

Java语言基础

Java语言基本程序设计知识

1, JavaAPI概述

1.1, 什么是API

- API是指应用程序接口(Application Program Interface, API), 故名思意就是程序的接口, 什么是接口呢? 我们在日常生活中经常会使用到U盘接口, 当我们的手机没有电的时候我们需要使用充电器通过手机的充电接口为我们的手机进行充电, 当我们要将我们手机上的照片发送到计算机上时, 我们需要使用数据线一端连接到我们手机的充电接口, 另一端连接到计算机的USB接口。



- 所以应用程序接口就是负责应用程序与应用程序之间沟通的通道。
- 程序中的接口是如何来的呢?

研发人员A开发了软件A, 研发人员B正在研发软件B。有一天, 研发人员B想要调用软件A的部分功能来用, 但是他又不想从头看一遍软件A的源码和功能实现过程, 怎么办呢? 研发人员A想了一个好主意: **我把软件A里你需要的功能打包好, 写成一个函数。你按照我说的流程, 把这个函数放在软件B里, 就能直接用我的功能了!** 其中, API就是研发人员A说的那个函数。

下面的例子可以更好的理解上面的由来。

假如我想要使用Java实现按下键盘输入"hello"然后在显示器通过控制台输出"hello", 此时应该如何实现呢?

太深入的思路就不说了, 就说使用Java语言吧, 如果我想要实现这样的功能我需要编写一个当我按下键盘的时候获取键盘的键值的代码, 然后我还有写一个将这个键值和ASCII码对应转换的程序当然对于键值来说除了数字键盘的键值和ASCII码中的不对应之外, 就字母键盘中的键值是完全对应于ASCII码的, 最后我要编写一个调用显示器的程序, 将查找到的字符通过控制台显示到显示器上。当然这也只是一种实现思路, 这里只是做一个举例的例子, 真正的实现需要编写输入输出流等一些复杂操作。

在这里Java提供了一个实现该功能的程序, 也就是Scanner类(后面会具体分析它的一些用法)它的一个作用是从键盘中获取输入的内容并使用nextLine()方法以字符串类型存储, 然后使用System类里面的println()方法将它输出到屏幕上, 那么现在我不想自己如此复杂的去实现这个功能, 我就希望使用Java提供给我的这个程序进行实现, 所以我需要将我的程序和它的程序连上中间的那根线, 然后去使用它的输入输出的功能, 此时程序可以这样写。

```

package codes;
public class ApiDemo {
    public static void main(String[] args) {
        //scanner在java.util包下通过该方式引用Scanner类，就相当于把线接起来
        java.util.Scanner scanner = new java.util.Scanner(System.in); //获取输入
        String input = scanner.nextLine(); //将输入内容以字符串类型存储
        System.out.println(input); //输出到控制台屏幕上
    }
}

```

最后的运行结果是：

```

C:\Users\EMCred\Desktop>java ApiDemo
hello
hello

```

这时我们可以说Scanner就是这个接口，nextLine就是我们所调用的这个接口里面的功能。

1.2, SDK和API

- SDK 就是 Software Development Kit 的缩写，翻译过来——软件开发工具包。这是一个覆盖面相当广泛的名词，可以这么说：辅助开发某一类软件的相关文档、范例和工具的集合都可以叫做 SDK。比如：有公司开发出某种软件的某一功能，把它封装成SDK（比如数据分析SDK就是能够实现数据分析功能的SDK），出售给其他公司做开发用，其他公司如果想要给软件开发出这种功能，但又不想从头开始搞开发，直接付钱省事。
- SDK和API有什么关系呢？

可以把SDK想象成一个虚拟的程序包，在这个程序包中有一份做好的软件功能，这份程序包几乎是全封闭的，只有一个小接口可以联通外界，这个接口就是API。比如：想要在企业ERP系统中增加某个功能（比如自动备份、数据分析、云存储等），但又不想耗费大量时间、也没那么多研发亲自去做这个功能。这时我们可以选择使用某个专业公司的“SDK”，把ERP系统连接上该SDK的API接口，就可以使用SDK软件包里的一些功能。

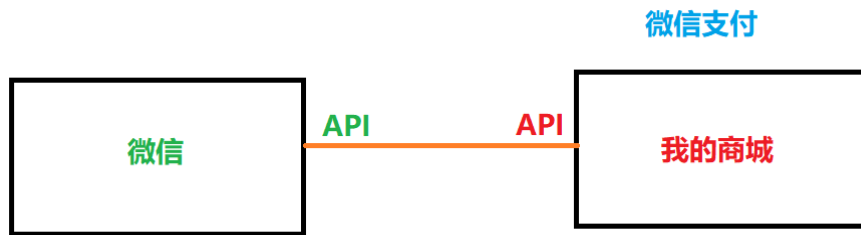
在实际开发中，比如京东和顺丰作为两家独立的公司，为什么会在京东网站上实时看到顺丰网站所归总的快递信息，这就要用到API，当查看自己的快递信息时，京东网站利用顺丰网站提供的API接口，可以实时调取信息呈现在自己的网站上。在这个过程中顺丰网站就是SDK，顺丰网站配备API接口可以对接外部的系统让京东不需要自己去弄一个物流团队和开发物流软件就能够时时查看自己平台的货物流通信息。

- SDK包含了API，除此之外它还包括一些关于该API使用的说明文档，范例，各种其他必要的组件和软件，API。

1.3, 使用API的优点

- 我们在开发购物网站的时候，经常会遇到需要使用线上支付的情况，但是如果从0开始搭建线上支付环境，就要考虑开发，与银行等签署协议，测试线上环境，如何保障用户资金安全等问题，这样算下来那将是巨大的开销的过长的开发时间，这时解决的方式就是使用其他有关线上支付软件开发的专业公司，比如微信，微信是我国互联网巨头之一的腾讯科技开发的一款即时通讯软件，它经过有长久的软件线上活动并且它提供人们经常使用到的微信支付功能，这个功能也是经过了长期的检验，安全可靠并且有责任管理约束，有效提高了保障，并且它可以外接到其他系统中使用，除此之外另一巨头阿里开发的支付宝也具有专业的线上支付功能，而且也有庞大的用户，并且也是长期运行，具有可靠保护，并且它可以外接到其他系统中使用，那么此时我们和他们经过协商并购买该功能的接口开放权力能够引入到我们的商城系统中使用，其中的线上支付功能就是API，微信和支付宝就是SDK。我们虽然接入了SDK但是我们只能用支付功能，所以可以把该API接口看成是支付功能，我们可以调用支付功能里面提供给我们的收款，转账等方法。

- 很好的例子就是当我们使用淘宝购物完成的时候提交订单时我们可以选择使用微信支付，支付宝支付或银行支付就是调用了他们提供API。

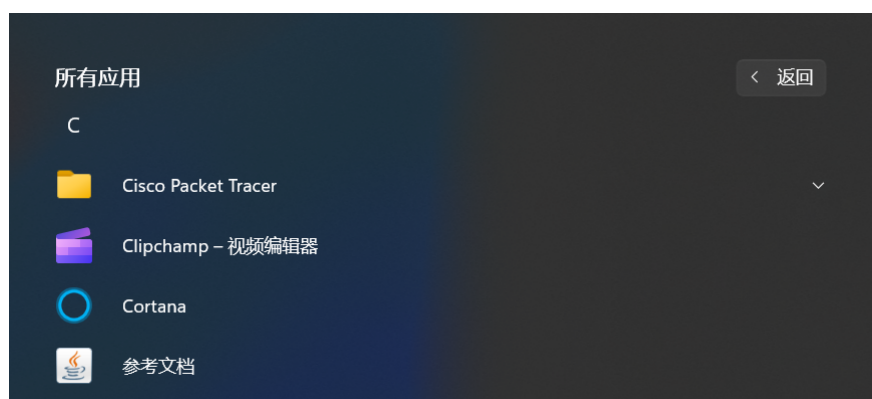


- 由此我们也可以知道API的优点就是
 1. 屏蔽了很多底层的细节，我们不需要去了解很深的内部实现机制，只需要知道这我们为什么需要这个API然后调用实现就行了。
 2. 增强了软件的复用性，就如上面的例子，如果说我开发的商城小程序就想要在线上支付的话，那么我就需要自己去写一个类似于微信支付的功能，这就非常麻烦，然而当我知道微信里面提供了这样的一个功能，并且它的这个接口是开放的任何程序如果需要都能够调用，此时我就会把它引入到我的程序当中直接使用即可。

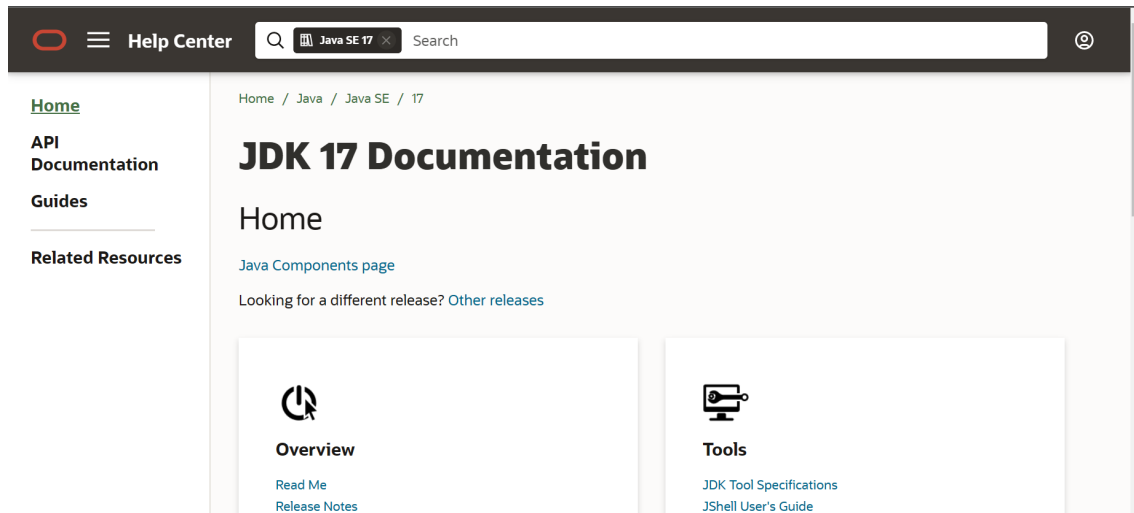
1.4, JavaAPI

- JavaAPI就是Java的应用程序接口，也称为库，这个库里面包含了为Java开发所预定义的一些类和接口，目前api任然在扩展当中，我们第三阶段将学习到java的一些核心的API的使用。上面所举得例子就是最明显的一个例子。
- Java实际上它和微信一样本质上它也是一种软件也就是应用程序，它的用途就是为专业的软件开发者提供一种开发的环境和工具，所以Java里面包含了许多开发Java的人员为我们这些使用Java进行开发的人员可能在开发的过程中会使用到的一些功能，这就是所预定义的一些类和接口。比如Scanner类中的nextLine方法。
- 上面我们说过了SDK，那么JDK实际上和SDK的意思是一样的，我们之前分析过JDK的文件目录，在JDK中包含了许多目录，在jre目录里面实际上存在一个lib目录，它是Java 运行环境所使用的核心类库、属性设置和资源文件。其中. rt.jar —— 引导类（运行时（RunTime）的类，包含了Java 平台的核心API），charsets.jar —— 字符转换类，除此之外在JDK中还存在一个lib文件目录，其中的tools.jar：包含了在JDK中工具和实用工具支持的非核心API，除此之外JDK中还包含了许多其他的说文文件和SDK的定义范围一致。
- 对于JDK来说我们开发人员最为关注的一点就是JavaAPI文档，JavaAPI文档记录了JavaAPI的一些用法，配置和使用环境可以说是JavaAPI的说明书，JavaAPI文档目前需要在官方网站上查阅，它以网页的形式展示。
- 如何查看JavaAPI文档？

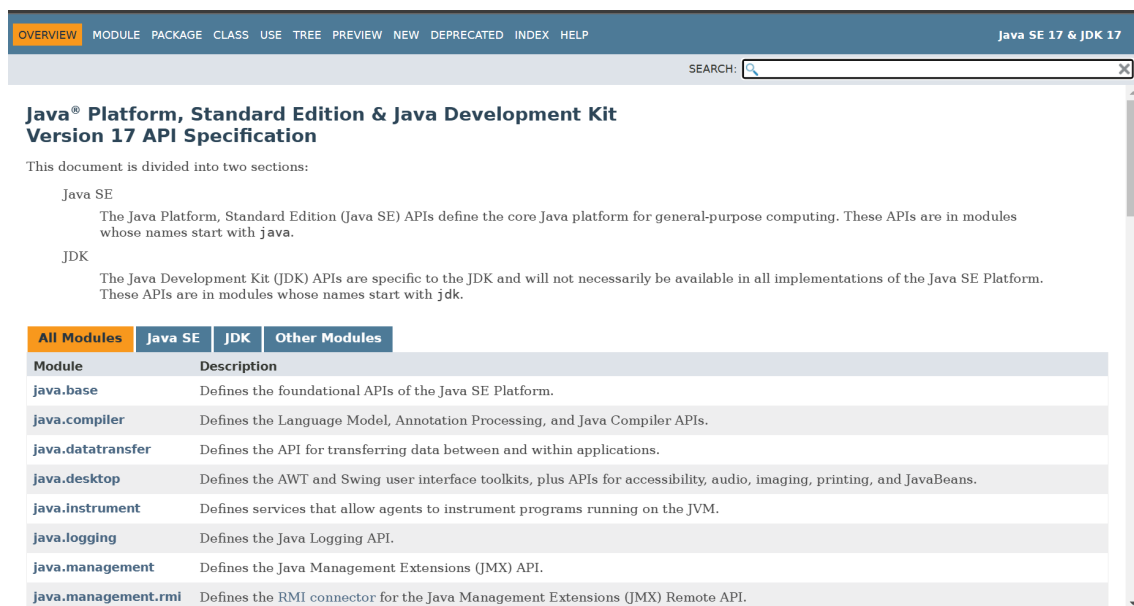
在Windows中，当我们下载了对应的JDK之后它会有一个对应的参考文档，这个参考文档实际上是一个url链接，我们可以在所有应用的C栏中找到。



点击参考文档会跳转到对应参考版本的网页中，比如我的跳转到了<https://docs.oracle.com/en/java/javase/17/>。



点击API Documentation即可查看对应的API文档。



对于文档现在只需要知道如何打开即可，在第三阶段中将具体的介绍文档的一些内容和如何阅读文档。

目前官方的文档只有英文版的，所以看官方的文档需要一定的英语阅读水平，当然对于程序员来说英语是一项重要的基础素养，需要我们慢慢的积累和培养，计算机学科实际上最早可以追溯到我们祖先使用的算盘工具，随着工业革命和第二次世界大战后的世界秩序，第一台现代的计算机在美国诞生，所以对于计算机学科来说很多的一些前沿科技和文档都是英文的，除此之外，英文作为使用最广泛的语言和最适合编程的语言来说都是计算机需要具备英语素养的原因。当然如果看不懂可以使用浏览器自带的翻译进行翻译，翻译出来的内容可能并不怎么专业，所以最好是自己能翻译就翻译不能翻译用浏览器自带翻译也是不错的选择。

使用线上文档查看的方式始终具有局限性，当出现断网等情况的时候我们就无法查看了，所以我们可以将API文档下载下来查看，当然下载的方式多种多样，可以去网上查找别人下好的（可以找到中文版或者是使用chm格式制作好的类似HTML查看器），也可以直接去官网进行下载。

现在我想去官网进行下载，进入下载JDK的网址<https://www.oracle.com/java/technologies/downloads/>。

选择想要下载文档的JDK版本，可以看到有一个文档下载按钮如下：

Documentation Download

然后它会下载一个.zip的文件到浏览器的默认下载目录，将该文件移动到非系统盘的一个文件目录中，然后使用解压工具解压即可。打开doc文件目录最终会出现如下所示的内容。

名称	修改日期
api	2023/1/19 17:48
legal	2023/1/19 17:48
resources	2023/1/19 17:48
specs	2023/1/19 17:48
index.html	2023/1/19 17:46

这些文件其实就是我们在线上看到的网页文件，只是下载到了我们本地而不是通过服务器发送过来的，index.html通常在web前端开发中作为网站内容的首页，我们只需要点击index.html文件即可像在线上看到的那样阅读文档。

当然这种直接下载的方式解压出来的文件有200M左右，可能有些大，当然也可以使用CHM工具进行转换，转换之后它的大小就是下载的压缩包的大小。

2，使用IDE开发Java程序

在之前已经介绍过了关于IDE的相关知识，同时开发Java程序我们主要使用到的IDE是eclipse和idea，所以现在来具体的介绍一下使用这两款IDE开发JavaSE程序。

2.1，路径

路径是操作系统的相关知识，在这里需要知道的有关路径的相关知识。

- 什么是路径？

路径可以说是一个通用的名词了，在每个地方它都有不同的含义，在计算机网络中路径是指从起点到终点的全部路由，在数据结构中路径一般指的是图的路径它表示由顶点和相邻顶点序偶构成的边所形成的序列，在日常生活中指的是道路。

我们这里的路径和上面的路径都不是一个概念，我们所说的路径更具体应该称为本机路径或者是物理路径。当我们在使用计算机的时候我们必须知道文件的位置，而表示文件位置的方式就是使用路径

比如在Windows下，我们去到JDK17的文件夹下，点击文件路径栏，可以看到。



上面的D:\ProgramFiles\JDK17就是JDK17文件夹所在的位置，也就是是JDK17文件夹的路径。

- Windows路径和Linux路径

对于不同操作系统的电脑来说，它们的路径表示是不同的。在实际开发中我们经常使用到的操作系统是Windows操作系统，Linux操作系统（服务器常用的RedHat，CentOS和我们的手机Android操作系统是基于Linux的），macOS操作系统（苹果系列的产品基于类Unix系统）。

- Windows文件和文件路径表示

在Windows操作系统中我们会经常使用C，D这样的盘来安装软件，在安装软件的时候我们一般是在D盘下面新建一个文件夹，然后当安装下载之后我们使用安装向导安装软件的时候会出现自定义安装选项，我们可以选择刚刚创建的目录然后进行安装，它的文件的格式是"盘符:\文件名"，比如"D:\ProgramFiles\JDK17"

其实文件的路径和操作系统中的文件管理有关，对于Windows操作系统来说，文件的管理属于多根目录的方式的，我们可以直观的感受到的根就是每个盘的盘符比如C和D，并且它的文件结构是树状的，比如说当我使用cmd进入到C盘下也就是在C根下，输入tree时就能够把C盘的所有的目录和文件都以树状的形式列出。


```
C:\>tree
文件夹 PATH 列表
卷序列号为 2AF7-8D2E
C:.
|--Intel
|--Kingsoft
|   |--office6
|   |   |--kdocermsoplugins
|   |   |--networkresult
|--PerfLogs
|--Program Files
```

当我们进入到一个具体的目录时也能够使用tree列出该目录下的内容。比如进入到D:\ProgramFiles\JDK17，这是因为Windows的文件结构是采用树状的文件结构。

```
D:\ProgramFiles\JDK17>tree
卷 新加卷 的文件夹 PATH 列表
卷序列号为 96B0-BA93
D:.
|--bin
|   |--server
|--conf
|   |--management
|   |--security
|   |   |--policy
|   |   |   |--limited
|   |   |   |--unlimited
```

Linux文件和文件路径表示

在Linux操作系统中没有所谓的盘符和C盘以及D盘，它的文件管理方式是单根目录的方式，它的文件结构也是树状结构，它的根文件从/开始。

它的文件路径格式是"/文件名"，比如"etc/sysconfig/network-scripts/"。

```
[root@localhost /]# cd etc/sysconfig
[root@localhost sysconfig]# ls
anaconda  console  ebttables-config  init  iptables-config  irqbalance  man-db  network  readonly-root  selinux
authconfig  cpupower  firewallld  ip6tables-config  kdump  modules  network-scripts  rsyslog  sshd
cbq  crond  grub  iptables-config  kernel  netconsole  rdisc  run-parts  wpa_supplicant
```

其实Linux操作系统是我们平时使用最多的操作系统，只是我们总会觉得计算机使用的是Windows操作系统，其实Linux操作系统在桌面操作系统中并不是主流的，但是它在服务器，嵌入式领域等用的特别的多，我们平时使用的手机不管是什么品牌的，基本上都是Android，Android操作系统是智能手机的主流操作系统，它的底层实际上是基于Linux的，那么我们平时使用手机的时候下载软件直接到应用商店进行下载即可，我们不需要关注这个软件它安装到哪里，但是我们点击手机APP的时候它就能进入使用。实际上如果我们去看手机的文件管理会发现它并没有所谓的C盘和D盘，因为它是基于Linux的操作系统，也就是说它是从/出发的一颗树，它承载了Linux的文件体系。这种方式实际上对普通用户是友好的，如果是一个完全不懂计算机的普通人都能够使用这样的系统而不用去关心文件在哪里的问题。

• 路径的分类

◦ 绝对路径

目标文件在硬盘上的真实的唯一的路径。

比如在Windows中的"D:\ProgramFiles\JDK17",这个位置在Windows中是唯一的，我可以通过这个唯一的位置找到该文件。

可以这样理解绝对路径，比如我有一个相隔多年的好友，这天他要我去参加他的婚礼，此时他会发给我"地球，中国，xx省，xx市，xx区，xx街道，xx饭店，xx厅，xx座位号"，此时这个位置在地球上唯一的指明了我要去坐的座位号。

在网络中的URL路径中"<https://www.baidu.com>"在网络上唯一的指明了该网站的位置。

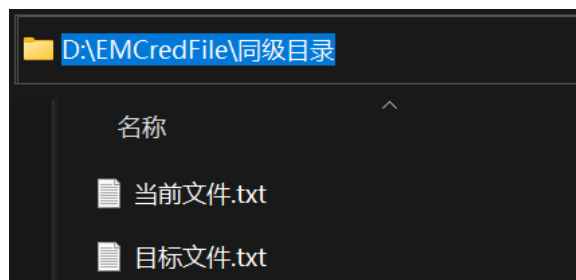
在实际的开发中，绝对路径的使用实际上还是很少的，因为我们开发的软件需要部署到服务器，智能手机，Windows端，Mac端等，如果我在Windows中开发的程序使用到了绝对路径比如"D:/project/images/cat.jpg"那么当我们部署到Linux的服务器上时就会出现Linux并不认识所谓的D，这时项目就无法被部署。

- 相对路径

相对于当前文件位置的路径。也就是说是从当前文件位置出发找到目标文件的过程。这里需要注意的是当前文件指的是打开正在使用的文件，比如我正在编辑Hello.java的源文件代码，此时的Hello.java就是当前文件，假如说我想在当前的代码中引入一个外部的文件，此时我们应该首先去看目标文件所在的位置，然后判断它相对于当前文件是什么等级的。一共可以分为三种等级。

- 同级目录

当前文件和目标文件在同一文件夹下。



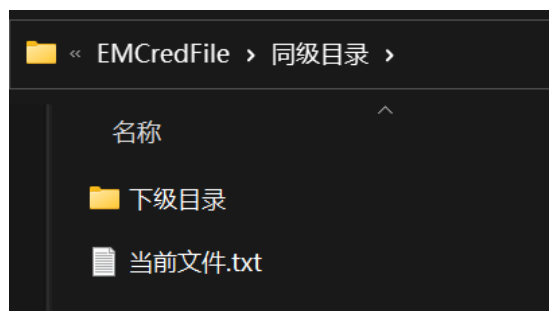
上述当前文件和目标文件在同一个文件夹下，所以它属于同级目录，如果要引用目标文件，可以使用。

```
"目标文件.txt"
"./目标文件.txt"
```

""."所代表的就是当前文件夹的意思，"/"代表进入的意思，"./"代表进入当前文件夹的意思。如果在同级目录下也可以省略。

- 下级目录

当前文件夹和包含目标文件的文件夹在同一个文件夹下。



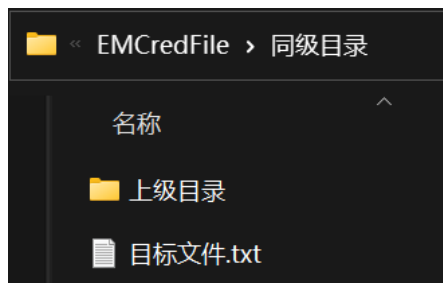
上述当前文件和包含目标文件的文件夹在同一个文件夹下，所以它属于同级目录，如果要引用目标文件，可以使用。

```
"./同级目录/下级目录/目标文件.txt"
```

首先进入当前文件所在文件夹，然后进入目标文件所在文件夹。

- 上级目录

包含当前文件所在文件夹和目标文件在同一文件夹下。



上述当前文件的所在文件夹和目标文件在同一个文件夹下，所以它属于同级目录，如果要引用目标文件，可以使用。

```
"../上级目录/目标文件.txt"
```

首先要从所在文件夹出来，然后找到目标文件进行引用。

".."表示返回到上一级目录。

在实际的开发中经常使用到该路径形式开发程序，我们在开发的时候都是将项目相关的内容放置在一个项目下的，也就是一个文件夹下，无论该文件夹是部署在Linux下的还是Windows下的，如果使用了相对路径的话，也能正常的找到对应的内容，但是这里会发现一个问题Windows的进入是"/"而Linux的是"\",在实际开发中根据上传到的操作系统进行转换即可。主要的是它至少能够找得到。

其实理解得更深入一点，为什么它能够找到呢，实际上是使用了命令的方式Windows中的可以使用cd .命令进入当前文件夹也就是它本身，使用cd ..退出当前文件夹到上一级，cd Desktop/hello，进入到当前文件夹下的Desktop文件夹下的hello文件夹，而在Linux中也是使用该命令进入退出的。

Windows下：

```
D:\>cd .  
  
D:\>cd EMCred  
  
D:\EMCred>cd..  
  
D:\>|
```

Linux下：

```
[root@localhost ~]# cd .  
[root@localhost ~]# cd etc  
[root@localhost etc]# cd ..  
[root@localhost ~]#
```

2.2, IDE搭建Java程序的基本结构

对于JavaSE的程序来说，一般分为三步

- 新建Java项目 (project)

对于Java项目的命名使用的是项目的名称，一般使用英文的名称。

在实际开发中一个项目或者说一个工程，它们都不是单独的开发人员完成的，一个项目是由一个项目负责组所承担的，比如和平精英这款游戏的开发项目承担是由光子工作室所承担的。

项目可以比作是小区，比如xx小区一期工程，也就是说这个项目可以根据市场的需求进行版本升级就是xx小区二期工程了。

- 新建Java包 (package)
 - 什么是包？

在生活中包的含义就是容纳特定物品的容器，比如书包，包裹。如果严谨的使用来说书包是用来装与学习相关的物品比如一系列课本的文具，包裹是用来装我们邮寄给他人的物品。

在Java中包和生活所理解的包含义是一样的，它所装的物品是功能相似或相关的一组类和接口。这是Java所提供的一种机制。

如果说项目是在建小区，那么包就相当于这个小区里的一栋单元楼。

- 为什么要使用包？

为了更好地组织类，Java 提供了包机制，用于区别类名的命名空间。

这里需要知道的是Java是使用类，接口，枚举和注解作为开发的基本单位的。在同一文件目录下，命名这些类，接口，枚举和注解的时候它们的名称是不能够重复的。这一点实际上我们平时在使用Windows的时候，如果在同一个文件夹下命名了相同文件名和后缀名的文件的时候，Windows会跳出提示是否替换原来的文件，其实无论是什么操作系统，在同一文件夹下这样的同名文件同类型文件是不存在的，这违反了操作系统所采用的磁盘管理格式的规定。

所以包其实可以总结为：

- 把功能相似或相关的类或接口组织在同一个包中，方便类的查找和使用。也就是把不同的 java 程序分类保存，更方便的被其他 java 程序调用。提高了软件的可维护性和复用性。
- 如同文件夹一样，包也采用了树形目录的存储方式。同一个包中的类名字是不同的，不同的包中的类的名字是可以相同的，当同时调用两个不同包中相同类名的类时，应该加上包名加以区别。因此，包可以避免名字冲突。实际上包就是一个文件夹。
- 包也限定了访问权限，拥有包访问权限的类才能访问某个包中的类。Java提供了访问修饰关键字来决定Java的访问权限，其中包访问权限为默认访问权限，在第二阶段将会学习到相关内容。

- 包的创建和使用

- package关键字声明包

关键字的讲解在后面的学习当中会详细讲解到。

package在英语中的意思为包和包裹，所以直接使用package声明所在的包。

比如之前提到过的Scanner类，它是Java所提供的的一个API，在它的源码中可以看到。

```
package java.util;
import java.io.*;
import java.math.*;
import java.nio.*;
import java.nio.channels.*;
import java.nio.charset.*;
import java.nio.file.Path;
import java.nio.file.Files;
import java.text.*;
import java.text.spi.NumberFormatProvider;
import java.util.function.Consumer;
import java.util.regex.*;
import java.util.stream.Stream;
import java.util.stream.StreamSupport;
import sun.util.locale.provider.LocaleProviderAdapter;
import sun.util.locale.provider.ResourceBundleBasedAdapter;
public final class Scanner implements Iterator<String>, Closeable
{.....}
```

可以看到在程序的最开始有一个package java.util;包实际上就是个文件夹那么它的路径是java/util/Scanner.java。

在我们的实际开发当中，上面说到一个项目是由一个项目组所负责的，在项目组中开发组的工作往往就是使用代码编写程序并在计算机中进行运行，一般使用Java语言作为开发的时候，常常需要给每个开发组的成员划分模块进行开发，每个模块由1到多个开发人员进行负责，而这里的模块就是包。

可以这样进行比喻，就像是我们在一个工厂制造车辆一样，这一个个的包就是组装车辆的一个个的零件，我们将这些零件生产出来然后进行组装成为我们自己品牌的一辆汽车。

当然Java的开发机制目前已经支持模块化的开发方式，之前在JDK目录中讲过，在JDK9版本中引入了模块系统但是这个模块实际上包含有许许多多的包所组织而成的，实际上可以将这个大模块看作是一个构件，比如：在组装汽车的时候，我们不需要从0开始把发动机搭建出来而是直接从发动机厂商购买发动机进行组装，其中构成发动机的每一个零件就相当于模块（包），而发动机这个整体就相当于一个构建（模块系统），它和其他构建相互组成为一辆汽车。

- 在实际的开发规范中，对包的命名需要做到

为了尽量使包名保持唯一性，包名通常采用小写、按倒写互联网址的形式进行定义。

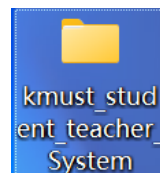
比如我们学校的名字是昆明理工大学，我们学校的官网是www.kmust.edu.cn，现在我们学校要使用Java作为后台开发语言开发一款学校教务系统，此时我们开发组接到该任务，我所负责的模块是student，所以我所使用的包名应该是cn.edu.kmust.student。

为什么可以使用互联网地址的形式作为包的命名方式呢？实际上我们所看到的www.kmust.edu.cn是互联网上的域名，在互联网中，每个域名都是唯一的，都要到对应的域名管理中心购买该域名进行使用，比如京东的域名现在是www.jd.com但是在之前这个域名是被其他的网站注册过的，京东原来的域名www.360buy.com，这个域名可以说跟京东没有任何的命名含义，所以为了获得jd这个名称，京东花了很多钱才将这个域名买过来。域名的名称是重要的，它需要包含网站的含义。正是因为域名的唯一性，并且每个有规模的公司或学校等组织目前都有自己的网站，所以为它们开发项目的时候常常使用这样的方式来保持包名的唯一性，否则如果清华大学也使用student作为一个模块的话当清华学生到昆工学生进行交流的时候就会出现包名冲突。

- 创建包

就拿上面的那个例子来说，我们使用最本质的方式创建包。

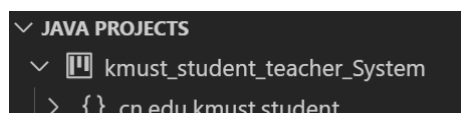
首选创建一个项目名，学生教师管理系统项目。



然后创建一个cn的文件夹，在cn中创建一个edu的文件夹，在edu的文件夹中创建一个kmust的文件夹，在kmust中创建student文件夹，(cn.edu.kmust.student)包中假如创建了一个学生成绩的Grades.java最后可以看到路径如下。

C:\Users\EMCred\Desktop\kmust_student_teacher_System\cn\edu\kmust\student

将该项目使用VScode打开，可以看到。



在Grades.java中它的写法是：

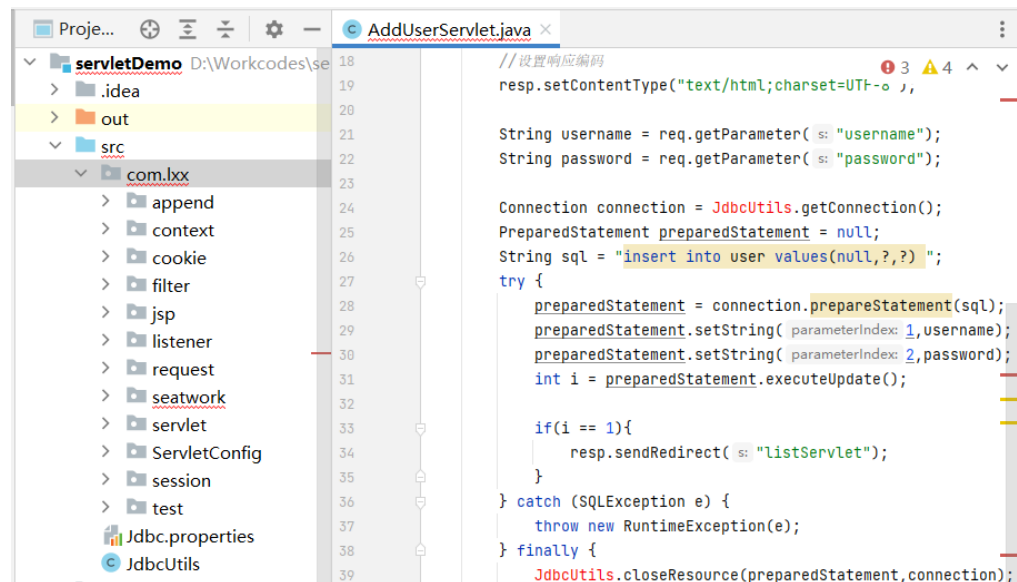
```
package cn.edu.kmust.student;
```

```
public class Grades{  
    .....  
}
```

包声明应该在源文件的第一行，每个源文件只能有一个包声明，这个文件中的每个类型都应用于它。

■ 无名包

所谓的无名包就是没有名字的包，对于这种包中的类或者是接口是无法被其他的类所直接引用的。比如，之后我们会学习在IDEA中创建项目，而包是放在src文件夹下的，如果在src文件夹下单独创建了一个.java文件的话它就属于无名包的范畴，而在其他包中的类将无法引用到它，下面的类将引用不到在包外创建的类。



所以在实际的开发项目中我们需要避免使用无名包。

■ import关键字导入包

import在英语中的意思是导入，引进，移入，进口的意思。

直接使用import关键字导入包。在上面我们讲过有关API的相关内容，那么连接两个API之间的那根线就是import。

在上面我们使用的ApiDemo的例子中我们直接使用java.util.Scanner 引入Scanner类，这其实就是用一根线将我们的程序和Java的API连接了起来。但是这种写法显得代码特别的冗余，在实际的开发中我们会在我们的程序中使用到大量的Java提供的API，如果都像上面那种方式去写的话，在一定程度上会出现大量冗余的代码使得代码的可读性变差，为了解决这个问题就引入了import。

import是写在package下的，它会说明用到了那些API。比如上面的ApiDemo程序，在该程序中使用到了Scanner类，它属于java.util包下的内容，所以如果使用import的话，它的代码形式为。

```

package codes
import java.util.Scanner;
public class ApiDemo {
    public static void main(String[] args) {
        //scanner在java.util包下通过该方式引用Scanner类，就相当于把线接起来
        Scanner scanner = new Scanner(System.in); //获取输入
        String input = scanner.nextLine(); //将输入内容以字符串类型存储
        System.out.println(input); //输出到控制台屏幕上
    }
}

```

这样就会显得十分的简洁，增强了代码的可读性，在实际开发中我们常常使用该关键字进行导包操作，实际上IDE会根据我们所写的类在自动的为我们进行导包的操作，我们不要去关注导包，所以慢慢的到后面开发中如果使用记事本开发时总会忘记导包这个操作。

现在或许会注意到为什么导入的只有java.util.Scanner呢，除了Scanner外，我们也并没有实现过String和System。为什么它们不用导入相应的包，其实这个问题的解答就是它们都是一些常用的操作，所以它们都被Java放在了java.lang包下，对于java.lang包来说它是默认导入的，也就是Java的**默认包**，也就是说我们不需要手动的去导入，我们一开始写程序的时候它就已经导入到了我们的程序中。这里关于Java所提供给我们的这些包的具体解释可以参看第三阶段相关内容。

实际上import只是编译器所认可的，那么什么是编译器所认可的呢？其实这个概念不难理解，所谓的编译器认可就是说我们所使用的import关键字实际上在经过编译的时候，编译器会帮助我们将代码还原成原始的模式然后生成字节码文件，比如上述的import java.util.Scanner;最终还是会被还原为使用到的类的地方也就是java.util.Scanner scanner = new Scanner(System.in);这样的形式，也就是说上面的代码会还原为没有使用import关键字的代码程序（也就是我的第一版的ApiDemo的写法）。编译器的认可模式在Java中采用的是十分广泛的一种模式，比如泛型，新循环等。到后面还会提到。

所以它所导入的实际上只是一个名称，它只会让编译器在编译这个java文件时把没有姓的类别加上姓，并不会把别的文件程序写进来。我们可以将java.util.Scanner中的util，也就是它所在的包的包名看作是姓，而Scanner看作是名，它就像我们的姓名一样，我们的姓就是百家姓中的一个，而名我们可以起的不同多种多样，比如张三，张思，当然也有叫李三，李思的，但是如果在1班有一个叫张思的，在2班也有一个叫做张思，我在走廊叫张思显然是不行的，但是我可以到要找的那个张思的门口去叫他，Java也是一个道理，它需要在一个具体的位置，而java.util.Scanner中的java就是那个位置，可能我们还会见到util.Scanner但是前缀会是sun.xxx...,javax.xxx...。在实际中1班也可能会有两个叫张远的，但是在Java里面这是不允许出现的。

这个姓也称为简名，它具有编译单元作用域。所谓的作用域也就是它所起作用的范围，对于简名来说它所起作用的范围是当前文件的任何位置。

其实对于import来说有两种导包模式：

- 单类型导入

所谓的单类型导入就是导入到具体的某个需要使用的类或接口，比如上述所导入的java.util.Scanner它直接指明了需要导入的是Scanner类。

- 按需类型导入

按需导入就是将某个包中的类或接口全部导入到程序当中。比如上述的java.util.Scanner，我们知道Scanner类在java.util包下，所以我们可以使用java.util.*将util中的所有类和接口名导入到我们的程序中，我们按需求进行选取使用。此时程序可以写成这样。

```
package codes
import java.util.*;
public class ApiDemo {
    public static void main(String[] args) {
        //scanner在java.util包下通过该方式引用Scanner类，就相当于把线
        接起来
        Scanner scanner = new Scanner(System.in); //获取输入
        String input = scanner.nextLine(); //将输入内容以字符串类型存
        储
        System.out.println(input); //输出到控制台屏幕上
    }
}
```

对于以上的两种导包模式实际上我们经常使用的是第一种模式，第二种模式往往在实际开发中不会经常使用的，一般使用在一些demo中。为什么呢？

编译速度：在一个很大的项目中，它们会极大的影响编译速度.但在小型项目中使用在编译时间上可以忽略不计。

命名冲突：解决避免命名冲突问题的答案就是使用全名。而按需导入恰恰就是使用导入声明初衷的否定。

说明问题：毕竟高级语言的代码是给人看的，按需导入看不出使用到的具体类型。

无名包问题：如果在编译单元的顶部没有包声明，Java编译器首选会从无名包中搜索一个类型，然后才是按需类型声明。如果有命名冲突就会产生问题。

- 新建Java类 (class)

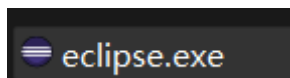
这里需要知道的是Java类是开发Java程序的最基本的单位，Java是由各种各样的类所组成的，同样用Java开发的程序也是由各种各样的类所组成的，对于Java类的具体阐述将在第二阶段进行说明。我们上述所写的ApiDemo或者是Scanner都是类。

当创建完包之后我们就需要实现具体的程序了，也就是写代码，而Java开发的基本单位是以类进行的，所以现在我们需要在包里面搭建各种各样的类。

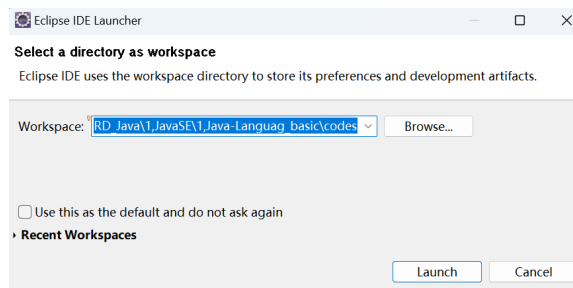
如果说包就是小区里的单元楼房的话，那么类就是这个楼房中的一间间的屋子。类就是具体实现整个Java程序的最基本开发单元。

2.3, 使用Eclipse开发Java程序 (Windows操作)

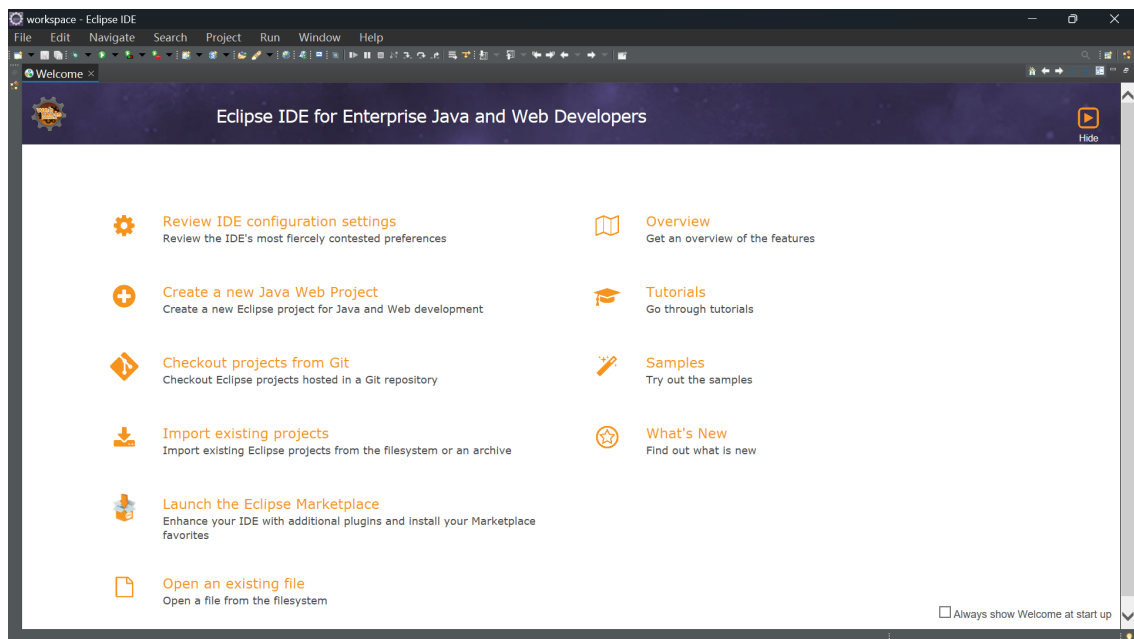
- 在之前，已经介绍过了有关Eclipse的相关知识，它的一款基于插件的IDE，当我们去官网下载下来之后，它就是个压缩包，我们只需要解压即可以使用。
- 解压之后进入它的文件目录中，找到它的可执行程序eclipse.exe，然后右键，发送到，桌面快捷程序。创建一个快捷链接到桌面上，当我们要使用的时候就不用每次都到该文件目录中寻找了。



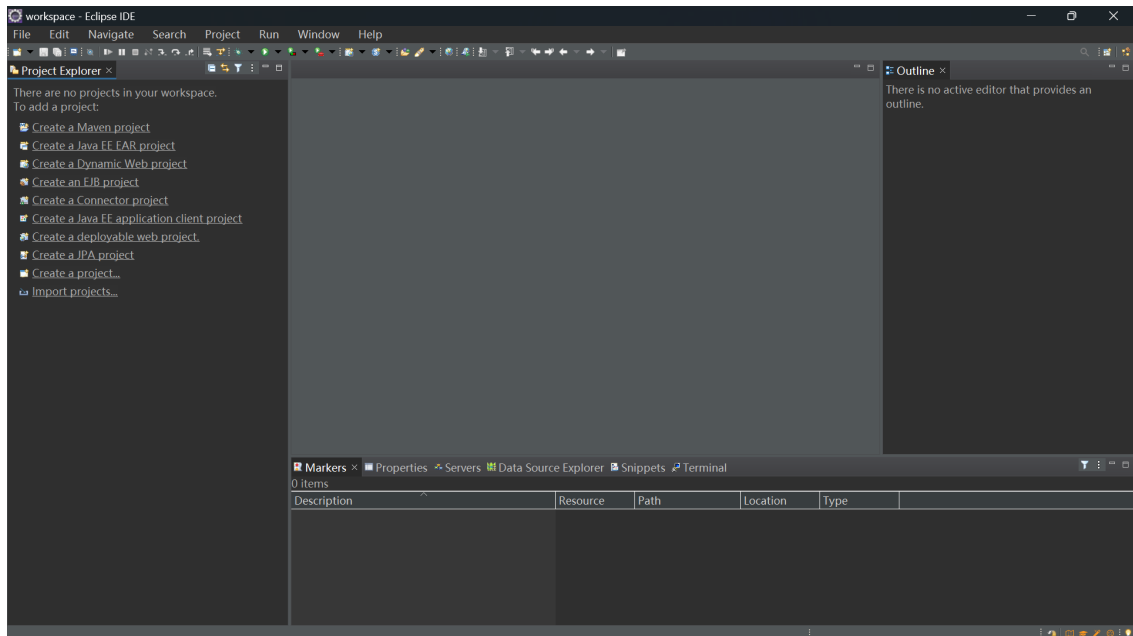
- 当我们第一次使用的时候，它会出现选择工作空间的选项。在Eclipse中存在一个称为工作空间的说法，所谓的工作空间实际上指的是我们所写的项目所放在的一个专门的文件夹中，这个文件夹就是所谓的工作空间，它的名称是eclipse-workspace，如果不选择eclipse默认会提供一个文件夹在C盘的用户文件夹的数据文件夹中创建eclipse-workspace文件夹，这里我一般是不用它默认的工作空间，而是首先会在非系统盘上，新建一个项目文件夹，当使用eclipse的时候直接将路径改为我的项目文件夹下使用。所以一般我不会勾选Use this as the default and do not ask again也就是说使用当前文件夹作为默认的工作目录下次点击的时候直接进入该目录中。当然在目前这个阶段，可以在非系统盘中新建一个eclipse-workspace然后勾选作为默认的使用空间也是完全可以的。



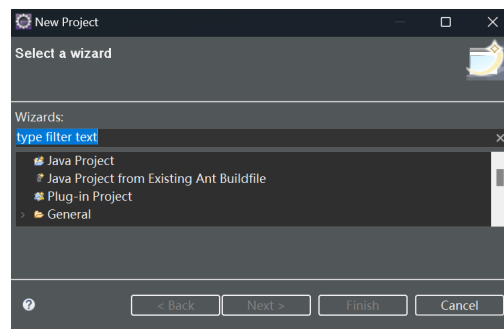
- 选择好工作空间之后，点击Launch也就是进入到工作空间。
- 第一次进入的时候会看到欢迎使用的页面。其实我们重新转换工作空间的时候它还是会出现的而且我们在之前的工作空间中的一些相关设置也会恢复默认值，需要重新的配置，这就是eclipse的灵活性，我们可以针对不同的工作空间进行设置。它的本质实际上它会在对应的工作空间中创建一个.metadata的文件夹，这个文件夹包含了当前工作空间当中的一些设置，而新的工作空间中并没有对应的文件夹，所以需要重新配置。



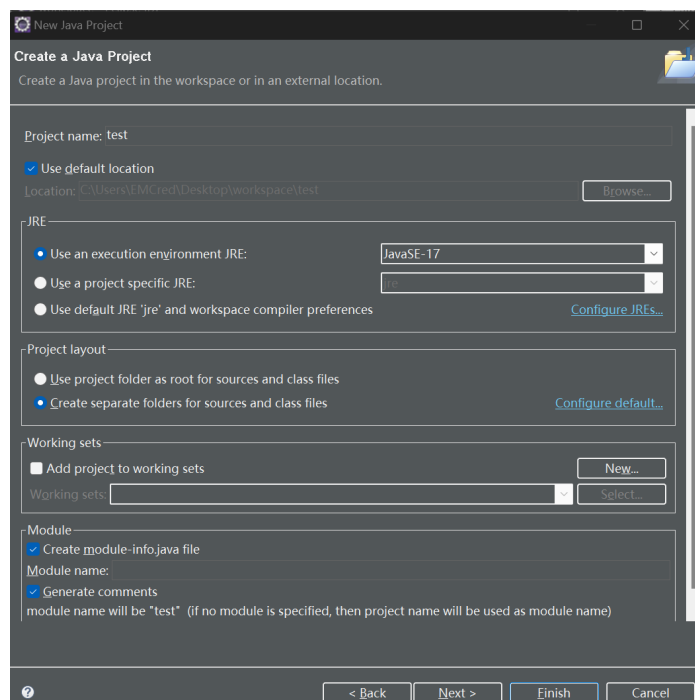
- 在该页面中Review IDE configuration setting就是配置该IDE的一些基本项，点击之后会出现6个基本项，Refresh Resources Automatically就是自动刷新资源（可选），Show Line Number in Editors在编辑器中展示行号（必选），Check Spelling in Text Editors在文本中检查拼写（可选），Execute Jobs in Background（可不选）Encode Text Files With UTF-8文本的编码使用UTF-8（可选）Enable PreferenceRecorder（可不选）然后点击Finish即可完成。Create a new Java web project就是创建javaweb的项目这属于JavaEE的内容，目前不会用到。Checkout projects from Git从Git中检测项目，git是一种协调开发的工具，也是软件开发人员必会的工具，在目前JavaSE阶段不会用到后面会详细的介绍git相关知识。import existing projects导入已经存在的项目，也就是说可以选择你之前已经开发好的项目导入到该工作空间中。Launch the Eclipse Marketplace登入到Eclipse的开源市场，我们知道Eclipse是基于插件的，所以可以登入到其中下载一些插件进行使用。我们找到welcome图标，位于左上角，将欢迎界面关闭即可进入IDE。



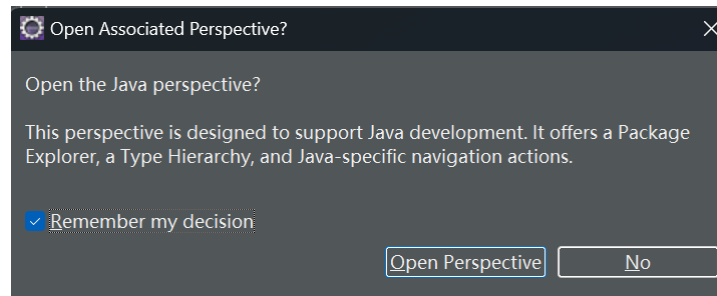
- 进入到IDE之后，我们看到左侧栏中可以创建很多的与Java相关的项目，但是在该阶段中，我们只需要创建Java的普通的项目即可，找到Create a project并点击。



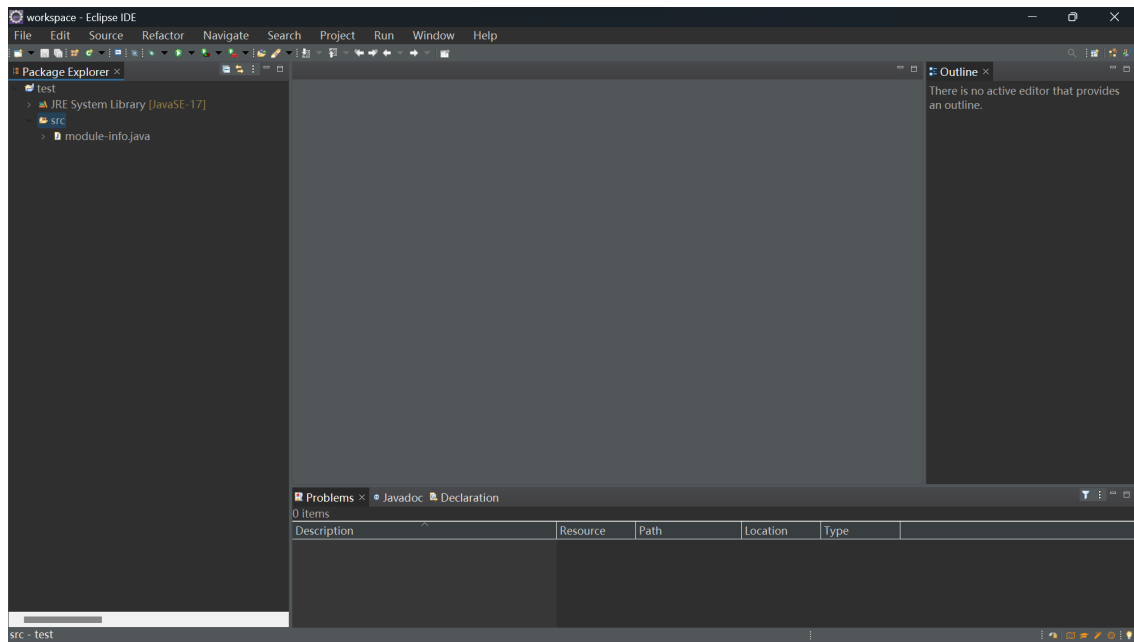
- 选择Java Project，点击Next就会看到JavaProject的创建引导栏，在该栏里的Project name填上该Java项目的名称，Use default location就是该项目的存放位置默认在所选择的工作区当中，JRE栏中Use an execution environment JRE就是选择对应的JRE版本，默认它会从环境变量中选择环境变量配置的JRE。Project layout是项目的文件布局方式，选择create separate folders for sources and class files也就是创建源代码文件和可执行字节码文件分别保存在不同的两个文件夹中的方式。Working sets是工作配置的意思，在该阶段不会使用，默认即可。Module是模块的意思，之前说过在JDK9的时候出现了一种模块系统，在该阶段不会使用，默认就行。点击Finish即可。



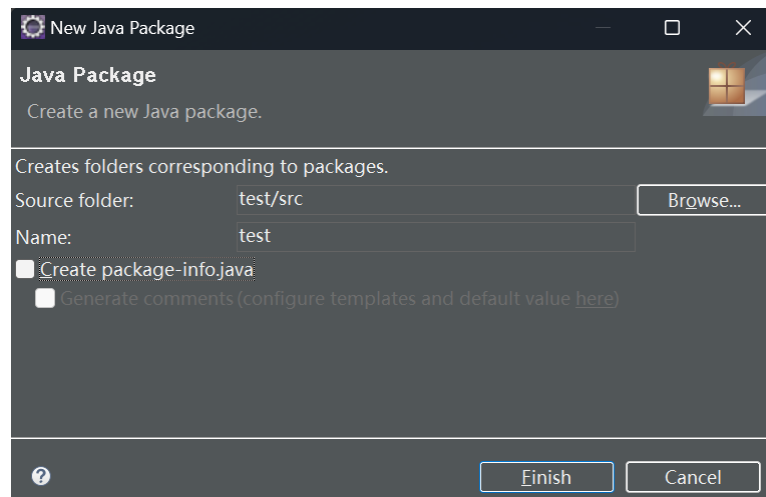
- 点击之后它会出现是否打开Java透视图的选项，所谓的透视图就是能够明确的看到该项目的结构，所以选择Remember my decision也就是记住我的选择，点击open perspective打开透视图。



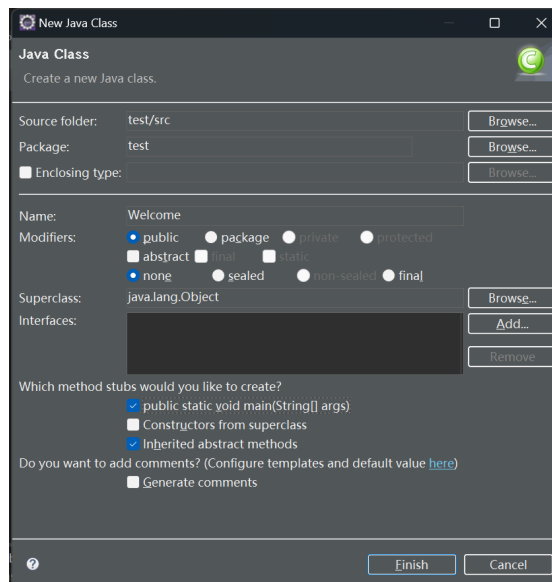
- 现在我们正式的进入到了该项目当中。



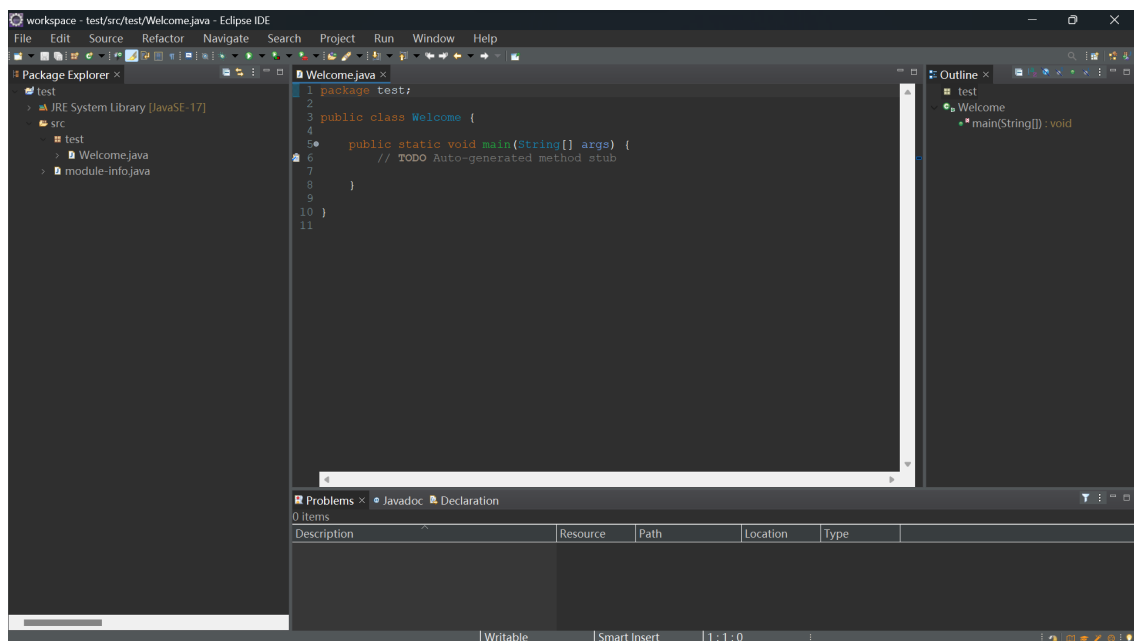
- 左侧栏中是项目的结构，我们的包和类文件都是搭建在src文件夹下的，src其实就是source的缩写，也就是项目的资源，接下来新建包，新建类，右键src，New，package。输入package的名称，create package-info.java是创建包信息的Java代码，是对包信息说明的Java文件，点击finish。



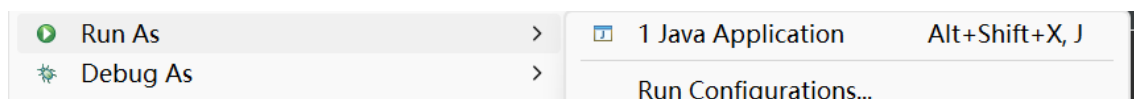
- 然后我们在包下，右键class，新建一个类为Welcome，在新建类的向导中的Name填上类的名称，我们可以看到它默认导入的类就是java.lang.Object，其他的选项默认即可，这里也可以勾选public static void main(String[] args)也就是默认会帮助你添加一个主方法。点击Finish。



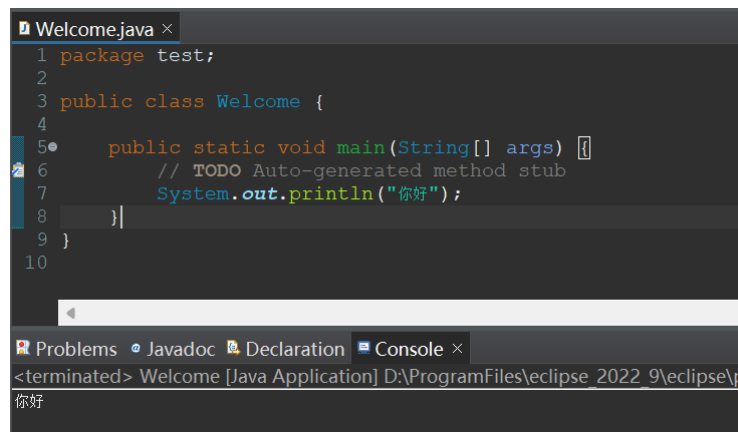
- 点击完Finish之后即可生成一个Welcome类。



- 可以看到左侧栏Package Explorer说明了该项目的结构，右侧栏outline时说明当前类中的结构，它只有一个主方法为main。
- 接下来是如何运行程序的问题，现在我们需要让它能够在控制台上输入“你好”，此时在主方法中写上System.out.println("你好");然后右键在Run as 中选择1 Java Application。



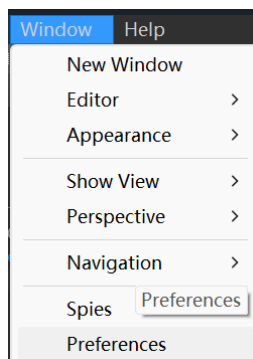
- 跳出的对话框点击“ok”即可，然后可以看到在下方的控制台中输出了对应的“你好”，其实在开发的时候首先要进行Debug也就是调试，看看程序有没有错误然后再进行实际的运行。



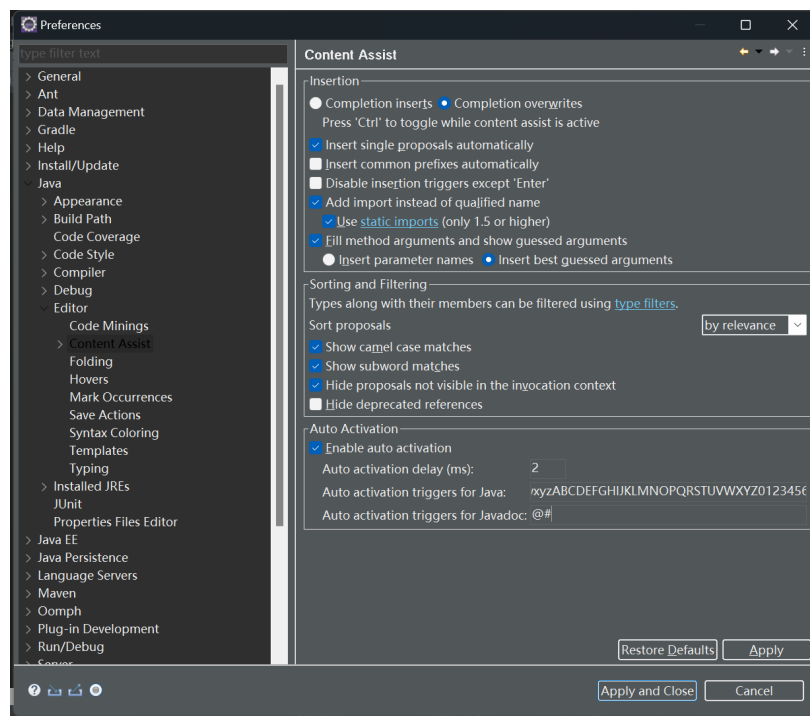
```
1 package test;
2
3 public class Welcome {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         System.out.println("你好");
8     }
9 }
10
```

Console: <terminated> Welcome [Java Application] D:\ProgramFiles\eclipse_2022_9\eclipse\...
你好

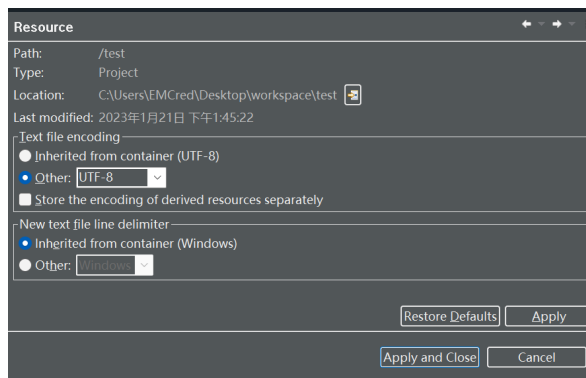
- 以上就是使用Eclipse搭建Java程序的基本步骤，在Eclipse中还可以做一个设置让我们编写代码的时候更加的顺畅，就是设置自动提示代码。
- 在IDE中的Windows选项中选择Preferences选项。



- 进入Java, Editor, Content Assist, 找到Auto Activation栏，设置Auto activation delay (ms) 自动提示响应的时间为2ms, Auto activation triggers for Java填入“abcdefghijklmnopqrstuvwxyzABCDE FGHIJKLMNOPQRSTUVWXYZ0123456789”也就是26个英文的小写字母和26个英文的大写字母以及10个阿拉伯数字。点击Apply，然后点击Apply and Close。



- 当我们写代码的时候它就会自动的进行提示，这样写起来就很方便了。
- 如果我们想看我们的项目所在的具体目录和更改字符编码的话，我们只需要在项目名哪里右键，点击Properties，就会出现以下的页面。



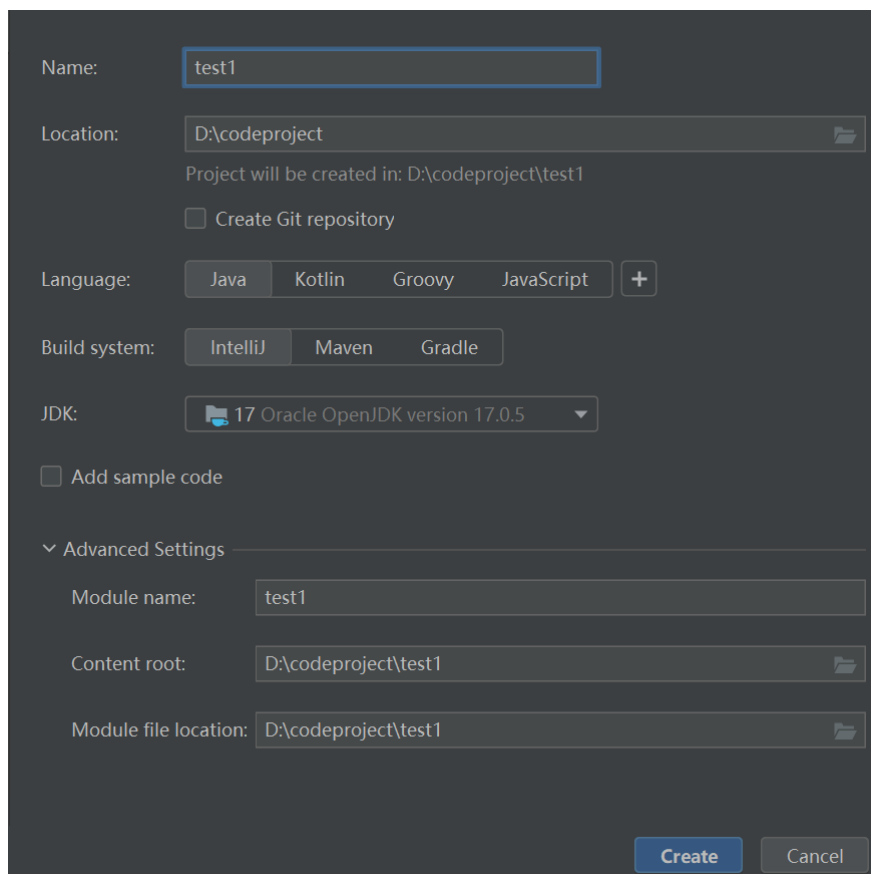
- 想要更改字符编码的话再Text file Encoding中的other的选择栏中进行更改即可。
- 想要看所在的具体目录，只需要在Location哪里点击转到的图标即可，我们可以看到在该项目中的目录结构就是我们刚刚所选的源文件和字节码文件存放在两个不同的文件夹中，其中的bin就是我们的字节码文件，bin文件夹一般在程序中指的是可执行文件夹，里面有一些二进制的可执行文件，而src就是源文件夹。

名称	修改日期	类型	大小
文件夹 .settings	2023/1/21 13:45	文件夹	
文件夹 bin	2023/1/21 13:58	文件夹	
文件夹 src	2023/1/21 13:58	文件夹	
文件 .classpath	2023/1/21 13:45	CLASSPATH 文件	1 KB
文件 .project	2023/1/21 13:45	PROJECT 文件	1 KB

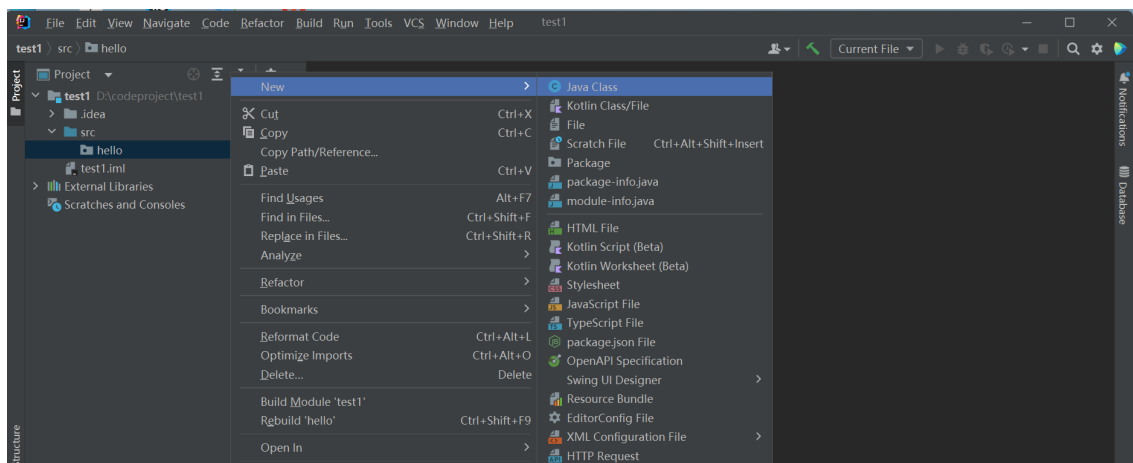
- 其实Eclipse的使用目前并不是主流的。虽然它不是主流的IDE但是它一般是入门学习的好工具，我以前一般在开发当中使用它撰写Demo和做一些笔记使用，除此之外它还是目前一些比赛的官方指定的IDE比如蓝桥杯。

2.4，使用IDEA开发基本的Java程序(Windows操作)

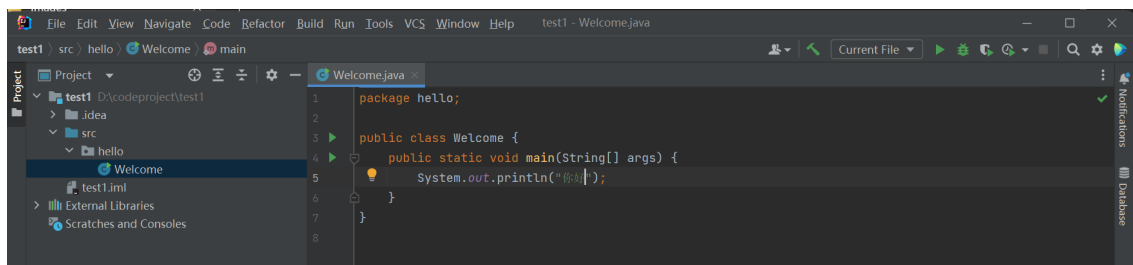
- 实际上，只要你会使用一种IDE其他的IDE也会很快就能学会如何去使用，IDEA作为目前主流的企业使用的开发工具，它自然也是具有特色的，它搭建Java项目的过程和一般的流程是一样的。它相对于Eclipse来说更好操作一些。
- 首先点击New project创建新项目。项目的Name就是项目名称，项目的位置Location进行选择，Create Git repository目前可以暂时不选择，language选择Java语言，Build System默认为IntelliJ的项目其他的Maven和Gradle是版本开发的项目，JDK选择我们本机安装的JDK版本，然后点击Create。



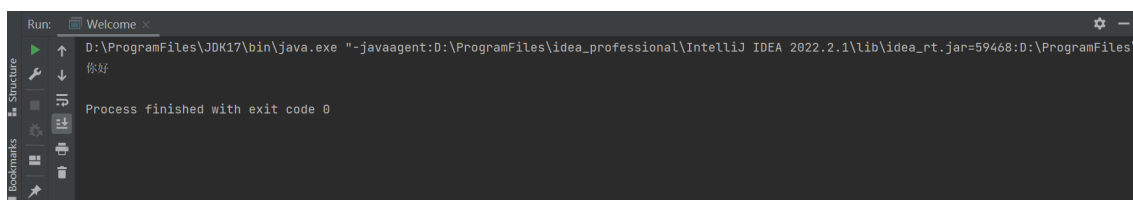
- 我们也可以像再Eclipse那样创建包和类。



- 它具有自动代码提示功能，并且它其实也是可以扩展插件的。

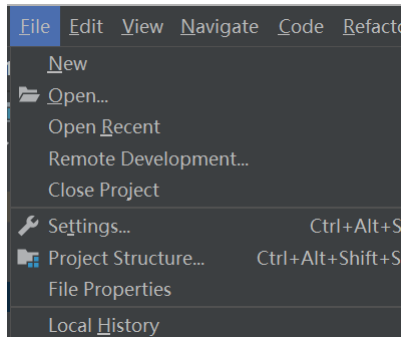


- 当我们运行或调试上述代码的时候只需要点击  即可运行或调试。

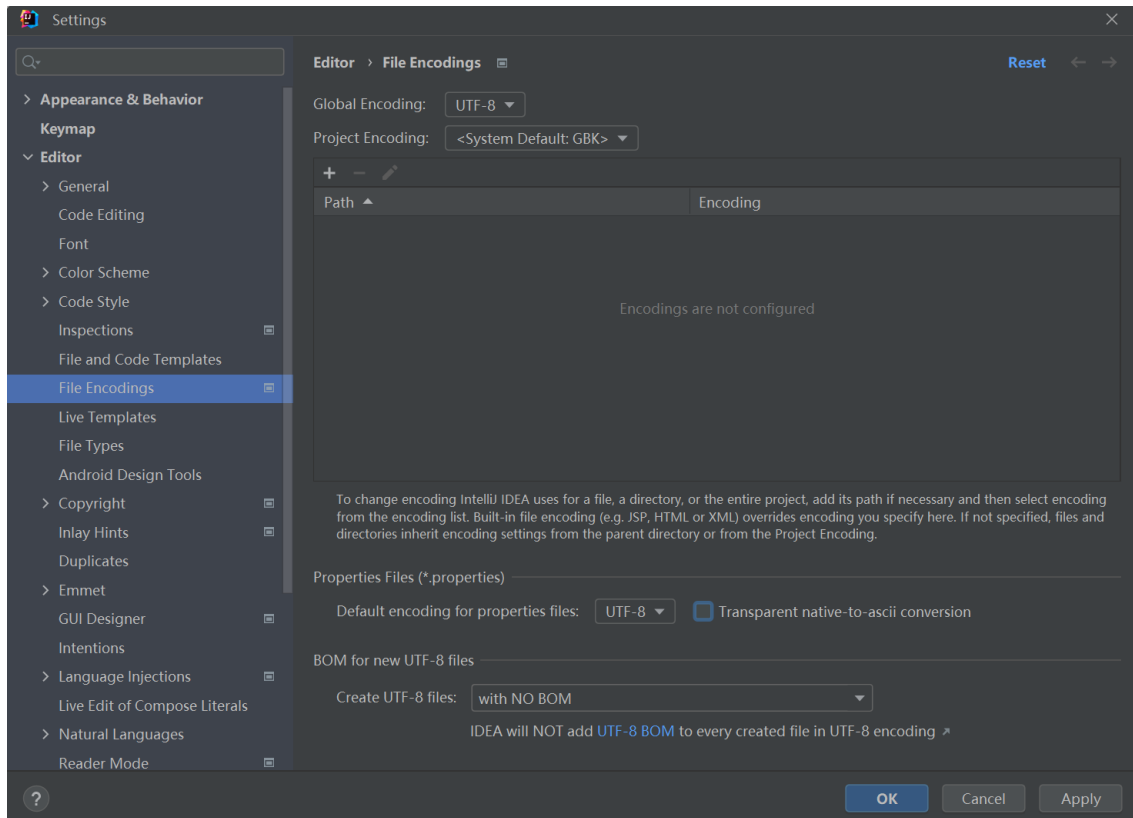


- 如果我们想看我们的项目所在的具体目录和更改字符编码的话。

- 如果想要更改字符编码，点击File，找到setting所在的位置。点击setting。



- 打开Editor，Code Style,File Encodings也就是文件的编码，Global Encoding是配置全局文件的编码，Project Encoding是本项目所采用的编码，政治与Properties文件的编码也默认选择为UTF-8的编码，选择不带BOM的UTF-8文件，点击Applay和OK。



- 如果想要查看当前项目文件夹所在的具体位置的话，右键项目，选择open in，Explorer。



- 当我们打开对应的文件夹之后可以看到。

名称	修改日期	类型	大小
.idea	2023/1/21 16:27	文件夹	
out	2023/1/21 15:57	文件夹	
src	2023/1/21 15:51	文件夹	
test1.iml	2023/1/21 15:50	IML 文件	1 KB

- 它实际上是默认帮助我们将源代码和字节码文件通过不同的文件夹进行分放的，out就是所输出的字节码的存放位置，实际上out指的是产生的资源文件夹，代码所产出的一些产品。我们在之后的学习当中主要会使用IDEA作为主要开发的开发工具。在之后的学习中我们会慢慢补充关于它的使用。

3, 程序设计错误

在我们设计程序的时候, 人无完人, 我们都会出现一些错误, 在程序中, 这些错误可以大分为以下三类。

3.1, 语法错误 (syntax error)

- 在编译过程中所出现的错误称为语法错误, 也可以称为编译错误。也就是说在我们书写源代码的过程中所产生的一些错误, 比如没在程序的结尾加上分号, 使用了中文的标点符号。
- 实际上现在我们使用IDE开发程序的时候, 这类错误可以直接被IDE标注出来。

```
//          语法错误在编译的时候就会由IDE自动的显示出来, 此时程序没有以分号结尾。  
int  a = 1
```

这里需要注意的是, 程序中的所有的程序设计标点符号都应该使用英文输入, 一些新手常常犯的错误就是使用了中文的一些标点符号, 导致程序出现错误。其实我们之前讲过关于字符编码的知识, 中文是采用双字节编码的而英文采用的是ASCII的单字节编码, 所以中文的标点符号和英文的是不一样的, 比如:我们切换到中文输入法下打出的", "和英文输入法下打出的","是不一样的。它明显要比英文的大一半。

3.2, 运行时错误 (runtime error)

- 程序通过了编译器语法的检测, 但是在程序运行的过程中出现了一个不可能执行的操作时就会出现运行时异常。运行时异常最明显的现象就是程序的中断。此时Java就会抛出异常。
- 比如我们使用0作为除数的时候。

```
19 //          运行时错误程序通过了编译器语法的检测, 但是在运行的过程中出现了一个不可能执行的  
20 //          操作时就会出现运行时异常。  
21 //          0作为除数时, 在正常的计算中我们都知道0不能够作为除数  
22 int num = 1/0;  
23  
24  
25
```

Problems Javadoc Declaration Console ×

<terminated> ProgramDemo [Java Application] D:\ProgramFiles\eclipse_2022_9\eclipse\plugins\org.eclipse.justj.openjdk
Exception in thread "main" java.lang.ArithmeticException: / by zero
at javase/java_language_basic.ProgramDemo.main(ProgramDemo.java:20)

- 除此之外运行时的错误还有很多, 比如我们要求用户输入的是数字, 而用户实际输入的是一个字符串的时候就会引发运行时错误。

3.3, 逻辑错误 (logic error)

- 程序正常通过了编译, 并且执行出了结果, 但是所执行出来的结果并不是我们想要的预期的结果。最根本的一个原因就是在执行的过程中程序的逻辑出现了错误。
- 比如我们想要计算 $3/2$ 的值, 我们都知道它的值为1.5。

```
25 //          逻辑错误程序正常通过了编译, 并且执行出了结果, 但是所执行出来的结果并不是我们想要的  
26 //          最根本的一个原因就是在执行的过程中程序的逻辑出现了错误。  
27 //          我们想要计算3/2的结果  
28 int result = 3/2;  
29 System.out.println(result);  
30
```

Problems Javadoc Declaration Console ×

<terminated> ProgramDemo [Java Application] D:\ProgramFiles\eclipse_2022_9\eclipse\plugins\org.eclipse.justj.openjdk
1

- 上述的出现虽然最后运行了出来, 但是并不是我们想要的结果, 很显然它出现了逻辑错误。纠正其原因就是它所使用的数据类型不正确, 应该使用浮点数类型进行接收。

实际上，在我们的开发中这三种错误是出现最为显著的，它影响着程序的正常运行，但是当我们的程序能够正常的运行并且满足了我们的预期结果之后，随着线上实际环境的部署它还是会出现各种各样的问题，比如大量用户在短时间内访问我们的网站时，网站可能会直接崩溃。虽然程序目前能够满足我们的运行，但是它运行时间花费很长，在一些需要高速运行的环境中，它不能够满足我们的需求。

针对软件中的各种错误和可能出现的各种问题，我们需要进行软件的测试，这属于另一门软件专业知识，软件测试的范畴。

软件是一种逻辑表现形式，对于逻辑来说，它的衡量实际上只能使用满足度来衡量，只能说逻辑满足某些场景，在某个时代下成为导向，但是随着时间的流逝，一些以前流行的逻辑就会慢慢的不适应时代，所以逻辑总会出现漏洞，所以软件也总会出现错误，没有完美的软件，软件当前未发现的一些错误和漏洞当被黑客等获取之后，公布大众我们普通人才可能知道，而这个过程或许凭借这个漏洞，许多的黑客已经将我们的个人隐私盗取并把我们的重要资料进行售卖的风险，所以我们要做出更可靠的系统需要不断的更新和发现当前软件存在的漏洞并且进行及时的修补。我们都有过每个一段时间Windows总会发出一些更新的消息，这是因为它发现了一些存在的漏洞或者软件bug，需要下载补丁进行修改，所以也告诉我们其实一些不太久，也不太新的系统相对更可靠一些，那些全新的系统才出来的时候一定会存在许多的漏洞需要经过很多检验才能被使用而那些太久的系统或许早就无法使用当前技术进行重构升级了，它的很多漏洞或许是作为典型的教材被使用。

4，注释，程序设计风格，关键字，标识符的规则及规范

4.1，注释

- 什么是注释

注释就是说明该程序是干什么的说明性文字，就像是在阅读一本书的时候会在书中的某一句话下面划上横线并写下有关这一句的解释说明的笔记。

- 为什么要使用注释

注释是一个程序员的良好习惯，在一些大型的项目中，往往都是很多的开发人员在一起进行合作开发的，书写注释既可以让自己看到，自己的代码逻辑和它为什么这么写是干什么的，也能够让其他的程序员看懂这段代码在干嘛。

- 单行注释

//注释的内容,一般使用单行注释来注释程序中的某一步骤或者说明某个变量或常量的含义

```
public static final double PI = 3.1415926; //它表示PI的常量值，可以直接用到程序中计算圆的面积
```

- 多行注释（块注释）

/*注释的内容*/，一般在程序中来注释程序中的某一块步骤

```
/*
 *计算两个整数相加的结果
 *将两个整数相加的结果返回给get方法，返回一个序号数
 *将序号数放入栈中
 */
int resultadd = add(num1,num2);
int num = get(resultadd);
push(num);
```

- 文档注释

文档注释是用来说明一个类中的相关的内容的描述说明。它通常放在一个类或者是一个方法的前面，用来说明这个类或者是方法的相关描述，使用，版本，参考等。它的格式是：


```

/**
 *描述的内容.....
 * .....
 *@标签 标签后说明的内容
 *@
 .....
 */
public class/interface/enum name{
    .....
}
or
public/private.. returntype methodname(parameter list){
    .....
}

```

使用文档注释可以生成Javadoc，所谓的Javadoc就是我们之前所下载的JavaAPI的文档，它是以HTML的形式展示的，所以文档注释在实际的开发中还是非常重要的一种方式，它能够直接根据注释内容生成相应代码的文档。

实际上我们去官网所下载的JavaAPI文档也是通过这样的方式生成出来的。在JDK的bin目录下存在一个javadoc.exe的程序，该程序就是生成JavaAPI文档的程序，我们可以使用命令的方式为我们的程序生成JavaAPI文档。

- 使用命令生成文档

假如我新建一个项目文件夹为test，在test文件夹中新建包JDBC，在包中新建了一个类为JdbcUtils类。下面是该类的具体内容。

```

package JDBC;
import java.io.IOException;
import java.io.InputStream;
import java.sql.*;
import java.util.Properties;
/**
 * JdbcUtils是针对于JDBC的工具类，是使用JDBC的相关知识实现的用于JDBC和数据库之间连接的工具类。
 * 它包含解析Properties文件获取数据库连接相关内容，创建连接数据库，关闭连接数据库，关闭Statement对象，关闭ResultSet对象和关闭连接的资源。
 * 它常常使用在需要连接数据库的场景中。
 * @author Emcred
 * @version 1.0
 * @since 1.8
 */
public class JdbcUtils {
    /**url是获取数据库的数据库类型，网络地址，端口号，数据库名称和连接的协议*/
    private static String url;
    /**password是连接数据库的密码*/
    private static String password;
    /**username是连接数据库时的用户名*/
    private static String username;
    /**driver是连接数据库时使用的驱动类型*/
    private static String driver;
    /**
     * 在该类加载的时候执行一次，然后一直连接数据库，直到释放资源
     * @throws IOException
     * @since 1.8
     * @see Properties
     */
}

```

```

static {
    //1,实例化properties对象
    Properties properties = new Properties();
    //2, 读取properties文件的输入流*****
    InputStream in =
JdbcUtils.class.getClassLoader().getResourceAsStream("Jdbc.properties");
    //3, 通过加载输入流解析properties文件
    try {
        properties.load(in);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    url=properties.getProperty("url");
    password=properties.getProperty("password");
    username=properties.getProperty("username");
    driver=properties.getProperty("driver");
}
/**
 * 当调用该方法的时候获取数据库连接
 * @since 1.8
 * @see Connection
 * @return Connection 返回一个Connection对象给调用者
 */
public static Connection getConnection(){
    try{
        //1,注册数据库驱动
        Class.forName(driver);
        //2,获取数据库连接对象
        Connection connect =
DriverManager.getConnection(url,username,password);
        return connect;
    }catch (Exception e){
        e.printStackTrace();
    }
    return null;
}
/**
 * 当调用该方法时关闭数据库连接
 * @param connection 外界传入的连接对象
 * @see Connection
 * @since 1.8
 */
public static void closeConnection(Connection connection){
    try {
        connection.close();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
/**
 * 调用该方式时关闭statement对象
 * @param statement 外界传入的Statement对象
 * @see Statement
 * @since 1.8
 */
public static void closeStatement(Statement statement){
    try {
        statement.close();
    }
}

```

```

        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}

/**
 * 调用该方法时关闭ResultSet对象
 * @param resultSet 外界传入的resultSet对象
 * @see ResultSet
 * @since 1.8
 */
public static void closeResultSet(ResultSet resultSet) {
    try {
        resultSet.close();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

/**
 * 调用该方法后关闭数据库连接
 * 该方法只能关闭只使用了Statement对象创建的数据库连接资源
 * @param statement 外界传入的Statement对象
 * @param connection 外界传入的连接对象
 * @see Statement
 * @see Connection
 * @since 1.8
 */
public static void closeResource(Statement statement, Connection
connection){
    //1, 先关闭statement
    closeConnection(connection);
    //2, 关闭connection
    closeStatement(statement);
}

/**
 * 调用该方法后关闭数据库连接
 * 该方法只能关闭只使用了Statement对象创建的数据库连接资源和使用ResultSet查
看了数据库数据的连接资源
 * @param resultSet 外界传入的resultSet对象
 * @param statement 外界传入的Statement对象
 * @param connection 外界传入的连接对象
 * @see ResultSet
 * @see Statement
 * @see Connection
 * @since 1.8
 */
public static void closeResource(ResultSet resultSet, Statement
statement, Connection connection){
    closeStatement(statement);
    closeConnection(connection);
    closeResultSet(resultSet);
}
}

```

接下来我们使用命令的方式来为该类生成一个API文档，在实际开发中，我们常常使用按文件，按包和按项目生成API文档。首先进入到项目所在目录。

```
javadoc JDBC(包名)
```

然后就会出现如下问题。

```
PS C:\Users\ROOTer\Desktop> test> javadoc JDBC
正在加载程序包JDBC的源文件...
.\JDBC\JdbcUtils.java:7: 错误: 编码GBK的不可映射字符
    * JdbcUtils鐫0抗漢班筇JDBC鑢動伐鋒風被鉅岬嶺沔跨駁JDBC鑢動舛鋒素燧駟嘴鑒扮玗璽木筇JDBC鍛岬隘鎰0旗泮
屨視杓煥鏗鑢動伐鋒風被鉅?
                                         ^
.\JDBC\JdbcUtils.java:8: 错误: 编码GBK的不可映射字符
    * 瀹富實錫B鐫愀roperties鎰困夾壘峰鑢鑢版嶼辜攪齡錙0 舛鋒沖哂瀉湍爻綵爰緩杓煥燧鑢版嶼鑢版嶼辜攪纈鋒鉅鉅
樞杓煥燧鑢版嶼辜攪纈鋒鉅鉅樞Statement漢渚薄鉅軒另閤環resultSet漢渚薄鉅軒另閤0齡錙0 峯墾勸簪鉅?
                                         ^
.\JDBC\JdbcUtils.java:9: 错误: 编码GBK的不可映射字符
    * 瀹富父甯鵠嬌璽厶濂闊?暇伙齡錙0 隘鎰0旗鑢動滿鐫0腑鉅?
                                         ^
.\JDBC\JdbcUtils.java:9: 错误: 编码GBK的不可映射字符
    * 瀹富父甯鵠嬌璽厶濂闊?暇伙齡錙0 隘鎰0旗鑢動滿鐫0腑鉅?
                                         ^
.\JDBC\JdbcUtils.java:15: 错误: 编码GBK的不可映射字符
    /**url鉅0咏鑒欏陞鎰0旗鑢動陞鎰0旗煖海濱潑爻綉緬淦滄鉅?銑出0壺e 佛銑岬隘鎰0旗錫錕0鍛舛齡錙0 玗璽他0
*/
```

这个问题已经讲过了，是由于字符编码所引起的问题，那么该如何解决呢也就是使用-encoding UTF-8参数进行编译，实际上还没有完，由于生成的文档属于HTML文档，所以需要设置HTML文档的字符编码也为UTF-8，所以需要参数-charset UTF-8。

```
javadoc -encoding UTF-8 -charset UTF-8 JDBC
```

可以看到它最后生成了一系列的HTML文件。

```
PS C:\Users\R00Ter\Desktop\test> javadoc -encoding UTF-8 -charset UTF-8 JDBC
正在加载程序包JDBC的源文件...
正在构造 Javadoc 信息...
标准 Doclet 版本 1.8.0_351
正在构建所有程序包和类的树...
正在生成.\JDBC\JdbcUtils.html...
正在生成.\JDBC\package-frame.html...
正在生成.\JDBC\package-summary.html...
正在生成.\JDBC\package-tree.html...
正在生成.\constant-values.html...
正在构建所有程序包和类的索引...
正在生成.\overview-tree.html...
正在生成.\index-all.html...
正在生成.\deprecated-list.html...
正在构建所有类的索引...
正在生成.\allclasses-frame.html...
正在生成.\allclasses-noframe.html...
正在生成.\index.html...
正在生成.\help-doc.html...
```

但是这一系列的HTML文件是默认生成到项目文件夹下，所以当我们进到项目文件夹后会看到许多HTML文件和我们的源文件显得十分的杂乱，所以我们想要把生成的文档放到一个文件夹下，那该怎么办呢？使用命令-d 文件路径指定生成的文档存放的文件位置。我想在项目文件夹下创建一个doc文件夹并且在doc文件夹中创建一个JDBCdoc文件夹用于存放JDBC包下的文档。

```
command:javadoc -encoding UTF-8 -charset UTF-8 -d ./doc/JDBCdoc JDBC
```

我们不需要去手动创建doc和DBCdoc文件夹，实际上当我们写上该路径后当计算机检测到没有这些文件它会自动帮助我们创建。

```
PS C:\Users\R00Ter\Desktop\test> javadoc -encoding UTF-8 -charset UTF-8 -d ./doc/JDBCdoc JDBC
正在加载程序包JDBC的源文件...
正在构造 Javadoc 信息...
正在创建目标目录: ".\doc/JDBCdoc\"
标准 Doclet 版本 1.8.0_351
正在构建所有程序包和类的树...
正在生成.\doc\JDBCdoc\JDBC\JdbcUtils.html...
正在生成.\doc\JDBCdoc\JDBC\package-frame.html...
正在生成.\doc\JDBCdoc\JDBC\package-summary.html...
正在生成.\doc\JDBCdoc\JDBC\package-tree.html...
正在生成.\doc\JDBCdoc\constant-values.html...
正在构建所有程序包和类的索引...
正在生成.\doc\JDBCdoc\overview-tree.html...
正在生成.\doc\JDBCdoc\index-all.html...
正在生成.\doc\JDBCdoc\deprecated-list.html...
正在构建所有类的索引...
正在生成.\doc\JDBCdoc\allclasses-frame.html...
正在生成.\doc\JDBCdoc\allclasses-noframe.html...
正在生成.\doc\JDBCdoc\index.html...
正在生成.\doc\JDBCdoc\help-doc.html...
```

此时我们到项目文件夹中可以看到。

doc	2023/1/22 13:04	文件夹
JDBC	2023/1/22 12:57	文件夹

此时我们也只需要打开index.html即可查看文档。但是我们会看到这样的网页标题显示。

生成的文档 (无标题)

此时为了更好的辨别是属于那个项目，所以我们需要网页标题为我们的项目名，此时使用-windowtitle "test"即可。

```
javadoc -encoding UTF-8 -charset UTF-8 -d ./doc/JDBCdoc -windowtitle
"test" JDBC
```

但是此时我们会发现我们生成的HTML文档还差点什么，我们可以打开官方的文档和我们自己的文档进行对比之后会发现。我们的少了一个文档的标题，这里的文档的标题使用的是“项目名&包名”的形式。这里我们可以使用-header "test & JDBC
" JDBC实现插入HTML元素到网页中。



```
javadoc -encoding UTF-8 -charset UTF-8 -d ./doc/JDBCdoc -header "
<br>test & JDBC</br><br>" JDBC -windowtitle "test" JDBC
```

除此之外关于javadoc的命令还有很多，我们可以打开终端（PowerShell或Shell），在终端上输入。

```
javadoc -help
```

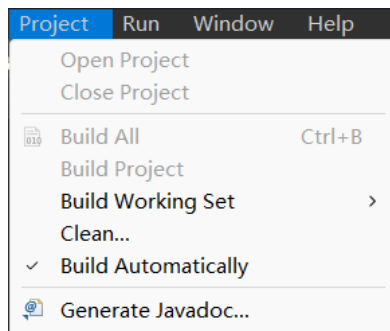
就可以看到与javadoc相关的参数了。

- 使用Eclipse生成文档

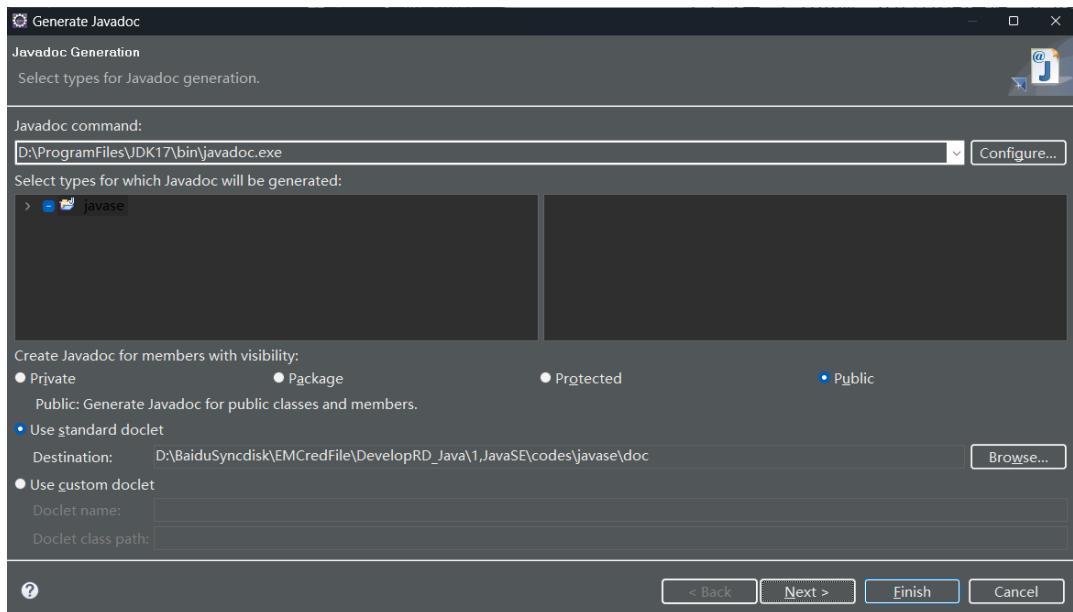
实际上我们发现IDE生成这些文档底层都是调用这些JDK命令的，IDE的一个好处就是这样，我们不需要使用很多的命令去自己配置，要不然它就不能够称作集成开发环境。

依旧使用上面的类作为演示文档注释的相关问题，这个时候将它的名字改为JavaDocDemo类，然后放在java_language_basic包下面。

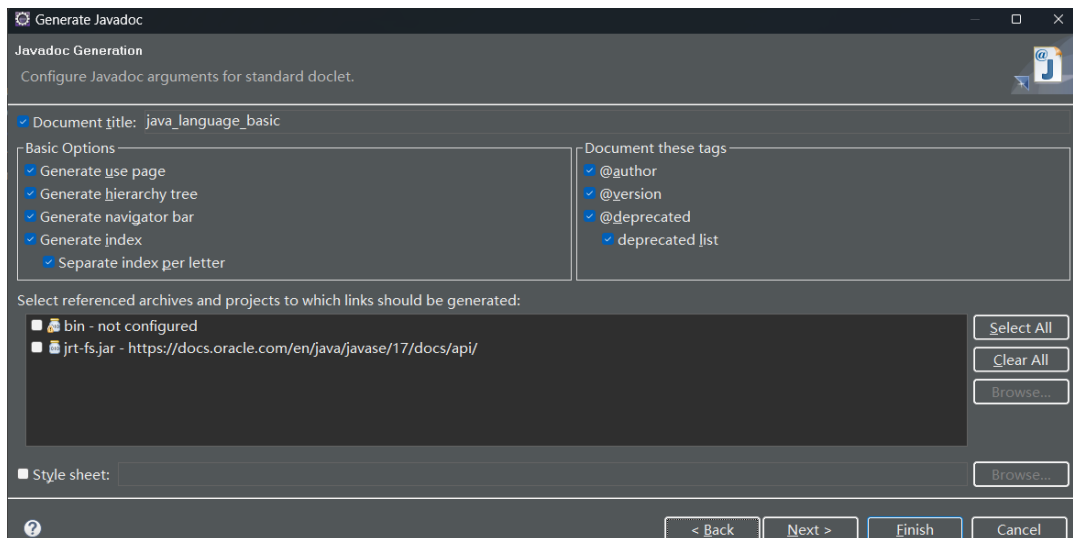
在Eclipse的Project选项下面点击Generate Javadoc，也就是生成Javadoc。



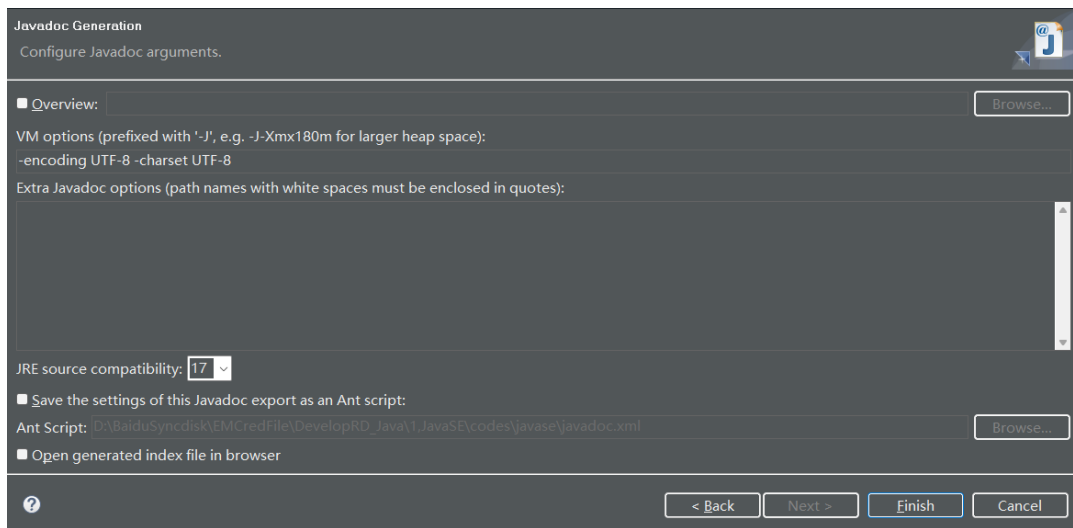
点击之后，它会出现一个生成的引导框，在该引导框中的Javadoc command中要选择的是 javadoc.exe 文件，也就是我们需要到JDK的安装目录中找到该文件，它在JDK目录的bin目录下，其他的选项默认，点击next。



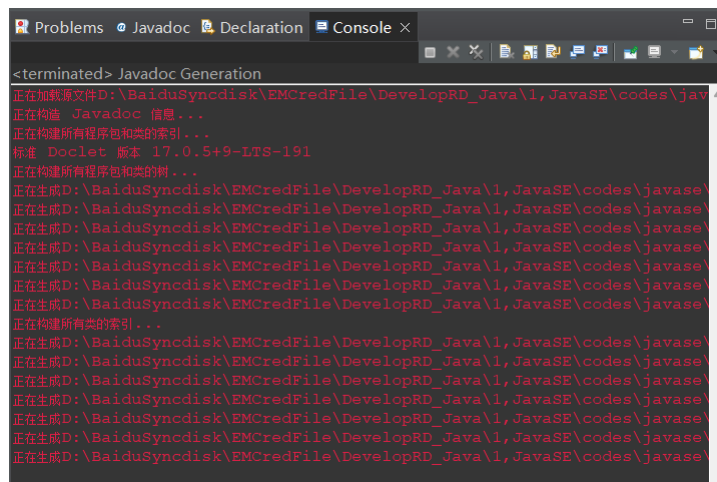
接下来进入到了第二个页面，这里只需要在Document title的地方写上我们的项目名称即可，其他的选项默认，点击next。



接下来需要注意的是配置编码的命令，否则会出现乱码问题，在VM options中配置命令-encoding UTF-8 -charset UTF-8，其他选项默认，Finish。



最后我们会看到控制台输出的创建过程。



现在我们去对应的项目文件夹中，可以看到产生了对应的doc文件夹，我们可以自行参看。

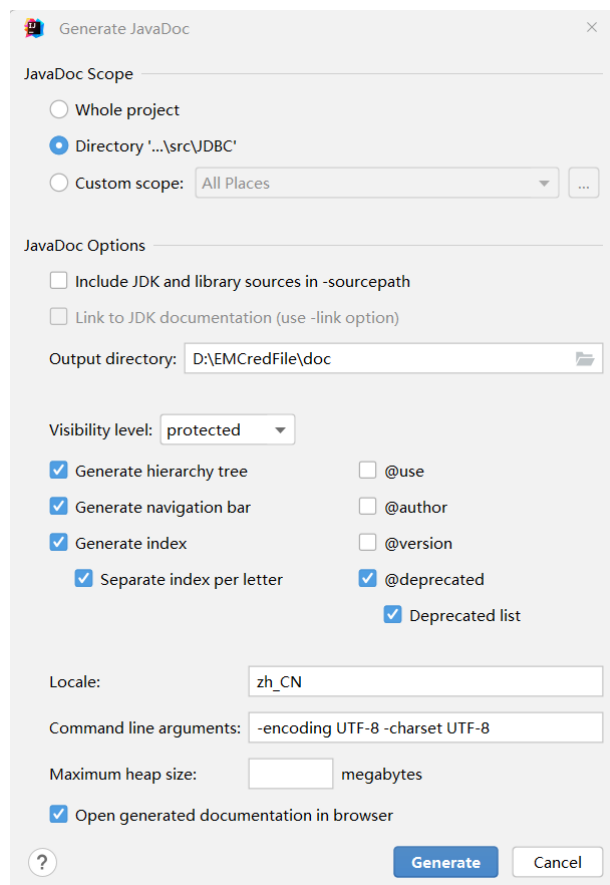
- 使用IDEA生成文档

找到IDEA中的Tools选项，在该选项下面找到Generate JavaDoc，点击进入可以看到javadoc的文档引导栏。

在引导栏中，JavaDoc Scope指的是按照范围生成javadoc，在其中有3项组成，Whole Project指的是按照整个项目来生成javadoc,File....."选项是按照文件为单位生成javadoc, Directory'.....'选项是按照包为单位生成javadoc文档的，Customer scope是按照用户自定义的范围来生成javadoc，在里面有很多的范围。

这里需要注意的是我们如果只想要某个文件的文档内容，我们需要先点击那个文档，再去打开Tools，我们需要某个包的文档内容，我们就需要先选中那个包的文件，再去打开Tools，否则不会出现File或Directory，Whole和Customer无论选不选中它都能出来。

比如下面我只想要JDBC包中导出文档，我需要先选中包文件夹，然后再打开Tools就会出现Directory选项。Output directory是输出文档的目录，Visibility level是导出的可见级别，这里默认即可，下面的选项也是默认即可，Locale指的是所选的HTML的语言这里填上zh_CN即可，command line arguments是相关的命令这里也就填入编码的命令就可以了-encoding UTF-8 -charset UTF-8实际上除了编码的命令还可以填上其他的命令,Open generated documentation in browser是创建完了之后直接在浏览器中打开index.html,然后选择Generate。



可以看到创建的过程，并且在浏览器中自动打开了对应的文档显示。

```
D:\ProgramFiles\JDK17\bin\javadoc.exe -locale zh_CN -protected -splitindex -encoding UTF-8 -charset UTF-8 -d D:\EMCredFile\doc @C:\Users\ROOTer\AppData\Local\Temp\javadoc_args
正在加载源文件D:\Workcodes\untitled\src\JDBC\JdbcUtils.java...
正在构造 Javadoc 信息...
正在构建所有程序包和类的索引...
标准 Doclet 版本 17.0.5+9-LTS-191
正在构建所有程序包和类的树...
正在生成D:\EMCredFile\doc\JDBC\JdbcUtils.html...
正在生成D:\EMCredFile\doc\JDBC\package-summary.html...
正在生成D:\EMCredFile\doc\JDBC\package-tree.html...
正在生成D:\EMCredFile\doc\overview-tree.html...
正在构建所有类的索引...
正在生成D:\EMCredFile\doc\allclasses-index.html...
正在生成D:\EMCredFile\doc\allpackages-index.html...
正在生成D:\EMCredFile\doc\index-files\index-1.html...
正在生成D:\EMCredFile\doc\index-files\index-2.html...
正在生成D:\EMCredFile\doc\index-files\index-3.html...
正在生成D:\EMCredFile\doc\index.html...
正在生成D:\EMCredFile\doc\help-doc.html...
```

实际上我们看到，它的底层就是调用JDK的javadoc工具所生成的一系列文档。

- 可能出现的错误和解决的方式

如果我们在Generate的时候出现了 **javadoc: 错误 - 无效的标记: --source-path** 这样的问题时，原因就是使用了低版本的JDK工具去生成高版本的JDK所写的源代码，此时我们需要将项目的JDK环境改为使用对应版本的JDK再次Generate就能正常的执行。

- 在实际的开发中我们常常写文档注释进行说明，对于文档注释的规范，这里可以说以下几点。

- 常使用的文档注释相关注解

标签	描述	示例
@author	标识一个类的作者	@author name
@deprecated	指名一个过期的类或成员	@deprecated description
@param	说明一个方法的参数	@param parameter- name description
@return	说明返回值类型	@return Type description
@see	指定使用该类的时候可以参考的其他类	@see classname
@since	表示该类或者方法或者成员变量的最低使用版本或者是某个时间被启用	@since 1.0
@exception/@throws	标志一个类抛出的异常	@exception exception- name description
@serial	说明一个序列化属性	@serial description
@serialData	说明通过writeObject() 和 writeExternal()方法写的的数据	@serialData description
@version	当前文件的版本	@version 1.0
{@link}	连接到一些相关的代码	{@link 包名. 类名/类名/# 方法名(参数 类型)}
{@code}	将文本标记为代码样式文本，一般在Javadoc中只要涉及到类名或者方法名，都需要使用@code进行标记	{@code 类名/ 方法名/关键字}
{@value}	用于标注在常量上用于表示常量的值	{@value}

标签	描述	示例
@inheritDoc	用于注解在重写方法或者子类上，用于继承父类中的Javadoc	基类的文档注释被继承到了子类 子类可以再加入自己的注释（特殊化扩展） @return @param @throws 也会被继承

■ 类的文档注释规范

方法的文档注释规范和类的文档注释规范差不多。以下是一种方法注释的模板。

类的文档注释一般书写的规范是：

1. 概要描述：通常用一段话简要的描述该类的基本内容。
2. 详细描述：通常用几大段话详细描述该类的功能与相关情况。
3. 文档标注：用于标注该类的作者、时间、版本、参略等信息。

```
/**
 *关于本类{@code JdbcUtils}是关于{@link sql}的具体实
 *现的工具类
 *
 * <p>主要包含有：获取数据库连接，静态加载数据库连接,创建连
 *接对象{@link Connect}, 创建Statement对象{@link
 *Statement}, 返回查询结果{@link ResultSet}, 关闭连接
 *资源{@link closeResource(ResultSet resultSet,
 *Statement statement, Connection connection)}和
 *closeResource(Statement statement,Connection
 *connection)
 *
 * @author <a href="mailto:xxx@xx.com">EMCred</a>
 * @see sql
 * @see Connection
 * @see Statement
 * @see ResultSet
 * @since 1.8 2022/12/2
 * @version 1.0
 */
public class JdbcUtils {
    .....
}
```

■ 方法的文档注释规范

实际上方法的文档注释规范和类的文档注释的规范相差不大。

```
/**
 *获取对应{@link forName(String name)}, {link
 *Connection}数据库的连接。
 *
 * <p>Note: 首先该方法注册相关的数据库驱动，注册成功之后创建
```

```

*一个数据库的连接对象{@code Connection}通过驱动对象
*{@code DriverManager}获取连接, 用户名和密码, 如
*果获取成功将会返回一个{@code Connection}否则将会返回
*{@code null}
*
*<p>示例代码</p>{@code
*Connection con = JdbcUtils.getConnection()
*if url,username,password exists return{@code
*Connect} else return {@code null}
*}</pre>
*
*@return {@code Connection} if the {@code
*Connection} is not{@code null}
*@see Connection
*@since 1.8 2022/12/2
*@version 1.0
*/
public static Connection getConnection(){
    try{
        //1,注册数据库驱动
        Class.forName(driver);
        //2,获取数据库连接对象
        Connection connect
=DriverManager.getConnection(url,username,password);
        return connect;
    }catch (Exception e){
        e.printStackTrace();
    }
    return null;
}

```

■ 成员的文档注释规范

成员一般指的是在类中的变量, 常量。

```

/**url是获取数据库的数据库类型, 网络地址, 端口号, 数据库名称和连接的协议*/
private static String url;

/** 默认数量 {@value} */
private static final Integer QUANTITY = 1;

```

实际上在我们的实际开发当中, 我们的文档注释需要只要能够讲的清楚代码说得代码的用途就可以了, 这里所使用到的注解是当javadoc进行解析的时候根据注解的内容输出对应的HTML对应的内容, 比如@see它就会输出另请参阅, 其实对于我们来说可以通过查看源码的方式学习这样的做法, 看一下专业的人员是如何书写程序和书写注释的, 这是一种重要的方式。

4.2, 程序设计风格

• 什么是程序设计风格

程序设计的风格决定着程序的外观。

实际上我们所写的代码如果写在一行, 它也是能够完全编译出来的。比如:


```
public void Method(){int a=1;int b=2;int sum=a+b;System.out.println(sum);}public int sub(int num1,int num2){return num1-num2;}
```

它实际上也是能够完全编译的，但是在一个大的项目中这样的书写方式就会显得代码十分的难看难读，阅读性非常的差。所以好的程序设计风格增加了程序的可阅读性，可维护性。

- 程序设计风格

- 正确的缩进和留白

所谓的缩进和留白就是在程序的书写过程当中的一种书写格式的规范。我们之前说过其实将程序写在一行它也能够正常的编译运行，但是这会显得程序的可读性十分得差，所以我们希望去规范一种书写得格式让程序更可读和可维护。

我们可以参考以下得重新案例学习缩进和留白得程序书写规范。

```
public class HelloWorld{
    System.out.println("hello world");
    int a = 1 + 2;
    if(a==1){
        System.out.println("HELLO WORLD");
    }
}
```

- 程序缩进

参考内容{}所在的位置，内容相对于{}一个tab位置书写。也就是说上面的System.out.println("hello world");属于第一层的花括号中的内容，它距离左边有一个tab的位置，一个tab也就是4个英文字母的位置，我们的键盘上也有tab键。System.out.println("HELLO WORLD");相对的是if的{}中也就是第二层{}它相当于第二层{}左边一个tab的位置。

- 程序的留白

我们看到int a = 1 + 2;并不是连着写在一起的，它们的中间都有一个空格隔开，这个空格就是程序的留白

上述程序的缩进和留白是对于一个专业的程序员的最基本的程序书写规范。

- 块风格

所谓的程序块就是一个{}中的代码，块风格就是{}的书写风格。比如下面的两种风格。

```
public class HelloWorld{
    System.out.println("hello world");
}
public class HelloWorld
{
    System.out.println("hello world");
}
```

- 行尾风格

行尾风格就是第一种代码的书写风格。它更节省空间，有层次感。我们写Java程序的时候常常用这种风格书写代码。我们可以参看JavaAPI的一些源码，它们的代码风格也是统一使用行尾风格进行书写的。这样书写出来的Java代码看着更专业一点。

- 次行风格

次行风格就是第二种代码风格。它的层次分明，程序更容易阅读。我们在写C/C++程序的时候常常采用该种风格来书写代码。对于一些专业的C/C++的开发工程师来说这种书写的风格师一种代表性的体现。它书写出来的C/C++代码更为专业一些。

实际上以上两种风格都可以按照个人偏好进行选取，但是在开发中还是需要一个团队统一好风格进行开发，为了与JavaAPI源码保持一致，所以一般选择行尾风格进行开发。

4.3，关键字（保留字）

- 什么是关键字（保留字）

关键字是程序语言中的一个术语，每一种语言都有着自己的关键字。关键字就是具有特定含义的英文单词，这些英文单词已经事先在程序语言中设定好了。

- 如何理解关键字（保留字）

我们已经提到的package,import,public,class,static,void,new,int,if,try...可以看到我们其实到目前为止所使用的案例中已经接触到了很多的关键字，它们都具有特定的含义，比如我们之前所讲到的package是用来声明包名的关键字，import是用来导入包的关键字，package和import在Java中是具有特定含义的（声明包和导包）的英文单词。它们事先在Java设计的时候就已经设计好的干什么用的单词。

实际上这些关键字的设计是为了体现出高级语言的特性，用我们日常的生活用语来进行开发，同时体现出不同的高级语言的特点，它们实际上屏蔽了更多的底层的计算机细节，用一些具有功能描述的单词来表示使用它能够做什么，相当于一些命令，比如使用javac能够将源代码编译为字节码文件。

4.4，标识符

- 什么是标识符

标识符，就是具有标识作用的符号。实际上标识符在程序中的含义就是为了标识程序中的类，方法，变量和常量的元素而采用的命名。标识符在是程序设计语言中的一种专业术语。

比如，在下面的这个例子中

```
package java_language_basic;
/**
 * <p>标识符案例
 *
 * <p>具有标识作用的符号。
 * 实际上标识符在程序中的含义就是为了标识程序中的类，方法，变量和常量的元素而采用的命名。
 * @author EMCred
 * @since 2023/1/23
 */
public class IdentifierDemo {
    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        String name = "张三";
        String address = "云南昆明";
        System.out.println(name+": "+address);
        System.out.println(a+b);
    }
}
```

其中的IdentifierDemo,a,b,name,address就是标识符，如果没有这些标识符，那么首先我们不可能区分对应的类，然后在程序中也不可能知道int在该程序中是1还是2还是两个独立的int我们无法区分，我们不能够去操作int = 1和int = 2，"张三"和"云南昆明"在程序设计时这两个字符串是什么含义，我们也无法去操作String = "张三"和String = "李四"。

所以我们可以看到标识符的最大作用就是命名，给一些在特定的程序设计时的名称，方便我们去标识这个程序是什么意思。

- 标识符的硬性规则

对于Java来说，Java的标识符的硬性规则是必须遵守的，否则会直接导致错误。

- Java是一种对大小写敏感的语言，所以String name和String Name是两个不同的名字。
- Java的标识符只能由数字（0123456789），字母（abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ，这里理解为英文字母，实际上中字也是可以的，这里的字母不仅仅指英文），下划线（_），美元符号（\$）组成，并且不能以数字开头。
- 标识符不能使用关键字进行命名。比如：int class = 1，int true = 2，int double = 1等
- 标识符的长度可以是任意的。

- 标识符的软性规则

在实际开发当中，我们在遵守Java标识符的硬性规则下，为了使得开发更加的规范，我们应当遵守Java的软性规则，所谓软性规则就是即使不遵守Java也能正常编译。

- 标识符的命名应该尽量采用英文的见名知意。实际上中文的汉字也可以作为标识符，比如：

```
String 你 = "you";  
System.out.println(你);
```

照样可以编译运行，但是它并不专业和规范，而且在有些情况下可能会出现一些字符编码的问题，所以我们应当做到命名的时候使用英文的见名知意。比如：姓名我们就应该写成String name，年龄int age等，最建议的是将英文的全称写出来。

- 对于项目，包的命名采用全小写字母进行命名。
- 对于类，接口，注解，枚举的命名应该采用帕斯卡命名法，全部单词的单词首字母都要大写，比如:String,InputStreamReader,Math,System等。并且类，接口，注解，枚举文件的文件名的命名就是类名，接口名，注解名，枚举名.java
- 对于变量，方法的命名采用的是驼峰命名法，也就是首单词的首字母小写，其他单词的首字母大写，比如：String name，InputStreamReader inputReader，public static void addNums()，public int getNums()等。
- 对于常量的命名采用的是所有字母全部大写的方式，多个单词之间使用_隔开。比如：public static final double PI = 3.1415926，public static final int STATE = true,public static final String OPEN_STATE。

- Java语言的基本语言规则总结

- Java语言是严格区分大小写的
- 所有的符号必须是英文模式
- 每句话必须以分号结尾

5，变量，常量和直接量（字面量）

5.1, 变量

- 什么是变量

所谓的变量，也就是可以改变的量，在程序运行中数据发生变化的量就是变量。变量是程序设计语言中的一个专业的术语。

其实变量我们最开始听到的是在数学中的变量，实际上程序中的变量和数学中的变量是一样的，比如，数学中线性方程。

$$y = x + 1$$

当x的值发生改变的时候y的值也会随之变化，y和x就是所称作的变量，它们的值能够不断的改变而1它无论y和x怎么变化都是1，在程序中1就称为常量。

- Java中定义变量（基本类型）

在Java中定义变量必须先声明变量然后初始化变量。所谓的声明就是使用Java中的数据类型进行声明，关于Java中的数据类型在下文中讲解。所谓的初始化指的是第一次给变量赋值，变量我们知道它的值可以多次的赋值。

- 即可声明一个变量，也可同时声明相同类型的多个变量。

```
int a; //声明一个整形变量
int b,c,d; //同时声明多个整形变量
```

- 即可边声明边初始化，也可先声明后初始化。

```
int e = 1; //声明了一个整形变量a并初始化为1
int f; //先声明了一个整形变量f
f = 1; //后将整形变量f初始化为1
```

- 同时声明多个变量时，可以按需边声明边初始化。

```
int h,i,j = 1; //同时声明了多个整形变量，并将整形变量j初始化为1
```

- 当为初始化过的变量中的值重新赋值时的含义是修改了变量中的值。

```
f = 2; //修改了f中的值为2
```

- 使用变量

- 对变量的操作就是对它所存的那个数的操作

```
int a1=5; //声明一个整型变量并且赋初值为5
int b1=a1+5; //取出a1的值为5再加5后赋给整型变量b
System.out.println(b1); //输出变量b1的值为10
a1=a1+5; //取出a1的值为5，加得到10后，再赋给a1，既在a1本身的基础上增加5
System.out.println(a1); //输出变量a1的值为10
```

- 变量的使用必须与数据类型匹配

```
double c1 = 3.14; //声明一个浮点型变量c1的值为3.14
a1 = c1; //将该浮点型变量的值赋给a1
```

- 变量在使用之前必须声明并初始化

```
System.out.println(m); //编译错误，未声明变量m的类型
int m;
System.out.println(m); //编译错误，未给变量m赋初值
```

5.2, 常量

- 什么是常量

在上述的变量的介绍中我们知道了变量就是可以改变的量，那么常量就与之相反是不能够改变的量，就如上面的例子中所说的1。

- Java中定义常量

Java中的常量只能够同时声明并且初始化。一旦定义了常量，那么常量的值将不能够通过其他的程序改变。

在Java中常常会在接口或者类当中定义常量，常量的一般形式是使用final关键字进行修饰，final关键字是Java中的权限修饰符，在第二阶段中将会详细介绍。而且常量的定义往往需要和static关键字进行连用，它常被定义为。

```
public static final double PI = 3.1415926;
```

在实际的开发中，它常常会使用在一些表示数据永远不变并且该数据需要经常被使用的情况下，比如状态值，我们知道，实际上我们进入到一个软件中，这个软件都会根据不同的状态来做出相应的判断和结果的，我们经常会到淘宝中去买东西，当我们点击购买并付款前，商品会显示待付款，当我们付款之后，商品会显示已付款，然后我们会经历代发货，已发货，已到货，已收货的状态来完成本次的交易，而以上的各种状态就需要状态值来判定，它们常常是一致的。

对于我们目前的阶段当中，实际上一些工具类就使用了常量，比如数学类中的PI值我们都知道，PI是一个固定的值，所以Java的Math类为我们提供了一个常量的调用，假如我们需要去计算圆的面积的时候，直接调用Math.PI即可。

```
double r = 12.3;
double area = Math.PI*r*r;
System.out.println(area);
```

- 常量的本质

实际上常量的本质就是那个数值，当编译器进行编译的时候会将常量直接替换为原来的值，它相对于变量来说运行的效率更快。

5.3, 直接量 (字面量Literal)

- 什么是直接量

直接量就是直接展示我们源代码中的值。

比如：int num = 1;其中的这个1就是直接量，public static final double PI = 3.1415926;中的3.1415926就是直接量，String name = "hello";中的hello就是直接量。为了更好的理解什么是直接量，我们可以参看下面的程序。

```
public class LiteralDemo {
    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        int num = a+b;
        System.out.println(num);
    }
}
```

其中1和2就是直接量。

6, Java的数据类型

6.1, 什么是数据类型

- 数据结构相关知识

数据结构是什么？

数据结构就是数据在计算机中怎么进行存储，也就是说它解决的是数据在计算机中如何存的问题。好的数据结构决定了好的程序，在开发程序之前一般会根据这个程序的要干什么来决定采用的数据结构。

所以数据结构在软件专业中具有核心的地位，实际上我们开发软件就是采用各种数据结构和算法来对数据进行处理然后把想要的结果输出给用户，实际上计算机就是输入内容输出结果的东西。而在输入到输出的这个过程中需要软件和硬件的相互配合才能更好的实现。对于软件的最优极限来说硬件就是软件的最优极限的瓶颈，无论我们设计的程序如何的精巧，但是如果运行该程序的硬件不好的话，那么该程序也无法发挥它的最大极限。所以硬件和软件的相互配合才能实现一个好的程序，两者要相辅相成。

数据结构的广泛定义就是 (D, S) 所谓的D就是数据元素的有限集，S就是D上的关系的有限集。数据结构就是数据的结构，那么什么是数据的结构呢，数据指的就是一组数据元素，结构指的就是这组数据元素之间的相互关系。可以这样说具有相互关系的一组数据就是一种数据结构。

对于数据结构来说它分为逻辑结构和物理结构，逻辑结构就是与计算机的存储没有关系，通过数学描述数据之间应该存在的结构，物理结构就是逻辑结构所描述出来的数据应该具有的结构应该在计算机中如何进行存储的问题。

对于逻辑结构来说它分为线性结构和非线性结构。

- 线性结构

线性结构实际上是一种关系型的结构，也就是二维表的结构，也可以称为线性表结构，什么是最直观例子就是打开电脑中的Excel，它所呈现的就是一张表。

我们首先知道数据结构中的数据，数据元素和数据项的基本定义。在线性表结构中构成线性表的可以是数据元素，比如26个英文字母就是一种线性表结构，构成该线性表结构的单位是数据元素也就是ABCD.....也就是单维表,除此之外，线性表结构中构成线性表的也可以是数据项，我们知道数据项是不可再分的数据单位，比如在学生基本信息表中，每一个学生为一个数据元素，它包括学号，姓名，性别，籍贯，专业等数据项组成也就是多维表，这也是一种线性表结构。

在程序中线性结构可以具体的分为：

线性表（26个英文字母单维表，学生基本信息表多维表），栈和队列（操作受限的线性表），字符串（又字符组成线性表，单维表的推广），数组（由数据元素组成线性表，单维或多维表的推广）和广义表（数据元素是一个线性表或单个数据元素，多维表的推广）。

- 非线性结构

非线性结构就是除了线性结构之外的其他结构。比如我们计算机中的图片，文件系统等这类由树或图结构组成的数据结构。非线性结构在程序中可以具体分为：

树结构：普通树结构（具有多个分支的层次结构，比如文件管理系统，人工智能对弈的决策树）和二叉树结构（具有两个分支的层次结构），图结构：有向图结构（边是顶点的有序对）和无向图结构（边是顶点的无序对）和集合结构（数据元素的关系仅属于一个集合之外没有其他的联系，比如判断一个学生是否属于某个班级）。

对于物理结构来说它分为顺序存储结构和链式存储结构。

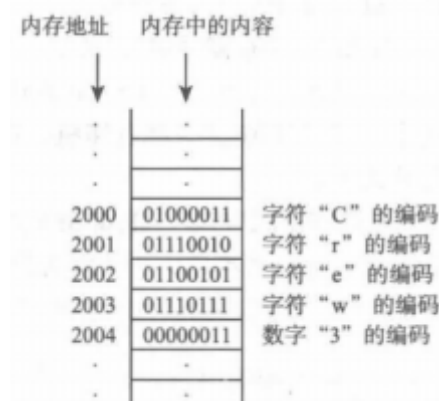
○ 内存

我们知道现代计算机的结构是按照冯 诺伊曼结构搭建的，在该结构中引入了存储器的相关概念，而在现代的计算机结构中的存储器结构就是存储数据的硬件结构，目前主要由内存和外存组成，除此之外还有高速缓存等存储设备组成，在计算机组成原理的存储器的层次结构中可以进行参考。

对于内存来说，什么是内存呢？我们在组装机台式机的时候常常需要用到内存条这样的硬件设备，实际上内存条就是我们组装机的内存。对于内存来说，我们需要知道的是

1. 内存是程序和数据运行时的空间，也就是说我们现在**正在运行**的程序和数据是在内存中运行的。程序不会在磁盘，硬盘等外部存储设备上运行的，这些外存是用来长久的保存数据的，当我们需要运行相应的程序的时候，操作系统会进行资源的调度和磁盘的调度，在磁盘和内存中进行数据的交换，把要使用的数据从磁盘或硬盘中取出来到内存中进行运行然后把处理过的数据，用不到的数据又交换到磁盘。只有将要运行的数据交换到内存中，接下来内存才能和CPU相互配合运行得数据。
2. 内存是掉电即失的，也就是说当我们关机之后，在内存中的数据就会随之丢失，我们常常遇到的是在我们编写文档的时候，突然停电，如果此时文档程序没有设置自动保存或者是上传云端等的话，当我们再次开机的时候可能打开文档就是一片空白，这是因为文档程序是运行在内存中的，内存掉电即失，所有数据都会丢失。这和内存得基本结构有关，广义上的内存分为随机存储器（RAM）和只读存储器（ROM），我们所说的内存条它实际上是RAM，而计算机的主板上还有BIOS，它属于ROM。对于RAM来说它又分为SRAM（静态随机存储器）和DRAM（动态随机存储器），SRAM由MOS管组成，DRAM由电容器组成，对于DRAM来说它需要刷新充电一次否则数据丢失，它们两者都断电及失。目前我们的台式计算机内存已经达到了8GB和16GB的甚至是32GB的。

最重要的是我们需要知道内存中的数据是如何进行存储的，当数据加载到内存中来的时候，它是按照字节的方式存储的，每个字节都有一个唯一的内存地址与之对应。可以这样理解，内存就相当于一个公寓，而公寓里的每个房间就相当于一块内存区域，公寓房间里住的人就相当于我们的数据或者说一个字节（8bit），当我想要去该公寓楼中找一个小明的人，那么我到该公寓楼下，我就需要知道小明住在公寓楼的房间号，我通过该房间号可以唯一的在这栋公寓楼中确定小明的房间而精准的找到他，而这里的房间号就是我们所说的内存地址，它唯一标记一块内存区域。



○ 顺序存储结构

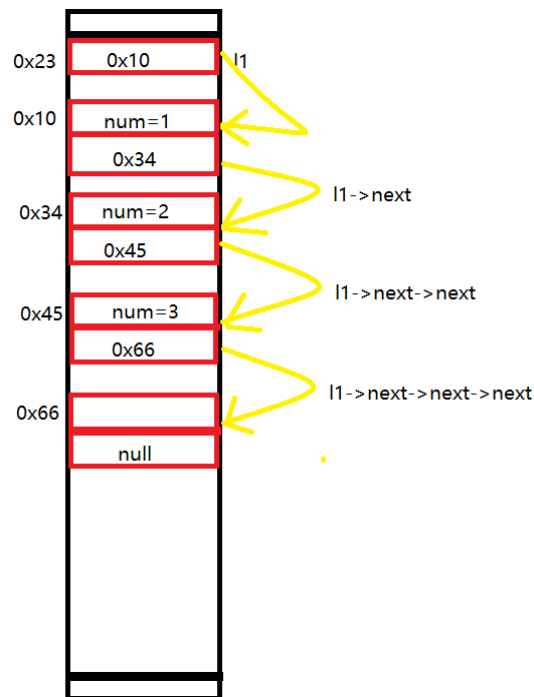
所谓的顺序存储结构是借助元素在存储器中的相对位置来表示的，也就是说它的存储地址是相对的有规律连续的，此时的数据元素采用结点来表示，比如一个个人信息表的节结点假设占50个字节，那么该个人信息元素在内存中的表示如下。它要求所有元素依次存放在一片连续的存储空间中。此时它占用了一整块的存储单元。

.....
0	小明，男，安徽
50	小李，男，云南
100	小红，女，北京
150	小张，男，深圳
.....

- 链式存储结构

所谓链式存储结构它的内存位置不是像顺序存储结构那样是连续的，它在存储器中是分散的，它们之间通过知道对方的地址来进行访问，就像是两个内存块之间有一根无形的链子相连起来一样，在程序设计语言中，常常通过指针这样的类型来表示。实际上指针就是一个地址。比如在C语言中的程序。它在内存中的存储时分散的，不需要一整块的存储空间。

```
#include <stdio.h>
#include <malloc.h>
typedef struct Node{
    int num;
    struct Node* next;
}L;
int main()
{
    L* l1 = (L*) malloc(sizeof(L));
    l1->num=1;
    l1->next = (L*) malloc(sizeof (L));
    l1->next->num=2;
    l1->next->next = (L*) malloc(sizeof(L));
    l1->next->next->num=3;
    l1->next->next->next=(L*) malloc((sizeof (L)));
    l1->next->next->next->next=NULL;
    L* l2 = l1;
    while(l2->next!=NULL){
        printf("%d ",l2->num);
        l2=l2->next;
    }
}
```



- 什么是数据类型

数据类型是和数据结构相关的一个概念。数据类型实际上可以看成是数据结构中的集合结构，它表示的是数据属于某种类型，它是程序语言的一种概念。

比如，在Java，C，C++中我们使用int来表示整型数据，使用double来表示浮点型数据，这就相当于一个集合结构所有的整型数据都使用int来表示，那么int就代表了整型数据的一个集合，不能声明int a = 3.14这样的数据。

实际上在计算机的存储中是无法表示所有的整型数据的，我们知道理论上的数是无穷的。所以在程序设计语言中，比如Java中对于整型数据来说它划分为byte,short,int,long类型来表示整型数据，这些类型都可以表示整型数据，区别就是它们能够表示的整数的范围是不一样的，也就是说Java这个语言划分了4种数据类型来表示整型类型的数据。之所以表示的范围不一样取决于这些类型的本身在内存中的权限，也就是说byte它在内存中只能申请到1字节的内存区域存放byte数据，1字节就是8bit，最高位作为符号位也就是说它能够表示的正整数和负整数的范围分别为00000000~01111111(0~127),10000000~11111111(-128~-1),所以它能够表示的整数的范围是-128~127。而int类型它能够在内存中申请得到4个字节的内存区域存放int数据，也就是32bit,它所能存储的数值范围是 $-2^{31} \sim 2^{31}-1$,这是一个相当大的范围，完全可以满足我们日常所能用到的整数。

在程序语言中会设计两类数据类型。

- 基本数据类型（内置数据类型）

对于基本数据类型来说，它是所有程序设计语言都必须设置的一类数据类型。这类数据类型是最基本的数据类型也就是整数，浮点数，字符。在程序设计语言中通过事先设置好的关键字来直接声明，比如int,double,char。这类数据类型是最为基本的数据类型是ADT抽象数据类型得基础，对于Java来说在实际的开发中这类数据类型直接使用得并不多，它们往往是包含在ADT数据类型中进行使用。

- ADT抽象数据类型

抽象数据类型是比基本数据类型更抽象一层的数据类型，那么什么是更抽象一层的数据类型呢？我们知道基本数据类型是在程序设计语言中早就定下来的，使用关键字所声明的一类最基本的数据类型，而抽象数据类型是我们用户自己可以定义的数据类型，它是由其他的抽象数据类型或者是基本数据类型组成的一种新的数据类型，这种数据类型是由我们用户自己定义的，它和int等一样是属于程序的一种类型。

ADT在数据结构中的定义是：

```

ADT{
    数据对象：（可以理解为各种的其他数据类型的集合）
    数据关系：（可以理解为在其中的各种数据类型之间的关系）
    操作：（该ADT具有的操作也就是程序中的函数或方法）
}

```

对于ADT来说，在许多的程序设计语言中都支持该种数据类型，比如在C语言中的结构体类型和Java中的类，接口等。就说Java的类来说，如果有个业务场景是为学校开发一款教务平台的，在其中对于学生来说我需要记录学生的姓名，入学年级，身份证号，籍贯，电话号码，学号和专业，然后我要求学生在该平台能进行的操作是可以登录平台，可以查看成绩，可以修改自己的电话号码和可以查看自己的姓名，入学年纪，身份证号，籍贯和专业信息。如果仅使用基本数据类型，我们也可以开发，但是非常难以开发和维护，那么在实际开发中，我们常常就会把该学生看作是一种数据类型，使用Java提供的类来自定义一个Student数据类型，它和int一样也是一种数据类型。我们可以按如下方式定义该类。

```

public class Student{
    private String name;
    private int state;
    private String ID;
    private String origin;
    private String profession;
    private String phoneNumber;
    private String stuID;
    private String passwords;
    public Student(String stuID,String passwords){
        .....
    }
    public void login(String passwords){
        .....
    }
    public Grades getGrades(String stuID){
        .....
    }
    public String getPhoneNumber(){
        .....
    }
    public void setPhoneNumber(){
        .....
    }
    .....
}

```

上述所定义的Student类仅仅演示，在实际开发过程中可能要比这个系统得多，根据不同学校得需求决定，对于上述Student类来说，它就是个我们自己定义的新类型。我们常常使用的是这种类型作为开发的基本单位进行开发，在后面组成一个ADT的类型基本上都是其他的ADT类型，基本数据类型基本都不会直接的使用而用其包装类替代。

我们可以看到ADT数据类型在Java中不仅仅是由基本数据类型和其他的抽象数据类型组成的，它还可以包含有对于这个类型来说的一些操作的定义。实际上每种不同的语言对于这种支持ADT数据类型是不同的，对于C语言来说它的抽象数据类型中是不包含有操作的定义的，ADT实际上就是一种定义抽象数据类型应该具有什么样的模型，在具体的程序设计语言中实现ADT的方式是不同的，名字也是不同的，在Java中这样的类型统称为引用类型而在C中这样的类型统称为派生类型。

除了上述定义的基本ADT模型外，为了实际在内存操作这种类型，C/C++在设计之初定义了指针类型进行操作，Java是由C和C++发展而来的一门语言，它摒弃了指针这样的类型，所以在Java中没有声明指针这样的类型，但是它的引用类型中实际上是将指针整合在其中了只是不需要像C和C++那样自己去使用指针操作，学习过C语言的人都知道C语言中的指针就是一个地址，指针类型就是定义这个类型的地址，比如：

```
int a = 1;
int* ptr = &a;
*ptr+=1;
```

在声明a的时候实际上在内存中为a开辟了一块空间，我们之前讲过每个内存空间都有唯一的内存地址，a也不例外，它此时是具有自己的内存地址，我们定义一个整型指针ptr指向a的地址，就是取出a的地址给到ptr，所以指针指向了a的地址也就指向了a，我们可以通过使用解引用来进行操作a的内存也就是使用*ptr直接访问内存a的数据加1最后a=2。

在Java中实际上也是这样的对于引用类型来说，引用就是指向，所以这也是它称为引用类型的原因。在后续的JVM中进行分析可以返回来看一下上述指针例子进行对照学习。

6.2, Java中的数据类型

- 基本数据类型（内置数据类型）

Java提供的基本数据类型包括：整数类型，浮点数类型，字符类型和布尔类型

整数类型：byte,short,int,long

浮点数类型：float,double

字符类型：char

布尔类型：boolean

- 为什么要Java要设计丰富的基本数据类型？

Java属于静态类型语言和强类型语言的范畴，所以它必须具体声明数据类型进行使用，我们可以看到仅仅整数类型就设置了4种类型，它们之间的区别就是范围表示不同，在内存中所占字节不同，所以设置丰富的数据类型是为了在程序员进行编程的时候针对一个问题使用合适的数据类型来有效使用内存资源和严谨的编程。

比如我要实现10以内的加减法，那么最大它是20最小是-9，所以我的数据类型选用byte即可。如果使用了int，那么它会浪费很大一部分空间。

- 基本数据类型分析

其实我们常常使用的基本数据类型并不是上述的所有，我们常常使用的基本数据类型是int，long，double，char和boolean。

- int

1. 类型：整数类型。
2. 内存占用：4Byte。
3. 直接量范围：-2³¹~2³¹-1。
4. 重要知识点：

- 整数直接量默认为int型并且整数直接量不能超出int的最大范围否则会出现编译错误。

理解：我们知道Java为了表示整型的数据（整型直接量）设计了4种基本的数据类型，byte,short,int,long，那么我们在源程序中所直接写的数值如果是整数的话，它的默认数据类型是整型。比如：

```
//整型直接量默认为int类型,int类型的最大值为2147483647, 如果直接量表示为2147483648就会出现编译错误
System.out.print(2147483647); //正常显示
System.out.println(2147483648); //出现编译错误
```

我们实际上不用去刻意的记住int的最大值和最小值, 如果我们想知道int的最大值和最小值, 可以使用包装类Integer中的常量值MAX_VALUE就是int的最大值, 最小值为MIN_VALUE。

```
//int的最大值和最小值
System.out.println("int的最大
值:"+Integer.MAX_VALUE); //2147483647
System.out.println("int的最小
值:"+Integer.MIN_VALUE); //-2147483648
```

- 两个整数相除, 结果还是整数, 小数部分无条件舍去。

理解: 也就是说, 两个int类型的数据比如3/2我们知道3/2最后是1.5, 但是如果使用一个int类型去接收的话, 比如int result = 3/2;最后输出的结果是1, 它不会四舍五入它最后只会保留整数部分。

```
int result = 3/2;
System.out.println(result); //1
```

最根本的原因是它的本质就是用来显示整型的数据, 不可能与浮点数出现什么关系, 如果最后的结果是浮点数的话, 最终它还是仅显示整数。

- 整数运算时若超出直接量范围就发生溢出, 不是错误, 但需要避免。

理解, 我们之间讲过编译错误(语法错误)编译器会直接提示错误, 比如上面写的int声明的变量如果所给的初始直接量超过了int的最大值或比int的最小值小的话编译器直接报错, 但是如果在运算后的结果超过了int的最大值的话它不是编译错误而称为溢出, 此时能够正常编译并且显示数值。比如:

```
//整数运算时若超出直接量范围就发生溢出, 不是错误, 但需要避免。
int num = 2147483647+1;
System.out.println(num); //-2147483648
```

我们使用int的最大值加1最后的结果是int的最小值-2147483648, 同样如果我们换为加2, 最后的结果是-2147483647, 感觉形成了一种循环, 时钟。

实际上这和计算机中数的表示方法有关, 在计算机中的数的表示方法有原码, 反码, 补码和移码, 这些知识属于计算机组成原理的CPU内容中的相关知识。这里需要说的是在Java中我们可以明显的看到整数在计算机内表示的方式应该是使用补码表示的, 什么是补码呢, 可以这样理解补码, 我们都知道时钟中的时针每转动一圈它就是12小时, 如果此时时针的指向为6点, 我们想要让它指向3点, 那么我们可以顺时针转动9, 同样也可以逆时针转动3, 如果记逆时针是-, 顺时针为+, 那么最后6+9=15和6-3=3最后的结果都是指向钟表上的3点, 此时我们可以说+9是-3以12为模的补数, 同理不仅有+9和-3, +10和-2, +11和-1, +12和0等, 这就是补码。可以找到一个与负数等价的正数通过模的关系来替代负数, 实现减法运算用加法实现。

上面我们可以简单的理解为2147483647+1和2147483647-4294967295也就是+1和-4294967295是以4294967296为模的补数。

- long

1. 类型：长整型
2. 内存占用：8Byte
3. 直接量范围： $-2^{64} \sim 2^{64}-1$
4. 重要知识点：

- 长整型直接量需要在数字的后面加上l或L否则还是整型数据。

比如：

```
long l1 = 2147483648; //编译错误，此时数据任然是整型数据，超出了整型数据的最大值
long l2 = 2147483648L; //此时的数据为长整型数据
```

我们说过只要是整型直接量它默认都是int类型。

- 运算时若有可能溢出，建议在第一个数字后加L

```
long l3 = 2147483647*2+2L; //0
long l3 = 2147483647L*2+2; //4294967296
System.out.println(l3);
```

首先需要知道的是数据类型之间会进行自动转换，自动类型转换也就是小的类型会往大的类型转，int比long小。

第一个l3它已经发生溢出了，也就是最大值乘于2之后它的值为-2，此时为整型数据，当-2+2L的时候，整型数据会自动往大的长整型数据转换，此时的0为长整型数据但是还是溢出了。第二个l3此时2147483647L就已经是长整型数据了，它和整型数据2相乘会自动转换为长整型，此时还在长整型的范围内，所以不会出现溢出，此时为4294967294再加上2就是4294967296，此时为正确答案。

- 显示当前时间接收，做一些跟计时，唯一数相关的工作。

再System类中的currentTimeMillis()方法用于返回一个从1970年1月1日00:00:00开始到当前时刻的毫秒数，这个毫秒数是通过long类型返回的，所以需要使用时long类型变量进行接收。

```
long currentTime = System.currentTimeMillis();
System.out.println("当前时间："+currentTime);
```

实际上1970年1月1日00:00:00是UNIX操作系统发布的时间，所以也成为UNIX时间戳，所谓的时间戳也就是时间开始记时的点。

- byte和short

1. 类型：字节类型和短整型
2. 内存占用：1Byte和2Byte
3. 直接量范围：-128~127和 $-2^{16} \sim 2^{16}-1$
4. 重要知识点：

- 整数直接量可以直接赋值给byte,short但不能超出byte和short的范围。虽然整数直接量默认为int。

```
///本来只要是整型数据，系统默认为Int类型，但Java规定直接量可以直接赋值给byte,short  
byte b1 = 123;  
short s1 = 23;
```

- byte,short类型变量数据参与运算，先一律自动转化为int再进行运算，所以此时需要使用int类型来接收或者进行强制类型的转换。

所谓的强制转换和自动转换是相对的一个概念，也就是说数据类型如果要从大类型往小类型转换的话需要我们自己手动添置转换的语法才能进行，否则就会出现编译错误，byte和short都比int小，在参与运算的时候参与运算的数据都变为了int，此时如果还用short或byte去接收的话就会出现编译错误。

比如下列的代码，最后s4会出现编译错误，这是因为s2和s3相加之后它进行了运算是int类型的数值，但是我们还使用了short类型进行接收。

```
short s2 = 123;  
short s3 = 124;  
short s4 = s2+s3; //出现编译错误
```

解决上述问题的办法要么使用int去接收，要么强制转换。

```
int s4 = s2 + s3;  
short s5 = (short)(s2 + s3);
```

- byte,short类型直接量数据参与运算，此时结果无需人为转换或者是使用int接收即可正常编译。

```
short s6 = 1+3; //正常编译
```

此时s6不会出现编译错误而正常编译，实际上这是它之间还是做了int到short的转换，但是我们使用了直接量进行书写，所以此时编译器会自动帮我们转换为short而不需要我们自己手动进行转换，这实际上就是一种编译器允许，它会在编译的时候自动帮我们变为(short)1+3，这是Java语言的一种特点。

■ double和float

1. 类型：双精度浮点型和单精度浮点型
2. 内存占用：8Byte和4Byte
3. 直接量范围：-1.798E+308~-4.9E-324,4.9E-324~1.798E+308和-3.4028235E+38~-1.4E-45,1.4E-45~-3.4028235E+38
4. 重要知识点：

- 浮点数直接量被默认为double类型，浮点数数据不能直接赋值给float类型，若要赋值给float数据的话就需要在浮点数后面加f或F。否则会出现编译错误。

```
///浮点数直接量被默认为double类型，若要将其转化为float类型时 就必须  
在数字后加上f或F。  
float f = 1.23f;  
float f1 = 1.23; //编译错误
```

- double 和float在运算时有可能会出现舍入误差

所谓的舍入误差说直白一点就是由于计算机的本地性质，在进行运算的场景下运算得到的数据和我们理想的数据产生一些偏差，这就是舍入误差，也就是说比如：

```
double d = 3.0;
double d1 = 2.98675;
double d2 = d - d1;
System.out.println(d2); //0.01325000000000206
```

最后我们希望的d2的结果是0.01325但是最后却输出了0.01325000000000206，这就是舍入误差，在一些精确运算的场合会出现很大的代码缺陷问题。

这是因为计算机是以二进制存储数值的，浮点数也不例外。Java 采用了IEEE 754 标准实现浮点数的表达和运算，比如0.1 的二进制表示为 0.0 0011 0011 ... (0011 无限循环)，再转换为十进制就是 0.100000000000000055511151231257827021181583404541015625。对于计算机而言，0.1 无法精确表达，这是浮点数计算造成精度损失的根源。所以在精确运算的场合中可以采用Java提供的BigDecimal类来实现。

- float在控制台中输出小数点后7位，double在控制台中输出小数点后16位。比如下面所示的例子。

```
float f1 = 3.1415926250312454286953F;
double d3 = 3.1415926250312454286953;
System.out.println(f1); //3.1415927
System.out.println(d3); //3.1415926250312456
```

- 可以使用科学计数法的方式来位float和double类型的变量赋值。也就是说如果我们想要表示的数为1345672.8的话我们可以表示为如下形式。

$$1.3456728 * 10^6$$

在程序中，我们可以使用E或e来代替10，也就是说上述的浮点型直接量我们想要在程序中直接赋值的话可以声明为如下的形式。

```
double a = 1.3456728E+6;
float b = 1.3456728E+6F;
```

如果指数为正数的话就是E+，反之为E-。

- 使用printf方法可以指定浮点数输出的位数。比如对于PI来说我仅想要输出它的前4位的话可以如下。

```
System.out.println(Math.PI); //3.141592653589793
System.out.printf("%.4f\n", Math.PI); //3.1416
```

我们可以看到最后输出的浮点数保留了后四位，对于printf来说，相信学习过C语言的都知道它的一些相关的用法，而上述按位输出浮点数的写法实际上和C语言是一样的写法。

当然这也仅仅是在屏幕上输出对应的指定位数的浮点数，实际上浮点数本身并没有改变为4位，它还是它本身也就是3.141592653589793。为了实际的能够转化为3.1416，我们需要借助Java的DecimalFormat类实现。这里不需要了解很深它怎么进行转换的，因为它是一个固定的套路。如果想要真正的去了解它的话可以在后续的学习中返回看它就能明白很多东西了。

```
double pi = Double.valueOf(new  
DecimalFormat("0.0000").format(Math.PI));  
System.out.println(pi);//3.1416
```

■ char

1. 类型：字符型
2. 内存占用：2Byte
3. 直接量范围：0~65535
4. 重要知识点：

- 字符直接量必须放在单引号中，而且只能放一个字符。

```
char c = '你';
```

- char的本质实际上就是int类型并且是 unsignedint类型也就是无符号的int类型，我们在之前讲过int类型会使用最高字节的位置作为自己的符号位，但是 unsignedint就是有几位就用几位没有所谓的符号位。
- 支持Unicode编码策略，一个字符对应一个码，常使用的字符和码的对应关系如下。

码（十进制）	字符
48	'0'
97	'a'
65	'A'

可以参考我们之前讲的字符编码进行对照学习。为了能够更直观的感受码和字的对应可以看如下例子。

由于char的本质就是int类型，所以对于char来说我们不仅仅可以输出字符还能输出字符所对应的码。除此之外我们还能输出码所对应的字符。

```
System.out.println((int)'a');//97  
System.out.println((char)97);//a
```

- char可以直接赋值整型数据，并且可以进行运算，最后输出的是运算结果所对应的字符。比如下面的例子。

```
/*char c1 = 94;  
char c2 = 3;  
char c3 = (char)(c1 + c2);*/  
char c3 = 94 + 3;  
System.out.println(c3);//a
```

因为char本质上就是int类型，当然可以做一些运算了。

- 0和'0'的关系理解

0：是一个整型数据，在计算机系统中按照补码的方式来进行存储，是一个int类型的数据，占4个字节，所对应的十进制表示为0。

'0': 是一个字符型数据，在计算机系统中按照ASCII码的方式来进行存储，是一个char类型的数据，占2个字节，所对应的十进制为48。

所以说0是0，'0'非0，'0'是48。

■ 转义字符（转义序列）

■ 什么是转义字符？

所谓的转义字符，实际上就是能够将字符的含义进行转换的字符就是转义字符。

比如现在我想要在程序中输出"Hello"的话我们最基本的考虑是如下所示的写法。

```
System.out.println("Hello");//编译错误
```

当进行编译的时候，编译器会去找到输出内容的起始"和结束"然后输出其中的内容，但是上述可以看到编译器第一个"和第二个"之间没有任何的内容，第三个"和第四个"之间也没有任何内容然后Hello和两个""没有使用字符连接符号并且Hello也不是变量，所以此时会出现编译错误。在上述中的"就是程序中的一种特殊的符号，它被赋予了特殊的含义，为了能够在程序中使用这个"的本义，我们就需要使用到转义字符，也就是\"就能将"转义为了原来的含义。

```
System.out.println("\\Hello\\");//Hello
```

它实际上解决了程序中的一些特殊符合和我们想要使用该特殊符号的正确含义之间的冲突。

■ Java里的常使用的转义字符

在实际的开发中常常使用的转义字符，实际上并不是很多，常常使用的转义字符如下。

转义字符	含义
\"	双引号
\"	单引号
\\	反斜杠
\\n	换行符
\\t	Tab

实际上转义字符不仅仅可以用来转义这些特殊字符的，当然转义字符的最大用途就是用来转义这些我们无法实际在程序中写入的字符，除此之外，实际上每一个字符都可以使用\转义出来，它不仅可以转义\"字符本身而且还可以使用字符对应的Unicode编码进行转义，比如对于"来说，我们想要转义，可以写成\"，也可以写成u0022，由于转义字符也可以写成如上的转义Unicode字符的模样，所以转义字符也可以称为转义序列，它不是单独的一个字符而是一种序列组成的。不仅仅可以使用Unicode编码进行转义，也可以使用\+16进制或8进制进行转义，当然这种方式并不常用。

```
System.out.println("\u0024");//$
```

对于转义字符本身来说它不仅仅是Java语言中的一种特点，在其他的编程语言中也可以看到，它实际上是一种程序语言、数据格式和通信协议的形式文法的一部分。转义序列通常有两种功能。第一个是编码一个句法上的实体，如设备命令或者无法被字母表直接表示的特殊数据。第二种功能，也叫字符引用，用于表示无法在当前上下文中被键盘录入的字符（如字符串中的回车符），或者在当前上下文中会有不期望的含意的字符（如Java中的字符串中的双引号字符"，不能直接出现，必须用转义序列表示）。

■ boolean

1. 类型：布尔类型
2. 内存占用：1Byte
3. 直接量：true或false
4. 重要知识点：
 - boolean类型的直接量只能是true或者是false并且true或者false只能是boolean类型的数值。
 - 对于boolean类型来说它只有true或者是false，也就是真或假，正确或错误，有或者没有.....，它是逻辑运算最后的结果，对于一个逻辑运算的最后结果只能是true或者是false，在程序中我们需要借助逻辑结构和逻辑运算来对程序的整体走向进行判断，所以boolean类型在后续的学习和开发中是最重要也是最常用的类型。我们在后续的学习中去深刻的理解和体会。

○ 基本数据类型之间的转换

这里所说的基本数据类型之间的转换，再开发中常常出现在直接量进行算数运算的过程中出现的现象，这里需要注意的是对于基本数据byte,short,int,long,float,double,char也可以称为数值类型，boolean类型也称为逻辑类型。数值类型和逻辑类型之间是不能够互相转换的，这里的基本类型实际上指的是数值类型。

■ 自动类型转换（隐式数据类型转换）

所谓的自动类型转换就是指在程序中的小类型向大的类型进行自动的转换过程，这个过程是编译器自动发生的，没有明显的变化。

那么什么是大类型？什么是小类型？

对于基本数据类型来说，它们的大小，实际上是根据它们所占的字节数来判定的，我们可以根据它们所占的字节数按照从小到大的排列顺序进行排列之后，如下：

byte,short,int,long,float,double
,char

对于自动类型转换的过程来说可以参考如下的例子进行理解。

```
System.out.println(3/2);//1  
System.out.println(3/2.0);//1.5
```

对于第一个来说3和2都是整型直接量，默认为int类型，那么毫无疑问两个整型直接量相除最后的结果任然是整型，对于小数部分无条件的舍去。对于第二个来说，3是整型直接量，2.0是浮点型直接量，对于3来说它默认为int类型，对于2.0来说，它默认为double类型，当3/2.0后的结果是什么类型呢？这个过程中int类型3会自动转换为double类型3.0，然后再和2.0相除之后的结果是1.5。这个过程并不是直接能够看到的，所以这个过程也称为隐式数据类型转换。

- 强制类型转换（显式数据类型转换）

所谓的强制类型转换就是指在程序中的大类型向小类型进行转换的过程，这个过程中需要使用强制转换的特定语法进行类型的转换，强制转换的语法如下。否则会出现编译错误。

（要强转的类型）强转的对象

比如如下的程序中我们想将一个boolean类型转换为一个int类型的数据我们可以如下定义。

```
double pi1 = 3.1415926;  
int piInt = pi1; //编译错误  
int piInt = (int)pi1;
```

我们可以看到强制转换需要我们人为的强制进行转换，它需要借助强制转换语法，这个过程我们可以明显的在程序中看到，所以它也被称为是显示转换。

对于强制转换来说，它会造成数据的丢失而导致精度损失，比如上述中所说的将double转换为int，最后它的小数部分就会丢失，所以对于程序语言来说，强制转换需要我们人为的手动进行操作，并且承担最后的结果导致的一些问题。

- 数值运算时的自动转化

在一个运算中，可能会发生多种数据类型的转化double a1=80/100;//结果为0.0，右边的数据都是整型数据，在运算后为0，然后自动转化为double类型为0.0double a2=80.0/100;//结果为0.8，右边有80.0，系统默认为double型数据，然后100也自动转化为double型数据，再将值赋给double型的变量a2。

- 引用数据类型

Java中的引用数据类型包括：类，数组，接口（本质上也是类），枚举类型（实际上也就是类），注解类型（本质也是类）

类：class

数组：[]

接口：interface

枚举：enum

注解：@interface

- 引用数据类型概述

在之前我们讲过关于数据结构和数据类型的相关知识，我们知道程序语言中不仅仅是有基本数据类型，许多的程序语言还提供了搭建ADT抽象数据类型的数据类型而在Java中引用数据类型就提供了这样的一种操作。同样对于为什么称为引用类型的时候，我们在使用C语言的指针进行类比的过程中提到过。

这里需要说的是引用数据类型可以说是Java的精髓所在，就像指针一样是C和C++语言的精髓所在，在后续的使用Java进行开发的过程中我们将会慢慢的体会引用数据类型的相关内容，我们可以看到上述Java中的引用数据类型中具体包含的内容，可以发现对于引用数据类型的最基本的实现是Java中的类，它十分的重要，是一个Java过程的最基本组成单位，可以看到上述的其他内容基本是类的衍生内容，对于类的具体讲解，我们需要到面向对象阶段进行讲解，Java的引用数据类型实际上就是为适合面向对象开发思想而设计的，在面向对象阶段中我们基本上就是讲述关于类的相关知识和面向对象思想的结合。

7, 运算符

对于程序设计语言来说除了提供基本的数据类型和抽象的数据类型支持之外，还提供了关于运算或者说是操作的符号，程序的本质实际上就是处理输入到程序中的数据然后输出结果，这也对应着计算机本质就是输入和输出的东西，现在可以支持的数据，那么接下来将的就是对数据的操作，也就是运算符或者说操作符。它分为以下几类。

7.1, 算数运算符

所谓的算数运算符就是操作与数据运算相关的运算符。

+: 加法运算符

```
int sum = 1+2;
System.out.println(sum);//3
```

-: 减法运算符

```
int sub = 2- 1;
System.out.println(sub);//1
```

*: 乘法运算符

```
int add = 1*6;
System.out.println(add);//6
```

/: 除法运算符

```
int del = 6/2;
System.out.println(del);//3
```

?: 取余运算符

- 取余运算适用于整数，小数和字符，但一般的话都是用于整数取余

```
System.out.println(3.5%2);//1.5
System.out.println('b'%'a');//1
System.out.println(20%12345);//20
System.out.println(12345%20);//5
```

++: 自增运算符

自增1，可在变量前也可在变量后使用。

- 单独使用时

在前在后都一样。也就是说单独使用时++a和a++的意思都是a = a + 1。比如下面的例子。

```
int a = 1;
a++;
System.out.println(a);//2
++a;
System.out.println(a);//3
```

- 被使用时

在前在后不一样。在前的先增加后使用，在后的先使用后增加。比如下面的例子。

```
int a1=5,b1=5;
//1)先将a1的值5赋给c1
//2)a1自增1后赋给a1
int c1=a1++;//此时a1++是被使用的
//1)先将b1的值增1后赋给d1
//2)再将b1的值赋给b1
int d1=++b1;
System.out.println(a1);//6
System.out.println(b1);//6
System.out.println(c1);//5
System.out.println(d1);//6

int a2=5,b2=5;
System.out.println(a2++);//5, 此时a2++被使用, 既输出操作使用
System.out.println(a2);//6
System.out.println(++b2);//6
System.out.println(b2);//6
```

--: 自减运算符

自减1, 可在变量前也可在变量后使用。

- 和自增运算符的用法是一样的, 可以参考自增运算符。

7.2, 赋值运算符

所谓的赋值运算符就是操作数据在内存中存储的运算符。也就是将数据赋给变量的运算符。分为基本赋值运算符和增强型赋值运算符。

=: 基本赋值运算符

```
int b = 1; //将值1赋给整型变量b
```

+=, -=, *=, /=, %=: 增强型赋值运算符

- 增强型运算符的作用就是简化广泛自增运算的写法, 我们在程序中经常需要使用到在自身的数值上增加某个数值的操作, 比如a = a+3;就是在a原来的基础上增加3。它可以被简化写为a+=3;
- 增强型赋值运算符自带强制转换。

我们在数据类型哪里的时候讲过short变量进行运算的时候是先转换为int类型进行运算的, 最后要接受的话要么使用int类型进行接收要么强制转换为short类型进行接收, 但是使用增强型赋值运算符后它自带强制转换。比如:

```
short b1 = 23;
b1 = (short)(b1 + 23); //需要强制转换接收
b1 +=23; //无需强制转换
System.out.println(b1);
```

7.3, 关系运算符

所谓的关系运算符就是在程序运算过程中出现的数据与数据之间的关系判断运算, 比如数据的大小, 在程序中的关键逻辑结点通过关系运算判定结果后进行下一步的程序, 关系运算的结果是布尔类型的数值。

>: 大于关系运算符

```
int i1 = 2,i2 = 3;  
System.out.println(i1>i2);//false
```

<: 小于关系运算符

```
System.out.println(i1<i2);//true
```

>=: 大于等于关系运算符

```
System.out.println(i1>=i2);//false
```

<=: 小于等于关系运算符

```
System.out.println(i1<=i2);//true
```

==: 等于关系运算符

```
System.out.println(i1==i2);//false
```

!=: 不等于关系运算符

```
System.out.println(i1!=i2);//true
```

7.4, 逻辑运算符

所谓逻辑运算符就是在程序中为进行程序的逻辑运算而提供的运算符，它常常会 and 关系运算符结合作为程序逻辑结点的判断，比如当 $a > b$ 并且 $a > c$ 时条件成立，这里的 $a > b$ 和 $a > c$ 就是关系运算而并且就是逻辑运算。逻辑运算的最后结果是布尔类型。

逻辑运算可以分为与，或，非和异或运算。

与运算：一假则假，也就是说只要有一方为假整个逻辑等式为假。

或运算：一真则真，也就是说只要有一方为真整个逻辑等式为真。

非运算：真真假假，也就是说如果是真则为假，是假则真。

短路的逻辑运算符：（常用）

&&: 与

```
System.out.println(2>3 && 2>1);//false  
System.out.println(2<3 && 2>1);//true
```

||: 或

```
System.out.println(2>3 || 2>1);//true  
System.out.println(2<3 || 2<1);//false
```

!: 非

```
System.out.println(!(2>3));//true
```

什么是短路的逻辑运算符？

所谓的短路的运算符就是当确定结果后结束执行也就形象的比喻为短路。可以这样理解，比如如下的程序例子。

```
int t1=3,t2=4;
boolean bool = t1>t2 && ++t1<3;
System.out.println(bool);//false
System.out.println(t1);//3
```

我们可以想到的是t1的结果最后肯定是自增的，那么输出的结果应该为4才对，但是最后输出了3，究其原因是因为&&是短路的，也就是说t1>t2是遇假则假，那么此时程序判断该语句为假就不会执行&&后面的内容了。如果上述程序改为如下所示。

```
int t1=3,t2=4;
boolean bool = t1<t2 && ++t1<3;
System.out.println(bool);//false
System.out.println(t1);//4
```

这时t1<t2是成立的，此时程序接着进行判断，++t1<3我们知道最后的结果可以转换为4<3，那么显然这是不正确的，所以此时bool的结果为false但是它执行了++t1，所以最后t1=4。

不短路的逻辑运算符：

&：与和|：或，它们的作用和&&和||是一样的，只不过它们是不短路的而已。可以看如下例子。

```
int t3=3,t4=4;
boolean bool1 = t3>t4 & ++t3<3;
System.out.println(bool1);//false
System.out.println(t3);//4
```

可以看到和短路的例子相比，只是最后++的结果发生了改变，它们的本质作用是没有变的，在这之中虽然t3>t4是错误的，但是它并没有不执行后面的++t3<3而是进一步的执行了，这就是短路与不短路的区别。

7.5，字符串连接运算符

字符串连接符是在Java中来连接两个字符串的。

+: 当一边存在""时是字符串连接符，用来连接两个字符串的。

```
System.out.println(1+2);//3，此时+两边是数值，所以这里的意思是加法
System.out.println(1+2+"20");//320，第一个+是加法第二个+的一边存在""所以是字符串连接符
System.out.println(""+23+34);//2334，变换过程为""+23+34->"23"+34->"2334"
```

7.6条件运算符（三目运算符）

条件运算符可以看作是一种if-else句式的简写形式，它和其他的运算符不一样，不是成对或者是单个基本的而是一种三个值的组合所以也称为三目运算，之前所接触的也称为二目运算比如t1<t2 && ++t1<3中的<只能是左右一个值和&&也是一样左右一个boolean句式，条件运算符的结果是布尔类型，根据结果返回最终的数值。

条件运算符的语法：boolean?e1:e2

用法：当boolean是true的时候返回e1，当boolean是false是返回e2。

```
int result = 1>2?2:1;
System.out.println(result);//1
```

案例：使用条件运算符实现闰年的判断

- 要求：使用Scanner类来接收用户的输入，当用户输入对应的年份的时候返回闰年或平年。
- 分析：能够被4整除**并且**不能被100整除**或者**能被400整除的年份。首先获取用户输入，然后进行闰年判断，最后返回结果。
- 编码：

```
package java_language_basic;
import java.util.Scanner;
/**
 * 使用三目运算符判断闰年
 * @author ROOTer
 *
 */
public class LeapYear {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("请输入年份: ");
        int year = scanner.nextInt();//获取用户输入的年份
        System.out.println(year%4==0 && year%100!=0 || year%400==0?"闰年":"平年");
    }
}
```

- 测试正确性：

测试用例：2000，期望结果：闰年



```
请输入年份:
2000
闰年
```

测试用例：2001，期望结果：平年



```
请输入年份:
2001
平年
```

7.7，位运算符

位运算符用来对二进制位进行操作,也就是说使用位运算符可以对二进制位进行直接的操作，这种操作更底层化一些，我们知道Java有JavaME，它面对的是一些嵌入式的设备，比如机顶盒，移动设备等，所以这里也不难理解Java还保留着对于位的直接操作符，位运算符最初是来源于C语言的，当然Java是由C和C++编写而成的，所以在一定程度上还保留着C和C++的一些影子。

位运算符在Java中只能用于整型数据也就是byte, short, int, long和char类型，它是按位运算，也就是按照二进制位根据运算符的类型一位位的进行运算。由于更加的接近于底层的操作由目前的研究表明它的速度和加法运算的速度相同，快于乘法运算。位运算符的运算规则是按照布尔代数的运算规则进行运算的，结果可以是布尔类型，用在整数时是整型。

这里需要注意的是布尔运算和布尔值是两个不同的概念，布尔运算是一种运算的规则，它的结果不是布尔值，布尔值只能是true和false。

当然对于我们程序员来说，这一点是有助于提升性能的，但是在实际的工作中我们还是使用算数运算符的时候较多，位运算符有其自身的局限性就是仅对于整型有效。

- 位逻辑运算符：

&：按位与运算符

如果相对应位都是1，则结果为1，否则为0。

```
int r = 3; //00000000 00000000 00000000 00000011
int r1 = 4; //00000000 00000000 00000000 00000100
int r2 = r & r1; //00000000 00000000 00000000 00000000
System.out.println(r2); //0
```

|：按位或运算符

如果相对应位都是0，则结果为0，否则为1

```
r2 = r | r1; //00000000 00000000 00000000 00000111
System.out.println(r2); //7
```

~：按位取反运算符

按位取反运算符翻转操作数的每一位，即0变成1，1变成0。

```
r2 = ~r2; //11111111 11111111 11111111 11111000
System.out.println(r2); //-8
```

^：按位异或运算符

如果相对应位值相同，则结果为0，否则为1

```
r2 = r ^ r1; //00000000 00000000 00000000 00000111
System.out.println(r2); //7
```

- 移位运算符：

<<：左移运算符

表示带符号左移，即符号位保持不变，次高位开始左移。比如int a = 3 << 1，存储为int类型的数据，那么它的二进制为00000000 00000000 00000000 00000011，从0开始向左移动1位变为00000000 00000000 00000000 00000110，也就是6相当于乘上了2。

```
r = r << 1; //00000000 00000000 00000000 00000110
System.out.println(r); //6
```

>>：右移运算符

表示带符号右移，即符号位保持不变，次高位开始右移。比如int a = 3 >> 1，存储为int类型的数据，那么它的二进制为00000000 00000000 00000000 00000011，从0开始向右边移动1位变为00000000 00000000 00000000 0000001也就是1。

```
r = r >> 1; //00000000 00000000 00000000 0000001
System.out.println(r); //1
```

>>>：无符号右移运算符

表示无符号右移，即符号位也参与右移。比如int a = 3 >>>1，存储为int类型的数据，那么它的二进制为00000000 00000000 00000000 00000011，从0开始向右边移动1位变为00000000 00000000 00000000 00000001也就是1。

```
r =3;
r = r >>> 1;//00000000 00000000 00000000 00000001
System.out.println(r);//1
```

7.8，运算符优先级和结合规则

- 运算符的结合规则
- 所有的数学运算都认为是从左向右运算的，Java 语言中大部分运算符也是从左向右结合的(左结合性)，只有单目运算符、赋值运算符和三目运算符例外，其中，单目运算符、赋值运算符和三目运算符是从右向左结合的，也就是从右向左运算（右结合性）。
- 运算符的优先级

优先级	运算符
1	()、[]、{}
2	!、+、-、~、++、--
3	*、/、%
4	+、-
5	«、»、>>>
6	<、<=、>、>=、 instanceof
7	==、!=
8	&
9	^
10	
11	&&
12	
13	boolean? e1: e2
14	=、+=、-=、*=、/=、&=、 =、^=、~=、«=、»=、>>>=

8，程序设计基本结构

8.1，什么是程序的基本结构

- 算法相关知识
 - 什么是算法？
- 在前面的学习中我们已经学习到了关于数据结构的相关知识，我们知道数据结构是解决数据如何存的问题，那么算法就是解决数据如何用的问题。抛开复杂的概念，从用户出发，计算机的本质就是输入和输出，输入需要的数据，输出想要的结果，这个过程我们在平时无论是使用手机还是计算机都是看不见的，我们只知道我输入需要的数据，计算机就能给我们想要的结果，

比如我们平时在打游戏的时候，在计算机上，我们都需要依靠于键盘和鼠标不断的劈里啪啦点击，敲击来让游戏角色放技能，走，跑等，在手机上我们需要触摸屏幕来操作游戏主角进行相关的操作，其中我们依靠的键盘，鼠标，触摸屏不断的向计算机输入而输出的结果就是游戏人物按照我们的想法进行行为，在这个过程中我们并不需要知道游戏是如何运行的，我按下键盘上的某个键或者是点击鼠标游戏人物为什么就会做出这样的行为，它的本质是什么。

现在作为一名编程人员我们需要知道的是游戏它也是一种程序，那么它之所以这样运行本质上就是程序是这样设计运行的，所谓的本质就是程序，那么什么是程序呢？在数据结构和算法这门学科中对程序的定义是“程序=数据结构+算法”，对于数据结构的概念我们在之前的学习中已经讲述过了并且还和数据类型之间进行进行了联系而更号的理解Java数据类型的本质，数据结构最本质的一句话可以说是解决数怎么存的问题，好的数据结构决定了好的程序，那么**算法所解决的就是数怎么用的问题**，也就是说它如何用我们输入的数据来进行一些列操作得到我们想要的数。

对于某一类的数据结构总要在其上施加相应的运算，通过对所定义运算的研究才能体现出数据结构的作用。算法是数据在计算过程中的组织方式，为了描述和实现某种操作，我们常常需要设计一种有限的操作序列来进行解决，这就是算法（为了解决某类问题而制定的一个有限长的操作序列），比如我们想要乘高铁，首先我们需要手机买票，然后提前到达车站，过安检，候车，乘车，到站下车，这一系列的操作才能实现乘高铁这样的问题。

- 算法的特性

1. 有穷性：对于任何的一个算法它都应该在有限的步骤之内完成并且每一步都应该在有限的时间内完成。

我们为了解决一个问题而设计的算法最基本的条件就应该满足问题本身的限制，任何一个待解决的问题，解决它的步骤都应该是朝着问题完成的方向进行的，也就是说问题总要在有限的步骤内解决，除此之外问题的解决还有一定的时间段，总不能说今天要参加高考我不想考我明天再考，这是不现实也不可能的，所以算法的有穷就是指解决问题的现实性。

2. 确定性：对于解决问题过程中的每一步操作都应该是确定的，不能有含糊不清和模棱两可的意思。

我们针对问题而解决的每一个操作不是这个操作好像能干这个而是这个操作就是干这个的意思，比如我要再淘宝上购物，我点击放到购物车这个功能，那么它就应该是把我们选择的东西放到购物车中而不是好像是把东西放在购物车中。

3. 可行性：算法的每一步都必须是可行的，也就是说，每一步都通过执行有限次数完成。

整个算法的步骤是有穷的，对于实现这个算法的每一步也是有穷的执行，所谓的可行就是说这个步骤是能够确切实现的，比如我想要开发可以控制地球自转的程序，当然是不可能实现的，我要开发一款控制飞行器的程序，它是可行的。

4. 输入：有零个输入或多个输入

计算机就是输入和输出，所以算法的设计需要实现输入，对于输入来说可以是0个输入，比如我们打剧情游戏，当我们打败游戏中的大boss之后，游戏会自动的进入到打败boss之后的场景，这个过程我们并没有输入任何东西，实际上这是一种事件驱动的编程思想，当我们实现到某个事件的时候程序就会执行一些操作，实际上这是程序帮助我们输入了内容。

5. 输出：有一个输出或多个输出

输出必须有1个输出，这也很好理解，如果没有输出什么结果，那么这个算法就是无意义的，它就是解决计算机的输入和输出的这个过程。

- 优秀的算法具有的标准

算法是解决一个问题的步骤，对于一个问题的解决我们可以有不同的步骤，对于每一步所执行的操作我们也可以不同，也就是说解决一个问题的算法不唯一，既然有那么多好的算法就有好的算法和差的算法之分，就如要制作一盘菜，不同的师傅做出来的也好也有坏，对于一个好的算法来说，它应当具有的标准是。

1. 正确性

在输入合理数据的情况下能正确的输出结果，这是最基本的算法要求，满足1的程序可以是合格的程序。

2. 可读性

一个好的算法首先要有好的可读性，也就是每一步是如何实现的，其他人一看都能一目了然其含义，对于复杂的步骤注释也是好的习惯，满足1，2的程序可以是中等的程序。

3. 健壮性

- 容错性：当输入的数据非法时，好的算法应当做出正确的处理而不是输出一串莫名其妙奇妙的结果，在Java中我们知道Java的异常机制就很好的满足了这一属性。
- 稳定性：算法的实现步骤应当是稳定的，所谓的稳定性就是在程序的运行中不会产生一些随机的错误。

满足1，2，3的程序可以是良好的程序。

4. 高效性

所谓的高效体现在空间和时间的合理利用上。

时间的高效是指在规定的时间内，对于同一问题而设计的不同算法中，执行速度最快结果正确的算法就是时间高效的算法，使用算法的时间复杂度进行衡量。

空间的高效是指算法占用内存空间的合理性来看待的，对于同一个问题而设计的不同算法中，所占用空间最小并且结果正确的算法就是空间高效的算法，使用算法的空间复杂度进行衡量。

根据具体的问题来均衡时间和空间之间的平衡点选择的算法才是优质的算法。

满足1，2，3，4的程序可以是优秀的程序。

5. 可扩展性

我们知道问题会随着时间的推移而进行变化，更为优秀的算法能够提前分析出问题的走向并且在原来的基础上为以后的扩展留出可扩展的余地。满足1，2，3，4，5的程序可以是非常优秀的程序。

• 什么是程序的基本结构

算法是实现一个程序问题的有限步骤，可以说算法是一个程序的灵魂，它是一个程序结构的组成。我们说软件的核心是程序，程序的核心就是数据结构和算法，数据结构和算法的核心就是算法。对于算法来说，无论多么复杂的算法结构它都可以由三种最基本的结构组成也就是顺序结构，选择结构和循环结构，这就是程序的基本结构。

不论是面试还是在实际开发中，可以说编程语言仅仅是编程人员实现的工具，数据结构和算法才是一个编程人员在实际的开发中的核心素养之一，所以作为程序员我们不仅仅要把目光放在编程语言上更重要的是对核心素养的培养。除了数据结构和算法，计算机组成原理，操作系统，计算机网络，编译原理，软件工程，使用各种工具等是计算机人员的专业素养核心。

8.2，顺序结构

• 算法描述方式

算法的描述方式可以分为三种，自然语言描述，流程图描述和伪代码描述。

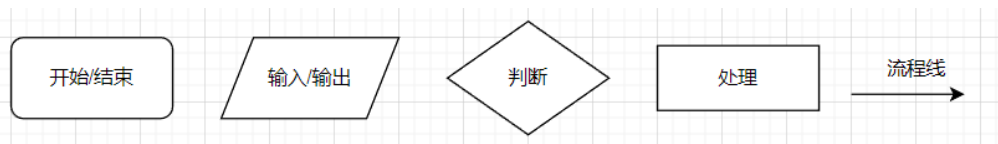
1. 自然语言描述

所谓的自然语言描述就是使用我们的日常用语去将一个算法的步骤表述出来。比如我们上述描述的坐高铁的步骤或者是老师上课讲述一道数学题时，他会讲这道题首先应该怎么做，然后怎么做，最后怎么做这样的接替步骤，这就是自然语言描述，说白了就是表述出这个算法如何实现的步骤。

2. 流程图描述

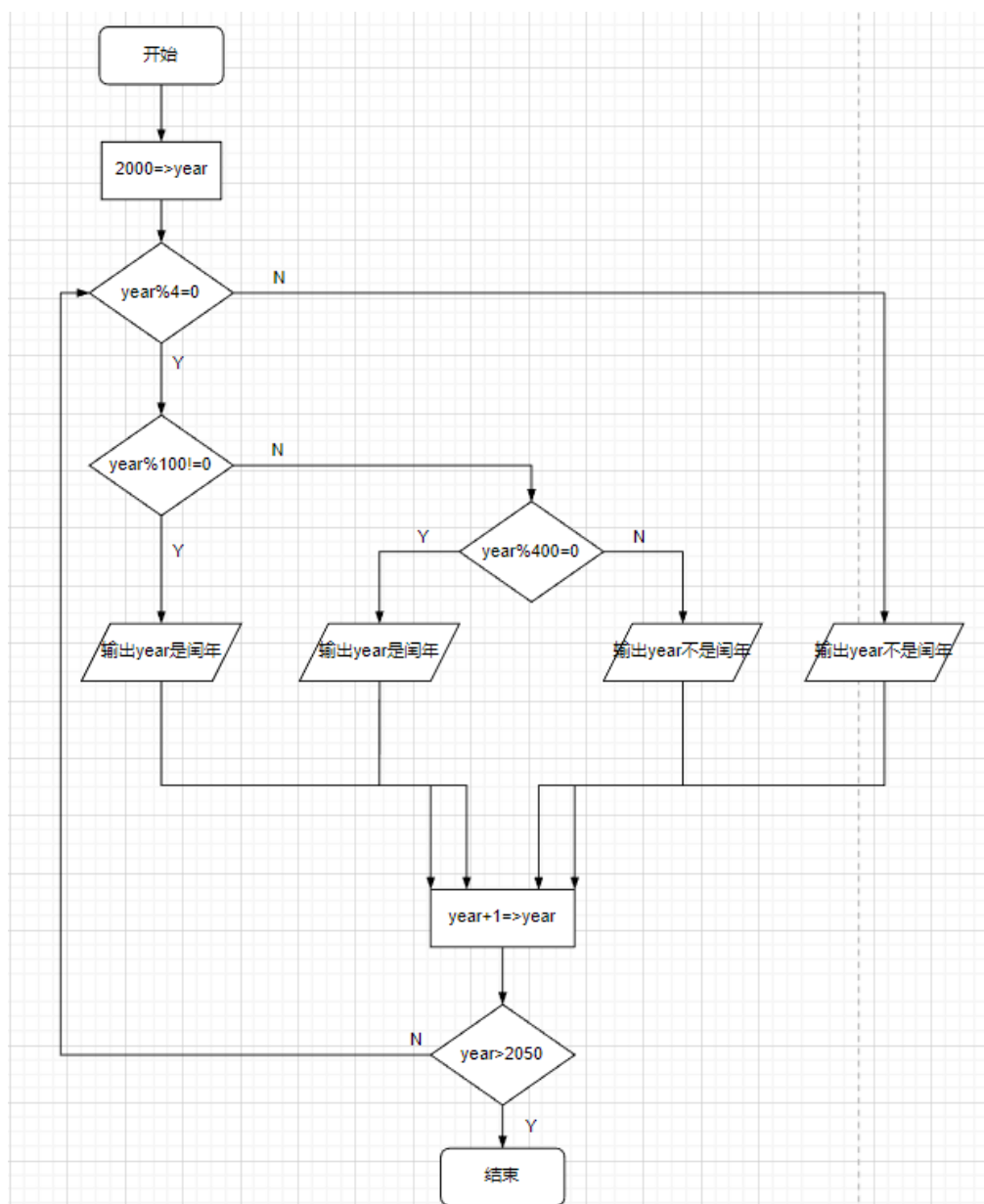
所谓的流程就是算法的每一个步骤，流程图就是用更为直观的图形化的形象来表述一个算法的实现过程。对于一个程序员来说，看流程图也体现着一个程序员的基本能力，这里的流程图应该称为算法流程图或程序流程图，在之后会学到类图，原型图，泳道图（跨功能流程图），控制流图等。所以对于一个程序员最初接触这个行业时就需要能够看懂程序和绘制基本的程序流程图。

■ 程序流程图基本元素



在流程图中=就是等于的含义，赋值使用=>符号

■ 使用程序流图描述判定2000到2050年是闰年的程序



3. 伪代码描述

所谓的伪代码就是一种介于自然语言和计算机程序语言之间的一种描述算法的方式。伪代码，所以它并不是真实的代码，不能够作为某种程序语言的源代码进行执行，它不是任何一种程序语言，没有固定严格的语法规则，它的核心就是把算法的意思表达清楚即可。比如我们接着描述判定2000到2050年是闰年的算法，如下：

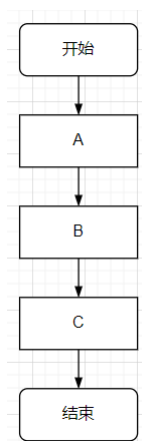
```
{
    int year=2000;
    while(year<2050){
        if(year%4==0){
            if(year%100!=0){
                输出是year闰年;
            }else if(year%400==0){
                输出是year闰年;
            }else{
                输出year不是闰年;
            }
        }else{
            输出year不是闰年;
        }
        year++;
    }
}
```

也可以不像我这样写，只要能够使用类似于代码的形式将算法表述清楚即可。

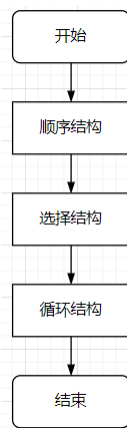
有了上述的算法的描述方式之后，我们接下来是使用具体的计算机语言来实现这些算法的描述，无论用何种的算法描述，它的最终目的就是说明这个程序所解决的问题，最后只要能使用计算机语言实现即可。

- 顺序结构基本概述

所谓的顺序结构从狭义的说代码的执行顺序是从上往下，顺序依次执行的，没有分支，每一句必走。它是最简单的结构也是最基本的结构。它的程序流程图为：



广义上说实际上我们所写的程序如果把每一个基本结构看作是一个模块，最后的整体可以看作是从上往下顺序执行的。



8.3, 选择结构

- 选择结构基本概述

有条件的去执行某个语句，并非每句必走。也就是说基于条件决定执行哪些语句。选择结构需要借助于布尔值也就是true和false作为结果条件进行选择那些语句该走，那些语句不该走。

- if结构

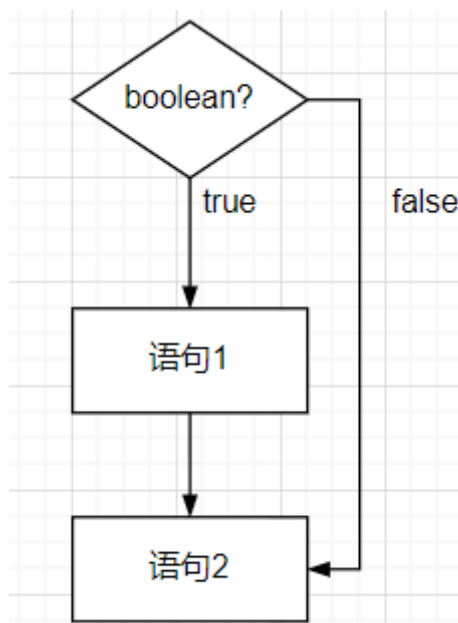
- 一条路的情况。也就是说满足条件就干事，不满足条件就不干事。比如满500打8折，不满500不打折。
- 结构表达

```
语句0;  
if(条件(逻辑表达式, 只要结果是boolean值即可)) {  
    语句1;  
}  
语句2;
```

- 结构分析

首先执行语句0;
当条件为true->执行if内语句1，然后执行语句2。
当条件为false->执行语句2。
有可能不会走。

- 程序流程图



- if-else结构

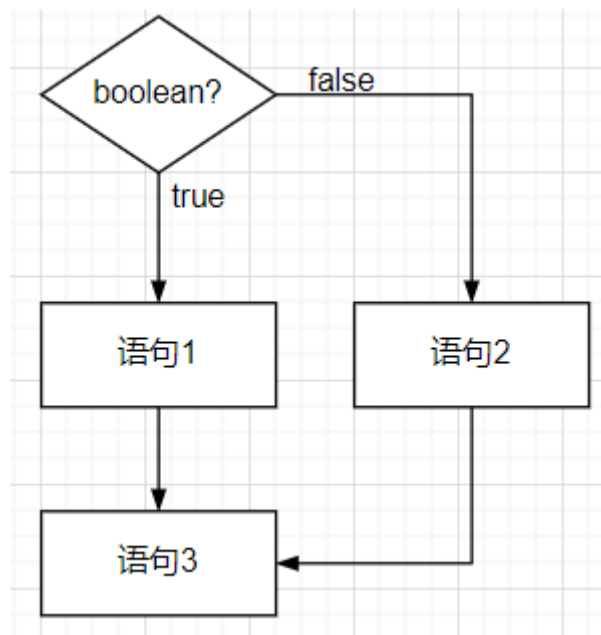
- 两条路的情况。也就是说满足条件就干事，不满足条件就干另一件事。比如满500打8折，不满500打9折。
- 结构表达

```
语句0;  
if(条件boolean condition){  
    语句1;  
}else{  
    语句2;  
}  
语句3;
```

- 结构分析

首先执行语句0;
当条件true->执行if内语句1，然后执行语句3。
当条件false->执行语句2，然后执行语句3。
至少走一路。

- 程序流程图



实际上到现在为止我们其实已经完全学习了关于选择结构的内容也就是if和if-else接下来所学习的是它们两者的衍生形式。

案例：收银台

- 要求：用户输入单价，数量和金额，程序按照满500打8折，不满500打9折的规则，如果用户支付金额满足输出总价和找零，不满则提醒用户钱不够还差多少。
- 分析：程序接收用户的输入，先通过单价和数量计算总价，根据总价按照满500打8折，不满500打9折的规则计算折扣得到最后的总价，最后根据用户输入的金额和最后的总价进行比较如果用户支付金额满足输出总价和找零，不满则提醒用户钱不够还差多少。
- 编码：

```
package java_language_basic;  
import java.util.Scanner;  
/**
```



```

* 收银台
* @author ROOTer
*
*/
public class Demo2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("请输入单价: ");
        double unitPrice = scanner.nextDouble();//获取输入的单价
        System.out.print("请输入数量: ");
        int amounts = scanner.nextInt();//获取输入的数量
        System.out.print("请输入金额: ");
        double money = scanner.nextDouble();//获取输入的金额
        //按照规则计算得最后的总价
        double totalPrice = unitPrice*amounts;//最初的总价
        if(totalPrice>500) {
            totalPrice = totalPrice*0.8;
        }else {
            totalPrice = totalPrice*0.9;
        }
        if(money>=totalPrice) {
            System.out.println("总价为: "+totalPrice+",应找零:"+(money-
totalPrice));
        }else {
            System.out.println("您的金额不足还差"+(totalPrice-money));
        }
    }
}

```

测试

测试用例: 249.5 2 449.1, 期望结果: 总价为449.1, 应找零0.0。

```

请输入单价: 249.5
请输入数量: 2
请输入金额: 449.1
总价为: 449.1, 应找零: 0.0

```

测试用例: 249.5 2 449, 期望结果: 您的金额不足还差0.1。

```

请输入单价: 249.5
请输入数量: 2
请输入金额: 449
您的金额不足还差0.100000

```

测试用例: 250 2 451, 期望结果: 总价为: 450.0,应找零:1.0。

```

请输入单价: 250
请输入数量: 2
请输入金额: 451
总价为: 450.0, 应找零: 1.0

```

案例: 判断奇数和偶数

- 要求: 用户输入一个整数然后输出该数是奇数还是偶数。
- 分析: 输入一个整数不是奇数就是偶数, 所以使用if-else结构进行书写, **在开发时先把if-else结构写好也就是if(){else{}**, 让整数和2取余如果余数为0则被2整除就是偶数, 否则为奇数。
- 编码

```

package java_language_basic;
import java.util.Scanner;
/**
 * 判断奇数和偶数
 * @author EMCred
 */
public class Demo2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("请输入一个整数: ");
        int getNum = scanner.nextInt();
        if(getNum%2==0) {
            System.out.println(getNum+"是偶数");
        }else {
            System.out.println(getNum+"是奇数");
        }
        System.out.println("over");
    }
}

```

○ 测试

测试用例：1，期望结果：1是奇数over。

```

请输入一个整数:
1
1是奇数
over

```

测试用例：2，期望结果：2是偶数over。

```

请输入一个整数:
2
2是偶数
over

```

• if-else if结构

- 多条路情况。也就是说满足条件就干事，不满足条件就干另一件事，都不满足就一件不干。比如满500打6折，满400打7折，满300打8折，那么此时如果只满200就不会打折。

○ 结构表达

```

语句0:
if(条件1boolean condition){
    语句1;
}else if (条件2boolean condation){
    语句2;
}else if(条件3boolean condition){
    语句3;
}else if(条件4boolean condition){
    语句4;
}
语句5:

```

○ 结构分析

首先执行语句0;

当条件1true->执行if内语句1, 然后执行语句5。

当条件1false->条件2true->执行if内语句2, 然后执行语句5。

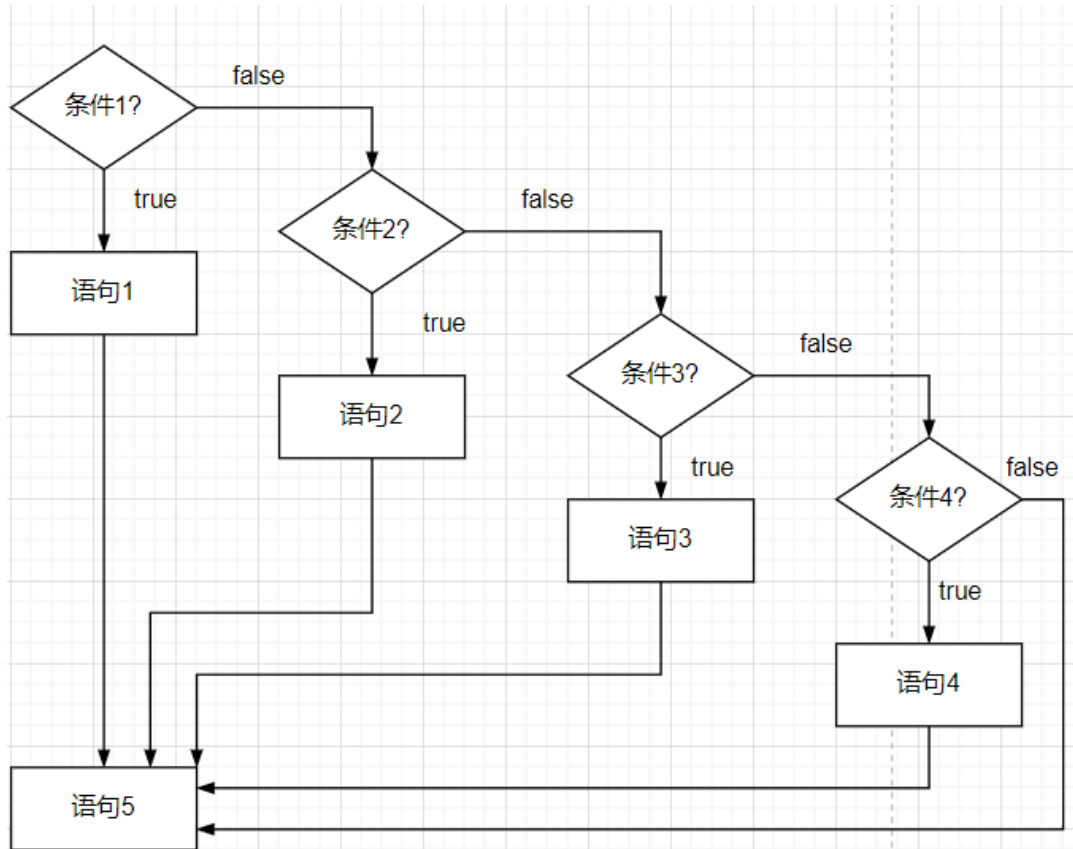
当条件2false->条件3true->执行if内语句3, 然后执行语句5。

当条件3false->条件4true->执行if内语句4, 然后执行语句5。

当条件4false->执行语句5。

有可能一路都不走。

○ 程序流程图



• if-else if-else结构

- 多条路情况。也就是说满足条件就干事, 不满足条件就干另一件事, 都不满足就干其他事。比如满500打6折, 满400打7折, 满300打8折, 不满300打9折。

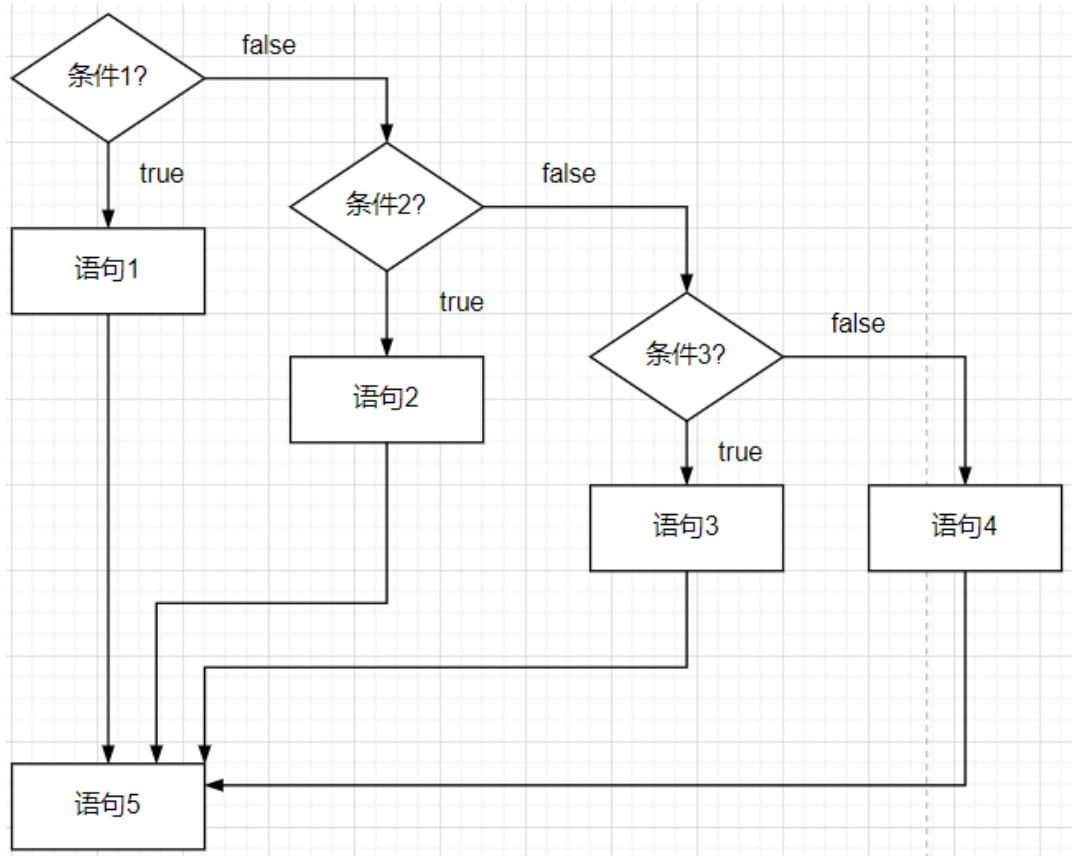
○ 结构表达

```
语句0;  
if(条件1boolean condition){  
语句1;  
}else if (条件2boolean condation){  
语句2;  
}else if(条件3boolean condition){  
语句3;  
}else{  
语句4;  
}  
语句5;
```

○ 结构分析

首先执行语句0；
当条件1true->执行if内语句1，然后执行语句5。
当条件1false->条件2true->执行if内语句2，然后执行语句5。
当条件2false->条件3true->执行if内语句3，然后执行语句5。
当条件3false->执行语句4，然后执行语句5。
至少走一路。

○ 程序流程图



○ 与if-else if结构的区别

最大的区别就是if-else if有可能一路都不会走，但是该结构至少走一路。

案例：学生成绩按等级输出

- 要求：学生成绩按照百分制，学生成绩分为4个等级，A:成绩 ≥ 90 B:成绩 $\geq 80 \& \& < 90$ C:成绩 $\geq 60 \& \& < 80$ D:成绩 < 60 ，当用户输入学习程成绩时需要判断数据的合法性，不合法则提醒用户输入合法数据，然后根据输入的数据按照等级输出。
- 分析：接收用户的输入数据，判断数据的合法性，根据等级进行输出。ABCD四个等级，必然走其中之一，所以使用if-else if-else结构。
- 编码：

```
package java_language_basic;
import java.util.Scanner;
/**
 * 学生成绩按等级输出
 * @author ROOTer
 *
 */
public class Demo3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double score = scanner.nextDouble();
```

```

        if(score<0 || score>100) {
            System.out.println("输入的数据不正确，请重新输入");
        }else {
            if(score>=90) {
                System.out.println("A");
            }else if(score>=80) { //当数据走到这里的时候说明数据<90，此处就相当于
80<=数据<90
                System.out.println("B");
            }else if(score>=60) {
                System.out.println("C");
            }else{
                System.out.println("D");
            }
        }
    }
}

```

- 测试：

测试用例-1和100，预期结果输入的数据不正确，请重新输入。

```

-1
输入的数据不正确，请重新输入

```

测试用例82，预期结果B。

```

82
B

```

- switch-case结构

- switch-case的作用本质

switch-case的作用本质实际上就是if-else if结构和if-else if-else结构。if-else if结构和if-else if-else结构可以完全替换为switch-case结构，但是switch-case结构不能完全替换为if-else if结构或者是if-else if-else结构。它们在底层的实现上有着不同的原理。

switch-case是判断byte,short,int,char和String类型（JDK1.7）的值相等然后进行选择的结构，也就是对整数和字符串判断相等的选择结构，其判断的依据还是依靠于boolean类型的值进行最后的选择。

- switch-case结构

- switch-case结构作用本质相当于if-else if结构，也就是说它有可能一步都不走。
 - 结构表达

```

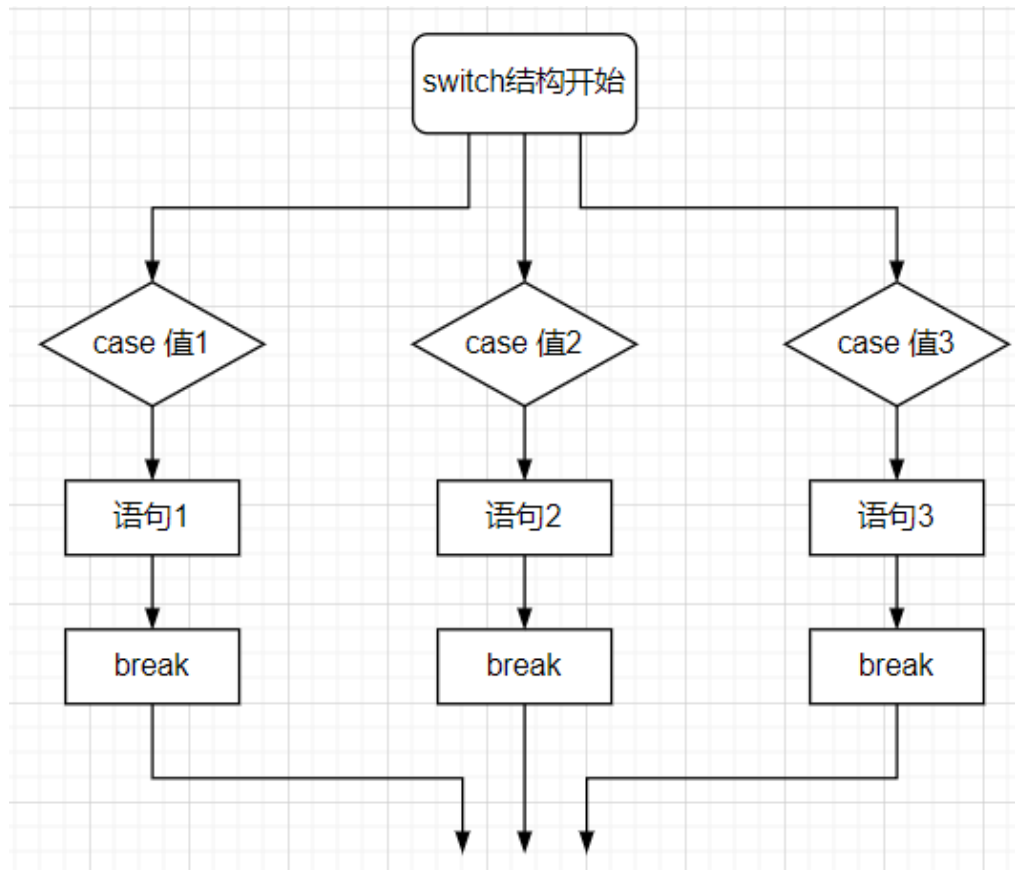
switch（表达式，只能是byte,short,int,char和String类型的值）{
    case 值1:语句1;break;
    case 值2:语句2;break;
    case 值3:语句3;break;
}

```

- 结构分析

表达式的值为值1->case 值1中的语句1然后break跳出switch-case结构。
 表达式的值为值2->case 值2中的语句2然后break跳出switch-case结构。
 表达式的值为值3->case 值3中的语句3然后break跳出switch-case结构。
 表达式的值为值4->表达式的值不在结构的值范围中->不执行switch-case中的任何语句。
 可能一句都不会走

- 程序流程图



- switch-case-default结构

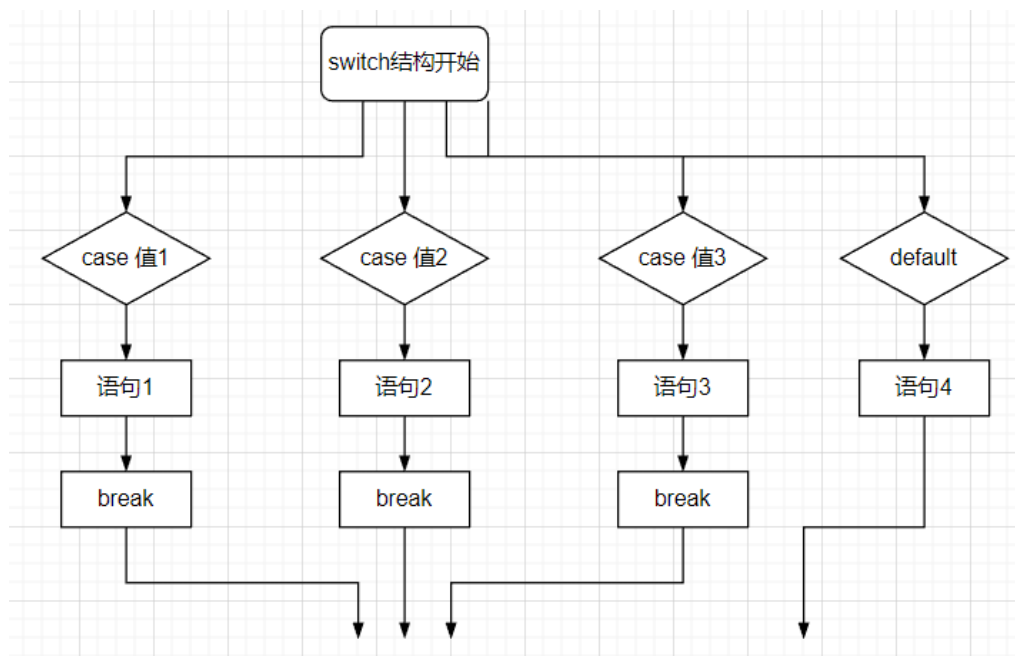
- switch-case-default结构相当于if-else if-else结构，也就是说它至少会走一路。
- 结构表达

```
switch (表达式, 只能是byte, short, int, char和String类型的值) {  
    case 值1: 语句1; break;  
    case 值2: 语句2; break;  
    case 值3: 语句3; break;  
    default: 语句4;  
}
```

- 结构分析

表达式的值为值1->case 值1中的语句1然后break跳出switch-case结构。
表达式的值为值2->case 值2中的语句2然后break跳出switch-case结构。
表达式的值为值3->case 值3中的语句3然后break跳出switch-case结构。
表达式的值为值4->表达式的值不在结构的值范围中但是存在default->执行default中的语句4然后跳出结构。
至少会走一路。

- 程序流程图



■ default的位置

default的位置实际上可以在switch结构中的任何位置，但是我们在使用switch结构的时候建议将default放在最后，在实际开发中这是一种开发的代码规范。

default关键字的意思是默认，其作用是当所有的值都匹配不上的时候才会选择从default中走，也就是说它是最后的选择也是必须的选择，我们之前讲过关于算法的优秀特性，在其中的可读性规定所写的程序代码应当能够一看便知其作用，实际上还有一层更深的意义就是代码的书写应当和我们的认知规范是一致的也就是说最后必须走也不得不走的一条道路应当是放在最后的一个办法，在语义上和我们的认知规范对应，所以default正和此意。

由于上述的语义限制，所以如果要将default放在其他的位置应该加上break关键字，否则程序的语义性和合理性就会出现问題，最好就是按照规范将default放在最后的位置。

○ switch中的break作用

我们在书写switch结构的时候可以看到，在switch结构中的每一条最后都会有一个break。

break的英文意思是冲破，突破，在这里break作为出现中的关键字可以理解为跳出，在这里说的是跳出switch结构，在之后的循环结构中我们还将深入理解循环中的break关键字，在这里break是作为switch结构的break而不是循环结构中的break，它们有着一些不同的区别，但是它们的作用都是跳出结构。

当程序执行到break时就会直接跳出结构接着执行结构下方的其他语句，也就是说在下面的语句中，num是2，进入switch结构中找到case 2，有case 2，执行case2中语句，当执行到break的时候它就会自己跳出switch结构，也就是说其他的case就不会被执行，然后执行System.out.println(num);//2

```
int num = 2;
switch (num) {
    case 1:
        System.out.print(1);
        break;
    case 2:
        System.out.print(2);
        num=2;
        break;
    case 3:
        System.out.print(3);
        break;
```



```
default:
    System.out.print(4);
}
System.out.println(num);//2
```

其实到这里会有一个误解，也就是可能会理解为break关键字是和switch结构配套的一个关键字，是必须加上的，实际上break关键字可以不写的。

当不写break的时候，执行的语句会从不写break的语句开始一直到break或者是到最后一句时才会执行结束。也就是说如下的情况。

```
int num = 1;
switch (num) {
    case 1:
        System.out.print(1);
    case 2:
        System.out.print(2);
        num=2;
    case 3:
        System.out.print(3);
        break;
    default:
        System.out.print(4);
}
```

此时程序会输出123，num=1，进入switch结构中寻找case 1，存在case 1，进入执行case 1的语句System.out.print(1);此时没有break，接着执行case 2的内容直到case 3到break的时候跳出switch结构，不在执行。

这里就会出现一种误解就是说如果我去掉case 3的break岂不是程序会无限的执行下去，因为程序无法跳出switch，实际上这是一种理解上的错误，我们看到switch结构中的内容如果没有break那就是一个标准的顺序结构，所以当它执行到最后一句时如果也没有break它也会结束，它不是循环结构还会回到开头接着执行。

```
int num = 1;
switch (num) {
    case 1:
        System.out.print(1);
    case 2:
        System.out.print(2);
        num=2;
    case 3:
        System.out.print(3);
    default:
        System.out.print(4);
}
```

此时程序会输出1234，然后结束switch结构，执行switch结构下面的语句。

关于switch中的break可以总结为如下：

- break的作用就是跳出当前的switch结构继续执行结构下方的其他语句。
- 实际上break关键字不是必须要求写的，它可以不写。

我们也可以称完全带有break的switch结构是标准的switch结构，在实际开发中我们会根据实际情况来使用标准switch结构和非标准switch结构，比如我们想要输入平年的月份获取天数这样的功能，那么非标准的switch结构比标准的switch结构更加的简洁。可以发现此时没写break的语句就相当于if条件中的或，而每一句就相当于if的条件。

```
int month =1;
switch (month) {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        System.out.println("31天");
        break;
    case 2:
        System.out.println("28天");
        break;
    case 4:
    case 6:
    case 9:
    case 11:
        System.out.println("31天");
        break;
    default:
        System.out.println("输入月份错误");
        break;
}
```

○ switch结构和if结构的区别

我们之前说过switch结构的作用和if-else if或if-else if-else的作用是一样的，用if结构也能够完全的实现switch结构的东西，比如上面的例子，我们可以写成

```
int month = 1;
if(month==1 || month==3 || month==5 || month==7 || month==8 || month==10 || month==12){
    System.out.println("31天");
}else if(month==2){
    System.out.println("28天");
}else if(month==4 || month==6 || month==9 || month==11){
    System.out.println("31天");
}else{
    System.out.println("输入月份错误");
}
```

那么if结构和switch结构的区别是什么？

- 语法结构不同：相对于if结构来说switch结构更加的简洁明白，当对于分支较多的if结构来说switch结构就显得特别的简洁明白，也就是可读性更好。
- 适用范围不同：if结构可以适用于所有的程序的逻辑结构的判断，但是switch结构只能用于整数的判断相等进行选择Java在JDK1.7之后可以进行字符串的相等判断，相对于if结构来说适用范围更小。
- 底层的执行原理不同：这里就只说if结构和switch结构的底层实现是不同的，由于底层实现不同两者在执行效率和占用空间上也就是不一样的，相对于if来说switch结构的效率更

高，它采用索引算法进行数据的查找，但是会生成一个中间的索引表，相对来说空间消耗会打一点，if结构它是每一个目标都要进行执行，所以相对来说效率会比switch慢。

- switch结构的最佳使用情况

根据上述所说的，switch结构在Java中的最佳适用情况就是如果需要判断整数相等或者是字符串相等的情况下使用最佳，所以对整数相等或字符串相等进行判断的时候首选switch结构。

案例：实现简单的两数计算器

- 要求：用户输入两个数，然后选择需要的加减乘除中的一个进行运算，最后输出结果并显示用户所使用的运算，需要进行非法输入的判断，也就是输入了除了+,-,*, /之外的其他符号要提醒用户。
- 分析：接收用户的输入和选择，由于设计+,-,*/，也就是需要判断输入的字符串，然后根据对应的字符串输出对应的结果，属于判断字符串相等的情况，可以使用if-else if-else结构但是首推switch-case-default结构。
- 编码

```
package java_language_basic;

import java.util.Scanner;

/**
 * 简单的两数计算器
 * @author EMCred
 *
 */
public class Demo5 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("请输入第一个数");
        double num1 = scanner.nextDouble();
        System.out.println("请输入第二个数");
        double num2 = scanner.nextDouble();
        System.out.println("请输入需要进行的运算只能是+-*/");
        String oper = scanner.next(); // 获取输入的字符串
        switch (oper) {
            case "+":
                System.out.println("加法: " + (num1 + num2));
                break;
            case "-":
                System.out.println("减法: " + (num1 - num2));
                break;
            case "*":
                System.out.println("乘法: " + (num1 * num2));
                break;
            case "/":
                System.out.println("除法: " + (num1 / num2));
                break;
            default:
                System.out.println("输入的操作符不正确!!! 只能是+-*/");
                break;
        }
    }
}
```

- 测试

测试用例1 2 +, 期望结果3.0

```
请输入第一个数
1
请输入第二个数
2
请输入需要进行的运算只能是+-*/
+
加法: 3.0
```

测试用例1 2 =,期望结果“输入的操作符不正确!!! 只能是+-*/”

```
请输入第一个数
1
请输入第二个数
2
请输入需要进行的运算只能是+-*/
=
输入的操作符不正确!!! 只能是+-*/
```

- 嵌套的选择结构

所谓的嵌套选择结构就是将多种选择结构嵌套在一起进行使用，我们到目前为止所接触的选择结构为if结构,if-else结构，if-else if结构，if-else if-else结构和switch-case结构，这些结构可以自由相互组合，根据业务的需求组合出合适的逻辑判断，然后进行使用，是算法组成中的选择部分。

8.4, 循环结构

- 循环结构概述

所谓的循环就是重复的干某件相同的事情，在程序中的循环结构基于条件的结果也就是布尔值来执行。说白了就是有条件的执行，并非每句必走，基于条件执行去反复地执行一段相同或相似的代码。

对于循环结构来说，它的组成可以总结为4个要素，在之后的学习中，我们将对循环结构的四个要素进行剖析。现在先来认识一下循环的四要素。

- 循环四要素

- 循环的变量：一直在变的那个数。
 - 循环的初始化：一直在变的那个数是从那个数开始的。
 - 循环的条件：一直在变的那个数以怎样的条件改变。
 - 循环的改变：一直在变的那个数是朝着什么样的方向改变直到结束的。

- while结构

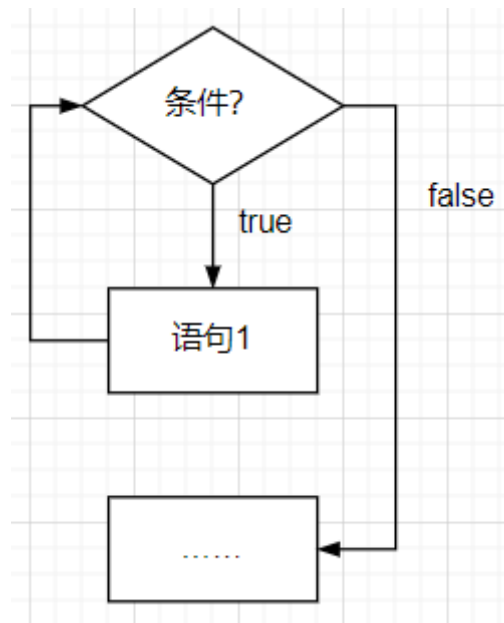
- 当型循环结构，可以理解为“当.....的时候”这样的意思就可以使用当型循环结构。
 - 结构表达

```
语句0;
while(条件boolean condition){
    语句1;
}
```

- 结构分析

首先执行语句0;
当条件的布尔值为true时->执行语句1。
当条件的布尔值为false时->跳出循环。
可能不会进入循环。

○ 程序流程图



○ while结构的循环四要素分析

```
int count = 0; (1)
while(count < 5) { (2)
    System.out.println("Hello"); (3)
    count++; (4)
}
```

- 循环的变量: count
- 循环的初始化: count = 0
- 循环的条件: count < 5
- 循环的改变: count++
- 循环分析:

(1):count=0 -> (2):count=0<5 -> (3) -> (4):count=1
->(2):count=1<5 -> (3) -> (4):count=2
->(2):count=2<5 -> (3) -> (4):count=3
->(2):count=3<5 -> (3) -> (4):count=4
->(2):count=4<5 -> (3) -> (4):count=5
->(2):count=5==5 -> over

案例：100以内的整数加法练习小程序

- 要求：运行程序时，程序自动的给出每道题目并指出当前的题号，等待用户输入正确的答案，根据用户输入的答案，程序自动判断对错，如果对则记10分，错误不记分并且将正确答案展示，当完成10道题之后，程序需要输出用户的得分和耗时，记录时间采用秒为单位进行记录时间。

- 分析：根据要求，这个业务需要重复的执行出题判题这样的代码，所以需要使用循环结构更为简洁，当完成10道题，所以这里采用while循环结构，这里的循环4要素是循环的变量就是题目的数量，变量名定为count，循环的初始化，题目是从第一道题开始的，所以循环的初始化为count = 1，循环的条件，当10道题结束之后，所以循环的条件是count<=10时进入while循环进行执行，循环的改变，每做完一道题题目数量count增1，所以循环的改变应该是count++。在循环中每次的题目应当是不一样的，所以需要随机生成两个100以内的整数，所以这里选择使用Math类中的random方法，为了生成100以内的整数，所以应该写成(int)(Math.random()*100);然后输出到控制台，接下来接收用户输入的答案，然后进行判断如果输入的答案和随机数之和的结果是一样的，那么判断为对，输出正确并且分数加上10分，否则输出为错误，正确答案为的形式。对于分数来说，它是一种基于条件的记录，也就是对加10分，错不加分。对于时间，使用我们之前讲到的System.currentTimeMillis()，它的每时每秒都是不一样的并且使用ms作为单位，所以在执行循环之前用变量startTime记录开始的时间，在循环结束之后使用变量endTime记录结束的时间，两者之差除以1000就是花费的秒数。
- 编码：

```
package java_language_basic;

import java.util.Scanner;

/**
 * 100以内的整数加法练习小程序
 *
 * @author ROOTer
 *
 */
public class Demo6 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int count=1;//循环的初始化
        int score = 0;//分数的初始化
        long startTime = System.currentTimeMillis();//开始时间
        while(count<=10) {
            int num1 = (int)(Math.random()*100);
            int num2 = (int)(Math.random()*100);
            System.out.println("第"+count+"题: "+num1+"-"+num2+"=");
            int input = scanner.nextInt();
            if(input==num1+num2) {
                score+=10;
                System.out.println("正确");
            }else {
                System.out.println("错误, 正确答案为"+(num1+num2));
            }
            count++;
        }
        long endTime = System.currentTimeMillis();//结束时间
        long time = (endTime-startTime)/1000;
        System.out.println("最终得分为: "+score);
        System.out.println("最终用时为"+time+"s");
    }
}
```

- 测试
- 正确性测试

```
Problems Javadoc Declaration Console X
<terminated> Demo6 [Java Application] D:\ProgramFiles\eclipse\ eclipse\
正确
第5题: 3+88=
91
正确
第6题: 68+39=
107
正确
第7题: 7+40=
47
正确
第8题: 3+67=
70
正确
第9题: 93+48=
141
正确
第10题: 21+53=
74
正确
最终得分: 100
最终用时: 50s
```

错误性测试

```
Problems Javadoc Declaration Console X
<terminated> Demo6 [Java Application] D:\ProgramFiles\ed
第5题: 46+34=
80
正确
第6题: 1+98=
99
正确
第7题: 55+12=
67
正确
第8题: 23+70=
93
正确
第9题: 90+37=
123
错误, 正确答案为127
第10题: 12+73=
85
正确
最终得分: 90
最终用时: 48s
```

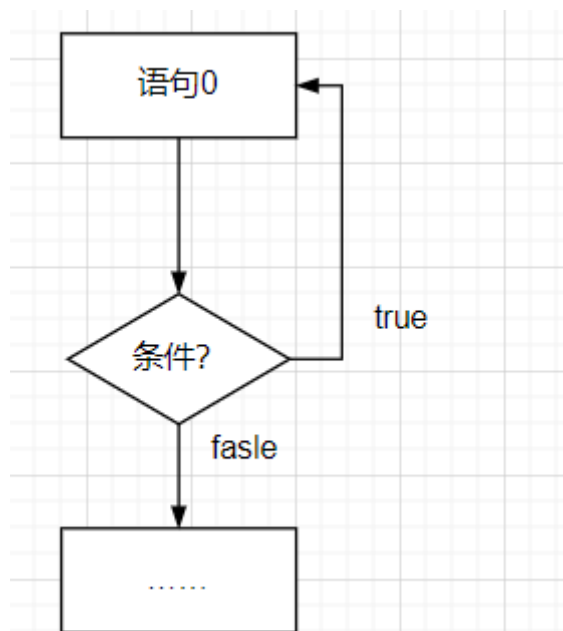
- do-while结构
 - 直到型循环结构, 可以理解为"直到.....就"这样的意思就可以使用do-while结构。
 - 结构表达

```
do{
    语句0;
}while(条件boolean condition);
```

- 结构分析

首先先执行语句0;
循环的条件为true是->接着执行语句0。
直到循环的条件为false时->跳出循环。
至少进入一次循环。

- 程序流程图



◦ do-while结构循环四要素分析

```

int count = 0; (1)
do{
    System.out.println("Hello"); (2)
    count++; (3)
}while(count<3 (4));
  
```

- 循环的变量: count
- 循环的初始化: count=0, count=1, count=2, count=3
- 循环的条件: count<10
- 循环的改变: count++
- 循环分析:

(1):count=0->(2)->(3):count=1->(4):count=1<3
 ->(2)->(3):count=2->(4):count=2<3
 ->(2)->(3):count=3->(4):count=3==3
 ->over

◦ do-while结构和while结构的区别

1. do-while结构至少走一次循环而while循环可能不会进入循环。
2. do-while结构的循环的初始化和循环的改变的值是相同的，也就是它的循环的要素2和循环要素4是相同的，while的则是不一样的不可两者兼具。

案例：实现猜数字小游戏

- 要求：程序自动生成0-1000的某个整数，用户输入猜测的整数，当输入的整数比猜测的整数大的时候，输出猜大了，当输入的整数比猜测的整数小的时候，输出猜小了，直到用户输入的整数和程序自动生成的整数一样时，输出恭喜您猜对了，对每次用户输入的整数需要判断其合理性，也就是输入的整数的范围只能在0-1000之间的，并且最后要输入用户一共猜了几次才猜对。
- 分析：在要求中说到直到“用户输入的整数和程序自动生成的整数一样时，输出恭喜您猜对了”，所以这个程序是重复的执行输入整数和判断整数的代码，所以需要使用循环结构，直到.....，这一点可以采用的循环结构是do-while结构，我们需要输入整数作为循环的初始化，所以我们需要设定一个input作为循环的变量，循环的条件是当input!=程序自动生成的整数时重复执行，当==时跳出循环，循环的改变是当input输入的整数和程序自动生成的整数不一样

的时候再次接收用户的输入，所以循环的改变就是循环的初始化，也就是循环四要素中的第2要素循环的初始化和第4要素循环的改变是一样的情况下，使用do-while结构更为合适。进行错误判断，当输入的整数小于0或者是大于1000的时候输出"输入错误，请重新输入，只能输入0-1000的整数"，此时不能算作是所输入的次数，所以这里需要使用docontinue关键字，在循环中的continue关键字的意思是跳过本次循环，执行下次循环，在之后会深入的讲解它和break关键字在循环结构中的作用。

- 编码

```
package java_language_basic;
import java.util.Scanner;
/**
 * 实现猜数字小游戏
 * @author EMCred
 */
public class Demo7 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int count = 0;
        int num = (int)(Math.random()*1000);
        System.out.println(num);
        int input;
        do {
            System.out.println("请输入0-1000之间的某个整数：");
            input = scanner.nextInt();
            if(input>1000||input<0) {
                System.out.println("输入错误，请重新输入，只能输入0-1000的整数");
                continue;
            }
            if(input>num) {
                System.out.println("猜大了");
            }else if(input<num) {
                System.out.println("猜小了");
            }
            count++;
        }while(input!=num);
        System.out.println("恭喜你猜对了!!!");
        System.out.println("一共猜了"+count+"次数");
    }
}
```

- 测试

正确性测试

```
请输入0-1000之间的整数:
100
猜小了
请输入0-1000之间的整数:
500
猜大了
请输入0-1000之间的整数:
200
猜小了
请输入0-1000之间的整数:
400
猜小了
请输入0-1000之间的整数:
450
猜大了
请输入0-1000之间的整数:
445
猜小了
请输入0-1000之间的整数:
446
猜小了
请输入0-1000之间的整数:
447
恭喜你猜对了!!!
一共猜了8次数
```

错误性测试

```
请输入0-1000之间的整数:
-1
输入错误, 请重新输入, 只能输入0-1000的整数
请输入0-1000之间的整数:
1001
输入错误, 请重新输入, 只能输入0-1000的整数
请输入0-1000之间的整数:
176
猜大了
请输入0-1000之间的整数:
175
恭喜你猜对了!!!
一共猜了2次数
```

- for结构

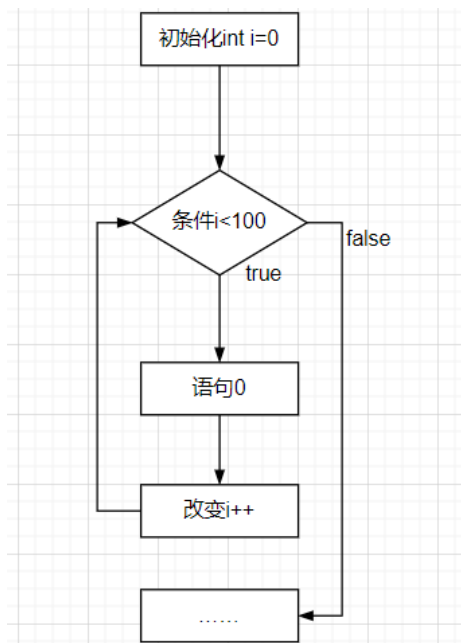
- 实际工作中使用最多的循环结构, 固定次数的循环, 到型循环, 到某个值的时候结束循环。
- 结构表达

```
for(int i=0初始化; i<100条件; i++改变){
    语句0;
}
```

- 结构分析

执行初始化操作->**int i=0;**
根据循环的条件进行判断, 循环条件为**true**->执行语句0;
执行完语句0之后执行改变->**i++;**
将执行的改变与条件进行比较, 条件为**true**时->接着执行语句0;
直到条件为**false**时->跳出循环。
一定会进入到循环并且执行完循环。

- 程序流程图



o for结构循环四要素分析

```

for(int i=0(1);i<5(2);i++(3)){
    System.out.println("Hello");(4)
}
  
```

- 循环的变量: i
- 循环的初始化: int i=0, i=1, i=2, i=3, i=4, i=5
- 循环的条件: i<5
- 循环的改变: i++
- 循环分析:

(1):i=0 -> (2):i=0<5 -> (4) -> (3):i=1

-> (2):i=1<5 -> (4) -> (3):i=2

-> (2):i=2<5 -> (4) -> (3):i=3

-> (2):i=3<5 -> (4) -> (3):i=4

-> (2):i=4<5 -> (4) -> (3):i=5

-> (2):i=5==5->over

o for循环的其他写法

实际上上述我们可以看到for循环直接规定了循环的四要素在其中,使得循环的结构更加明确。在实际开发中所书写的for循环的格式就是如上所示的标准格式,实际上for循环不仅可以这样写,还可以写成如下的写法。

```

//三种循环的解耦写法
//在一定程度上瓦解了for程序的简洁明白性
//循环的初始化解耦
int i = 0;
for(;i<100;i++){
    System.out.println("Hello");
}
//循环的改变解耦
for(int i = 0;i<100;){
    System.out.println("Hello");
}
  
```

```

        i++;
    }
    //循环的改变和初始化解耦
    int i = 0;
    for(;i<100;){
        System.out.println("Hello");
        i++;
    }
    //为了能够让循环不进入无限循环，至少都要保留循环的条件，当然如果完全解耦并且想要不进入到无限循环中的话其实还可以这样写
    int i=0;
    for(;;){
        if(i<100){
            break;
        }
        System.out.println("Hello");
        i++;
    }

    //for循环的组合写法
    //for(循环的初始化;循环的条件;语句1,循环的改变);
    //这样写增加了程序的耦合性，所以使得程序的可读性十分的差
    for(int i=0;i<100;System.out.println("Hello"),i++);

    //多循环的初始化和多循环的改变写法
    //for(循环的初始化1,循环的初始化2.....;循环的条件;循环的改变1，循环的改变2.....){语句1}
    //这样的多循环初始化和多循环的改变的写法实际上在开发中用到的并不是很多
    for(int i=0,j=0;i<100;i++,j++){
        System.out.println("Hello");//输出50次
    }

```

o for循环使用i++和++i的区别

我们在之前说过了关于i++和++i的区别，在单独使用时使用i++和++i的效果是一样的，当被使用的时候，使用i++和++i的效果是不一样的，在for循环中，我们可以看到的是i++或++i是被使用的，那么使用++i和i++有区别吗？

实际上这里我们虽然看到是被使用的，但是在for循环中i++和++i实际上只是形式不一样，它们的作用和单独使用的时候是一样的，这和for循环的实现机制有关，实际上我们可以看到程序流程图中就能很明白的解读了其中含义。

接下来我们也可以做一个实验，如下：

```

package java_language_basic;
/**
 * for中的i++和++i的使用区别
 * @author ROOTer
 *
 */
public class ForJava {
    public static void main(String[] args) {
        int count = 0;
        for(int i=0;i<10;i++){
            System.out.println(i);//0 1 2 3 4 5 6 7 8 9
            count++;
        }
        System.out.println(count);//10
    }
}

```

```
}
```

```
package java_language_basic;

/**
 * for中的i++和++i的使用区别
 * @author ROOTer
 */
public class ForJava {
    public static void main(String[] args) {
        int count = 0;
        for(int i=0;i<10;++i){
            System.out.print(i+" "); //0 1 2 3 4 5 6 7 8 9
            count++;
        }
        System.out.println();
        System.out.println(count); //10
    }
}
```

可以看到此时和单独使用时的情况时一样的。

案例：显示1-1000之间的素数。

- 要求：显示1-1000之间的素数，每行显示15个素数。
- 分析：所谓的素数就是大于1的整数并且只能被1和自己整除的数就是素数比如2，3，5，7，11等，最小的素数是2，我们在设计算法的时候可以这样进行思考，既然2是最小的素数，所以我们循环的初始化应该是从2开始的。那么我们要显示1-1000之间的素数，所以我们的程序是固定从2到1000的，因为不知道1000是不是素数，所以需要 ≤ 1000 ，也就是说要包含1000，也就是说循环的条件应该是 < 1000 。我们需要显示1-1000之间的素数，那么我们就需要每次将当前数字x进行判断处理也就是一直1000，所以循环的改变是 $x++$ 。对于固定次数的循环我们采用for循环结构更为合适。

接下来我们需要考虑到任何数除以1它最后都是本身，所以为了确定一个数是不是素数，我们应该将当前的数字也就是x从2开始逐个相除，也就是说我们还需要设置一个循环在其中让x除以从2开始的每一个整数，此时该循环的初始化为 $num=2$ ，我们可以考虑当 x/num 时最大也就是除到x本身的大小，如果 num 比 x 大的话那么最后的结果就是小数，在程序中就是0所以该循环的条件应当设置为 $num \leq x$ ，循环的改变是一个个的数字与当前数字相除进行验证，所以为 $num++$ 。此时为固定循环每次都应当 $\leq x$ ，所以使用for循环结构更加合适。

根据素数的定义，只能被1和自身整除的大于1的整数，大于1的整数和被1整除已经满足了已经满足了，接下来考虑的是只能被自己所整除。这里有一个重要的也是不好理解的一点，就是如果它没有除到滋生就被整除的话，那么它就一定不是素数，也就是说如果 $x \% num == 0$ 时就可以判断是不是素数了，当它是素数时那么此时的当前数字x应该是 $== num$ 的。比如：如果当前是2，也就是 $x=2$ ， $2 \% 2 = 0$ 此时就可以进行判断 $2=2$ ，所以2是素数，如果当前是4，也就是 $x=4$ ，当 $num=2$ 的时候 $4 \% 2 = 0$ ，此时可以进入判断4是不是素数， $4 > 2$ 不满足 $num \geq x$ 所以4不是素数，如果当前是3，也就是 $x=3$ ， $3 \% 3 = 0$ 此时就可以进行判断是不是素数， $3=3$ 所以此时3是素数。

- 编码：

```
package java_language_basic;

/**
 * 显示1-1000之间的素数
 * @author ROOTer
 */
```

```

*
*/
public class Demo10 {
    public static void main(String[] args) {
        int num;
        int count=0;
        for (int x = 2; x <= 1000; x++) { //x为当前数字
            for (num = 2; num <= x; num++) { //每次进入获取当前数字第一个取余为0
                的数字num
                if (x % num == 0) {
                    break;
                }
            }
            if (num == x) { //判断num和x是不是一致的也就是它本身
                System.out.printf("%4d", x);
                count++;
                if(count==15) {
                    count=0;
                    System.out.println();
                }
            }
        }
    }
}

```

○ 测试:

```

2   3   5   7   11  13  17  19  23  29  31  37  41  43  47
53  59  61  67  71  73  79  83  89  97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173 179 181 191 193 197
199 211 223 227 229 233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349 353 359 367 373 379
383 389 397 401 409 419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509 521 523 541 547 557 563 569 571
577 587 593 599 601 607 613 617 619 631 641 643 647 653 659
661 673 677 683 691 701 709 719 727 733 739 743 751 757 761
769 773 787 797 809 811 821 823 827 829 839 853 857 859 863
877 881 883 887 907 911 919 929 937 941 947 953 967 971 977
983 991 997

```

- continue和break关键字详解

- break关键字

我们在之前的switch-case结构中见到过break关键字，当时我特别的强调是switch结构中的关键字，它的作用的跳出当前的switch结构，而在循环中也是可以使用break关键字的，它的作用可以总结为如下几点：

- break关键字只能在循环中使用，这里的break强调的是循环中的break不是switch结构中的break，并且必须配合选择语句在循环中进行使用。

理解：break关键字的作用是跳出当前的循环，那么应该是循环执行到了某个条件之后才会跳出当前的循环，否则循环需要一直持续下去，所以这里的break满足一层语义当某个条件成立的时候提前结束循环。所以它必须配合选择语句进行使用，比如上述的显示1-1000之间的素数案例，当内层循环满足x%num==0时接下来的num就不需要继续找了，已经找到了，此时跳出循环。在其中的break是在一个判断语句中使用的，所以break配合选择语句在循环中使用。

- break关键字的含义是跳出当前循环，在多层循环嵌套结构中只能跳出一层循环。

理解：break的含义是跳出当前循环，所以也就是只能跳出当前的循环，无论它包含在多少循环之中，比如上面的显示1-1000之间的素数案例，我们可以看到它实际上是由两个for循环进行嵌套的，但是它在满足 $x \% num == 0$ 的条件下仅仅跳出内层的循环，并没有跳出外层的循环，所以才能使得满足对当前数字的每次验证都是从2开始的。

- break关键字的目的是做到某些特殊的操作，不仅仅是Java在其他的语言中也是有break关键字的，break关键字在实际的开发中使用的频率相对较高。

- continue关键字

continue关键字我们在do-while案例：实现猜数字小游戏中使用过，我们在当时说过它的意思是跳过本次循环，关于它的作用可以总结为如下几点：

- continue关键字也只能在循环结构中进行使用并且也必须配合选择语句进行使用。
- continue关键字的含义是跳出本次循环,也就是执行到continue的时候它下面的语句不再执行直接进入下次循环，同样也只对当前循环起到作用。

理解：如何理解跳出本次循环？

比如我想要累加1-100的和，跳过个位为3的数，那么此时程序可以如下编写：

```
int sum = 0;
for(int i=1;i<=100;i++) {
    if(i%10==3) {
        continue;//跳过剩余语句而进下次循环
    }
    sum=sum+i;
}
System.out.println("sum="+sum);
```

如何判断个位为3的数字？

对于个位是3的数字，那么可以采用除10取余的方式进行判断，比如3， $3 \% 10 = 3$ ， $13 \% 10 = 3$ ，这里的除10就相当于获取个位，这是一种程序的编写套路，那么除于100取余就相当于获取十位由此推理。

此时程序的运行过程如下分析：

```
sum=0
i=1->i<100->i%10=1->sum=0+1=1->i+=2
i=2->i<100->i%10=2->sum=0+1+2=3->i+=3
i=3->i<100->i%10=3->i+=4
i=4->i<100->i%10=4->sum=0+1+2+4=7->i+=5
.....
```

- 在实际开发中continue关键字的使用频率实际上并不是很高，它可以被其他的语句进行替换。

理解：比如上面的累加1-100的和，跳过个位为3的数，它的程序还可以这样写：

```
int sum=0;
for(int i=1;i<=100;i++) {
    if(i%10!=3) {
        sum=sum+i;//跳过剩余语句而进下次循环
    }
}
System.out.println("sum="+sum);
```

也就是说既然 $i \% 10 == 3$ 的不行，那么我就只记录 $i \% 10 \neq 3$ 的，这种思考方式是从反面进行推理的。

○ 借助continue关键字理解for循环和while循环的区别

我们已经很具体的讲过了关于for循环，while循环和do-while循环了，那么for循环和它们有什么区别呢？

- 最明显的就是关键字不同
- 最本质的是执行循环的改变方式是不同的

理解：如果我们使用while结构来写累加1-100的和，跳过个位为3的数，我们可能会这样书写

```
int i=1;
int sum = 0;
while(i<=100) {
    if(i%10==3) {
        continue;
    }
    sum=sum+i;
    i++;
}
System.out.println(sum);
```

实际上当运行这个程序的时候会发现，程序可能跑很久它都不会执行出来，这是为什么呢？我们来实际进行分析。

```
i=1->i=1<100->i%10=1->sum=0+1=1->i+=2
i=2->i=2<100->i%10=2->sum=0+1+2=3->i+=3
i=3->i=3<100->i%10=3->continue->此时continue下方的循环内语句不再执行也就是说i=3循环没有驱动，此时循环的改变是个定值，所以程序陷入了一种无限循环的局面
```

我们和上面的for循环所写的程序进行对比之后会发现两者几乎是一样的过程为什么会出现这样的不同点，这就是我们要说的for结构和while结构的循环的改变之间的区别，for的循环的改变是写在()内的，而while和do-while的循环的改变是写在循环的代码块中的{}。

for循环的循环的改变在()中，所以执行完{}中的内容并不影响for的循环的改变而while结构的循环的改变是直接写在{}中的也就是说循环的改变会受到执行的影响这和它们之间的底层实现机制是不一样的。

对于for来说：

```
for(int i=1(1);i<=100(2);i++(3)) {
    if(i%10==3(4)) {
        continue;
    }
}
```

从(1)->(2)->(4)->(3)的过程中执行到(4)如果进入到了continue,continue后的循环内语句本次不再执行一直到当前循环结构的}结束，执行下次循环，首先执行(3),然后变为(3)->(2)->(4)。

对于while来说：

```

int i=1;(1)
while(i<=100(2)) {
    if(i%10==3(3)) {
        continue;
    }
    i++;(4)
}

```

从(1)->(2)->(3)->(4)的过程中执行到(3)如果进入到了continue, continue后的循环内语句本次不再执行一直到当前循环结构的}结束,此时(4)就包含在其中, 那么循环的改变就是一个定值, 循环陷入到无限循环中。

我们也可以通过观察while和for的执行次数来判断这个说法:

```

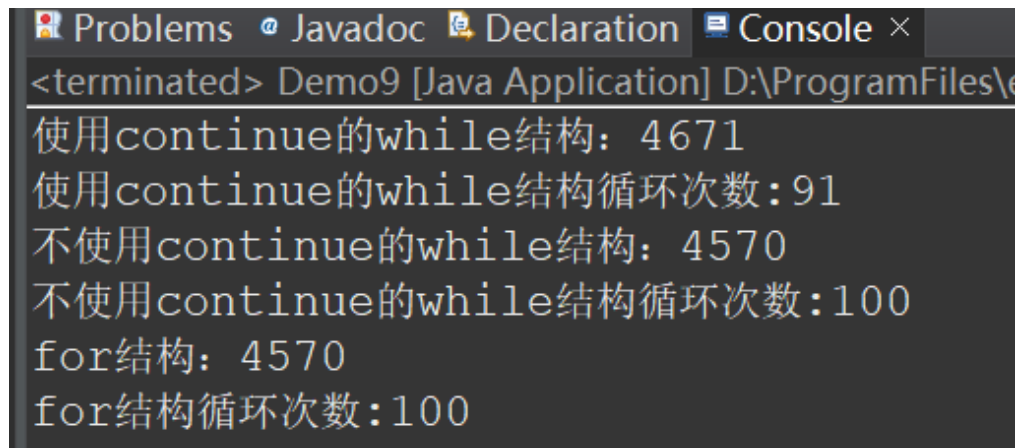
package java_language_basic;
/**
 * for和while结构的区别
 *
 * @author EMCred
 *
 */
public class Demo9 {
    public static void main(String[] args) {
        int num=0;
        int count = 0;
        int sum=0;
        while(num<=100) {
            num++;
            if(num%10==3) {
                continue;
            }
            sum=sum+num;
            count++;
        }
        System.out.println("使用continue的while结构: "+sum);//4671
        System.out.println("使用continue的while结构循环次数:"+count);//91
        num=1;
        count=0;
        sum=0;
        while(num<=100) {
            if(num%10!=3) {
                sum=sum+num;
            }
            num++;
            count++;
        }
        System.out.println("不使用continue的while结构: "+sum);//4570
        System.out.println("不使用continue的while结构循环次数:"+count);//100
        sum=0;
        count=0;
        for(int i=1;i<=100;i++,count++) {
            if(i%10==3) {
                continue;
            }
            sum=sum+i;
        }
    }
}

```

```

    }
    System.out.println("for结构: "+sum);//4570
    System.out.println("for结构循环次数:"+count);//100
}
}

```



```

<terminated> Demo9 [Java Application] D:\ProgramFiles\
使用continue的while结构: 4671
使用continue的while结构循环次数: 91
不使用continue的while结构: 4570
不使用continue的while结构循环次数: 100
for结构: 4570
for结构循环次数: 100

```

- for循环的循环要素2和要素4也就是循环的初始化和循环的改变在循环过程中是一致的，这一点和do-while一致。
- 三种循环结构使用最佳情况

对于三种循环的最佳使用情况的说明可以总结为如下的一个规则，当然在实际开发中还是以自己的想法为主，这个最佳使用情况只是一种推荐使用的方式。

根据业务的转换语义：

如果业务可以转换为“当.....”的时候优先选择使用while循环，如果业务可以转换为“直到.....”的时候优先选择do-while循环，如果业务可以固定次数循环的话优先选择for循环。

根据循环的四要素：

业务是否固定循环？

固定->优先使用for循环

不固定->业务的循环的改变和循环的初始化是否在循环过程中可以一致？

可以->优先使用do-while循环

不可以->优先使用while循环

案例：使用for循环改写猜数字小游戏

- 要求：参看之前的猜数字小游戏的要求，在do-while循环那章，只不过这次是使用for循环进行改写。
- 分析：我们知道for循环的要素2和要素4在循环的过程中也是一致的，所以可以参照着do-while循环进行改写，具体的实现可以参考do-while结构如何实现的，现在来具体的实现for循环，当然也可以自己想着写出来。下面是它的一种实现方式。
- 编码

```

package java_language_basic;

import java.util.Scanner;

/**
 * for循环实现猜数字小游戏
 * @author EMCred
 *
 */

```

```

public class Demo11 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int num = (int)(Math.random()*1000);
        int count=0;
        System.out.println("请输入猜测的数字0-1000: ");
        for(int
input=scanner.nextInt();input!=num;System.out.println("请输入猜测的数字0-
1000: "),input=scanner.nextInt(),count++) {
            if(input<num) {
                System.out.println("猜小了");
            }else{
                System.out.println("猜大了");
            }
        }
        System.out.println("恭喜你猜对了,一共猜了"+count+"次");
    }
}

```

- 测试

```

请输入猜测的数字0-1000:
880
猜大了
请输入猜测的数字0-1000:
875
猜大了
请输入猜测的数字0-1000:
872
猜大了
请输入猜测的数字0-1000:
871
恭喜你猜对了,一共猜了10次

```

- 最小化数值错误

- 什么是最小化数值错误

所谓的最小化数值错误就是使用浮点数进行逻辑运算的时候, 由于涉及浮点数的数值误差是不可避免的, 因为浮点数在计算机中本身就是近似表示的, 所以这就会导致使用浮点数进行逻辑运算时, 逻辑运算的结果可能会出现错误, 为了降低这种错误, 我们称之为最小化数值错误。

- 解决

1. 想要解决最小化数值的错误, 那么最好的办法就是不使用浮点数参与逻辑运算, 也就是说在循环结构不使用浮点数作为循环的初始化, 条件, 改变在选择结构中不使用浮点数作为条件。
2. 我们常常会使用到循环结构对浮点数进行操作, 比如我们想要获得从0.1+0.2+0.3.....1的值, 那么我们还是需要使用到循环结构进行操作, 此时我们应该从小到大添加数字, 并且使用整数作为循环的四要素。

理解: 比如从0.1.....1这个过程一共是10个数相加, 所以需要增加10次, 次数固定, 这个时候我们使用for循环结构, 循环的变量为count, 循环的初始化为int count=0, 循环的条件为count<10, 循环的改变为count++, 最后可以写成如下的形式。这里需要知道的是0.1+0.2+0.3+.....+1.0=5.5。

```
double num = 0.1;
double sum=0;
for(int i=0;i<10;i++) {
    sum+=num;//0+0.1+0.2+0.3+.....+1.0
    num+=0.1;
}
System.out.println("从小的加到大的: "+sum);//5.5
```

最后的结果为5.5，这是从小数值往大数值的方向去加的，它的结果相对来说错误概率更小一些。

接下来我们反过来进行加，也就是从1.0+0.9+0.8+.....+0.1来进行计算，它的结果是

```
sum=0;
num=1;
for(int i=0;i<10;i++) {
    sum+=num;//1.0+0.9+0.8+.....+0.1
    num-=0.1;
}
System.out.println("从大的加到小的: "+sum);//5.500000000000001
```

可以看到最后的结果相对于从小到大的加所得的结果误差更大一些。

- 无限循环

- 什么是无限循环

所谓的无限循环就是指循环陷入了一种无限的情况，也就是说循环不会出来了，没有结果，之前我们在解读continue关键字的时候举过一个while结构陷入无限循环的例子，所谓的无限循环的体现就是当时所说的程序跑了很长时间都不可能出结果。

无限循环也称为死循环。

- 为什么要使用无限循环

实际上在开发过程中我们可能会使用到无限循环，无限循环的最根本的原因就是当条件永远为true的时候循环将会永不停止执行，之所以使用无限循环是因为在某些时候，我们很难知道循环的最终条件但是又不得不使用循环的情况下才使用无限循环。

实际上我们在实际开发中所使用的无限循环并不是真的就无限循环下去了，它常常配合着内部的选择语句和break一起使用，索然无法预期循环的条件但是当循环到某个点的时候，我们就需要跳出无限循环继续执行接下来的语句，可能现在对其体会并不深刻，在之后的实际的项目开发中，如果碰到对应的代码可以返回来查看，从实践中体会。

能不用无限循环的情况下，最好是不用无限循环，无限循环常常用于比较复杂的问题处理，一方面它对计算机的资源占用消耗很大，另一方面在Linux操作系统使用无限循环可能会出现一些问题，具体的遇到之后再说。

- while, do-while和for无限循环

无限循环的本质就是循环的改变得到的布尔结果是个定值也就是true，实际工程中的无限循环并不是无限的，所以对于循环结构来说它们的无限循环结构常常有以下几种写法：

```
while结构:
while(true){
    if(条件){
        break;
    }
}
```

```

do-while结构:
do{
    if(条件){
        break;
    }
}while(true);

for结构:
for(;;){
    if(条件){
        break;
    }
}
for(;;true;){
    if(条件){
        break;
    }
}

```

- 嵌套的循环结构

所谓的嵌套循环就是循环之中嵌套循环，比如我们之前所写过的案例：显示1-1000之间的素数，再其中我们使用外层的for循环选定当前的数值，然后当前的数值进入到内层的循环从2开始逐个相除取余直到找到第一个取整的数。对于嵌套循环可以总结为如下：

1. 嵌套循环的最大特点就是外层走一遍，里层走所有遍。
2. 一般嵌套循环再实际开发中最多嵌套三层循环，基本上嵌套循环也就是嵌套两层循环，如果不得不多层嵌套的话在一定程度上会造成CPU负载加重对于Java来说甚至会导致JVM崩溃对于我们的内存来说也会造成内存的负荷。
3. 对于双层循环来说，它可以构成一种二维表的关系，外层是行，内层是列。

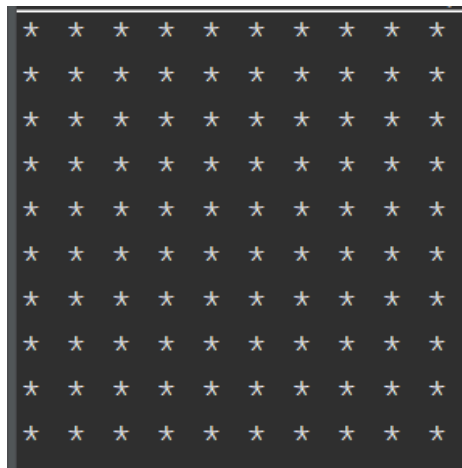
理解，现在我想要使用程序打印出一个10X10的正方形。

```

package java_language_basic;

/**
 * 嵌套循环
 * @author EMCred
 */
public class Demo13 {
    public static void main(String[] args) {
        int i=0;
        while(i<10) { //外层控制行，针对每个i
            int j=0;
            while(j<10) { //内层控制列，内层取遍0, 1, 2.....9
                System.out.print("* "); //输出形状的组成元素
                j++;
            }
            i++;
            System.out.println();
        }
    }
}

```

案例：输出9X9乘法表

要求：使用程序输出一张9X9乘法表

分析：输出9X9乘法表，核心思想是需要使用到两个for循环进行嵌套，外层for循环输出行，内层for循环输出列。针对于外层的每一个i，内层取遍每个j，外层走一遍，内层走所有遍，争对每一行的i输出对应的每一列的i*j。

编码：

```
package java_language_basic;
/**
 * 输出9X9乘法表
 * @author EMCred
 *
 */
public class Demo8 {
    public static void main(String[] args) {
        /*
        *表头
        */
        for(int i=1;i<=9;i++) {
            System.out.print("-----");
        }
        System.out.println();
        System.out.println("          乘法表");
        for(int i=1;i<=9;i++) {
            System.out.print("-----");
        }
        System.out.println();
        /*
        *外层的装饰
        */
        System.out.print(" ");
        for(int i=1;i<=9;i++) {
            System.out.print("    "+i);
        }
        System.out.println();
        System.out.print(" ");
        for(int i=1;i<=9;i++) {
            System.out.print("----");
        }
        System.out.println();
        /*
```

```

*具体的乘法表
*/
for(int i =1;i<=9;i++) {
    System.out.print(i+"|");
    for(int j=1;j<=9;j++) {
        System.out.printf("%4d",i*j);
    }
    System.out.println();
}
}
}

```

测试:

乘法表									
	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81