《Android 系统下 Java 编程详解》

作者：华清远见

第 3 章 标识符、关键字与数据类型

本章简介

本章主要介绍 Java 中使用注释的方法和使用规则，包括特殊注释：javadoc 注释的使用方法；分号，空格等分隔符； Java 中标识符的定义规则； Java 中的数据类型；简单类型间的数据类型转换；分析了对象的构造和初始化；局部变量、全局变量的定义、声明及其作用范围； Java 中通过值传递方式传递参数；最后介绍了使用 Java 编写出优雅代码的编程一般规则。

专业始于专注 卓识源于远见

3.1 Java 注释

在编写程序时，为了说明某段代码的用途、某个方法的功能或者某个方法的参数，输入/输出值等的含义，需要在程序的关键部分加一些注释来说明。在各种编程语言中，都提供了各自的用于放置到程序代码中的注释语句，这些语句和程序语句混杂在一块，因此，需要一种特殊的机制让注释和代码不会在编译时发生冲突和混淆。比如，在 VB 中，用单引号“’”表示单行的注释等。而在 Java 中，也同样提供了用于注释。和其他语言相比较，Java 提供的注释方式更灵活、更多样、更强大。

在 Java 中，提供了 3 种注释方式：短（单行）注释、块（多行）注释及文档注释。单行和多行注释很容易理解，将注释符之间的内容当做注释，在编译和运行时将这部分内容忽略。

在 Java 中，比较特殊的是 javadoc 注释，包含在这部分中的注释可以通过 javadoc 命令来自动生成 API 文档。通过 javadoc 工具，可以保证程序代码和技术文档的同步。在修改了程序中的注释后，只需要通过 javadoc，就可以方便地生成相应的技术文档。下面介绍单行注释和多行注释的方法。

3.1.1 知识准备：Java 注释使用规则

（1）单行注释：单行注释就是在程序中注释一行代码。

注释规则：在代码中单起一行注释， 注释前最好有一行空行，并与其后的代码具有一样的缩进层级。如果单行无法完成，则应采用块注释。

注释格式：

```
// 注释内容
```

（2）多行注释：一次将程序中的多行注释掉。

注释规则：注释若干行，通常用于提供文件、方法、数据结构等的意义与用途的说明，或者算法的描述。一般位于一个文件或者一个方法的前面，起到引导的作用，也可以根据需要放在合适的位置。

注释格式：

```
/*
注释内容
*/
```

来看一个单行注释和多行注释的例子。

源文件：MessageComment.java

```
//这是一个单行注释
/*
 这是一个
 多行注释
 */
public class MessageComment {
 public static void main(String[] args) {
      System.out.println("发信息");
      // System.out.println("此条信息不会显示");
 }
}
```

3.1.2 知识准备：利用 javadoc 来产生 API 文档

我们知道，在软件开发过程中，文档编写的重要性不亚于程序代码本身。如果代码与文档是分离的，那么在每次修改代码时，都需要修改相应的文档就会是一件很麻烦的事情。 所以通过 javadoc 将代码同文档“连接”起来。在 Java 中，还有一种特别的注释方式：文档注释。利用这种注释，可以从 Java 源文件中提取这些注释的内容，来产生 HTML 格式的 API 文档。

文档注释的基本格式如下：

```
/**
```

```
文档注释内容
*/
```

注意把文档注释和多行注释区分清楚，文档注释的开始标记是“/**”，而多行注释标记的开始标记是“/*”。

由于文档注释最重要的一个功能就是用它来生成 HTML 格式的 API 文档，因此，很多用于 HTML 格式化的 HTML 标记也可以用在文档注释中，在从这些注释中提取注释生成 HTML 文件的时候，在生成的 HTML 文件中，将使用这些 HTML 标记来格式化 HTML 文件内容。常用的 HTML 标记有…、<code>…</code>等。关于这些 HTML 标记及其他的 HTML 标记，请读者参考相关的 HTML 资料。

和多行注释不同的另一个地方是，文档注释并不是可以放在 Java 代码的任何地方，javadoc 工具在从 Java 代码中提取注释生成 API 文档时，主要从以下几项内容中提取信息。

- 包。
- 公有（public）类与接口。
- 公有方法和受保护（protected）方法。
- 公有属性和受保护属性。

因此，文档注释也应该放到相应的位置中。

1．文档注释位置

（1）类注释。类注释用于说明整个类的功能、特性等，它应该放在所有的“import”语句之后，在 class 定义之前。

这个规则也适用于接口（interface）注释。

（2）方法注释。方法注释用来说明方法的定义，比如，方法的参数、返回值及说明方法的作用等。方法注释应该放在它所描述的方法定义前面。

（3）属性注释。默认情况下，javadoc 只对公有（public）属性和受保护属性（protected）产生文档——通常是静态常量。

（4）包注释。类、方法、属性的注释都直接放到 Java 的源文件中，而对于包的注释，无法放到 Java 文件中去，只能通过在包对应的目录中添加一个 package.html 的文件来达到这个目的。当生成 HTML 文件时，package.html 文件的<BODY>和</BODY>部分的内容将会被提取出来当做包的说明。关于包注释，后面还会有更进一步的解释。

（5）概要注释。除了包注释外，还有一种类型的文档无法从 Java 源文件中提取，就是对所有类文件提供概要说明的文件。同样的，也可以为这类注释单独新建一个 HTML 文件，这个文件的名字为“overview.html”，它的<BODY>和</BODY>标记之间的内容都会被提取。

2．javadoc 标记

在 javadoc 注释中，常用@来表示一个 javadoc 标记，常用的 javadoc 标记如下：

- @author：作者。
- @version：版本。
- @docroot：表示产生文档的根路径。
- @deprecated：不推荐使用的方法。
- @param：方法的参数类型。
- @return：方法的返回类型。
- @see：用于指定参考的内容。
- @exception：抛出的异常。
- @throws：抛出的异常，和 exception 同义。

需要注意这些标记的使用是有位置限制的。其中可以出现在类或者接口文档注释中的标记有：@see、@deprecated、@author、@version 等。可以出现在方法或者构造器文档注释中的标记有：@see、@deprecated、@param、@return、@throws、@exception 等。可以出现在属性文档注释中的有：@see、@deprecated 等。

3．javadoc 命令语法

javadoc 的命令行语法如下：

```
javadoc [ options ] [ packagenames ] [ sourcefiles ] [ @files ]
```

参数可以按照任意顺序排列。下面对这些参数作一些说明。

（1）packagenames 包列表：这个选项可以是一系列的包名（用空格隔开），例如，java.lang java.lang.reflect java.awt。因为 javadoc 不递归作用于子包，不允许对包名使用通配符；所以必须显式地列出希望建立文档的每一个包。

（2）sourcefiles 源文件列表。这个选项可以是一系列的源文件名（用空格隔开），可以使用通配符。javadoc 允许 4 种源文件：类源代码文件、包描述文件、总体概述文件、其他杂文件。

- 类源代码文件：类或者接口的源代码文件。
- 包描述文件：每一个包都可以有自己的包描述文件。包描述文件的名称必须是“package.html”，与包的.java 文件放置在一起。包描述文件的内容通常是使用 HTML 标记写的文档。javadoc 执行时将自动寻找包描述文件。如果找到，javadoc 将首先对描述文件中<body></body>之间的内容进行处理，然后把处理结果放到该包的 Package Summary 页面中，最后把包描述文件的第一句（紧靠<body>）放到输出的 Overview Summary 页面中，并在语句前面加上该包的包名。
- 总体概述文件：javadoc 可以创建一个总体概述文件描述整个应用或者所有包。总体概述文件可以被任意命名，也可以放置到任意位置。-overview 选项可以指示总体概述文件的路径和名称。总体概述文件的内容是使用 HTML 标记写的文档。javadoc 在执行的时候，如果发现-overview 选项，那么它将首先对文件中<body></body>之间的内容进行处理；然后把处理后的结果放到输出的 Overview Summary 页面的底部；最后把总体概述文件中的第一句放到输出的 Overview Summary 页面的顶部。
- 其他杂文件：这些文件通常是指与 javadoc 输出的 HTML 文件相关的一些图片文件、Java 源代码文件（.java）、Java 程序（.class）、Java 小程序（Applets）、HTML 文件。这些文件必须放在 doc-files 目录中。每一个包都可以有自己的 doc-files 目录。例如，你希望在 java.awt.Button 的 HTML 文档中使用一幅按钮的图片（Button.gif）。首先，必须把图片文件放到 java\awt\doc-files\中；然后在 Button.java 文件中加入以下注释：

```
/**
* This button looks like this:
* <img src="doc-files/Button.gif">
*/
```

- files 包含文件。为了简化 javadoc 命令，可以把需要建立文档的文件名和包名放在一个或多个文本文件中。例如，为了简化以下命令：

```
javadoc -d apidoc com.oristand.college com.oristand.school
```

可以建立一个名称为 mypackage.txt 的文件，其内容如下：

```
com.oristand.college
  com.oristand.school
```

然后执行以下命令即可：

```
javadoc -d apidoc @mypackage.txt
```

- options 命令行选项。

① public 只显示公共类及成员。

② protected 只显示受保护的和公共的类及成员。默认选项。

③ package 只显示包、受保护的和公共的类及成员。

④ private 显示所有类和成员。

-classpath classpathlist 指定 javadoc 查找“引用类”的路径。引用类是指带文档的类加上它们引用的任何类。javadoc 将搜索指定路径的所有子目录，classpathlist 可以包含多个路径（使用“;”隔开）。

一切就绪后，就可以使用 JDK 中的“javadoc”工具来生成相关的 API 文档了。

3.1.3 任务一：使用 javadoc 注释，生成 API 文档

1．任务描述

写一段代码，加入 javadoc 注释，并使用 javadoc 工具生成相关 API 文档。

2．技能要点

（1）添加 javadoc 注释。
（2）使用 javadoc 命令生成 API 文档。

3．任务实现过程

（1）编写一个 JavaDoc 类，声明变量，并加入 javadoc 注释。

源文件：JavaDoc.java

```
/**
 * javadoc 演示程序--<b>JavaDoc</b>
 *
 * @author Alex Wen
 * @version 1.0 2009/12/15
 */
public class JavaDoc {
  /**
   * 在 main()方法中使用的显示用字符串
   *
   * @see #main(java.lang.String[])
   */
  static String SDisplay;

  static String 变量;

  /**
   * 显示 JavaDoc
   *
   * @param args
   * 从命令行中输入字符串
   */
  public static void main(String args[]) {
       SDisplay = "Hello World ";
       变量 = "test";
       System.out.println(SDisplay + 变量);
  }
}
```

（2）用如下的 javadoc 命令来生成 API 文档：

```
javadoc -private -d doc -author -version JavaDoc.java
```

在控制台上将会列出正在生成的文件。
从图 3-1 中可以看出生成了哪些文件。

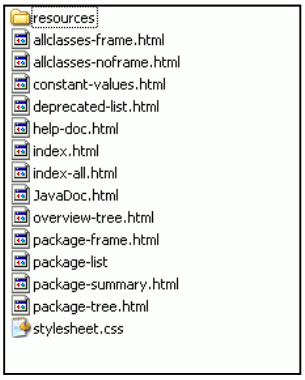

图 3-1　生成的 API 文档

（3）打开 index.html 文件（图 3-2 显示为打开的结果），因为没有包，所以，没有包列表文件。

图 3-2　javadoc 生成的 index.html 显示结果

3.2　分隔符和标识符

在 Java 中，有一类特殊的符号称为分隔符，包括空白分隔符和普通分隔符。

3.2.1　知识准备：空白分隔符

空白分隔符包括：空格、回车、换行和制表符 Tab 键。空白分隔符的主要作用是分隔标识符，帮助 Java 编译器理解源程序。例如：

```
int a;
```

若标识符 int 和 a 之间没有空格，即 inta，则编译程序会认为这是用户定义的标识符，但实际上该语句的作用是定义变量 a 为整型变量。

另外，在代码的编排时，适当的空格和缩进可以增强代码的可读性。看看下面 HelloAndroid.java 这段代码。

```
public class HelloAndroid{
  public static void main(String args[]){
       System.out.println("Hello Android!");
  }
}
```

在这个程序中，用到了大量的用于缩排的空格（主要是制表符和回车），如果不使用缩排空格，这个程序可能会是如下的模样。

```
public class HelloAndroid{
public static void main(String args[]){
System.out.println("Hello Android!");
}
}
```

相比较上一个程序，这个程序没有使用制表符来做缩排，显然在层次感上差了很多，甚至，还可能是如下情况。

```
public class HelloAndroid{public static void main(String args[]){System.out.println("Hello Android!");}}
```

这个程序可读性更差了：所有的语句都写在同一行上。在语法上，这个程序是正确的，但是，在可读性上，没有比这更差的了。因此，在写程序的时候，一定要灵活地使用空格来分隔语句或者做格式上的缩排等。但是，也要小心不要滥用它，所以使用空白分隔符要遵守以下规则：

- 任意两个相邻的标识符之间至少有一个分隔符，以便编译程序能够识别；变量名方法名等标识符不能包含空白分隔符。

- 空白分隔符的多少没有什么含义，一个空白符和多个空白符的作用相同，都是用来实现分割功能的。
- 空白分隔符不能用非普通分隔符替换。

3.2.2 知识准备：普通分隔符

普通分隔符具有确定的语法含义，有如下 7 种普通分隔符，如表 3-1 所示。

表 3-1 分隔符功能说明表

分隔符	名称	功能说明
{ }	大括号（花括号）	用来定义块、类、方法及局部范围，也用来包括自己初始化的数组的值。大括号必须成对出现
[ ]	中括号（方括号）	用来进行数组的声明，也用来撤销对数组值的引用
( )	小括号（圆括号）	在定义和调用方法时，用来容纳参数表。在控制语句或强制类型转换的表达式中用来表示执行或计算的优先权
;	分号	用来表示一条语句的结束。语句必须以分号结束，否则即使一条语句跨行或者多行，仍是未结束的
,	逗号	在变量生命中用于分隔变量表中的各个变量，在 for 控制语句中，用来将圆括号里的语句链接起来
:	冒号	说明语句标号
.	圆“点”	用于类/对象和它的属性或者方法之间的分隔。例如，圆点“.”就起到了分隔类/对象和它的方法或者属性的作用

3.2.3 知识准备：Java 语言标识符的组成规则

在 Java 中，标识符是赋予变量、类或方法的名称。程序通过这些名称来访问或修改某个数据的值。标识符可从一个字母、下画线(_)或美元符号($)开始，随后也可跟数字。在这里，字母的范围并不局限于 26 个英文字母，而是包括任何一门语言中的表示字母的任何 Unicode 字符。标识符未规定最大长度。

在定义和使用标识符时需要注意，Java 语言是大小写敏感的。比如，“abc”和“Abc”是两个不同的标识符。

在定义标识符的时候，需要注意以下问题：

（1）标识符不能有空格。

（2）标识符不能以数字开头。

（3）标识符不能是 Java 关键字。

（4）不能有@、#等符号。

问：定义标识符可以用中文吗？

答：可以使用中文名称作为标识符，但是并不建议这么做。因为在 Java 中，使用中文容易引起一些编码方面的问题。

3.2.4 任务二：综合使用 Java 分隔符和标识符

1. 任务描述

编写程序，输出手机开机问候语，体会 Java 分隔符的作用和标识符的使用规范。

2. 技能要点

- 了解各类分隔符的功能。

- 掌握标识符命名规范。

3．任务实现过程

（1）编写一个类名为 OpenGreetings，类中定义了一个方法 theDate()，用于打印日期和开机问候语。在 main()方法中调用 theDate 方法，并传入当天的日期作为参数。

（2）声明并初始化日期变量时，注意标识符的命名规则，当使用@开头，数字开头或者关键字时会报错。

源文件：OpenGreetings.java

```
public class OpenGreetings {

 public static void main(String[] args) {
      int day = 20, month = 5, year = 2011;
      //以下 3 种命名标识符不合法
      //int @day; int 12abc; int private;
      OpenGreetings og = new OpenGreetings();
      og.theDate(day, month, year);
      }

 public void theDate(int theDay,int theMonth,int theYear){
      String greetings = "Welcome To Android World~!";
      System.out.println("Today is "+theYear+"/"+theMonth+"/"+theDay+"\n"+greetings);
 }
}
```

（3）运行程序，运行结果如下：

```
Today is 2011/5/20
Welcome To Android World~!
```

3.3 Java 关键字/保留字

3.3.1 知识准备：Java 关键字使用规范

Java 中一些赋以特定的含义、并用做专门用途的单词称为关键字（keyword）。在定义自己的标识符的时候，不要和这些关键字重名，否则，在编译时将会出现错误。比如，下面的变量定义就是错误的：

```
int byte;
```

在这个例子中，试图定义一个 int 类型的变量“byte”，但是因为“byte”是关键字，所以不能用来作为变量名。

所有 Java 关键字都是小写的，TURE、FALSE、NULL 等都不是 Java 关键字，goto 和 const 虽然从未被使用，但也作为 Java 关键字保留。Java 中一共有 51 个关键字，如表 3-2 所示。

表 3-2 Java 关键字

abstract	assert	boolean	break	byte	continue
case	catch	char	class	const	double
default	do	extends	else	final	float
for	goto	long	if	implements	import
native	new	null	instanceof	int	interface
package	private	protected	public	return	short
static	strictfp	super	switch	synchronized	this
while	void	throw	throws	transient	try

volatile					

3.3.2 知识准备：重点关键字解析

- abstract：Java 中的一个重要关键字，可以用来修饰一个类或者一个方法为抽象类或者抽象方法。
- extends：表示继承某个类，继承之后可以使用父类的方法，也可以重写父类的方法。
- super：super 关键字表示超（父）类的意思。
- this：代表对象本身。
- interface：声明一个接口。
- implements：实现接口关键字。
- private：访问控制修饰符，声明类的方法，字段，内部类只在类的内部可访问。
- protected：访问控制修饰符，声明类成员的访问范围是 package 包内可访问。
- public：访问控制修饰符，声明类成员对任何类可见。
- static：表示应用它的实体在声明该实体的类的任何特定实例外部可用。
- final：用来修饰类或方法，表示不可扩展或重写。

3.4 数据类型

3.4.1 知识准备：简单类型

Java 有 8 种简单类型：四种整型、两种浮点型、一种字符型、一种用于表示 true/false 的布尔类型。表 3-3 列出了这 8 种简单数据类型。

表 3-3 简单数据类型

数据类型	数据类型名称	大小（bits）	默认值
boolean	布尔类型	1	false
char	字符型	16	0
byte	字节型	8	0
short	短整型	16	0
int	整型	32	0
long	长整型	64	0
float	单精度浮点型	32	0.0
double	双精度浮点型	64	0.0

在这些数据类型中，int、short、byte、long 都是整型数据，而 double 和 float 是浮点型数据。char 也可以看成是整型数据，但它是无符号的（没有负数）。

1. 布尔类型

与 C 语言不同，Java 定义了专门的布尔类型，用于表示逻辑上的“真”或“假”。布尔类型用关键字 boolean 来定义，数据只能取值 true 或 false，注意不能用整型的 0 或 1 来替代。布尔类型不是数值型变量，它不能被转化成任意一种类型。布尔类型常常用在条件判断语句中。

```
boolean sendMsg = true;
boolean recieveMsg = false;
```

注意：

这里的 true 和 false 是不能用双引号引起来的，如果用双引号引起来，就成了 String 类型的数据了。

2．字符类型

字符（Char）类型数据用来表示通常意义上的“字符”。字符常量是用单引号括起来的单个字符。

Java 与众不同的特征之一就是 Java 对各种字符的支持，这是因为 Java 中的字符采用 16 位的 Unicode 编码格式，Unicode 被设计用来表示世界上所有书面语言的字符。Char 简单类型保存一个 16 位的 Unicode 字符。Unicode 字符通常用十六进制编码形式表示，范围是“\u0000”~“\Uffff”，其中前缀“\u”标志着这是一个 Unicode 编码字符，其中前 256 个（“\u0000”~“\u00FF”）字符与 ASCII 码中的字符完全重合。

下面来看一个字符型数据的应用实例：

源文件：CharTest.java

```
public class CharTest {
 public static void main(String args[]) {
      char Msg1 = 'M';
      char Msg2 = '中';
      char Msg3 = '5';
      char Msg4 = '\u0001';
      System.out.println(Msg1);
      System.out.println(Msg2);
      System.out.println(Msg3);
      System.out.println(Msg4);
 }
}
```

char 型数据只能记录单个的字符值，不能表述更多的文字信息，Java 语言还提供了 String 类型——记录由多个字符组成的字符串。String 常量的表达形式为双引号引起来的 0~多个字符，例如：

```
String s = "Java 小能手";
System.out.println("Hello,Android!");
```

注意：

String 不是基本数据类型，而属于引用类型。

char 类型的数据用单引号表示，注意它和 String 类型数据的区别，例如，“A”表示的是一个 char 类型的数据，而“A”表示的是一个 String 类型的数据，它们的含义是不同的。

Java 语言中还允许使用转义字符“\”来将其后的字符转变为其他的含义，例如，如果需要在输出结果时换行，应给编译器换行指令 n，但是如果直接在程序中写 System.out.println("n");则不会起到换行的效果。此时，需要在 n 之前输入“\”，这时编译器会知道“n”是被转义的字符。

还有一种特殊情况，在 Java 中使用一个绝对路径：c:\learning\java，如果直接在程序中写 String path = “c:\learning\java”，则不会得到你期望的结果，因为 Java 将“\”当成转义符了。所以，这时候应该这样写：

```
String path = "c:\\learning\\java";
```

这时，第一和第三个“\”都是表示转义符，表示后面的那个字符（此处都为“\”）有特殊的含义。

3．整数类型

整数类型分为 4 类：byte、short、int 及 long，它们的差别在于所占用的内存空间和表数范围不同。表 3-4 列出了这 4 种整数类型数据的存储空间及表数范围。

表 3-4　整数类型的存储空间和表数范围

类　型	占用存储空间	表数范围
byte	1 字节	−128 ~ 127
short	2 字节	$-2^{15} \sim 2^{15}-1$ （−32768~32767）
int	4 字节	$-2^{31} \sim 2^{31}-1$ (−2147483648~2147483647)
long	8 字节	$-2^{63} \sim 2^{63}-1$

通常情况下，int 是最常用的一种整型数据，它也是 Java 中整数常量的默认类型。在表示非常巨大的数字时，则需要用到更大范围的 long。对于前面 3 种整数数据类型的数据，只需要直接写出数据就可以了，而对于长整形（long）数据，需要在长整型数据后面加上 L 或 l 来表示。

整型常量虽然默认为 int 类型，但在不超过其表数范围的情况下，可以将 int 类型的数据直接赋给 char、byte、short 类型的变量。

下面是这些整形数据类型的一些例子：

```
byte b = 12;
short s = 12345;
int i = 1000000;
long l = 1000000000L;
```

Java 中允许使用 3 种不同的进制形式表示整型变量：八进制、十进制、十六进制。

（1）十进制整数，如 123、-456、0。

（2）八进制整数，以 0 开头，如 0123 表示十进制数 83，-011 表示十进制数-9。

（3）十六进制整数，以 0x 或 0X 开头，如 0x123 表示十进制数 291，-0X12 表示十进制数-18。

4．浮点类型

Java 浮点类型有两种：float 和 double。Java 浮点类型有固定的表数范围和字段长度。和整数类型一样，在 Java 中，浮点类型的字段长度和表数范围与机器无关。表 3-5 列出了符点类型数据的存储空间和表数范围。

表 3-5　符点类型数据的存储空间和表数范围

类　　型	占用存储空间	表数范围
float	4 字节	-3.403E38~3.403E38
double	8 字节	-1.798E308~1.798E308

double 类型的浮点类型数据正如它的名字所揭示的，它表示精度是 float 的两倍（因此也将 double 类型的数据称为双精度类型的数据）。表示 float 类型的数据需要在数字后面加上 F，用于和 double 类型数据相区别。

Java 语言浮点类型常量有两种表示形式：

- 十进制数形式，必须含有小数点，例如 3.14、314.0、0.314。否则将被当做 int 型常量处理，例如 314。
- 科学计数法形式，如 3.14e2、3.14E2、314E2。注意，只有浮点类型才能采用科学计数法表示，因此，314E2 也是浮点型常量，而不是 int 型。

Java 语言的浮点型常量默认为 double 型，要声明一个常量为 float 型，则要在它数字的后面加 f 或 F。例如：

3.0 表示一个 double 型常量，占 64 位内存空间。

3.0f 表示一个 float 型常量，占 32 位内存空间。

3.4.2　知识准备：非 boolean 简单数据类型之间的转换

在 Java 程序中，一些不同的数据类型之间可以进行数据类型的相互转换。简单数据类型的转换一般分为两种：

（1）低级到高级的自动转换。

（2）高级到低级的强制类型转换。

二者的区别主要在于数据类型的表述范围是不同的。比如，有一个 int 类型的数据，赋给一个 long 类型的变量，或者反之。这就类似于将水（数据）从一个容器（某种数据类型）倒入到另一个容器（另一种数据类型）一样，因为容器的大小不同，能够装盛的水也是不同的。如果将从小容器中的水倒入到大容器中，不会有什么问题，但是，如果将大容器中的水倒入到小容器中，就可能会造成部分水溢出。同样的，在数据类型转换上面，也有类似的问题，如果将表数范围比较小的数据类型数据转换成表数范围大的数据

类型，则可以顺利转换；反之，则有可能发生数据的溢出（损失一部分信息）。

在图 3-3 所示的数据类型的转换中，实线条表示这种转换不会引起信息的损失，而虚线条表示此种转换可能会引起信息的损失。

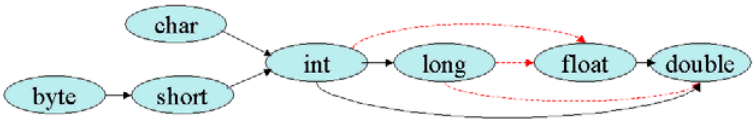

图 3-3　不同数据类型之间的合法数据转换

如果数据的转换按照图 3-3 中箭头所示的方式来完成，则程序会自动转换，不需要在程序中干预，这种转换是低级到高级的自动转换，也成为“扩展转换（Widening Conversion）”。但是，如果不按照图中的方向来转换，则可以通过“强制类型转换”的方式来完成，此时，可能会引起信息的丢失。当按照图 3-3 中箭头所示的反方向来转换时，非常有可能造成数据精度的损失，这种转换也经常称为“缩小转换（Narrowing Conversion）”。

例如，int 类型的数据在必要时可以自动转换成 double 的数据，但是，如果需要将 double 类型的数据转换成 int 类型的数据，则需要通过强制类型转换来完成。下面这条语句可以实现这个功能：

```
double d = 1.2345;
int i = (int)d;
```

这样，就可以将 double 类型的数据 d 转换成 int 类型的数据，此时，i 的值为 1，显然，小数后面的值都丢失了。

3.4.3　任务三：简单数据类型转换实例

1．任务描述

编写程序验证低级到高级的自动转换和高级到低级的强制类型转换。

2．技能要点

- 理解类型转换的原理。
- 掌握强制类型转换的方法和用途。

3．任务实现过程

（1）编写源文件：DataOper.java，使 int 类型变量自动转化为 double 类型变量，double 类型变量强制转化为 int 类型变量。

源文件:DataOper.java

```
public class DataOper {
  public static void main(String[] args) {
       // int 类型数据将会自动转换成 double 类型
       double db1;
       int i = 123;
       db1 = i;
       System.out.println("db1=" + db1);
       // double 类型数据转换成 int 时，将会损失精度
       double db2 = 1.234;
       int j = (int) db2;
       System.out.println("j=" + j);
  }
}
```

（2）编译并运行这个程序，可以得到如下的输出：

```
Db1=123.0
j=1
```

int 类型的数据已经自动转换成了 double 类型的数据，而 double 类型数据在强制转换成 int 类型数据的时候，小数点后面的值已经损失了。

3.4.4 知识准备：引用类型

Java 语言中除了 8 种基本数据类型以外的数据类型称为引用类型，或复合数据类型。如第 2 章中所述，引用类型的数据都是以某个类的对象的形式存在的，在程序中声明的引用类型变量只是为该对象起的一个名字，或者说是对该对象的引用，变量的值是对象在内存空间中的存储地址而不是对象本身。

Java 中的所有对象都要通过对象引用访问，引用类型数据以对象的形式存在，其构造和初始化及赋值的机制都与基本数据类型的变量有所不同。声明基本数据类型的变量时，系统同时为该变量分配存储器空间，此空间将直接保存基本数据类型的值。而声明引用类型变量时，系统只为该变量分配引用空间，并未创建一个具体的对象或者说并没有为对象分配存储器空间，将来在创建一个该引用类型的对象后，再使变量和对象建立对应关系。

这就好比用遥控器（引用）操纵电视机（对象），只要控制遥控器就可以保持与电视机的连接。如果需要换台，实际上操控的是遥控器（引用），再用遥控器操作电视机（对象）。此外，即使没有电视机，遥控器也可以独立存在。也就是说，你拥有一个引用，并不是一定需要有一个对象与它关联。因此，如果想操纵一个词，可以创建一个 String 引用：

```
String s;
```

这里所创建的只是引用，并不是对象。如果此时向 s 发送一个消息，就会返回一个运行时的错误，因此，一种比较安全的做法是创建引用的同时就进行初始化。

```
String s = "Hello android";
```

在这里用到了 Java 语言的一个特性：字符串可以用带引号的文本初始化。通常，对其他对象需要一种更通用的初始化方法。参看任务四的引用过程。

3.4.5 任务四：引用类型程序示例

1. 任务描述

编写程序定义引用类型变量，了解引用类型的使用方法和特点。

2. 技能要点

- 声明引用类型。
- 了解引用类型和普通类型在使用上的区别。

3. 任务实现过程

（1）构造有引用类型成员变量的类 Email，在这个类中，定义了 3 个属性：address、title、group，以及一个具有 3 个参数的构造器，用于在创建对象时初始化 3 个属性。通过定义这个类，也就定义了一个引用类型的数据类型：Email。

源文件: Email.java

```
public class Email {
 // 定义属性
 String address;

 String title;

 int group;


 // 定义一个构造器
 public Student(String Email_address, String Email_title, int Email_group)
 {
      address = Email_address;
      title = Email_title;
      group = Email_group;
          }
```

```
    // 定义属性“address”的设置方法
    public void setAddress (String Email_address) {
         address = Email_address;
    }

    // 定义属性“title”的获取方法
    public String getTitle() {
         return title;
    }
    // 其他属性的设置和读取方法从略
}
```

（2）再定义一个 TestEmail.java 类，用于说明引用类型的用法。

源文件：TestEmail.java

```
public class TestEmail {
    public static void main(String[] args) {
         Email e1;
         e1 = new Email("Android@gmail.com", "capter 3", 2);
         System.out.println("第一封邮件地址是：" + e1.getAddress());
         Email e2;
         e2 = e1;
         e1.setAddress("Java@gmail.com");
         System.out.println("第二封邮件地址是：" + e2.getAddress());
    }
}
```

执行上面的 TestEmail 应用程序，可以在控制台上得到如下的输出结果：

第一封邮件地址是：Android@gmail.com。

第二封邮件地址是：Java@gmail.com。

3.4.6 技能拓展任务：分析对象的构造和初始化

1．任务描述

分析任务四的代码，了解对象的构造和初始化的原理和过程。

2．技能要点

- 理解引用类型变量的初始化过程和内存分配过程。

3．任务实现过程

分析一下 TestEmail.java 中的代码段：

```
         Email e1;
         e1 = new Email("Android@gmail.com", "capter 3", 2);
         System.out.println("第一封邮件地址是：" + e1.getAddress());
         Email e2;
         e2 = e1;
         e1.setAddress("Java@gmail.com");
         System.out.println("第二封邮件地址是：" + e2.getAddress());
```

这个代码段的作用是建立并初始化了两个 Email 引用类型数据，以对象 e1 为例讲解引用类型数据的初始化过程（对象的初始化过程）。

（1）执行语句“Email e1;”时，系统为引用类型变量 e1 分配引用空间（定长 32 位），此时只是定义了变量 e1，还未进行初始化等工作，因此还不能调用 Email 类中定义的方法，如图 3-4 所示为此时内存的分配情况。

（2）执行语句“e1 = new Email("Android@gmail.com", "capter 3", 2);”，先调用构造方法创建一个 Email 类的对象——新对象分配的内存空间用来存储该对象所有属性（address，title，group）的值，并对各属性的值进行默认的初始化（关于默认初始化请参考 3.5 节内容）。注意，在这个程序中，因为 address 和 title

的类型是 String 类型，也是属于引用类型，所以它们的默认初始值也为 null，图 3-5 为此时内存中的情况。

图 3-4　步骤（1）执行后的内存情况

图 3-5　步骤（2）执行后的内存情况

（3）接下来执行 Email 类的构造方法，继续此新对象的初始化工作，构造方法中又要求对新构造的对象的成员变量进行赋值，因此，此时 address、title、group 的值变成了"Android@gmail.com"、"capter 3"和 2，图 3-6 所示为此时的内存情况。

图 3-6　步骤（3）执行后的内存情况

（4）至此，一个 Email 类的新的对象的构造和初始化构成已完成。最后再执行“e1 = new Email("Android@gmail.com", "capter 3", 2)”中的“=”号赋值操作，将新创建对象存储空间的首地址赋值给 Email 类型变量 e1，如图 3-7 所示为此时的内存情况。

图 3-7　执行步骤（4）的内存情况

于是引用类型变量 e1 和一个具体的对象建立了联系，称 s1 是对该对象的一个引用。最后，总结一下对象的构造及初始化程序的步骤：

（1）分配内存空间。
（2）进行属性的默认初始化。
（3）进行属性的显式初始化。
（4）执行构造方法。
（5）为引用型变量赋值。

3.5　变量及其初始化

在编写程序时，通常需要使用一个“别名”来表示某种类型的可变值，这就是“变量”。在前面的程序中已经在很多地方用到了“变量”。比如，任务四中，就定义了变量“address”、“title”等。

3.5.1　知识准备：局部变量

大多数过程语言都有作用域的概念，作用域决定了在其内的变量名的可见性和生命周期。Java 允许变量在任何程序块内被声明。变量的作用域在指定的局部程序块（可以是方法体、程序段等），这种类型的变量通常被称为“局部变量”。要注意的是，变量在其作用域内被创建，离开其作用域时被撤销。因此，一个变量的生存期就被限定在它的作用域中，超出了作用域局部变量无效。

1．在程序块中声明局部变量

所谓程序块，就是用“{”和“}”包含起来的代码块。它是一个单独的模块，和方法有点类似，但不能像方法一样可以用方法名来调用。在这个程序块中的变量也是局部变量，即使这个程序体处于类的定义中。如下程序段：

```
public class Clock{
  int getUp = 6;
  {
       int goBed = 11;
  }
  public void printit() {
       System.out.println(getUp);
       System.out.println(goBed);// 出错，因为 goBed 是局部变量，不能用于变量所在程序块范围之外
  }
}
```

其中的变量 goBed 就是局部变量，它只在自己所处的代码块中起作用。

在 Java 中，所有的变量必须先声明再使用。基本的变量声明方法如下：

```
<data_type> var_name [=var_value]
```

其中 data_type 是变量类型，它可以是基本类型之一，或类及接口类型的名字。变量类型后面，跟着此变量的名称，它的名称必须是一个符合 Java 命名规范的标识符。然后，可以给这个变量赋一个值，注意，这个值的数据类型必须和变量的数据类型一致（或者兼容）。下面是变量声明的几个例子：

```
int i;
double d1 = 123.4;
double d2 = 123;
```

声明同一类型的多个变量时，使用逗号将各变量分隔开：

```
int i,j,k;
int a=1,b,c=100;
```

2．在方法体中声明局部变量

此时，此变量的作用范围只局限于此方法体内。示例如下：

```
public class ClockTest{
  static int getUp = 6;
  {
       int goBed = 11;
  }
  public void alarm() {
       int al = 20;
  }
  public void close() {
       System.out.println(al);// 出错，al 的作用范围只局限于方法 alarm 中
  }
}
```

因为变量 al 是定义在方法 alarm 中的，所以它的作用范围只局限在方法 alarm 中。在方法 close 中试图使用这个变量，在编译的时候将会报错。

局部变量在方法或代码块执行时创建，方法或代码块执行结束时销毁。局部变量在使用前必须初始化。

3.5.2 知识准备：成员变量

不在方法体也不在程序块中的变量，称为成员变量。成员变量定义在类中，是类成员的一部分，整个类都可以访问它。只要类的对象被引用，成员变量就将存在。Java 中成员变量说明的形式如下：

```
[修饰符] 成员变量类型 成员变量名列表;
```

成员变量的修饰符有以下几种：默认访问修饰符、public、protected、private、final 和 static 等。默认访问修饰符的成员变量可以被同一包（Package）中的任何类访问。

下面是成员变量的一个例子：

源文件：Contact.java

```
public class Contact{
  public static void main(String args[]) {
       // 实例化 SendMsg 后可以访问具有访问权限的成员变量
       SendMsg send = new SendMsg();
       System.out.println(send.word);
  }
}

class SendMsg {
  int word = 12;// 成员变量

  public int getWord() {
       return word;
  }
}
```

在这个例子中，定义了一个类 SendMsg，在这个类里面定义了一个 int 类型的成员变量： word，在用于测试的类 Contact 中，首先实例化这个类，然后再通过“实例名.变量名”的方式来访问它。

3.5.3 知识准备：变量初始化

所有的局部变量在使用之前，都必须初始化，也就是说，必须要有值。

在初始化变量时，应该把变量名写在左边，随后是赋值操作符“=”，然后再在右边加上一个恰当的 Java 表达式或值。

变量的初始化有两种方法：一种是在声明变量的时候同时给它赋一个值：

```
int i  = 4;
```

还有一种是，先声明变量，然后再在适当的时机给它赋值：

```
int k;
…
k = 10;
```

3.5.4 知识准备：局部变量的初始化

局部变量也可以像成员变量一样，先声明，再初始化；或者在声明的同时，就对其进行初始化。也可以一次声明几个同一数据类型的变量。但是，系统不会对局部变量进行默认的初始化，因此，局部变量在使用之前，必须对其进行显式初始化。

下面的代码中，因为 n 没有初始化，所以，这个时候如果对它进行操作，将会报错：

```
public class Test{
  …
  public void aMethod(int j){
       int m,n,k;
       m = j;
       k = 100;
       System.out.println(m);
       System.out.println(n);//Error
       System.out.println(k);
  }
}
```

注意，在这个程序中的变量 m，它根据方法的传入参数 j 来初始化，所以这个时候在方法体中是可以对其进行任何和其数据类型相匹配的操作了。

3.5.5 知识准备：成员变量的初始化

变量在使用之前，必须首先对它进行初始化。首先来看一下成员变量的几种初始化方式：

- 成员变量的初始化可以在声明时初始化，那么创建一个新对象时，此成员变量的值固定不变。

- 成员变量在类的构造方法或其他方法中动态的初始化。
- 成员变量的默认初始化值。

默认初始化是 Java 成员变量的特性，可以不需要手动地显式初始化成员变量。因为系统在创建一个新的对象的时候，会给这些成员变量赋一个初值。在这里需要注意的是，对于引用变量，它的默认初值是 null 而非相应的引用类型对象，也就是说，它并不指向任何对象的首地址。特别是不能通过该引用类型变量去调用任何方法或属性，否则，将会出现错误。

虽然成员变量可以不用显式初始化即可使用，但是，系统给不同数据类型的成员变量初始化时，它的初始化的值是不同的，比如，int 类型的初始化值是 0，boolean 类型的初始化值是 false。这与 3.4 节中给出的基本类型默认初始值相同。

注意：
系统只对全局（成员）变量指定默认的值，不会对局部变量赋值。

3.5.6 任务五：成员变量的 3 种初始化方式

1. 任务描述

编写一个类，有 3 个成员变量，分别用上述 3 种方式进行初始化，打印它们的值，查看初始化效果，并验证 Java 实行了成员变量的默认初始化。

2. 技能要点

- 掌握 3 种初始化方式。
- 了解 3 种初始化方式的区别和应用场合。

3. 任务实现过程

（1）编写一个名为 Initialize 的类，该类中声明 3 个 int 类型变量：a、b、c。对 a 进行声明时的初始化，确定 a 的值；对 b 在构造方法中进行动态初始化；对 c 不进行初始化。

（2）在 main()方法声明并初始化一个 Initialize 对象，打印其成员变量 a、b、c 的值。

源文件：Initialize.java

```
public class Initialize {

 int a = 10;//成员变量 a 在声明时初始化
 int b;
 int c;
 public Initialize(int i){
      this.b = i;//构造方法中初始化成员变量 b
 }
 public static void main(String[] args) {
      Initialize init = new Initialize(20);
      System.out.println("声明时初始化变量 a="+init.a+
                "\n 构造方法初始化变量 b="+init.b+"\n 变量默认初始化 c="+init.c);
 }
}
```

（3）运行程序，输出结果如下：

```
声明时初始化变量 a=10
构造方法初始化变量 b=20
变量默认初始化 c=0
```

3.6 值传递和引用传递

3.6.1 知识准备：Java 中的值传递

在程序中，经常需要将一个变量的值赋给另一个变量，赋值后，两个变量的值相同，那么，在 Java 中，它的实现机制是怎样的呢？首先来看一个例子：

源文件：CallByValuePri.java

```
/**
 * <DL>
 * <DT><b>功能：</b>
 * <DD>简单类型数据传值调用演示</DD>
 * </DL>
 */
public class CallByValuePri {
 void half(int n) {
      n = n / 2;
      System.out.println("half 方法 n=" + n);
 }

 public static void main(String args[]) {
      CallByValuePri cb = new CallByValuePri();
      int m = 10;
      System.out.println("Before the Invocation,m=" + m);
      cb.half(m);
      System.out.println("After the Invocation,m=" + m);
 }
}
```

编译并用如下程序执行这个程序：

```
java CallByValue
```

可以得到如下的输出：

```
Before the Invocation,m=10
half 方法 n=5
After the Invocation,m=10
```

可以看到，在调用方法的前后，m 的值不变。因为在 Java 中，传递的是值，这样，只是将 m 的值传递给了方法 half()，而 m 本身的值并没有发生改变。按值传递的重要特点：传递的是复制的值，也就是说，传递后就互不相关了。

3.6.2 知识准备：Java 中的引用传递

引用传递指的是在方法调用时，传递的参数是按引用进行传递，其实传递引用的地址也就是变量所对应的内存空间的地址。示例如下：

源文件：PassByReference.java

```
public class PassByReference {

 private void printNumber(AddressBook a){
        a.number = 20;
        System.out.println("printNumber 方法中的 numner="+a.number);
      }
      public static void main(String[] args) {
           PassByReference p = new PassByReference();
           AddressBook a = new AddressBook();
           a.number = 10;
           System.out.println("执行 printNumber 方法前 main 方法中的 numner="+a.number);
          p.printNumber(a);
          System.out.println("执行 printNumber 方法后 main 方法中的 numner="+a.number);
        }
      }
      class AddressBook{
           //电话簿中联系人数量
          public int number;
      }
```

运行结果如下：

```
执行 printNumber 方法前 main 方法中的 numner=10
printNumber 方法中的 age=20
执行 printNumber 方法后 main 方法中的 age=20
```

可以看到，当 printNumber 方法将引用对象 a 的成员变量 number 值改变后，方法外的对象 a 的成员变量 number 值同样改变。传递的是值的引用，也就是说，传递前和传递后都指向同一个引用（也就是同一个内存空间）。

注意：

在 Java 里面只有基本类型和按照下面这种定义方式的 String 是按值传递，其他的都是按引用传递。就是直接使用双引号定义字符串方式：String str = "This is Java";

3.7 Java 编码规范

想象一下，在一个大型的项目中，如果每个程序员在给包、类、变量、方法取名的时候没有一点约定，只是随心所欲，可能会带来哪些问题？

（1）程序可读性极差。

（2）在相互有交互的程序中，给其他程序员理解程序带来很大的麻烦。

（3）对于测试员来说，在测试中如果需要检查源程序，将会感到无从下手。

（4）在后续的维护中，可能因为程序根本没法看懂，而不得不重新编写一个新的程序。

因此，程序设计的标准化非常重要，原因在于这能提高开发团队各成员的代码的一致性，使代码更易理解，这意味着更易于开发和维护，从而降低了软件开发的总成本。为实现此目的，和其他语言类似，Java 语言也存在非强制性的编码规范。

3.7.1 知识命名规范

命名惯例也称命名约定，在声明包名、类名、接口名、方法名、变量名、常量名时，除必须符合标识符命名规则外，还应尽量体现各自描述的事物或属性、功能等。例如，可定义类 Student 描述学生信息。一般性命名约定如下：

- 尽量使用完整的英文单词或确有通用性的英文缩写。
- 尽量采用所涉及领域的通用或专业术语。
- 词组中采用大小写混合使之更易于识别。
- 避免使用过长的标识符（一般小于 15 个字母）。
- 避免使用类似的标识符，或者仅仅是大小写不同。

具体命名规范如表 3-6 所示。

表 3.6 Java 编程命名规范

元素名称	命名规则	示例
包（Package）	采用完整的英文描述符，应该都是由小写字母组成。对于全局包，将 Internet 域名反转并接上包名	com.srt.moa.action
类（Class）	类名一般使用（动）名词或（动）名词组合来表示，并且各个名词的首字母大写，其他字母小写	Customer, SavingsAccount
接口（Interface）	接口名与类名类似，一般使用（动）名词或（动）名词组合来表示，并且各个名词的首字母大写，其他字母小写	Contactable, Prompter
方法	方法一般用一个动宾短语组成，表示一个动作。	public void balanceAccount(int

（Interface）	方法的首字母小写（通常是动词的首字母），而其他单词的首字母大写（通常是一个名词的首字母）	deposit){...}
异常（Exception）	通常采用字母 e 表示	Exception e
变量名	变量的名称一般使用一个（动）名词或其组合来表示，并且首字母小写，其他单词的首字母大写	private int age; private String studentName;
静态常量字段（static final）	全部采用大写字母，单词之间用下画线分隔	static final int MAX_SIZE=10;

数组	数组应该总是用下面的方式来命名：
	`byte[] buffer;`
	而不是：
	`byte buffer[];`
		byte[] buffer

3.7.2 代码编写格式规范

代码编写格式规范如下：

- 缩进一般是每行 2 个或 4 个空格，使排版整齐，语句可读。
- 关键词和操作符之间加适当的空格。
- 相对独立的程序块与块之间加空行。
- 较长的语句、表达式等要分成多行书写。
- 若函数或过程中的参数较长，需要进行适当的划分。
- 一行只写一条语句，即使是短语句也要分行。
- 程序块由大括号{...}界定，大括号必须成对出现，且编程时“{”和“}”应各占独立一行，同时与引用它们的语句左对齐。例如：

```
if (i>0) { i ++ }; // 错误, { 和 } 在同一行

if (i>0)
{
i ++
};      // 正确, “{”单独作为一行
```

3.8 本章小结

本章介绍了 Java 注释、分隔符和标识符、关键字、数据类型等 Java 编程基础知识；分析了变量的种类和几种初始化方法；描述了 Java 中值传递和引用传递两种传递方式的方法和区别。最后给出了 Java 编码规范。本章所述均为 Java 编程中最基础但是关键的技术要点，读者通过学习，完成章节任务，可以对 Java 编程的理解深入一步，并为之后的学习打下基础。

课后练习题

一、选择题

1．一个 byte 表示的数据范围是（ ）。

A．-128 ~127　　B．-32768 ~32767

C．-255 ~ 256　　D．跟 Java 虚拟机相关

2．下面（ ）是不合法的 Java 标识符。

A．example123　　B．_for　　C．$apple　　D．#student

3．下面（ ）不是 Java 的保留字。

A．goto　　B．null　　C．While　　D．native

4．关于 javadoc 命令正确的说法是（　　）。

A．javadoc 可以提取所有的注释生成注释文档

B．javadoc 生成的注释文档是一个 word 文件

C．javadoc 生成的注释文档是一个 HTML 文件

D．javadoc 只能提取/**...*/之间的注释

5．下面（　　）不是 Java 基本数据类型。

A．String　　B．boolean　　C．float　　D．byte

二、简答题

1．简述代码块的概念。

2．简述标识符和保留字的区别。

3．简述 Java 中的数据类型。

4．简述局部变量、成员变量的区别。

5．简述参数的值传递和引用传递。

三、编程题

编写一个类，它有一个 char 类型的属性和一个 int 类型的属性，不对它们进行初始化，打印它们的值，以验证 Java 执行了默认的初始化。

联系方式

集团官网：www.hqyj.com　　嵌入式学院：www.embedu.org　　移动互联网学院：www.3g-edu.org

企业学院：www.farsight.com.cn　　物联网学院：www.topsight.cn　　研发中心：dev.hqyj.com

集团总部地址：北京市海淀区西三旗悦秀路北京明园大学校内 华清远见教育集团

北京地址：北京市海淀区西三旗悦秀路北京明园大学校区，电话：010-82600386/5

上海地址：上海市徐汇区漕溪路 250 号银海大厦 11 层 B 区，电话：021-54485127

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-22193762

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218