lastName:张三 email：123@qq.com###### 封装的优点

类中的字段可以被设置为只读或只写。
类可以完全控制它字段里面所存储的东西。
类的使用者不用知道类是如何存储数据的。类可以改变字段的数据类型而类的使用者不需要改变何之前的代码。

<h4 id="多态">多态</h4>
<h4 id="定义">定义</h4>
指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同而采用多种不同行为方式。（发送消息就是函数调用）；
实现多态的技术称为：动态绑定（dynamic binding），是指在执行期间判断所引用对象的实际型，根据其实际的类型调用其相应的方法。
多态的作用：消除类型之间的耦合关系。
<h6 id="多态存在的三个必要条件">多态存在的三个必要条件</h6>
一、要有继承；
二、要有重写；
三、父类引用指向子类对象。
<h6 id="多态的好处-">多态的好处：</h6>

可替换性（substitutability）。多态对已存在代码具有可替换性。例如，多态对圆 Circle 类工作对其他任何圆形几何体，如圆环，也同样工作。
可扩充性（extensibility）。多态对代码具有可扩充性。增加新的子类不影响已存在类的多态性继承性，以及其他特性的运行和操作。实际上新加子类更容易获得多态功能。例如，在实现了圆锥、

圆锥以及半球体的多态基础上，很容易增添球体类的多态性。
接口性（interface-ability）。多态是超类通过方法签名，向子类提供了一个共同接口，由子类完善或者覆盖它而实现的。如图 8.3 所示。图中超类 Shape 规定了两个实现多态的接口方法，compu eArea()以及 computeVolume()。子类，如 Circle 和 Sphere 为了实现多态，完善或者覆盖这两个口方法。
灵活性（flexibility）。它在应用中体现了灵活多样的操作，提高了使用效率。
简化性（simplicity）。多态简化对应用软件的代码编写和修改过程，尤其在处理大量对象的运和操作时，这个特点尤为突出和重要。

Java 中多态的实现方式：接口实现，继承父类进行方法重写，同一个类中进行方法重载。
实例一：
public class Wine {
public void fun1(){
System.out.println("Wine 的 Fun.....");
fun2();
}
<pre><code class="highlight-chroma">public void fun2(){
    System.out.print n("Wine 的Fun2...");
}
</code></pre>
}
public class JNC extends Wine{
/**

@desc 子类重载父类方法

<pre><code class="highlight-chroma"> 父类中不存在该方法，向上转型后，父类是不能引用该方法的
</code></pre>

@param a
@return void
*/
public void fun1(String a){
System.out.println("JNC 的 Fun1...");
fun2();
}

<pre><code class="highlight-chroma">/**
* 子类重写父类方

* 指向子类的父类用调用fun2时，必定是调用该方法
*/
public void fun2(){

    System.out.print n("JNC 的Fun2...");
}
</code></pre>
}

public class Test {
public static void main(String[] args) {
Wine a = new JNC();
a.fun1();
}
}运行结果：
Wine 的 Fun.....
JNC 的 Fun2... 实例二（多态==晚绑定）：
不要把函数重载理解为多态。因为多态是一种运行期的行为，不是编译期的行为。
比如 Parent p = new Child();
当使用多态方式调用方法时，首先检查父类中是否有该方法，如果没有，则编译错误；如果有，去调用子类的该同名方法。（注意此处，静态 static 方法属于特殊情况，静态方法只能继承，不能重 Override，如果子类中定义了同名同形式的静态方法，它对父类方法只起到隐藏的作用。调用的时用谁的引用，则调用谁的版本。）
如果想要调用子类中有而父类中没有的方法，需要进行强制类型转换，如上面的例子中，将 p 转为子类 Child 类型的引用。因为当用父类的引用指向子类的对象，用父类引用调用方法时，找不到父中不存在的方法。这时候需要进行向下的类型转换，将父类引用转换为子类引用。
结合实例说明，主要讲讲两种类型转换和两种编译时候的错误：
public class PolyTest
{
public static void main(String[] args)
{
<pre><code class="highlight-chroma">   //向上类型转换
    Cat cat = new at();
    Animal animal cat;
    animal.sing();

    //向下类型转换
    Animal a = new Cat();
    Cat c = (Cat)a;
    c.sing();
    c.eat();

    //编译错误
    //用父类引用调父类不存在的方法，Cat类中定义了eat()方法，但是Animal类中没有这个方法，a1引用是Animal类，所以找不到。
    //Animal a1 = ew Cat();
    //a1.eat();

    //编译错误
    //向下类型转换只能转向指向的对象类型，因为父类引用指向的是Cat类的对象，而要强制转换成Dog类，这是不可的。
    //Animal a2 = ew Cat();
    //Cat c2 = (Dog

a2;

}
</code></pre>
}
class Animal
{
public void sing()
{
System.out.println("Animal is singing!");
}
}
class Dog extends Animal
{
public void sing()
{
System.out.println("Dog is singing!");
}
}
class Cat extends Animal
{
public void sing()
{
System.out.println("Cat is singing!");
}
public void eat()
{
System.out.println("Cat is eating!");
}
}
多态部分原文链接：<a href="https://ld246.com/forward?goto=https%3A%2F%2Fblog.csdn.ne
%2Fa78270528%2Farticle%2Fdetails%2F80018602" target="_blank" rel="nofollow ugc">https
//blog.csdn.net/a78270528/article/details/80018602
<h4 id="接口">接口</h4>
接口，英文称作 interface，在软件工程中，接口泛指供别人调用的方法或函数。定义接口如下
public interface InterfaceEmployee{
...
}通俗来讲就是你买了一个 usb 接口的鼠标想要使用就必须插在你笔记本的 usb 接口上，不可能接在
ga 接口上把？
简单来说接口更像是一种标准和规范，接口之与 Java 就像是社会之于法律的关系。
接口 多继承的功能（继承接口） 防盗门必须实现门的功能和锁的功能，
接口中可以含有变量和方法。但是要注意，接口中的变量会被隐式地指定为 public static final
量（并且只能是 public static final 变量，用 private 修饰会报编译错误）；
方法会被隐式地指定为 public abstract 方法且只能是 public abstract 方法（用其他关键字，
如 private、protected、static、 final 等修饰会报编译错误），并且接口中所有的方法不能有具体
实现，也就是说，接口中的方法必须都是抽象方法。
要让一个类遵循某组特地的接口需要使用 implements：
public Employee implements Interface(){
...
}#### 包
在 Java 中使用包是为了防止命名冲突，来控制访问，使得搜索/定位和类、接口、枚举和注释等

使用更为简单。
包可以被定义为一组相关的类型（类、接口、枚举和注释），提供访问保护和命名空间管理。</
>
<h6 id="包的好处-">包的好处：</h6>
1 易于查找和使用
2 防止命名冲突
3 允许在更广的范围内保护类和方法
使用 package 来声明每个包；
使用 import 来导入包的路径
import java.util.Scanner; import java.util.*
String 是引用数据类型，但是每次使用的时候不需要导入，因为他在 java.lang 包下，系统会自
导入
public 公共的 private 私有的 protected 受保护的，default 默认的
private 修饰的成员变量是私有的，不能直接访问或者赋值，这时我们需要一个公有的的方法来实现
们的目的
<h4 id="重载">重载</h4>
构造方法的重载 就像你们毕业之后面试，填写入职申请表。面试了 N 家，每一家的入职申请表
不一样。
有的信息多，有的信息少。不管你们写的多，写的少，都是入职申请表。都实例化成为一张填完过的
职申请表
教给 hr。
普通的重载：在一个类中定义多个名称相同的方法，但参数的类型或者个数不同
<h4 id="重写">重写</h4>
如果一个类从它的父类继承了一个方法，如果这个方法没有被标记为 final，就可以对这个方法
行重写。
重写的好处是：能够定义特定于子类类型的行为，这意味着子类能够基于要求来实现父类的方法

重写就像是重新写作业，使用参照物的
public class Animal{
public void move(){
System.out.println("Animals can move");
}
}
public class Dog extends Animal{
public void move(){
System.out.println("Dogs can walk and run");
}
}
public class TestDog{
public static void main(String args[]){
Animal a = new Animal(); // Animal reference and object
Animal b = new Dog(); // Animal reference but Dog object
a.move();// runs the method in Animal class
b.move();//Runs the method in Dog class
}
}显示如下：
Animals can move
Dogs can walk and run 在上面的例子中，你可以看到尽管 b 是 Animal 类型，但它运行了 dog 类
方法。原因是：在编译时会检查引用类型。然而，在运行时，JVM 会判定对象类型到底属于哪一个对
。
因此，在上面的例子中，虽然 Animal 有 move 方法，程序会正常编译。在运行时，会运行特定
象的方法。
考虑下面的例子：
class Animal{

public void move(){
System.out.println("Animals can move");
}
}
class Dog extends Animal{
public void move(){
System.out.println("Dogs can walk and run");
}
public void bark(){
System.out.println("Dogs can bark");
}
}
public class TestDog{
public static void main(String args[]){
Animal a = new Animal(); // Animal reference and object
Animal b = new Dog(); // Animal reference but Dog object
a.move();// runs the method in Animal class
b.move();//Runs the method in Dog class
b.bark();
}
}这将产生如下结果：
TestDog.java:30: cannot find symbol
symbol : method bark()
location: class Animal
b.bark();
^
这个程序在编译时将抛出一个错误，因为 b 的引用类型 Animal 没有一个名字叫 bark 的方法。
<h5 id="重写规则">重写规则</h5>

重写方法的参数列表应该与原方法完全相同。
返回值类型应该和原方法的返回值类型一样或者是它在父类定义时的子类型。
重写函数访问级别限制不能比原函数高。举个例子：如果父类方法声明为公有的，那么子类中的写方法不能是私有的或是保护的。
只有被子类继承时，方法才能被重写。
方法定义为 final，将导致不能被重写。
一个方法被定义为 static，将使其不能被重写，但是可以重新声明。
一个方法不能被继承，那么也不能被重写。
和父类在一个包中的子类能够重写任何没有被声明为 private 和 final 的父类方法。
和父类不在同一个包中的子类只能重写 non-final 方法或被声明为 public 或 protected 的方法
一个重写方法能够抛出任何运行时异常，不管被重写方法是否抛出异常。然而重写方法不应该抛比被重写方法声明的更新更广泛的已检查异常。重写方法能够抛出比被重写方法更窄或更少的异常。
构造函数不能重写。

<h5 id="使用super关键字">使用 super 关键字</h5>
当调用父类的被重写的方法时，要用关键字 super。
class Animal{
public void move(){
System.out.println("Animals can move");
}
}
class Dog extends Animal{
public void move(){

super.move(); // invokes the super class method
System.out.println("Dogs can walk and run");
}
}
public class TestDog{
public static void main(String args[]){
Animal b = new Dog(); // Animal reference but Dog object
b.move(); //Runs the method in Dog class
}
}这将产生如下结果:
Animals can move
Dogs can walk and run#### 异常处理
Throwable： 是所有异常的祖先类
Error： 指的是程序无法处理的错误
Exception： 指的是程序本身可以处理的异常
try{
警察抓坏人，首先你要锁定坏人所在的区域
可能发生异常的代码块;可能有错误的区域；查几句，它就会在这几句代码快的范围内找，其他的代码就不管了
}catch(语句可能遇到的异常的类型 对象){
抓到换人以后，该怎么处理
异常处理代码块; 比如说 System.out.print("");
}finally{
无论是否发生异常都会执行的代码块
无论是否抓到坏人，都会收队
} Java 的异常可以分为 检查时异常 和 运行时异常
运行时异常 : RuntimeException 及其所有子类
<h4 id="String类">String 类</h4>
<pre><code class="highlight-chroma"> //直接赋值的方法
  String temp = "bc";
  //构造方法赋值
  String temp1 = new String("构造赋值");
  //用+号进行字串的拼接
  String temp2 = emp+temp1;
  //字符串可以和他类型的变量进行连接，但是连接之后的结果也是String型的
  //如果连接字符的操作较多的时候，不建议用"+"号拼接，用StringBuffer类进行操作
  System.out.print(temp2+(num+num1));

  String hello = "ello world";
  //字符串的lengt()方法，用户获取字符串的长度
  int len = hello.lngth();
  System.out.print("字符串的长度是"+len);

    //返回指定索引的 char 值。索引范围为从 0 到 length() - 1。序列的第一个 char 值位于索引 0 处，第二个位于索引 处，依此类推，这类似于数组索引。
    char ctemp = ello.charAt(0);
    System.out.print n(ctemp);

    String hello = " ello world";
    //返回指定子字串在此字符串中第一次出现处的索引。
    int num = hello. ndexOf("o");
    //返回指定子字串在此字符串中最后一次出现处的索引。
    int lastNum = h llo.lastIndexOf("o");
    System.out.print n(num);
    System.out.print n(lastNum);
    //返回一个新的符串，它是此字符串的一个子字符串。该子字符串从指定索引处的字符开始，直到此字符串末尾。
    String subTemp = hello.substring(2);
    System.out.print n(subTemp);
    //返回一个新字串，它是此字符串的一个子字符串。该子字符串从指定的 beginIndex 处开始，直到索引 endIndex - 处的字符。因此，该子字符串的长度为 endIndex-beginIndex。
    String subTem 1 = hello.substring(2,5);
    System.out.print n(subTemp1); 程序代码没有运行的时候是放在硬盘里面的。

</code></pre>
<h4 id="Stringbuffer">Stringbuffer</h4>
<pre><code class="highlight-chroma">   String str1 = "hel";
    String strUp = Hel";
    //假设一个验证的场景，提示输入四个验证码，不论输入的大小，小写，都可以验证通过。
    //java是大小写感的，所以用equals肯定是不行的
    boolean sta2 = tr1.equals(strUp);
    //将此 String 另一个 String 比较，不考虑大小写。如果两个字符串的长度相同，
    //并且其中的相字符都相等（忽略大小写），则认为这两个字符串是相等的。
    boolean sta = s r1.equalsIgnoreCase(strUp);

```
</span></span><span class="highlight-line"><span class="highlight-cl">    System.out.print
n(str1+"和"+strUp+"是否相等"+sta2);
</span></span><span class="highlight-line"><span class="highlight-cl">    System.out.print
n(str1+"和"+strUp+"用IgnoreCase比较是否相等"+sta);
</span></span><span class="highlight-line"><span class="highlight-cl">    String res = "HE
lo WOrLd";
</span></span><span class="highlight-line"><span class="highlight-cl">    //想得到小写的r
s
</span></span><span class="highlight-line"><span class="highlight-cl">    System.out.print
n("没转换之前的是"+res);
</span></span><span class="highlight-line"><span class="highlight-cl">    System.out.print
n("LowerCase用转换之后的是"+res.toLowerCase());
</span></span><span class="highlight-line"><span class="highlight-cl">    //想得到大写的r
s
</span></span><span class="highlight-line"><span class="highlight-cl">    System.out.print
n("UpperCase用转换之后的是"+res.toUpperCase());
</span></span></code></pre>
```