第 2 章

Java 核心面试基础

本章导读

从本章开始主要带领读者学习 Java 的基础知识以及在面试和笔试中常见的问题。本章先告诉读者要掌握的重点知识有哪些，然后将教会读者应该如何更好地回答这些问题，最后总结了一些在企业的面试及笔试中较深入的真题。

知识清单

本章要点（已掌握的在方框中打钩）

- ☐ 数据类型和变量
- ☐ 运算符和流程控制语句
- ☐ 面向对象的特性
- ☐ 抽象类和抽象方法
- ☐ 接口的使用

2.1 Java 核心知识

本节主要讲解 Java 中的基本数据类型、局部变量和成员变量、运算符和表达式以及流程控制语句等基础知识。读者只有牢牢掌握这些基础知识才能在面试及笔试中应对自如。

2.1.1 数据类型

Java 中有两大数据类型，分别为基本数据类型和引用数据类型。

基本数据类型如表 2-1 所示。

表 2-1 基本数据类型

数 据 类 型	位数/b	表示及作用
byte（位）	8	有符号的、以二进制补码表示的整数，数值取值范围是–128 ～ 127

续表

数 据 类 型	位数/b	表示及作用
short（短整数）	16	有符号的、以二进制补码表示的整数，数值取值范围是-32 768～32 767
int（整数）	32	有符号的、以二进制补码表示的整数，数值取值范围是-2 147 483 648 ～ 2 147 483 647。一般的整型变量默认为 int 类型
long（长整数）	64	有符号的、以二进制补码表示的整数，数值取值范围是-9 223 372 036 854 775 808 ～ 9 223 372 036 854 775 807。该类型主要用于比较大的整数的系统
float（单精度）	32	单精度、符合 IEEE 754 标准的浮点数，在储存大型浮点数组的时候可节省内存空间。数值取值范围是 1.4E-45～3.402 823 5E38
double（双精度）	64	双精度、符合 IEEE 754 标准的浮点数，浮点数的默认类型为 double 类型。数值取值范围是 4.9E-324～1.797 693 134 862 315 7E308
boolean（布尔）	1	表示一位的信息，只有 true 和 false 两个值。这种类型只作为一种标志来记录 true/false 的情况
char（字符）	16	char 类型是一个单一的 Unicode 字符，数值取值范围是 0～65 535；char 数据类型可以储存任何字符

引用数据类型包括类、接口、数组等，这些在之后的章节中将会介绍到。

在 Java 中数据类型的转换有两种方法：

（1）自动类型转换。编译器自动完成类型转换，不需要在程序中编写代码。

（2）强制类型转换。强制编译器进行类型转换，必须在程序中编写代码。

由于基本数据类型中 boolean 类型不是数字型，所以基本数据类型的转换是除了 boolean 类型以外的其他 7 种类型之间的转换。

自动转换类型的情况如下：

（1）整数类型之间可以相互转换，如 byte 类型的数据可以赋值给 short、int、long 类型的变量；short、char 类型的数据可以赋值给 int、long 类型的变量；int 类型的数据可以赋值给 long 类型的变量。

（2）整数类型转换为 float 类型，如 byte、char、short、int 类型的数据可以赋值给 float 类型的变量。

（3）其他类型转换为 double 类型，如 byte、char、short、int、long、float 类型的数据可以赋值给 double 类型的变量。

- 自动类型转换规则：从存储范围小的类型到存储范围大的类型，即 byte→short（char）→int→long→float→double。

☆**注意**☆ 在整数之间进行类型转换时，数值不发生改变，而将整数类型（尤其是比较大的整数类型）转换成小数类型时，由于存储方式的不同，可能存在数据精度的损失。

- 强制类型转换规则：从存储范围大的类型到存储范围小的类型，即 double→float→long→int→short（char）→byte。

语法格式：

```
（type）value
```

其中，type 是要强制类型转换后的数据类型。例如：

```
int a = 123
```

```
byte b = (byte)a
```

2.1.2　常量和变量

1. 常量

常量即在程序运行过程中一直不会改变的量。常量在整个程序中只能被赋值一次，并且一旦被定义，它的值就不能再被改变。声明常量的语法格式如下：

```
final 数据类型 变量名[=值]
```

常量名称通常使用大写字母。常量标识符可由任意顺序的大小写字母、数字、下画线（_）和美元符号（$）等组成，标识符不能以数字开头，也不能是 Java 中的保留字和关键字。

当常量用于一个类的成员变量时，必须给常量赋值，否则会出现编译错误。

Java 还允许使用一种特殊形式的字符常量值来表示一些难以用一般字符表示的字符，这种特殊形式的字符是以“\”开头的字符序列，称为转义字符。

Java 中常用的转义字符及含义如表 2-2 所示。

表 2-2　Java 中常用的转义字符及含义

转 义 字 符	含　义
\ddd	1～3 位八进制数所表示的字符
\uxxxx	1～4 位十六进制数所表示的字符
\'	单引号字符
\"	双引号字符
\\	双斜杠字符
\r	回车
\n	换行
\b	退格
\t	横向跳格

2. 变量

变量代表程序的状态，程序通过改变变量的值来改变整个程序的状态。

在程序中声明变量的语法格式如下：

```
数据类型 变量名称;
```

☆**注意**☆　数据类型和变量名称之间需要使用空格隔开，空格的个数不限，但是至少需要一个；语句使用“;”作为结束。

1）变量的命名规则

（1）变量名必须是一个有效的标识符。

（2）变量名不可以使用 Java 中的关键字。

（3）变量名不能重复。

（4）选择有意义的单词作为变量名。

2）变量的分类

根据作用域的不同，一般将变量分为成员变量和局部变量。

（1）成员变量。

成员变量又分为全局变量和静态变量。

全局变量不需要使用 static 关键字修饰，而静态变量则需要使用 static 关键字进行修饰。

全局变量在类定义后就已经存在，占用内存空间，可以通过类名来访问，因此不需要实例化。

（2）局部变量。

局部变量是指在方法或者方法代码块中定义的变量，其作用域是其所在的代码块。可分为以下三种：

方法参数变量（形参）：在整个方法内有效。

方法局部变量（方法内定义）： 从定义这个变量开始到方法结束这一段时间内有效。

代码块局部变量（代码块内定义）：从定义这个变量开始到代码块结束这一段时间内有效，常用于 try…catch 代码块中。

2.1.3 运算符和表达式

程序是由许多语句组成的，而语句的基本单位就是表达式与运算符。表达式是由操作数与运算符组成的。操作数可以是常量、变量，也可以是方法，而运算符就是数学中的运算符号，如“+”“-”“*”“/”“%”等。

1. 算术运算符

常用的算术运算符及含义如表 2-3 所示。

表 2-3 常用的算术运算符及含义

操 作 符	含 义
+	加法，把运算符两侧的值相加，即 a+b
-	减法，用左操作数减去右操作数，即 a-b
*	乘法，把操作符两侧的值相乘，即 a*b
/	除法，用左操作数除以右操作数，即 a/b
%	取余，左操作数除以右操作数的余数，即 a%b
++	自增，操作数的值增加 1，即 a++
--	自减，操作数的值减少 1，即 a--

2. 关系运算符

关系运算符也称比较运算符，是指对两个操作数进行关系运算的运算符，主要用于确定两个操作数之间的关系。常用的关系运算符及含义如表 2-4 所示。

表 2-4 常用的关系运算符及含义

操 作 符	含 义
==	检查两个操作数的值是否相等，如果相等即 a=b，则条件为真
!=	检查两个操作数的值是否相等，如果值不相等即 a!=b，则条件为真

续表

操　作　符	含　　义
>	检查左操作数的值是否大于右操作数的值，如果大于即 a>b，则条件为真
<	检查左操作数的值是否小于右操作数的值，如果小于即 a<b，则条件为真
>=	检查左操作数的值是否大于或等于右操作数的值，如果 a>=b，则条件为真
<=	检查左操作数的值是否小于或等于右操作数的值，如果 a<=b，则条件为真

3. 逻辑运算符

逻辑运算符用来把各个运算的变量连接起来，组成一个逻辑表达式，判断编程中某个表达式是否成立，判断的结果是 true 或 false。常用的逻辑运算符及含义如表 2-5 所示。

表 2-5　常用的逻辑运算符及含义

操　作　符	含　　义
&&	逻辑与运算符，当且仅当两个操作数都为真，条件才为真，即 a&&b
||	逻辑或操作符，如果两个操作数中的任何一个数为真，则条件为真，即 a||b
!	逻辑非运算符，反转操作数的逻辑状态。如果条件为 true，则逻辑非运算符将得到 false，即!（a&&b）

4. 赋值运算符

赋值运算符就是为各种不同类型的变量赋值，简单的赋值运算符由等号（=）来实现，即是把等号右边的值赋给等号左边的变量。常用的赋值运算符及含义如表 2-6 所示。

表 2-6　常用的赋值运算符及含义

操　作　符	含　　义
=	简单的赋值运算符，将右操作数的值赋给左操作数，即 c=a+b
+=	加和赋值操作符，把左操作数和右操作数相加并赋值给左操作数，即 c += a 等价于 c =c+a
-=	减和赋值操作符，把左操作数和右操作数相减并赋值给左操作数，即 c-=a 等价于 c=c-a
*=	乘和赋值操作符，把左操作数和右操作数相乘并赋值给左操作数，即 c*=a 等价于 c=c*a
/=	除和赋值操作符，把左操作数和右操作数相除并赋值给左操作数，即 c/=a 等价于 c=c/a
%=	取模和赋值操作符，把左操作数和右操作数取模后赋值给左操作数，即 c%=a 等价于 c=c%a

5. 位运算符

位运算符主要用来对操作数为二进制的位进行运算，按位运算表示按每个二进制位来进行运算，其操作数的类型是整数类型以及字符型，运算的结果是整数类型。常用的位运算符及含义如表 2-7 所示。

表 2-7　常用的位运算符及含义

操　作　符	含　　义
<<=	左移位赋值运算符
>>=	右移位赋值运算符
&=	按位与赋值运算符
^=	按位异或赋值操作符
|=	按位或赋值操作符

2.1.4　流程控制语句

1. 顺序语句

顺序语句的执行顺序是自上而下，依次执行。

2. 条件语句

1）if 语句

```
if(条件表达式){
    条件表达式成立时执行该语句;
}
```

如果条件表达式的值为 true，则执行 if 语句中的代码块，否则执行 if 语句块后面的代码。

2）if…else 语句

```
if(条件表达式){
条件表达式成立时执行该语句;
}else{
条件表达式不成立时执行该语句;
}
```

3）if 嵌套语句

```
if(条件表达式 1){
   if(条件表达式 2){
     语句 1;
          }else{
     语句 2;
    }
          }else{
     语句 3;
    }
```

3. 选择语句

switch 语句判断一个变量与一系列值中某个值是否相等，每个值称为一个分支。

```
switch(表达式){
    case "表达式的结果 1":
        语句 1;
    break;
    case "表达式的结果 2":
        语句 2;
    break;
    default:
        语句 3;
    break;
 }
```

（1）switch 语句中的变量类型可以是 byte、short、int 或者 char。

（2）switch 语句可以有多个 case 语句。每个 case 后面跟一个要比较的值和冒号。

（3）case 语句中值的数据类型必须与变量的数据类型相同，而且只能是常量或者字面常量。

（4）当变量的值与 case 语句的值相等时，case 语句之后的语句开始执行，直到 break 语句出现才会跳出 switch 语句。

（5）当出现 break 语句时，switch 语句中止。程序跳转到 switch 语句后面的语句执行。case 语句不包含 break 语句。如果没有 break 语句出现，则程序会继续执行下一条 case 语句，直到出现 break 语句为止。

（6）switch 语句可以包含一个 default 分支，该分支必须是 switch 语句的最后一个分支。default 在没有 case 语句的值和变量值相等的时候执行。default 分支不需要 break 语句。

4. 循环语句

1）while 语句

while 语句的执行过程是先计算表达式的值，若表达式的值为真（非零），则执行循环体中的语句，继续循环；否则退出该循环，执行 while 语句后面的语句。循环体可以是一条语句或空语句，也可以是复合语句。

```
while(循环条件){
    循环体;
}
```

2）do…while 语句

```
do{
    循环体;
}while(循环条件)
```

☆**注意**☆　while 语句属于先判断后执行，而 do…while 语句先执行一次，而后再进行判断。do…while 循环和 while 循环能实现同样的功能。然而在程序运行过程中，这两种语句还是有差别的。如果循环条件在循环语句开始时就不成立，那么 while 循环的循环体一次都不会执行，而 do…while 循环的循环体还是会执行一次。

3）for 语句

```
for (初值; 判断条件; 赋值增减量)
{
    语句 1;
    …
    语句 n;
}
```

for 关键字后面()中包括了三部分内容：初始化表达式、循环条件和操作表达式。它们之间用“;”分隔，{}中的执行语句为循环体。

（1）最先执行初始化步骤。可以声明一种类型、初始化一个或多个循环控制变量，也可以是空语句。

（2）判断条件。如果为 true，则循环体被执行。如果为 false，则循环中止，开始执行循环体后面的语句。

（3）执行一次循环后，更新循环控制变量。

（4）再次检测判断条件。循环执行上面的步骤。

2.2 面向对象

Java 是一种面向对象的程序设计语言，了解面向对象的编程思想对于学习 Java 开发尤其重要。面向对象技术是一种将数据抽象和信息隐藏的技术，它使软件的开发更加简单化，不仅符合人们的思维习惯，而且降低了软件的复杂性，同时提高了软件的生产效率，因此得到了广泛的应用。

2.2.1 面向对象的三大特性

几乎所有面向对象的编程设计语言都离不开封装、继承和多态。

1. 封装

面向对象的核心思想就是封装。封装是指将对象的属性和行为进行包装，不需要让外界知道具体实现的细节。封装可以使数据的安全性得到保证，当把过程和数据封装后，只能通过已定义的接口对数据进行访问。

（1）属性：Java 中类的属性的访问权限的默认值不是 private，通过加 private（私有）修饰符来隐藏该属性或方法，从而只能在类的内部进行访问。对于类中的私有属性，要对其给出方法（如 getXxx()、setXxx()）来访问私有属性，保证对私有属性操作的安全性。

（2）方法的封装：对于方法的封装，既需要公开也需要隐藏。方法公开的是方法的声明（定义），只需要知道参数和返回值就可以调用该方法；隐藏方法的实现会使实现的改变对架构的影响最小化。

（3）封装的优点：良好的封装能够减少耦合；类内部的结构可以自由修改；可以对成员变量进行更精确的控制；隐藏信息，实现细节。

2. 继承

继承主要指的是类与类之间的关系。通过继承，可以效率更高地对原有类的功能进行扩展。继承不仅增强了代码的复用性，提高了开发效率，更为程序的修改、补充提供了便利。

Java 中的继承要使用 extends 关键字，并且 Java 中只允许单继承，即一个类只能有一个父类。这样的继承关系呈树状，体现了 Java 的简单性。子类只能继承在父类中可以访问的属性和方法，实际上父类中私有的属性和方法也会被继承，只是子类无法访问。

子类并不能全部继承父类的成员变量或成员方法，规则如下：

（1）能够继承父类的 public 和 protected 成员变量（方法），但不能继承父类的 private 成员变量（方法）。

（2）对于父类的包访问权限成员变量（方法），如果子类和父类在同一个包下，则子类能够继承；否则，子类不能够继承。

（3）对于子类可以继承父类型的成员变量（方法），如果在子类中出现了同名称的成员变量（方法），则会发生隐藏现象，即子类的成员变量（方法）会屏蔽掉父类的同名成员变量（方法）。如果要在子类中访问父类中同名成员变量（方法），则需要使用 super 关键字来进行引用。

3. 多态

多态是同一个行为具有多个不同表现形式或形态的能力。

多态是把子类型对象主观地看作是其父类型的对象，因此其父类型就可以是很多种类型。

多态的特性：对象实例确定则不可改变（客观不可改变）；只能调用编译时的类型所定义的方法；运行时会根据运行时的类型去调用相应类型中定义的方法。

2.2.2　类和对象

类是一个模板，它描述一类对象的行为和状态。

对象是类的一个实例，有状态和行为。

1. 类

1）类的声明

在使用类之前，必须先声明。类的声明格式如下：

```
[标识符] class 类名称
{
    //类的成员变量
    //类的方法
}
```

- 声明类需要使用关键字 class，在 class 之后是类的名称。
- 标识符可以是 public、private、protected 或者完全省略。
- 类名应该是由一个或多个有意义的单词连缀而成，每个单词首字母大写，单词之间不要使用其他分隔符。

2）类的方法

类的方法有四个要素，分别是方法名称、返回值类型、参数和方法体。定义一个方法的语法格式如下：

```
修饰符 返回值类型 方法名称（参数列表）
{
    //方法体
return 返回值;
}
```

方法包含一个方法头和一个方法体。方法头包括修饰符、返回值类型、方法名称和参数列表。

- 修饰符：定义了该方法的访问类型，是可选的。
- 返回值类型：指定了方法返回的数据类型。它可以是任意有效的类型，如果方法没有返回值，则其返回类型必须是 void，不能省略。方法体中的返回值类型要与方法头中定义的返回值类型一致。
- 方法名称：要遵循 Java 标识符命名规范，通常以英文中的动词开头。
- 参数列表：由类型、标识符组成，每个参数之间使用逗号分隔开。方法可以没有参数，但方法名称后面的括号不能省略。
- 方法体：指方法头后{}内的内容，主要用来实现一定的功能。

2. 对象

对象是根据类创建的。在 Java 中，使用关键字 new 来创建一个新的对象。创建对象的过程如下：

（1）声明：声明一个对象，包括对象名称和对象类型。

（2）实例化：使用关键字 new 来创建一个对象。

（3）初始化：使用 new 创建对象时，会调用构造方法初始化对象。

对象（object）是对类的实例化。在 Java 的世界里，“一切皆为对象”，面向对象的核心就是对象。由类产生对象的格式如下：

```
类名 对象名= new 类名();
```

访问对象的成员变量或者方法的格式如下：

```
对象名称.属性名
对象名称.方法名()
```

3. 构造方法

在创建类的对象时，对类中的所有成员变量都要初始化。Java 允许对象在创建时进行初始化，初始化的实现是通过构造方法来完成的。

在创建类的对象时，使用 new 关键字和一个与类名相同的方法来完成，该方法在实例化过程中被调用，成为构造方法。构造方法是一种特殊的成员方法，主要特点如下：

（1）构造方法的名称必须与类的名称完全相同。

（2）构造方法不返回任何数据类型，也不需要使用 void 关键字声明。

（3）构造方法的作用是创建对象并初始化成员变量。

（4）在创建对象时，系统会自动调用类的构造方法。

（5）构造方法一般用 public 关键字声明。

（6）每个类至少有一个构造方法。如果不定义构造方法，Java 将提供一个默认的不带参数且方法体为空的构造方法。

☆**注意**☆ 类是对某一类事务的描述，是抽象的、概念上的定义，对象是实际存在的该类事务的个体。对象和对象之间可以不同，改变其中一个对象的某些属性，不会影响到其他的对象。

2.2.3 抽象类和抽象方法

在面向对象中，所有的对象都是通过类来实现的，但是反过来，并不是所有的类都是用来描绘对象的。若一个类中没有包含足够的信息来描绘一个具体的对象，这样的类就是抽象类。抽象方法指一些只有方法声明，而没有具体方法体的方法。抽象方法一般存在于抽象类或接口中。

1. 抽象类

1）抽象类的使用原则

（1）抽象方法必须为 public 或者 protected，默认为 public。

（2）抽象类不能直接实例化，需要依靠子类采用向上转型的方式处理。

（3）抽象类必须有子类，使用 extends 继承，一个子类只能继承一个抽象类。

（4）子类如果不是抽象类，则必须重写抽象类之中的全部抽象方法。

（5）抽象类不能使用 final 关键字声明，因为抽象类必须有子类，而 final 定义的类不能有子类。

2）抽象类在应用的过程中需要注意的事项

（1）抽象类不能被实例化，如果被实例化，就会报错，编译无法通过。只有抽象类的非抽

象子类可以创建对象。

（2）抽象类中不一定包含抽象方法，但是有抽象方法的类必定是抽象类。

（3）抽象类中的抽象方法只是声明，不包含方法体，就是不给出方法的具体实现也就是不给出方法的具体功能。

（4）构造方法、类方法（用 static 修饰的方法）不能声明为抽象方法。

（5）抽象类的子类必须给出抽象类中的抽象方法的具体实现，除非该子类也是抽象类。

2. 抽象方法

1）抽象方法的声明

声明一个抽象方法的语法格式如下：

```
abstract 返回类型 方法名([参数表]);
```

☆**注意**☆　抽象方法没有定义方法体，方法名后面直接跟一个分号，而不是花括号。

2）抽象方法的实现

继承抽象类的子类必须重写父类的抽象方法，否则，该子类也必须声明为抽象类。最终，必须有子类实现父类的抽象方法，否则，从最初的父类到最终的子类都不能用来实例化对象。

2.2.4　接口

接口在 Java 编程语言中是一个抽象类型，是抽象方法的集合，接口通常以 interface 来声明。一个类通过继承接口的方式来继承接口的抽象方法。

1. 接口的声明

```
interface 接口名称 [extends 其他的接口名] {
    //声明变量
    //抽象方法
}
```

2. 接口的实现

当类实现接口时，类要实现接口中所有的方法。否则，类必须声明为抽象类。类使用 implements 关键字实现接口。在类声明中，implements 关键字放在 class 声明后面。

```
class 类名称 implements 接口名称[, 其他接口]{
…
}
```

3. 接口与抽象类的异同

1）相同点

（1）都可以被继承。

（2）都不能被直接实例化。

（3）都可以包含抽象方法。

（4）派生类必须实现未实现的方法。

2）不同点

（1）接口支持多继承；抽象类不能实现多继承。

（2）一个类只能继承一个抽象类，而一个类却可以实现多个接口。

（3）接口中的成员变量只能是 public、static、final 类型的；抽象类中的成员变量可以是各

种类型的。

（4）接口只能定义抽象方法；抽象类既可以定义抽象方法，也可以定义实现的方法。

（5）接口中不能含有静态代码块以及静态方法（用 static 修饰的方法）；抽象类可以有静态代码块和静态方法。

2.3 精选面试、笔试题解析

根据前面介绍的 Java 基础知识，本节总结了一些在面试或笔试过程中经常遇到的问题。通过本节的学习，读者将掌握在面试或笔试过程中回答问题的方法。

2.3.1 Java 基本数据类型之间如何转换

题面解析：本题主要考查应聘者对基本数据类型的熟练掌握程度。看到此问题，应聘者需要把关于数据类型的所有知识在脑海中回忆一下，其中包括基本数据类型有哪些、数据类型的作用等，熟悉了数据类型的基本知识之后，数据类型之间的转换问题将迎刃而解。

解析过程：

数据类型之间的转换有两种方式：自动转换和强制转换。

1. 自动转换

自动转换规则：从存储范围小的类型转换到存储范围大的类型，即 byte→short（char）→int→long→float→double。

（1）存储范围小的类型自动转换为存储范围大的类型。如 byte 类型的数据可以赋值给 short、int、long 类型的变量；short、char 类型的数据可以赋值给 int、long 类型的变量；int 类型的数据可以赋值给 long 类型的变量等。

（2）存储范围大的类型转换为存储范围小的类型时，需要加强制转换符。

（3）byte、short、char 之间不会互相转换，并且三者在计算时首先转换为 int 类型。

（4）实数常量默认为 double 类型，整数常量默认为 int 类型。

2. 强制转换

强制转换规则：从存储范围大的类型到存储范围小的类型，即 double→float→long→int→short（char）→byte。

语法格式：

```
（type）value
```

其中，type 是要强制类型转换后的数据类型。

2.3.2 谈谈你对面向对象的理解

题面解析：本题是对面向对象知识点的考查，应聘者在回答该问题时，不能照着定义直接背出来，而是要阐述自己对面向对象概念的理解，另外，还要解释关于面向对象更深一层的含义。

解析过程：

在解释面向对象之前，先介绍一下什么是对象。

在 Java 语言中，把对象当作一种变量，它不仅可以存储数据，还可以对自身进行操作。每个对象都有各自的属性及方法，Java 就是通过对象之间行为的交互来解决问题的。

在我看来，面向对象就是把构成问题的所有事务分解成一个个的对象，建立这些对象去描述某个事务在解决问题中的行为。而类就是面向对象中很重要的一部分，类是很多个具有相同属性和行为特征的对象所抽象出来的，对象是类的一个实例。

类还具有三个特性，即封装、继承和多态。

（1）封装：将一类事务的属性和行为抽象成一个类，只提供符合开发者意愿的公有方法来访问这些数据和逻辑，在提高数据的隐秘性的同时，使代码模块化。

（2）继承：子类可以继承父类的属性和方法，并对其进行拓展。

（3）多态：同一种类型的对象执行同一个方法时可以表现出不同的行为特征。通过继承的上下转型、接口的回调以及方法的重写和重载可以实现多态。

2.3.3　Java 中的访问修饰符有哪些

题面解析：本题主要考查应聘者对修饰符的掌握程度，知道访问修饰符有哪些以及它们的使用范围和区别等。

解析过程：

Java 中有四种访问修饰符，分别为 public、private、protected 和 default。

（1）public：公有的。用 public 修饰的类、属性及方法，不仅可以跨类访问，而且允许跨包（package）访问。

（2）private：私有的。用 private 修饰的类、属性以及方法只能被该类的对象访问，其子类不能访问，更不允许跨包访问。

（3）protected：介于 public 和 private 之间的一种访问修饰符。用 protected 修饰的类、属性以及方法只能被类本身的方法及子类访问，即使子类在不同的包中也可以访问。

（4）default：默认访问模式。在该模式下，只允许在同一个包中进行访问。

☆**注意**☆　protected 修饰符所修饰的类属于成员变量和方法，只可以被子类访问，而不管子类是不是和父类位于同一个包中。default 修饰符所修饰的类也属于成员变量和方法，但只可被同一个包中的其他类访问，而不管其他类是不是该类的子类。protected 属于子类限制修饰符，而 default 属于包限制修饰符。

2.3.4　重载和重写

试题题面：什么是方法的重载和重写？它俩之间有什么区别？

题面解析：本题属于对概念类知识的考查，在解题的过程中需要先解释方法重载和重写的概念，然后介绍各自的特点，最后再分析方法重载和重写之间的区别。

解析过程：

1. 方法重载

构成方法重载的必要条件：定义在同一个类中，方法名相同，参数的个数、顺序、类型不同构成重载。

方法重载的目的：解决参数的个数、类型、顺序不一致，但功能一致、方法名一致的重名问题的情况。

方法重载的特点有以下几点：

（1）发生在同一个类中。

（2）方法名称相同（参数列表不同）。

（3）参数的个数、顺序、类型不同。

（4）和返回值类型以及访问权限修饰符、异常声明没有关系。

（5）重载是多态的一种表现形式。

（6）重载的精确性原则，就是赋给变量值的时候要按照变量的规则赋值。

2. 方法重写

如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程称为方法的重写。

方法重写的目的：父类的功能实现无法满足子类的需求，需要重写。

方法重写的特点：

（1）发生在具有子父类两个关系的类中。

（2）方法名称相同。

（3）参数的列表完全相同。

（4）返回值类型可以相同或者是其子类。

（5）访问权限修饰符不能够严于父类。

（6）重写是多态的必要条件。

（7）抛出的异常不能比父类的异常大。

（8）私有修饰的方法不能够被继承，就更不可能被重写。

（9）构造方法不能被重写。

2.3.5 什么是构造方法

题面解析：本题主要考查应聘者对 Java 中构造方法的理解，因此应聘者不仅需要知道什么是构造方法、构造方法有哪些特点，而且还要知道怎样使用构造方法。

解析过程：

构造方法是指定义在 Java 类中的用来初始化对象的方法。通常使用“new+构造方法”的方式来创建新的对象，还可以给对象中的实例进行赋值。

1. 构造方法的语法规则

（1）方法名必须与类名相同。

（2）无返回值类型，不能使用 void 进行修饰。

（3）可以指定参数，也可以不指定参数；分为有参构造方法和无参构造方法。

例如，调用构造方法：

```
Student s1;
s1 = new Student();//调用构造方法
```

2. 构造方法的特点

（1）当没有指定构造方法时，系统会自动添加无参的构造方法。

（2）构造方法可以重载：方法名相同，但参数不同的多个方法，调用时会自动根据不同的参数选择相应的方法。

（3）构造方法是不被继承的。

（4）当手动指定了构造方法时，无论是有参的还是无参的，系统都将不会再添加无参的构造方法。

（5）构造方法不但可以给对象的属性赋值，还可以保证给对象的属性赋一个合理的值。

（6）构造方法不能被 static、final、synchronized、abstract 和 native 修饰。

2.3.6　局部变量与成员变量有什么区别

题面解析：本题主要考查局部变量和成员变量的区别，应聘者需要掌握变量的基础知识，包括什么是变量、什么是常量、变量的命名规则以及它们之间的区别等内容。看到问题时，应聘者脑海中要快速想到关于变量的各个知识点，以至于能够快速、准确地回答出该问题。

解析过程：

局部变量是指在方法或者方法代码块中定义的变量，其作用域是其所在的代码块。

成员变量是指在类的体系结构的变量部分中定义的变量。

局部变量和成员变量的区别：

（1）定义的位置。

局部变量：定义在方法的内部。

成员变量：定义在方法的外部，即直接写在类中。

（2）作用范围。

局部变量：只适用于方法中，描述类的公共属性。

成员变量：整个类中都可以通用。

（3）默认值（初始化）。

局部变量：没有默认初始值，需要手动进行赋值之后才能使用。

成员变量：有默认初始值，如 int 类型的默认值为 0；float 类型的默认值为 0.0f；double 类型的默认值为 0.0。

（4）内存的位置。

局部变量：位于栈内存。

成员变量：位于堆内存。

（5）生命周期。

局部变量：在调用对应的方法时，局部变量因为执行创建语句而存在，超出自己的作用域之后会立即从内存消失。

成员变量：成员变量随着对象的创建而创建，随着对象的消失而消失。

2.3.7 解释一下 break、continue 以及 return 的区别

题面解析：本题是在笔试中出现频率较高的一道题，主要考查应聘者是否掌握循环控制语句的使用。在解答本题之前需要知道 break、continue 和 return 的用法，经过对比，进而就能够很好地回答本题。

解析过程：

1. break

break 用于完全结束一个循环，跳出循环体。无论是哪种循环，只要在循环体中有 break 出现，系统会立刻结束循环，开始执行循环之后的代码。

break 不仅可以结束其所在的循环，还可结束其外层循环。在结束外层循环时，需要在 break 后加一个标签，这个标签用于标识外层循环。Java 中的标签就是一个紧跟着英文冒号（:）的标识符，且必须把它放在循环语句之前才有作用。例如：

```
for (int i = 0 ; i < 10 ; i++ ){
     //内层循环
     for (int j = 0; j < 5 ; j++ ){
          System.out.println("i 的值为:" + i + " j 的值为:" + j);
     if (j == 1){
     //跳出 outer 标签所标识的循环
          break outer;
        }
     }
}
```

2. continue

continue 用于终止本次循环，继续开始下次循环。continue 后的循环体中的语句不会继续执行，下次循环和循环体外面的语句都会执行。

continue 的功能和 break 有相似的地方，但区别是 continue 只是终止本次循环，接着开始下一次循环，而 break 则是完全中止循环。例如：

```
//简单的 for 循环
for (int i = 0; i < 5 ; i++ ){
          System.out.println("i 的值是" + i);
     if (i == 2){
        //忽略本次循环的剩下语句
        continue;
      }
        System.out.println("continue 后的输出语句");
}
```

3. return

return 并不是用于跳出循环，而是结束一个方法。如果在循环体内的一个方法内出现 return 语句，则 return 语句将会结束该方法，紧跟着循环也就结束。与 continue 和 break 不同的是，return 将直接结束整个方法，不管这个 return 处于多少层循环之内。例如：

```
for (int i = 0; i < 5 ; i++ ){
     System.out.println("i 的值是" + i);
     if (i == 2){
       return;
     }
```

```
        System.out.println("return 后的输出语句");
    }
```

2.3.8　Java 中的基本数据类型有哪些

题面解析：本题通常出现在面试中，考官提问该问题主要是想考查应聘者对基本数据类型的熟悉程度。数据类型是 Java 最基础的知识，只有掌握了基础知识，才能在以后的开发工作中应用自如。

解析过程：

Java 中的基本数据类型分为整数类型、浮点数类型、字符类型和布尔类型四种。

1. 整数类型

1）byte

byte 是数据类型为 8 位、有符号、以二进制补码表示的整数，用于表示最小数据单位；取值范围为-2^7～2^7-1，其中默认值为 0。

2）short

short 是数据类型为 16 位、有符号、以二进制补码表示的整数；取值范围为-2^{15}～$2^{15}-1$，其中默认值为 0。

3）int

int 是数据类型为 32 位、有符号、以二进制补码表示的整数；取值范围为-2^{31}～$2^{31}-1$，其中默认值为 0；一般整型变量默认为 int 类型。

4）long

long 是数据类型为 64 位、有符号、以二进制补码表示的整数；取值范围为-2^{63}～$2^{63}-1$，其中默认值为 0L；long 主要使用在需要比较大整数的系统上。

2. 浮点数类型

1）float

float 是数据类型为单精度、32 位、符合 IEEE 754 标准的浮点数，其中默认值为 0.0f。

浮点数不能用来表示精确的值。

2）double

double 是数据类型为双精度、64 位、符合 IEEE 754 标准的浮点数，其中默认值为 0.0d；浮点数的默认类型为 double 类型，double 类型同样不能表示精确的值。

3. 字符类型

字符类型是一个单一的 16 位的 Unicode 字符；取值范围为\u0000（0）～\uffff（65535）。

char 数据类型可以存储任何字符，但需要注意不能为 0 个字符。

4. 布尔类型

布尔（boolean）数据类型表示一位的信息；boolean 数据类型只有 true 和 false 两个值，只作为一种标志来记录 true/false 的情况，其中默认值为 false。

2.3.9 Java 中 this 的用法

题面解析：本题不仅会出现在笔试中，而且在以后的开发过程中也会经常遇到。因此掌握 this 的用法是非常重要的。

解析过程：

this 在类中代表当前对象，可以通过 this 关键字完成当前对象的成员属性、成员方法和构造方法的调用。

Java 的关键字 this 只能用于方法体内。当一个对象创建后，Java 虚拟机就会给这个对象分配一个引用自身的指针，这个指针的名字就是 this。因此，this 只能在类中的非静态方法中使用，静态方法和静态的代码块中绝对不能出现 this，并且 this 只和特定的对象关联，而不和类关联，同一个类的不同对象有不同的 this。

那么什么时候使用 this 呢？

当在定义类中的方法时，如果需要调用该类对象，就可以用 this 来表示这个对象。

this 的作用：

（1）表示对当前对象的引用。

（2）表示用类的成员变量，而非函数参数。

（3）用于在构造方法中引用满足指定参数类型的构造方法，只能引用一个构造方法且必须位于开始的位置。

2.3.10 接口和抽象类

试题题面：接口是否可以继承接口？抽象类是否可实现接口?抽象类是否可继承实体类？

题面解析：本题属于在笔试中高频出现的问题之一，主要考查关于接口和抽象类的知识点，在解答本题之前需要了解什么是接口、什么是抽象类、什么是抽象方法，同时还需要把接口和抽象类区分开来，以防混淆。

解析过程：

- 接口。接口属于一种约束形式，只包括成员定义，不包含成员实现的内容。
- 抽象类。抽象类主要是针对看上去不同但是本质上相同的具体概念的抽象。抽象类不能用来实例化对象，声明抽象类的唯一目的是将来对该类进行扩充。一个类不能同时被 abstract 和 final 修饰。如果一个类包含抽象方法，那么该类一定要声明为抽象类，否则将出现编译错误。
- 抽象方法。抽象方法是指一些只有方法声明而没有具体方法体的方法。抽象方法一般存在于抽象或接口中。抽象方法不能被声明成 final 和 static；任何继承抽象类的子类必须实现父类的所有抽象方法，除非该子类也是抽象类；如果一个类包含若干个抽象方法，那么该类必须声明为抽象类，但抽象类可以不包含抽象方法；抽象方法的声明以分号结尾。

（1）接口可以继承（extends）接口。通过关键字 extends 声明一个接口是另一个接口的子接口。由于接口中的方法和常量都是 public，子接口将继承父接口中的全部方法和常量。例如：

```
public interface InterfaceA{
}
interface InterfaceB extends InterfaceA{
```

```
}
```

（2）抽象类可以实现（implements）接口。当一个类声明实现一个接口而没有实现接口中所有的方法，那么这个必须是抽象类，即 abstract 类。例如：

```
public interface InterfaceA{
}
abstract class TestA implements InterfaceA{
}
```

（3）抽象类可继承（extends）实体类，但前提是实体类必须有明确的构造函数。例如：

```
public class TestA{
}
abstract class TestB extends TestA{
}
```

2.4　名企真题解析

接下来，我们收集了一些各大企业往年的面试及笔试题，读者可以根据以下题目来作参考，看自己是否已经掌握了基本的知识点。

2.4.1　值传递和引用传递

【选自 WR 笔试题】

试题题面：当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是按值传递还是按引用传递？

题面解析：本题题目比较长，有些读者可能觉着回答很费劲。其实可以换一种方式来想该问题，即 Java 中是按值传递还是引用传递？本题的重点是在最后的按值传递还是按引用传递。接下来详细讲解按值传递和按引用传递。

解析过程：

先来讲解一下什么是值传递和引用传递。

（1）值传递：在方法调用时，实际参数（即实参）把它的值传递给对应的形式参数（即形参），方法执行中，对形式参数值的改变不影响实际参数的值。

按值传递就是将一个参数传递给一个函数时，函数接收的是原始值的一个副本。因此，如果函数修改了该参数，仅改变副本，而原始值保持不变。

（2）引用传递：也称为传地址。方法调用时，实际参数的引用被传递给方法中相对应的形式参数，在方法执行中，对形式参数的操作实际上就是对实际参数的操作，方法执行中形式参数值的改变将会影响实际参数的值。

按引用传递是当将一个参数传递给一个函数时，函数接收的是原始值的内存地址，而不是值的副本。因此，如果函数修改了该参数的值，调用代码中的原始值也随之改变。如果函数修改了该参数的地址，调用代码中的原始值不会改变。

在 Java 中只有值传递参数。

（1）当一个对象实例作为一个参数被传递到方法中时，参数的值就是该对象引用的一个副本。对象的内容可以在被调用的方法中改变，但对象的引用是不会发生改变的。

Java 中没有指针，因此没有引用传递。但可以通过创建对象的方式来实现引用传递。

（2）在 Java 中只会传递对象的引用，按引用传递对象。

（3）在 Java 中按引用传递对象并不意味着会按引用传递参数。参数可以是对象引用，而 Java 是按值传递对象引用的。

（4）Java 中的变量可以为引用类型和基本类型。当作为参数传递给一个方法时，处理这两种类型的方式是相同的，两种类型都是按值传递。

2.4.2 什么是类的反射机制

【选自 GG 面试题】

题面解析：本题主要考查 Java 中的反射机制，我们需要知道什么是反射机制、反射机制的功能都有哪些，另外就是怎样运用反射机制来创建类的对象等。全面地了解该问题所涉及的知识，回答问题会更加容易。

解析过程：

反射机制是 Java 语言中的一个重要的特性，反射机制不仅允许程序在运行时进行自我检查，而且还允许对其内部的成员进行操作。由于反射机制在运行时能够实现对类的装载，因此能够提高程序的灵活性，但是如果使用反射机制的方法不当，则可能会严重影响系统的性能。

反射机制提供的功能如下：

（1）得到一个对象所属的类。

（2）获取一个类的所有成员变量和方法。

（3）在运行时创建对象。

（4）在运行时调用对象的方法。

反射机制最重要的一个作用就是可以在运行时动态地创建类的对象，其中 Class 类是反射机制中最重要的类。

获取 Class 类的方法如下：

```
(1) Class class1 = Class.forName("com.reflection.User");
(2) Class class2 = User.class;
(3) User user = new User();
    Class class3 = user.getClass();
```

获取对象实例的方法如下：

```
(1) user1 = (User)class1.newInstance();
    user1.setName("a");
    user1.setAge("15");
(2) Constructor constructor = class2.getConstructor(String.class, Integer.class);
    user2 = (User)constructor.newInstance("b", 11);
```

2.4.3 Java 创建对象的方式有哪几种

【选自 BD 面试题】

题面解析：本题也是在大型企业的面试中最常问的问题之一，主要考查创建对象的方式。

解析过程：

共有四种创建对象的方法。

（1）通过 new 语句实例化一个对象。

使用 new 关键字创建对象是最常见的一种方式，但是使用 new 创建对象会增加耦合度。在使用 new 时需要先查看 new 后面的类型，然后再决定分配多大的内存空间；接着可以通过调用构造函数，来对对象的各个域进行填充；根据构造方法的返回值进行对象的创建，最后把引用地址传递给外部。例如：

```
package test;
/**使用 new 关键字创建对象*/
public class NewClass
{
   public static void main(String[] args)
   {
      Hello h = new Hello();
      h.sayWorld();
   }
}
```

（2）通过反射机制创建对象。

使用反射机制的 Class 类的 newInstance()方法。

（3）通过 clone()方法创建一个对象。

在使用 clone()方法时，不会调用构造函数，而是需要有一个分配了内存的源对象。在创建新对象时，首先应该分配一个和源对象一样大的内存空间。

（4）通过反序列化的方式创建对象。

序列化就是把对象通过流的方式存储到文件里面，那么反序列化就是把字节内容读出来并还原成 Java 对象，这里还原的过程就是反序列化。在使用反序列化时也不会调用构造方法。