# JavaSE阶段

## 语言基础

### 变量的概念

计算机内存中的一块存储空间，是存储数据的基本单元。

变量的组成：数据类型、变量名、值

### 变量的定义流程

1. 声明：

数据类型 变量名； //开辟变量空间

1. 赋值：

变量名 = 值； //将值赋给变量

注意：Java 是强类型语言， 变量的类型必须与数据的类型一致

### 变量的定义方式

1. 先声明，在赋值（常用）：

数据类型 变量名；

变量名 = 值；

1. 声明并赋值（常用）：

数据类型 变量名 = 值;

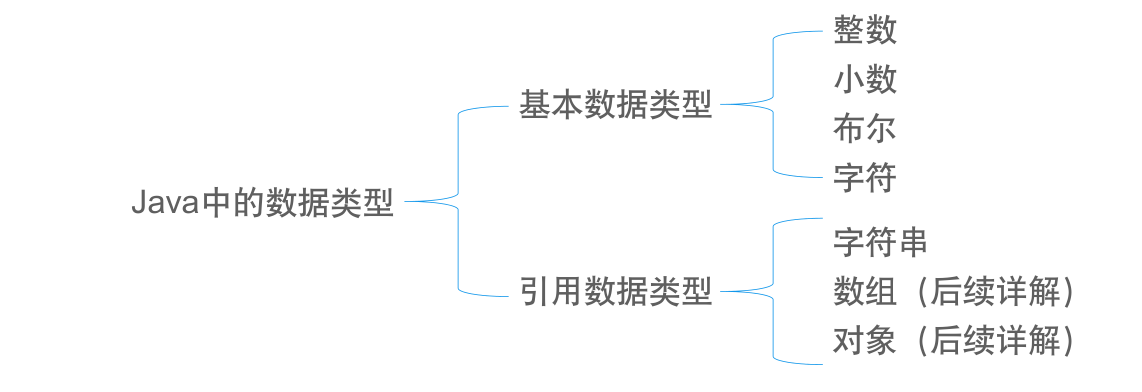
1. 多个同类型变量的声明与赋值（了解）：

数据类型 变量1, 变量2 , 变量 3 = 值 3 , 变量 4 , 变量 5 = 值; 5;

### 数据类型

Java 中的变量具有严格的数据类型区分。（强类型语言）

在 Java 语言中，任何一个值，都有其对应类型的变量。



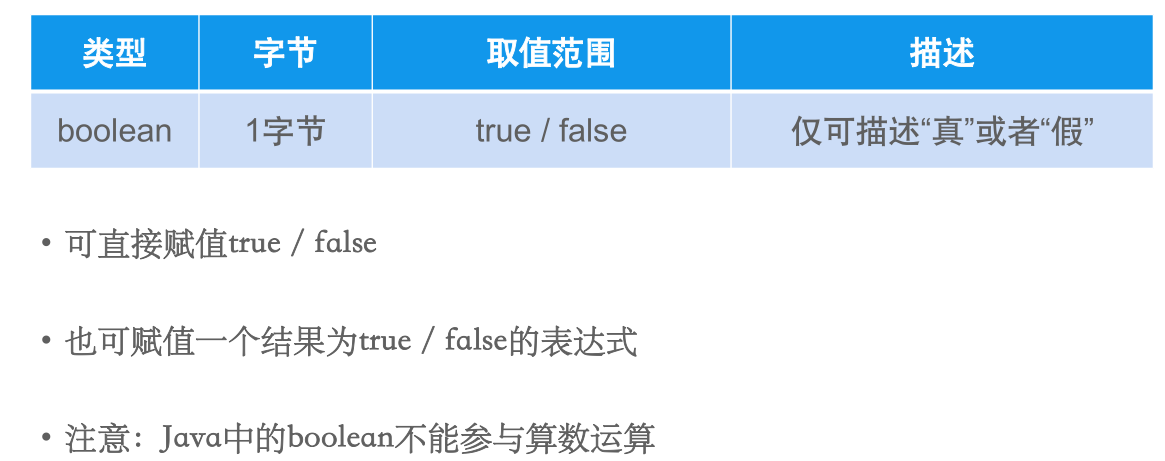
### 基本数据类型（整数）



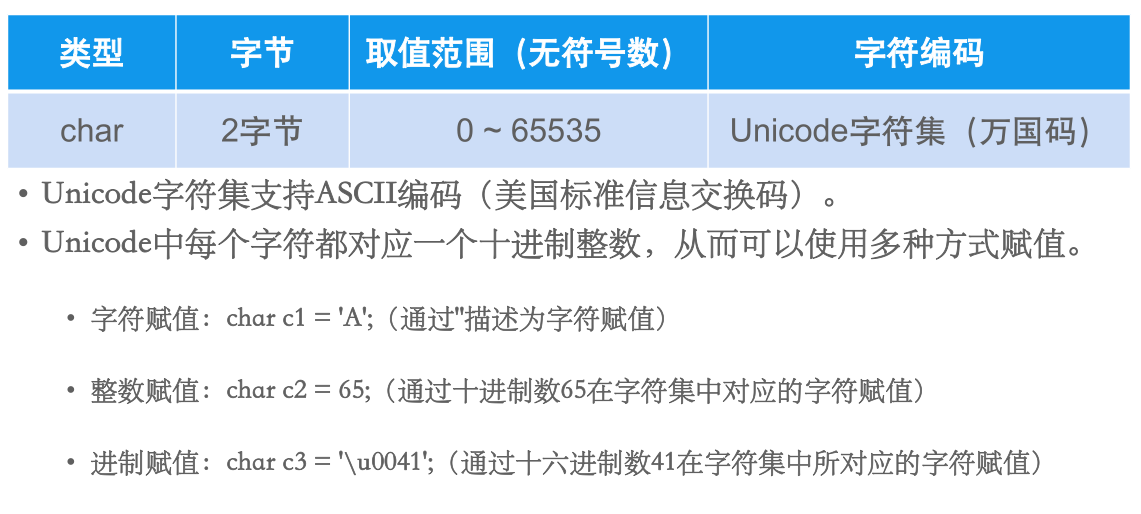
### 基本数据类型（小数/浮点数）



### 基本数据类型（布尔）



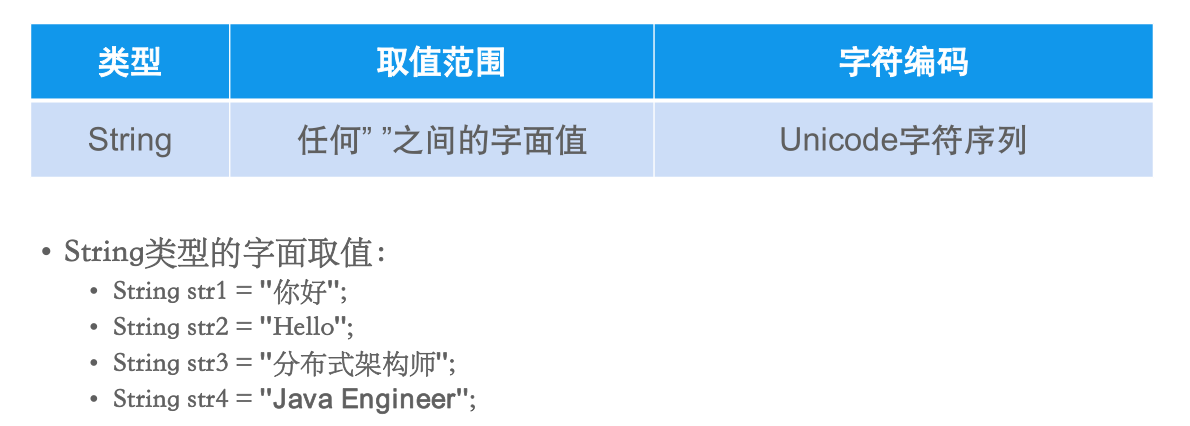
### 基本数据类型（字符）



### 转义字符



### 10.引用数据类型（字符串）

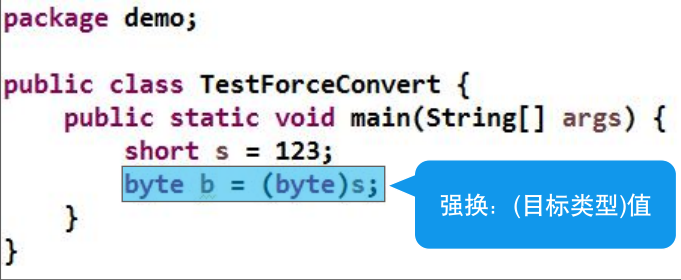


### 11类型转换

1. 自动类型转换
2. 两种类型相互兼容
3. 目标类型大于源类型



1. 强制类型转换
2. 两种类型相互兼容
3. 目标类型小于源类型



强制类型转换规则：

1. 整数长度足够，数据完整

例： int i = 100; ; byte b = (byte)i; //b = 100

1. 整数长度不够，数据截断

例： int i = 10000; ; byte b = (byte)i; //b = 16 （符号位变化，可能变为负数）

1. 小数强转整数，数据截断

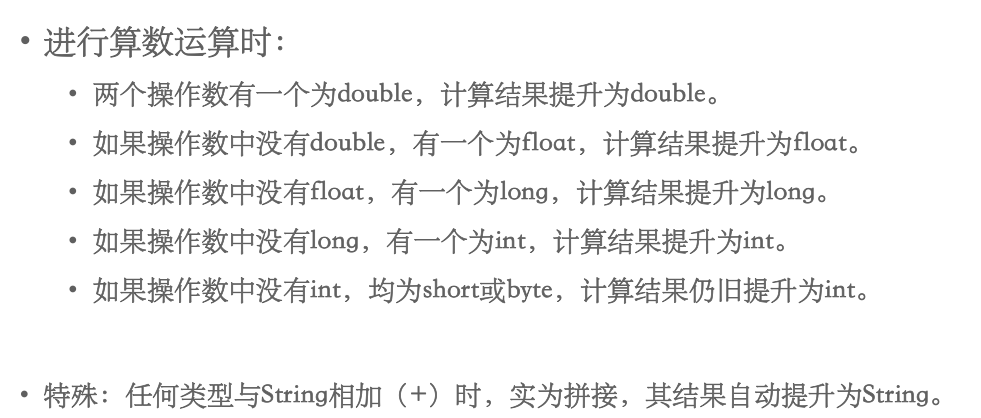
例： double d = 2.5; int i = (int)d; //i = 2 （小数位舍掉）

1. 字符整数互转，数据完整

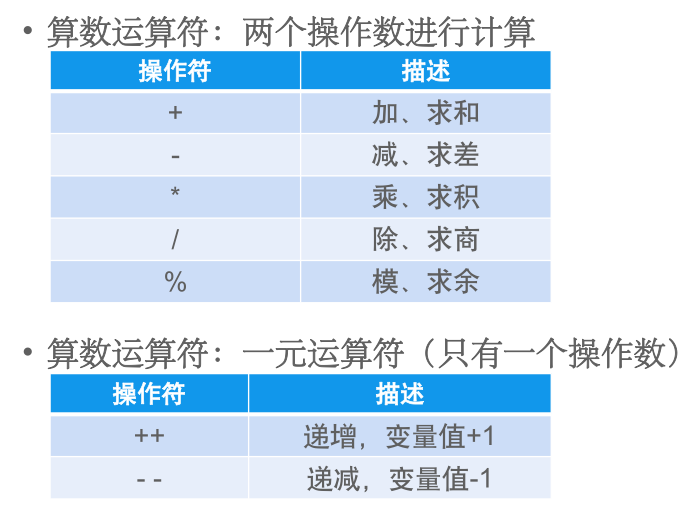
例： char c = 65; int i = c ; //i = 65

1. boolean的取值为true/false ，不可与其他类型转换

### 12.自动类型提升

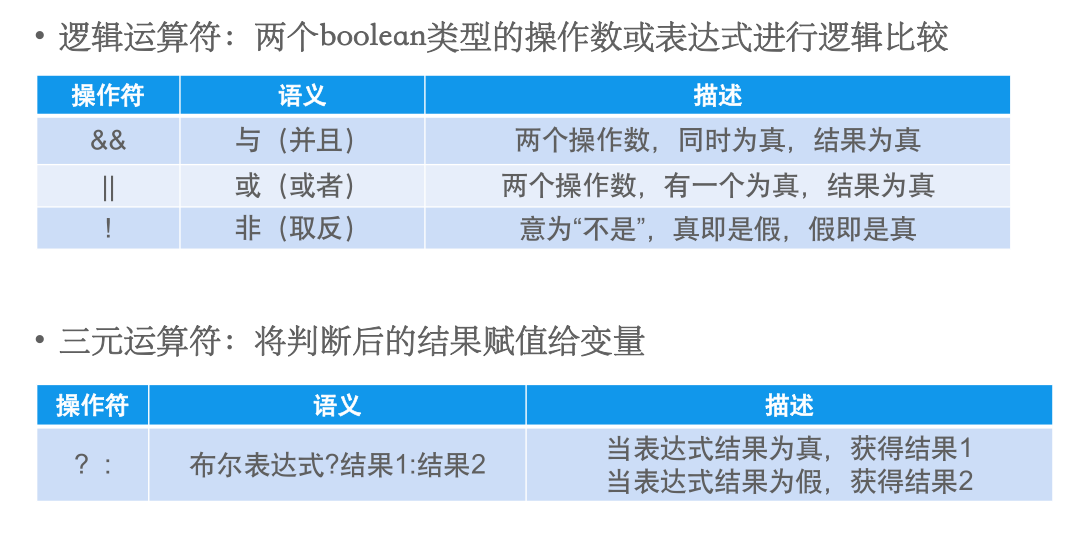


### 13.运算符









### 14.控制台输入

程序运行中，可在控制台（终端）手动录入数据，再让程序继续运行。

导包语法： import 包名. 类名; // 将外部 class 文件功能引入到自身文件中

使用顺序：

1. 导入java.util.Scanner;
2. 声明Scanner类型的变量

Scanner input = new Scanner(System.in);

3)使用Scanner类中对应的函数（区分类型）：

1. int num = input.nextInt(); //接收整数
2. double num = input.nextDouble(); //接收小数
3. String num = input.next(); //接收字符串
4. Char num = input.next.charAt(0); //接收首个字符

注意：如果输入了不匹配的数据，则会产生java.util.InputMismatchException

## 选择结构与分支结构

### 1.基本if选择结构

语法：

if（布尔表达式）{

//代码块

}

后续代码……

执行流程：

1. 对布尔表达式进行判断
2. 结果为true，则先执行代码块，再执行后续代码
3. 结果为false ，则跳过代码块，直接执行后续代码

### 2.if else选择结构

语法：

if（布尔表达式）{

//代码块1

}else{

//代码块2

}

后续代码…….

执行流程：

1. 对布尔表达式进行判断
2. 结果为true ，则先执行代码块1，再退出整个结构，执行后续代码。
3. 结果为false ， 则先执行代码块2，再退出整个结构，执行后续代码。

### 3.多重if选择结构

语法：

if(布尔表达式1){

//代码块1

}else if(布尔表达式2){

//代码块2

} else if(布尔表达式3){

//代码块3

}else{

//代码块4

}

执行流程：

1. 表达式1为 true ，则执行代码块1，再退出整个结构
2. 表达式2为 true ，则执行代码块2，再退出整个结构
3. 表达式3为 true ，则执行代码块3，再退出整个结构
4. 以上均为false，则执行代码块4，再退出整个结构。

注意：相互排斥，有一个为 true ，其他均不再执行，适用于区间判断。

### 4.嵌套if选择结构

语法：

if(外层表达式){

if(内层表达式){

//内层代码块1

}else{

//内层代码块2

}

}else{

//外层代码块

}

执行流程：当外层条件满足时，再判断内层条件。

注意：一个选择结构中，可嵌套另一个选择结构。嵌套格式正确的情况下，支持任意组合 。

### 5.swich分支结构

语法：

swich(变量|表达式){

case值1：

逻辑代码1;

break;

case值2：

逻辑代码2;

break;

…………….…..

case值n：

逻辑代码n;

break;

default:

未满足时的逻辑代码;

Break;

}

可判断类型：byte、short、int、char、String（JDK7+）

执行流程：

1. 如果变量中的值等于值1，则执行逻辑代码1
2. 如果变量中的值等于值2，则执行逻辑代码2
3. 如果变量中的值等于值n，则执行逻辑代码n
4. 如果变量中的值没有匹配的case值时，执行default中的逻辑代码。

注意：所有 case 的取值不可相同

### 6.局部变量

概念：声明在函数内部的变量，必须先赋值再使用。

作用范围：定义行开始到所在的代码块结束

注意：多个变量，在重合的作用范围内，不可出现重名（命名冲突）

## 多线程

### 1.什么是进程？

程序是静止的，只有真正运行时的程序，才被称为进程。

单核CPU在任何时间点上，只能运行一个进程；宏观并行、微观串行。

### 2.什么是线程？

线程，又称轻量级进程（Light Weight Process）。程序中的一个顺序控制流程，同时也是CPU的基本调度单位。

进程由多个线程组成，彼此间完成不同的工作，交替执行，称为多线程。

Java虚拟机是一个进程，当中默认包含主线程（Main），可通过代码创建多个独立线程，与Main并发执行。

### 3.线程的组成

1. CPU 时间片：操作系统（OS）会为每个线程分配执行时间。
2. 运行数据：

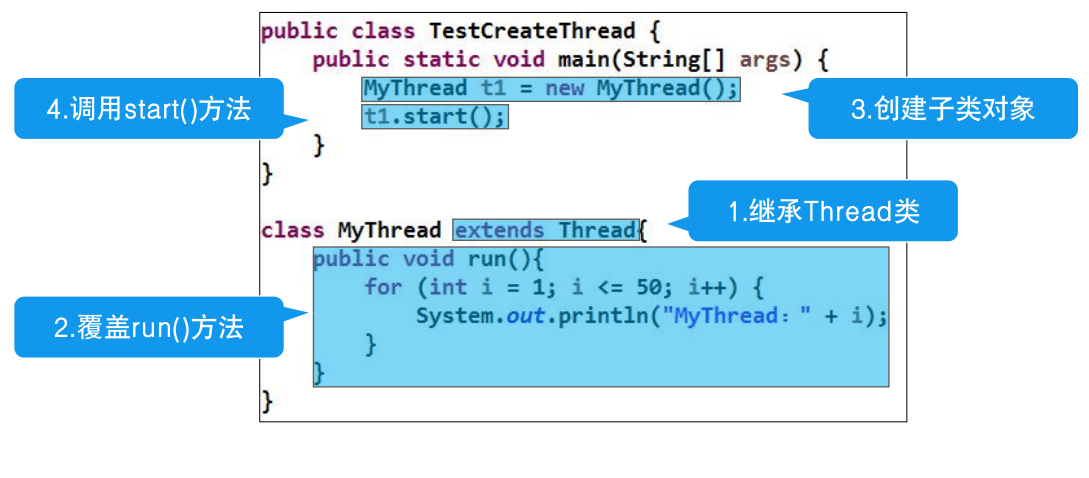
堆空间 ：存储线程需使用的对象，多个线程可以共享堆中的对象。（堆空间共享）

栈空间 ：存储线程需使用的局部变量，每个线程都拥有独立的栈。（栈空间独立）

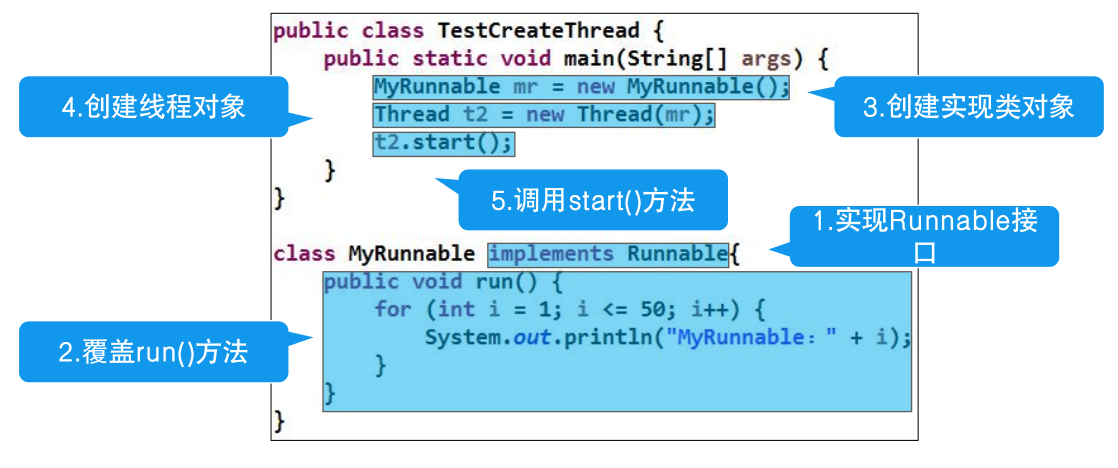
1. 线程的逻辑代码

### 4.创建线程的方式

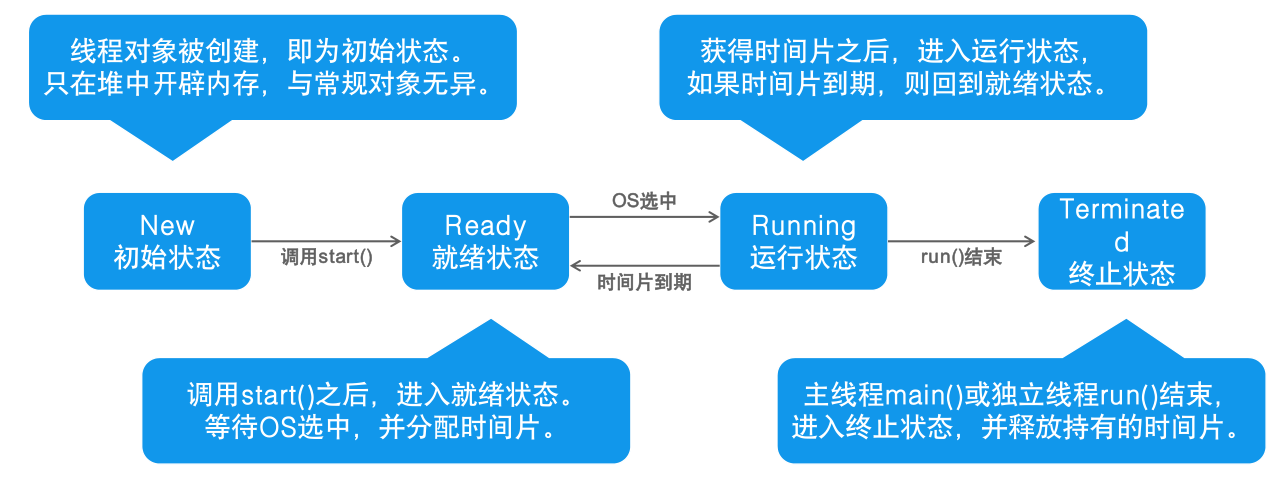
第一种方式：继承Thread类



第二种方式：实现Runnable接口



### 5线程的状态（基本）



### 6.常见方法

1. 休眠

public static void sleep(long millis);

当前线程主动休眠millis毫秒。

1. 放弃

public sytatic void yield();

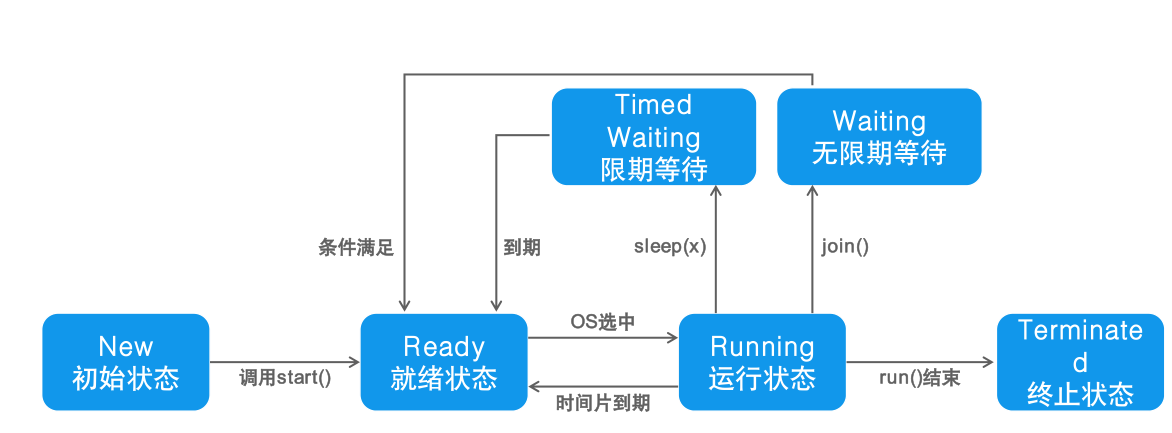
当前线程主动放弃时间片，回到就绪状态，竞争下一次时间片。

1. 结合（插队）

public final void join();

允许其他线程加入到当前线程中。

### 7.线程的状态（等待）



### 8.线程安全问题

当多线程并发访问临界资源时，如果破坏原子操作，可能会造成数据不一致。

临界资源：共享资源（同一对象），一次仅允许一个线程使用，才可保证其正确性。

原子操作：不可分割的多步操作，被视作一个整体，其顺序和步骤不可打乱或缺省。

1. 保证线程安全方法一：

同步代码块：

Synchronized(临界资源对象){ //对临界资源对象加锁

//代码（原子操作）

}

注意：

1. 每个对象都有一个互斥锁标记，用来分配给线程的 。
2. 只有拥有对象互斥锁标记的线程，才能进入对该对象加锁的同步代码块。
3. 线程退出同步代码块时，会释放相应的互斥锁标记。
4. 保证线程安全方法二：

同步方法：

Synchronized 返回值类型 方法名（形参列表）{//对当前对象（this）加锁

//代码（原子操作）

}

注意：

1. 只有拥有对象互斥锁标记的线程，才能进入该对象加锁的同步方法中。
2. 线程退出同步方法时，会释放相应的互斥锁标记。

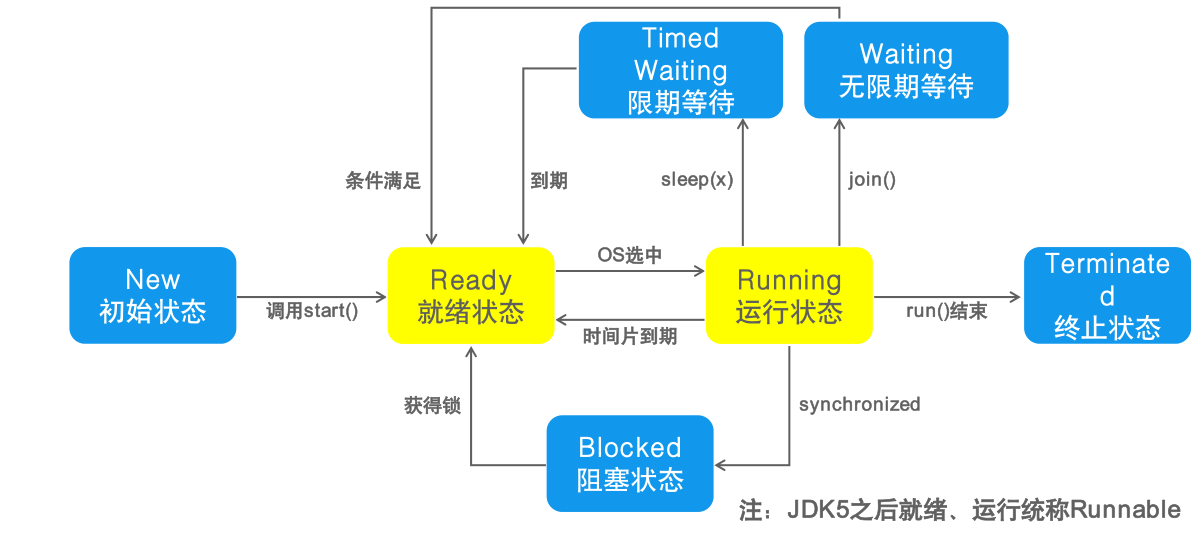
### 9.同步规则

1）只有在调用包含同步代码块的方法，或者同步方法时，才需要对象的锁标记。

2）如调用不包含同步代码块的方法，或普通方法时，则不需要锁标记，可直接调用。

已知JDK中线程安全的类：StringBuffer、Vector、Hashtable，这几种类中的公开方法均为synchronized修饰同步的方法。

### 10.线程的状态（阻塞）



### 11.等待通信

1. 等待

public final void wait();

public final void wait(long timeout);

必须在对obj（对象）加锁的同步代码块中，在一个线程中，调用obj.wait()时，此线程会释放其拥有的所有锁标记。同时此线程阻塞在o的等待队列中。释放锁，进入等待队列。

1. 通知

public final void notify();

public void final notifyAll();

必须在对 obj 加锁的同步代码块中。从 obj 的 Waiting 中释放一个或全部线程。对自身没有任何影响。

## 高级多线程

### 1.线程池概念

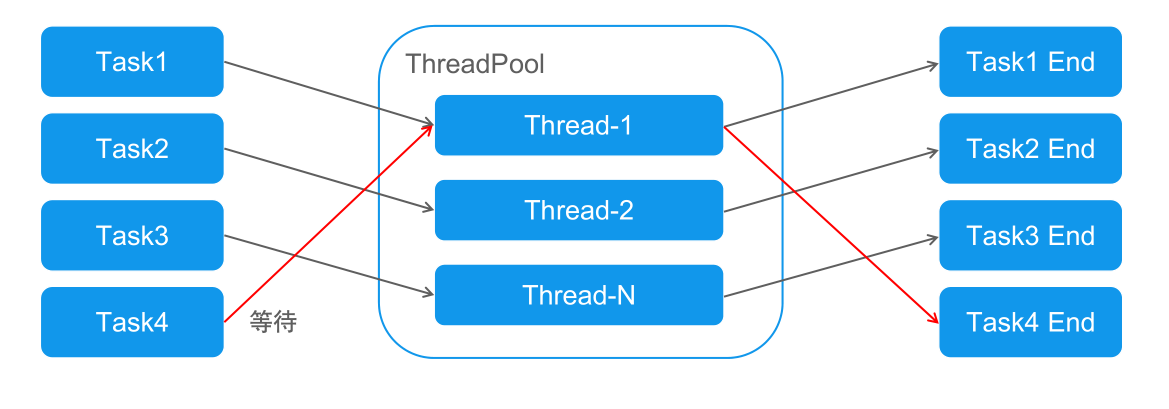
1）线程容器，可设定线程分配的数量上限。

2）将预先创建 的 线程对象存入池中，并重用线程池中的线程对象。

3）避免频繁的创建和销毁。

### 2.线程池原理

将任务提交给线程池，由线程池分配线程、运行任务，并在当前任务结束后复用线程。



### 3.获取线程池

常用的线程池接口和类(所在包 java.util.concurrent)：

Executor ：线程池的顶级接口。

ExecutorService ：线程池接口，可通过 submit(Runnable task) 提交任务代码。

Executors工厂类：通过此类可以获得一个线程池

通过 newFixedThreadPool(int nThreads) 获取固定数量的线程池。参数：指定线程池中线程的数量。

通过 newCachedThreadPool()获得动态数量的线程池，如不够则创建新的，没有上限。

### 4.Callable接口

public interface Callable<V>{

public V call() throws Exception;

}

JDK5 加入，与 Runnable 接口类似，实现之后代表一个线程任务。

Callable 具有泛型返回值、可以声明异常。

### 5.Future接口

概念：异步接收 ExecutorService.submit() 所返回的状态结果，当中包含了call()的返回值。

方法： V get() 以阻塞形式等待Future中的异步处理结果（call() 的返回值）。

### 6.线程的同步与异步

同步：形容一次方法调用，同步一旦开始，调用者必须等待该方法返回，才能继续。

注意：单条执行路径

异步：形容一次方法的调用，异步一旦开始，像是一次消息传递，调用者告知之后立刻返回。二次竞争时间片，并发执行。

注意;多条执行路径。

### 7.Lock接口

JDK5加入，与synchronized相比，显示定义，结构更为灵活；

提供更多使用性方法，功能更强大、性能更优越。

常用方法：

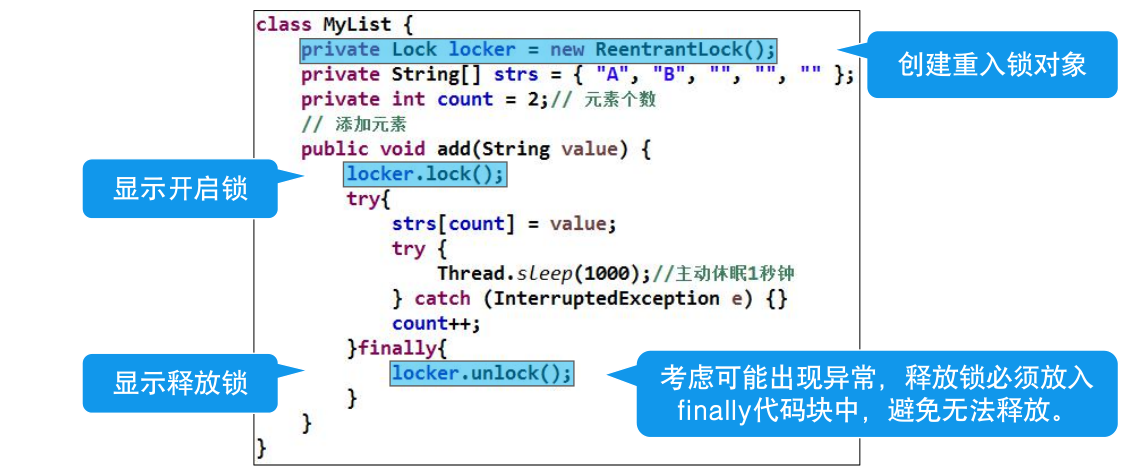
void lock() // 获取锁，如锁被占用，则等待。

boolean tryLock() // 尝试获取锁（成功返回 true， 失败返回false，不阻塞）

void unlock() //释放锁

### 8.重入锁

Reentrantlock：Lock接口的实现类，与synchronized一样有互斥锁功能。



### 9.读写锁

ReentrantReadWriteLock:

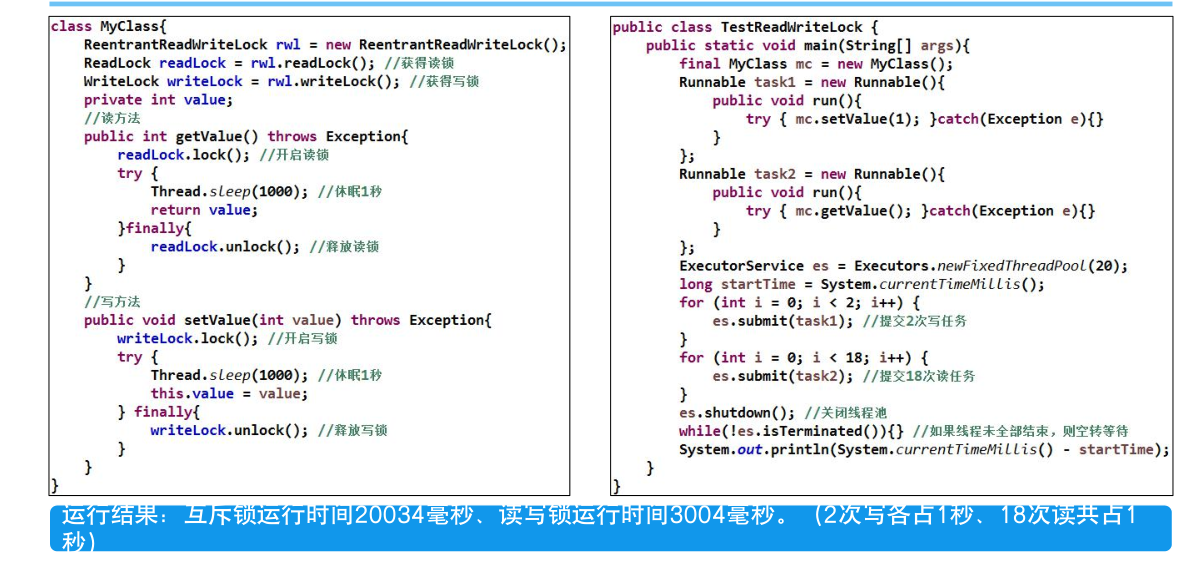
一种支持一写多读的同步锁，读写分离，可分别分配，读锁，写锁。

支持多次分配读锁，使多个读操作可以并发执行。

互斥规则：

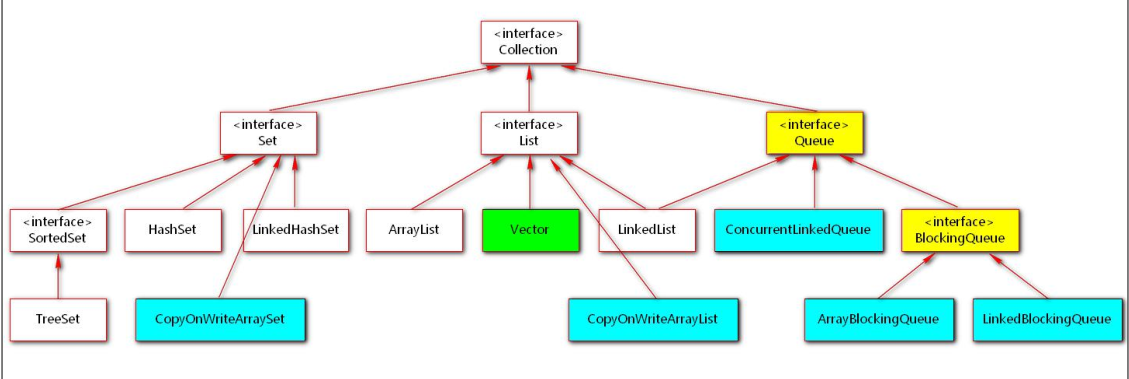
1. 写——写：互斥、阻塞；
2. 读——写：互斥，读阻塞写，写阻塞读；
3. 读——读：不互斥、不阻塞。

在读操作远远高于写操作的环境中，可在保障线程安全的情况下，提高运行效率。



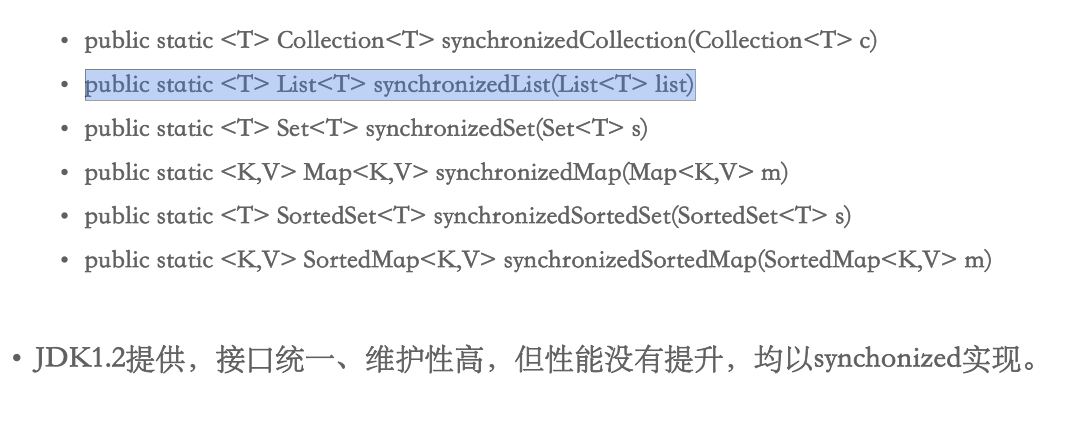
### 10.线程安全的集合

Collection体系集合下，除Vector以外的线程安全集合。



### 11.Collections中的工具方法

Collections 工具类中提供了多个可以获得线程安全集合的方法。



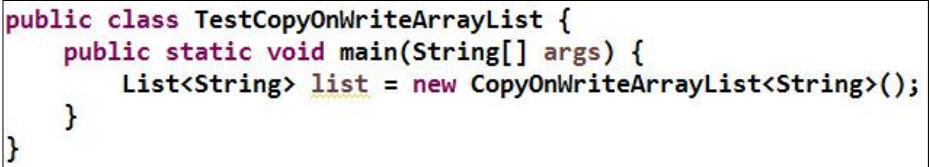
### 12.CopyOnWriteArrayList

线程安全的ArrayList，加强版读写分离

写有锁，读无锁，读写之间不阻塞，优于读写锁

写入时，先copy一个容器副本，再添加新元素，最后替换引用

使用方式与ArrayList无异

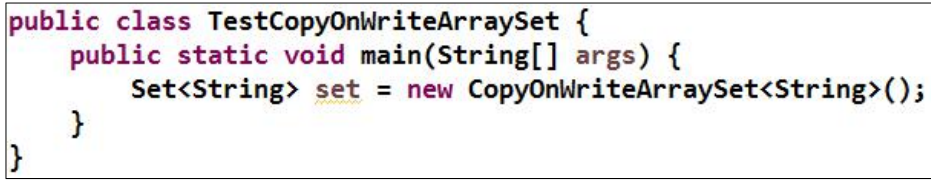


### 13.CopyOnWriteArraySet

线程安全的set，底层使用CopyOnWriteArrayList实现

唯一不同在于，使用addIfAbsent()添加元素，会遍历数组

如存在元素，则不添加（扔掉副本）



### 14.ConcurrentHashMap

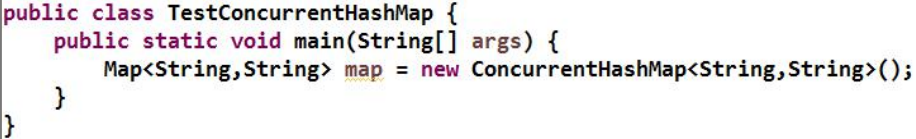
初始容量默认为16段（Segment）,使用分段锁设计

不对整个Map加锁，而是为每个Segment加锁

当多个对象存入同一个Segment时，才需要互斥

最理想状态为16个对象分别存入16个Segment，并行数量16

使用方式与HashMap无异



### 15.Queue接口（队列）

Collection的子接口，表示队列FIFO（First In First Out）

常用方法：

抛出异常：

boolean add(E e) // 顺序添加一个元素（到达上限后，再添加则会抛出异常）

E remove() // 获得第一个元素并移除（如果队列没有元素时，则抛异常）

E element() // 获得第一个元素但不移除（如果队列没有元素时，则抛异常）

返回特殊值：推荐使用

boolean offer(E e) // 顺序添加一个元素 （到达上限后，再添加则会返回false）

E poll() // 获得第一个元素并移除 （如果队列没有元素时，则返回null）

E peek() // 获得第一个元素但不移除 （如果队列没有元素时，则返回null）

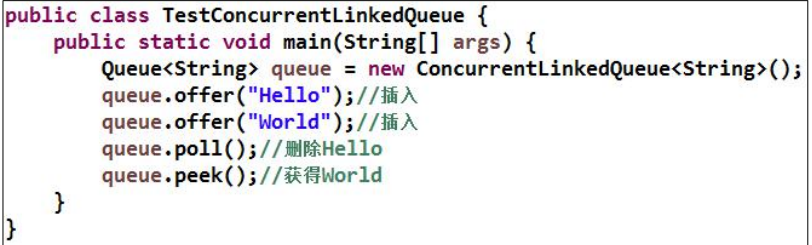
### 16.ConcurrentLinkedQueue

线程安全、可高效读写的队列，高并发下性能最好的队列。

无锁、CAS 比较交换算法，修改的方法包含三个核心参数（V,E,N）

V ：要更新的变量、E ：预期值、 N ：新值。

只有当 V==E 时， V=N ；否则表示已被更新过，则取消当前操作。



### 17.BlockingQueue接口（阻塞队列）

Queue的子接口，阻塞的队列，增加了两个线程状态为无限等待的方法

方法：

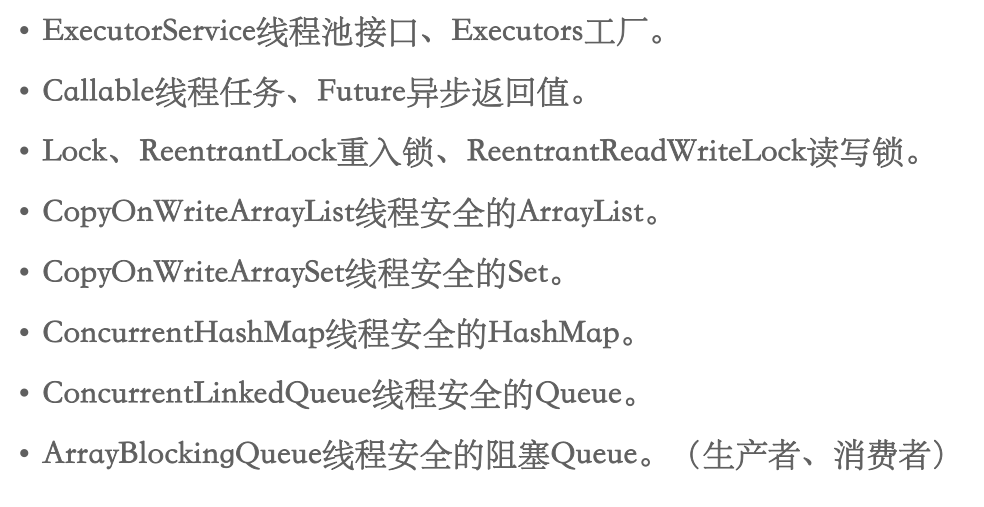
void put(E e) // 将指定元素插入此队列中，如果没有可用空间，则等待。

E take() // 获取并移除此队列头部元素，如果没有可用元素，则等待。

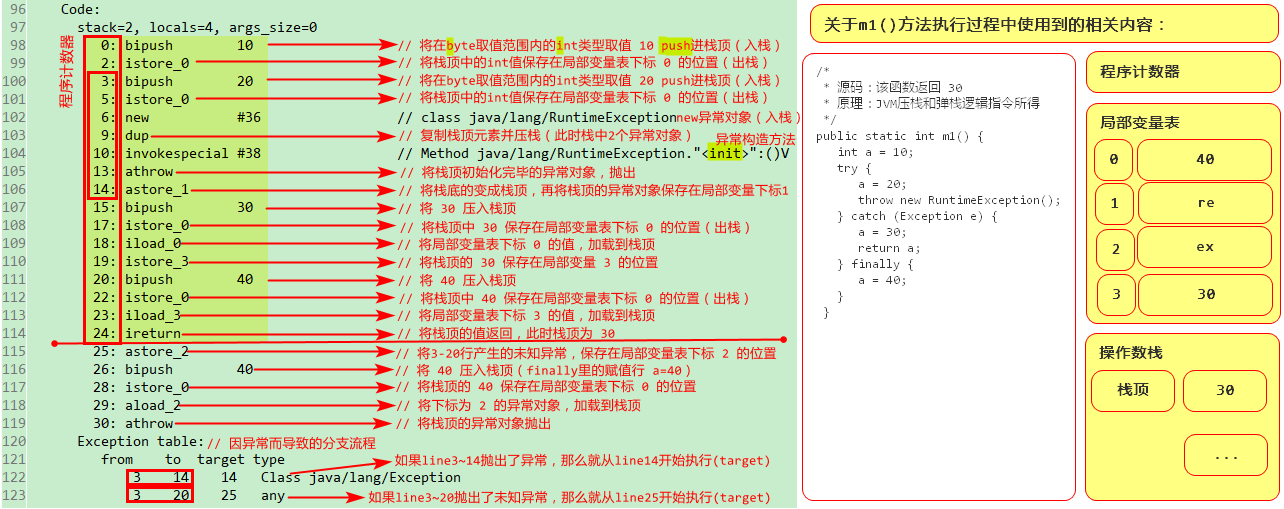
可用于解决生产生、消费者问题。



### 18.高级线程总结



### 19.反编译、栈



## IO框架

### 什么是流？

内存与存储设备之间传输数据的通道。（水借助管道传输，数据借助流传输）

### 流的分类

1. 按方向（重点）
2. 输入流：将<存储设备>中的内容读入到<内存>中。
3. 输出流：将<内存>中的内容写入到<存储设备>中。

2）按单位

1. 字节流：以字节为单位，可以读写所有数据。
2. 字符流：以字符为单位，只能读写文本数据。

3）按功能

1. 节点流：具有实际传输数据的读写功能。
2. 过滤流：在节点流的基础上增强功能。

### 字节流

字节流的父类（抽象类）：

1. InputStream:字节输入流
2. public int read(){}
3. public int read(byte[] b){}
4. public int read(byte[] b,int off,int len){}
5. OutputStream:字节输出流
6. public void write(int n){}
7. public void write(byte[] b){}
8. public void write(byte[] b,int off, int len){}

### 字节节点流

1. FileOutputStream:

public void write(byte[] b) //一次写多个字节，将b数组中的所有字节，写入输出流

1. FileInputStream:

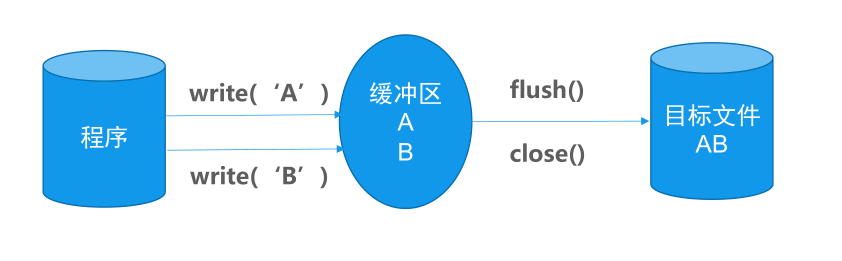
public int read(byte b) //从流中读取多个字节，将读到内容存到b数组，返回实际读到的字节数；如果文件达到尾部，则返回-1.

### 字节过滤流

1）缓冲流：BufferedOutputStream/BufferedInputStream

提高IO效率，减少访问磁盘的次数

数据存储到缓冲区中，flush是将缓冲区的内容写入到文件中，也可以直接close



2) 对象流：ObjectOutputStream/ObjectInputStream

增强了缓冲区功能

增强了读写8种基本数据类型和字符串功能

增强了读写对象的功能：

1. readObject() //从流中读取一个对象
2. writeObject(Object obj) //向流中写入一个对象

注意：使用流传输对象的过程成为序列化、反序列化

### 对象序列化

1. 必须实现Serializable接口
2. 必须保证其所有属性均可序列化
3. Transient修饰为临时属性，不参与序列化
4. 读取到文件尾部的标志：java.io.EOFException

### 字符编码

1）ISO-8859-1 收录除ASCII外，还包括西欧、希腊语、泰语、阿拉伯语、希伯来语对应的文字符号

2）UTF-8 针对Unicode的可变长度字符编码

3）GB2312 简体中文

4）GBK 简体中文、扩充

5）BIG5 台湾，繁体中文

注意：当编码方式和解码方式不一致时，会出现乱码

### 字符流

字符流的父类（抽象类）：

1. Reader:字符输入流：
2. public int read(){}
3. public int read(char[] c){}
4. public int read(char[] b,int off,int len){}
5. Writer:字符输出流：
6. public void write(int n){}
7. public void write(String str){}
8. public void wrute(char[] c){}

### 字符节点流

1. FileWriter:

public void write(String str) //一次写多个字符，将数组中所有字符，写入输出流

1. FileReader：

public int read(char[] c) //从流中读取多个字符，将读到内容存入c数组，返回实际读到的字符数；如果达到文件的尾部，则返回-1。

### 字符过滤流

1. SufferedWriter/PrintWriter(封装了print()、println()方法，支持写入后换行)
2. BufferedReader

支持输入换行符

可一次写一行、读一行

### 桥转换流

1. InputStreamReader
2. OutputStreamWriter

作用：

1. 可将字节流转换为字符流
2. 可设置字符的编码方式

使用方式：

1. 创建节点流
2. 【创建过滤流，设置字符编码集】
3. 封装过滤流
4. 读写数据
5. 关闭流

### File类

概念：代表物理盘符中的一个文件或者文件夹。

方法：

1）createNewFile() //创建一个新文件

2）Mkdir() //创建一个新文件夹

3）Delete() //删除文件或空目录

4）Exists() //判断File对象所代表的对象是否存在

5）getAbsolutePath() //获取文件的绝对路径

6）getName() //获得名字 文件名.后缀

7）getParent() //指定文件的上一级目录

8）isDirectory() //是否是文件夹

9）isFile() //是否是文件

10）length() //获得文件的长度

11）listFiles() //列出文件夹中的所有内容

12）renameTo() //修改文件名为

13）canExecute() //所有可以打开的文件或文件夹，都是可执行的

14）canWrite() //能不能修改文件

15）canRead() //能不能执行文件

16）getPath() //得到相对路径

17）getFreeSpace() //获取硬盘的空闲空间（字节）

18）getTotalSpace() //获取硬盘的总空间（字节）

19）isHidden() //判断文件是否为隐藏

20）(System.currentTimeMillis() - file.lastModified()) / 1000 / 60 ////获取文件最后一次修改的时间

### 13.FileFilter接口

public interface FileFilter{

boolean accept(File pathname)；

}

当调用File类中的listFiles()方法时，支持传入FileFilter接口接口实现类，对获取文件进行过滤，只有满足条件的文件的才可出现在listFiles()的返回值中。

### IO框架总结



## 网络编程

### 什么是网络？

由点和线构成，表示诸多对象间的相互联系。

### 什么是计算机网络?

为实现资源共享和信息传递，通过通信线路连接起来的若干主机（Host）。

互联网： （Internet）点与点相连

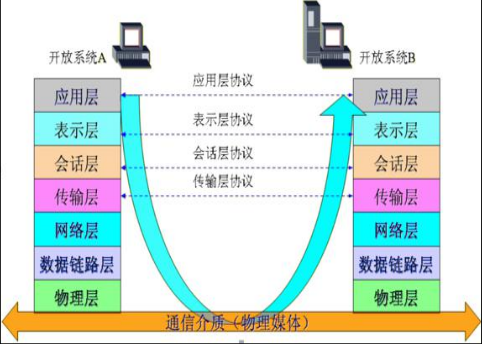
万维网：（WWW —— World Wide Web ）端与端相连

物联网：（IoT —— Internet of things ） 物与物相连

网络编程：让计算机与计算机之间建立连接、进行通信

### 网络模型

OSI （Oper System Interconnection） 开放式系统互联



第七层：应用层负责文件访问和管理、可靠运输服务、远程操作服务。（HTTP、FTP、SMTP）

第六层：表示层负责定义转换数据格式及加密，允许选择以二进制或ASCII格式传输。

第五层：会话层负责使应用建立和维持会话，使通信在失效时继续恢复通信.（断点续传）

第四层：传输层负责是否选择差错恢复协议，数据流重用、错误顺序重排。（TCP、UDP）

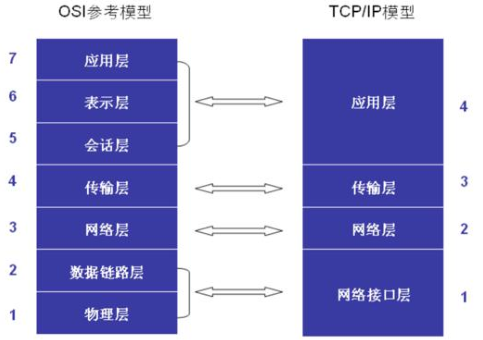
第三层：网络层负责定义了能够标识所有网络节点的逻辑地址。（IP地址）

第二层：链路层在物理层上，通过规程或协议（差错控制）来控制传输数据的正确性。（MAC）

第一层：物理层为设备之间的数据通信提供传输信号和物理介质。（双绞线、光导纤维）

1. TCP/IP模型

一组用于实现网络互连的通信协议，将协议分成四个层次。



第四层：应用层负责传送各种最终形态的数据，是直接与用户打交道的层，典型协议是HTTP、FTP等。

第三层：传输层负责传送文本数据，主要协议是TCP、UDP协议。

第二层：网络层负责分配地址和传送二进制数据，主要协议是IP协议。

第一层：接口层负责建立电路连接，是整个网络的物理基础，典型的协议包括以太网、ADSL等。

### TCP/UDP协议

TCP协议：Transmission Control Protocol 传输控制协议

是一种面向连接的、可靠的、基于字节流的传输层通信协议。数据大小无限制。建立连接的过程需要三次握手，断开连接的过程需要四次挥手。

UDP协议：User Datagram Protocol 用户数据报协议

是一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务，每个包的大小为64KB。

### IP

IP协议：Internet Protocol Address 互联网协议地址/网际协议地址

分配给互联网设备的数字标签（唯一标识）。

IP地址分为两种：

1. IPV4：4字节32位整数并分成4段8位的二进制数，每8位之间用圆点隔开，每8位整数可以转换为一个0～255的十进制整数。

格式：D.D.D.D 例如：255.255.255.255

1. IPV6：16字节128位整数，并分成8段十六进制数，每16位之间用圆点隔开，每16位整数可以转换为一个0～65535的十进制数。

格式：X.X.X.X.X.X.X.X 例如：FFFF.FFFF.FFFF.FFFF. FFFF.FFFF.FFFF.FFFF

### IPV4的应用分类

A类：政府机构，1.0.0.1～ 126.255.255.254

B类：中型企业，128.0.0.1 ～ 191.255.255.254

C类：个人用户，192.0.0.1 ～ 223.255.255.254

D类：用于组播，224.0.0.1 ～239.255.255.254

E类：用于实验，240.0.0.1 ～255.255.255.254

回环地址：127.0.0.1，指本机，一般用于测试使用。

查看IP命令：ipconfig（进入cmd使用）

测试IP命令： ping D.D.D.D

### Port（端口）

端口号：在通信实体上进行网络通讯的程序的唯一标识。

端口分类：

1. 公认端口：0～1023
2. 注册端口：1024～49151
3. 动态或私有端口：49152～65535

常用端口：

1. MySql：3306
2. Oracle：1521
3. Tomcat：8080
4. SMTP：25
5. Web服务器：80
6. FTP服务器：21

### InetAddress类

概念：表示互联网协议（IP）地址对象，封装了与该IP地址相关的所有信息，并提供获取信息的常用方法。

方法：

1）public static InetAddress getLocalHost() //获得本地主机地址对象

2）public static InetAddress getByName(String host) //根据主机名称获得地址对象

3）public static InetAddress[] getAllByName(String host) //获得所有相关地址对象

4）public String getHostAddress() //获取Ip地址字符串

5）public String getHostName() //获取IP地址主机名

### 10.基于TCP的网络编程（Socket编程）

Socket编程：

Socket（套接字）是网络中的一个通信节点。

分为客户端Socket和服务器ServerSocket

通信要求：IP地址 + 端口号

开发步骤：

1. 建立通信连接（会话）：

创建ServerSocket，指定端口号

调用accept()等待客户端接入

1. 客户端请求服务器：

创建Socket，指定服务器IP + 端口号

使用输出流，发送请求数据给服务器

使用输入流，接收响应数据到客户端（等待）

1. 服务器响应客户端：

使用输入流，接收请求数据到服务器（等待）

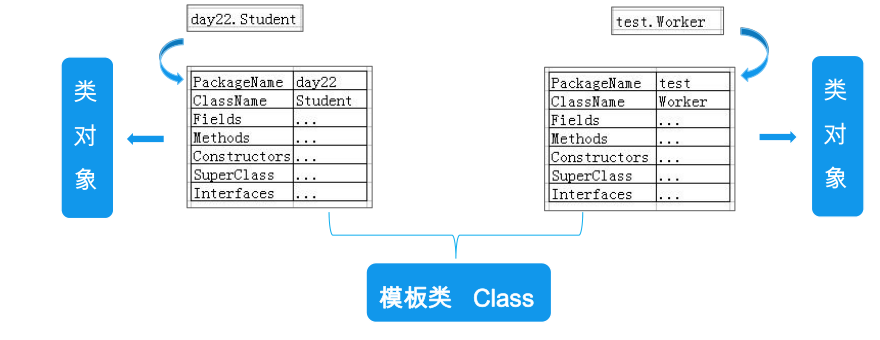
使用输出流，发送响应数据给客户端

## 反射

### 什么是类对象？

类的对象：基于某个类new出来的对象，也成为实例对象

类对象：类加载的产物，封装了一个类的所有信息（类名、父类、接口、属性、构造方法、方法）



### 获取类对象的方式

1. 通过类的对象，获得类对象

Student s = new Student();

Class c = s.getClass();

2）通过类名获取类对象

Class c = 类名.class();

3）通过静态方法获取类对象

Class c = Class.forName(“包名.类名”);

### 常用方法

1）String getName() //返回由 类对象表示的实体（类，接口，数组类，原始类型或空白）的名称，作为 String 。

2）static Class<?> forName(String className) //获取类对象名

3）Package getPackage() //获取类对象的包名

4）Class<? super T> getSuperclass() //获取父类的类对象名

5）Class<?>[] getInterfaces() //获取接口的类对象名

6）Constructor<?>[] getConstructors() //获取构造方法

7）Class<?>[] getParameterTypes() //获取方法(构造/成员)的参数类型列表

8）Field[] getFields() //获取属性（自身+父类的所有public公开属性）

9）Field[] getDeclaredFields() //获取属性（自身所有的属性）

10）Method[] getMethods() //获取方法（自身+父类单继承叠加的所有public公开方法）

11）Method[] getDeclaredMethods() //获取方法（自身所有的方法）

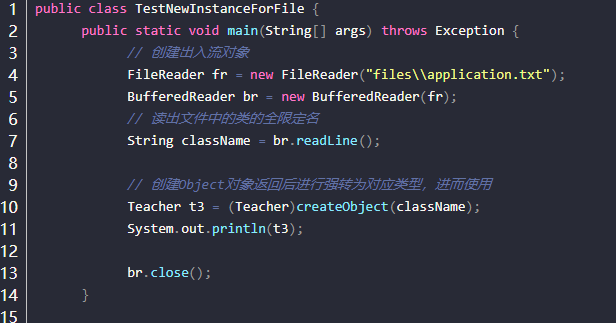
12）T newInstance() //创建由此类对象表示的类的新实例（此类对象必须有无参构造）。

### 4.工厂设计模式

开发中有一个非常重要的原则“开闭原则”，对拓展开放、对修改关闭

工厂模式主要负责对象创建的问题

可通过反射进行工厂模式的设计，完成动态的对象创建。

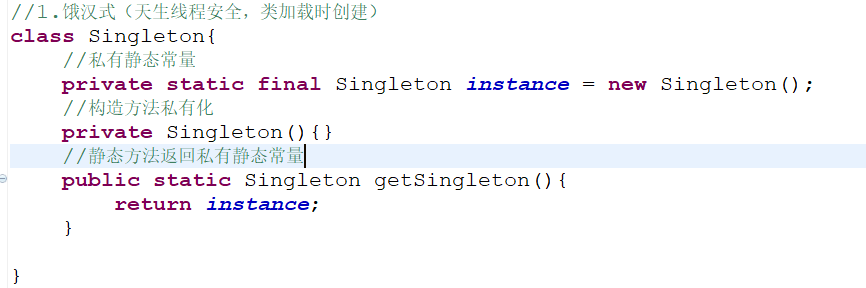
 

### 5.单例模式

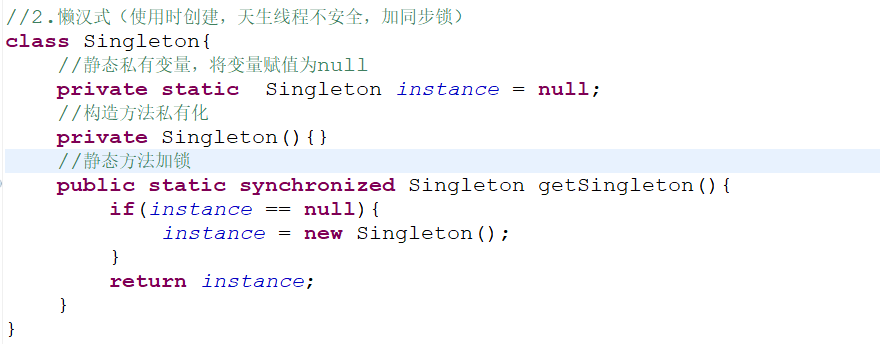
单例（Singleton）：只允许创建一个该类的对象。

创建方式：

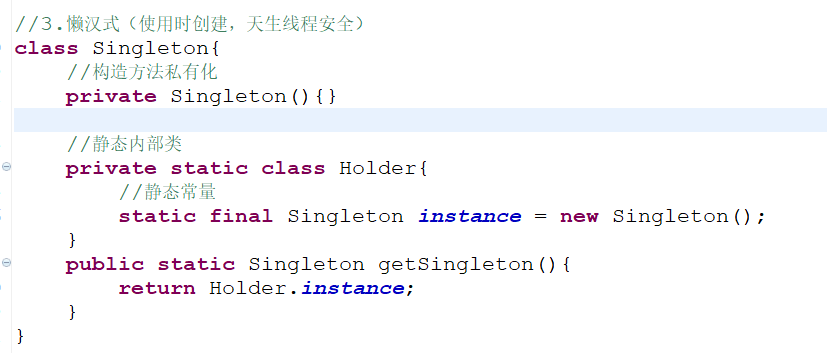
1. 饿汉式（类加载时创建，天生线程安全）



1. 懒汉式（使用时创建，线程不安全，加同步锁）

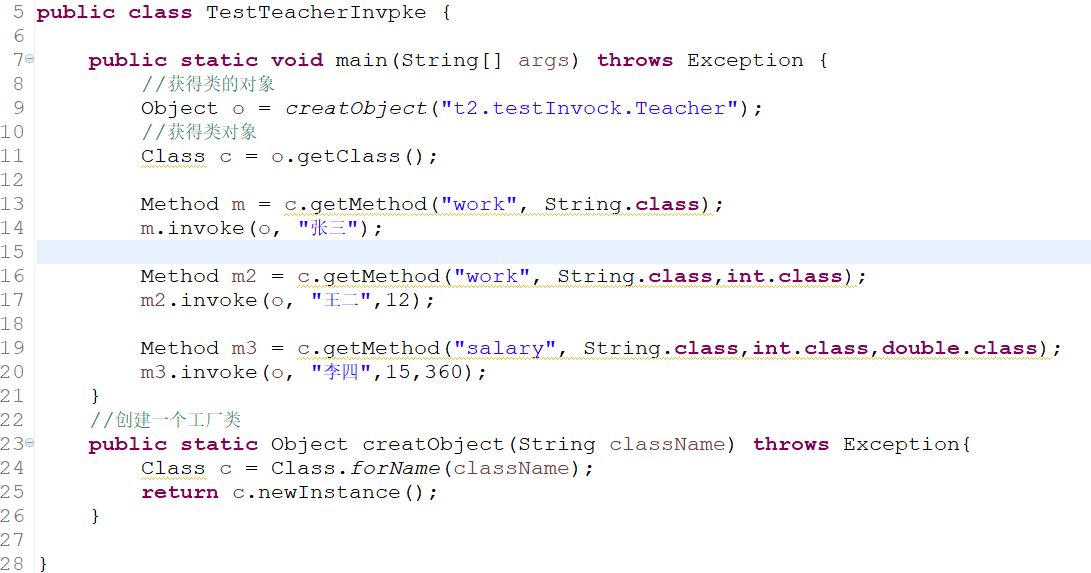


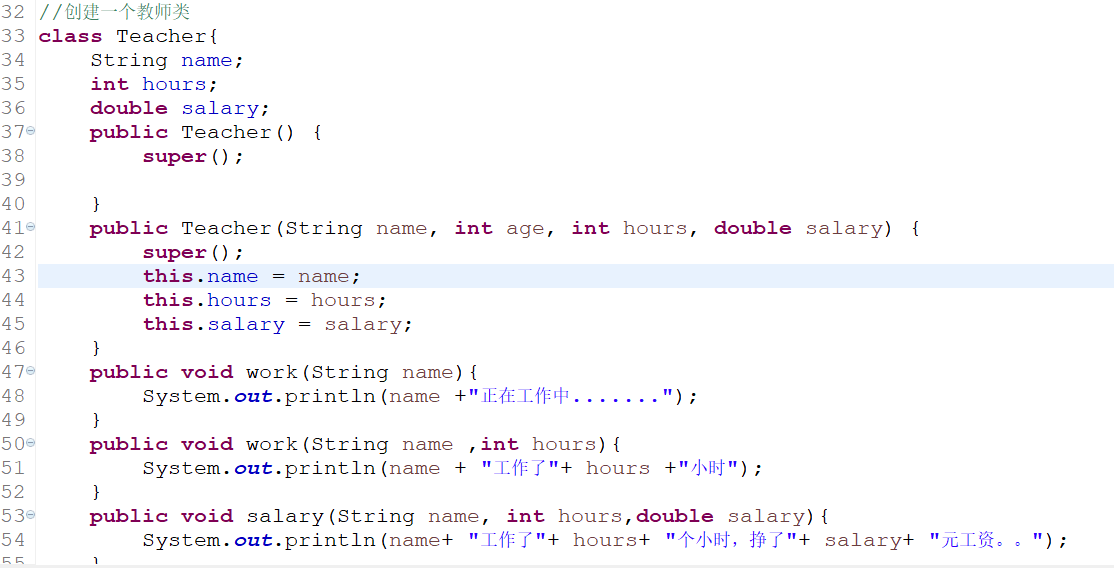
1. 懒汉式（使用时创建，线程安全）



### 6.Invoke方法

invoke(Object obj,Object … args) //对带有指定参数的指定对象调用由此Method对象表示的底层方法。





## JDK8

### 什么是Lambda?

概念：允许把函数作为一个方法的参数。（函数作为参数传递到方法中去）

格式： <函数式接口> <变量名> = （参数1，参数2….）-> {

//方法体

}

注意事项：

新的操作符：-> （箭头操作符）

（参数1，参数2）->表示参数列表

-> { } 方法体

1）形参列表的数据类型会自动推断

2）如果形参列表为空，只需保留（）

3）如果形参只有一个，（）可以省略，只要参数名字即可

4）如果执行语句只有一句，且无返回值，{}可省略

5）若有返回值，仍想省略{}，return也可省略。保证执行语句只有一句

6）Lambda表达式不会生成单独的内部类文件

7）Lambda访问局部变量时，变量要修饰final，如果没加，会自动添加

### 2.函数式接口

如果一个接口只有一个抽象方法，则称该接口为函数式接口。

为了确保接口达到要求，可以添加@FunctionalInterface注解。（当接口方法大于一个时，会爆红提醒）

内置四个核心函数式接口：

1. Consumer <T> 消费型接口 ：void accept(T t);
2. Supplier <T> 供给型接口 : T get();
3. Function <T,R> 函数型接口 ： R apply(T t);
4. Predicate <T> 断言型接口 ：boolean test(T t);

### 3.方法引用

方法引用是Lambda表达式的一种简写形式，如果Lambda表达式方法体中只是调用一个特定的已存在的方法，则可以使用方法引用。

使用 ：： 操作符将对象或类和方法名的名字分隔开来。

1. 对象 ：： 实例方法
2. 类 ：： 静态方法
3. 类 ：： 实例方法
4. 类 ：： new

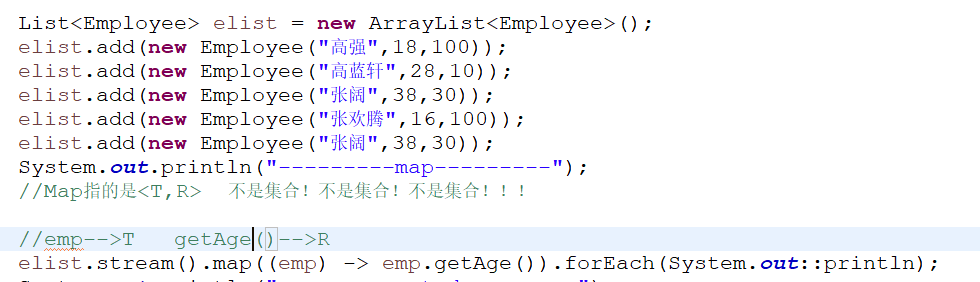
注意：调用的方法参数列表与返回值类型，要与函数型接口中的方法参数列表与返回值类型一致。

### Stream API

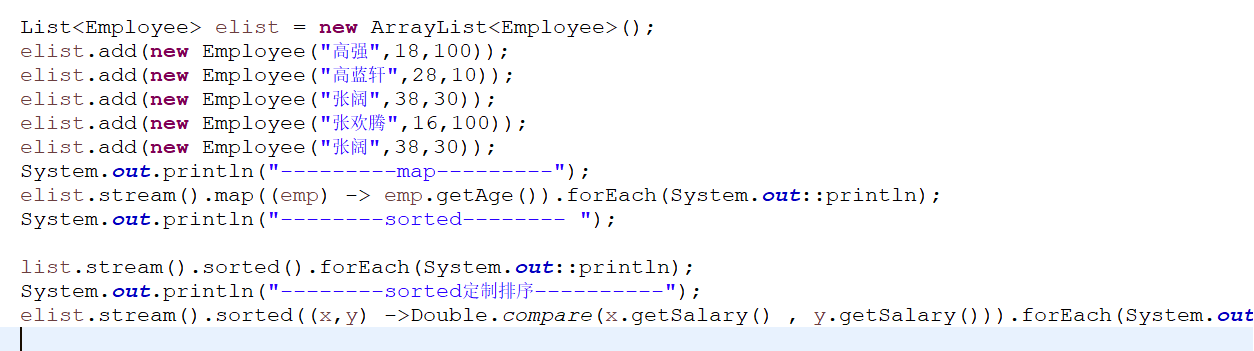
Stream是Java8中处理数组、集合的抽象概念。

可以执行非常复杂的查找、过滤、映射等操作。

1. 常用方法：中间操作
2. Stream<T> filter(Predicate<? super T> predicate); //过滤
3. Stream<T> limit(long maxSize); //截断，不超过给定数量
4. Stream<T> distinct(); //筛选（去重），利用hashCode和equals
5. <R> Stream<R> map (Function<? super T,? extends R > mapper); //不是map集合，



1. Stream<T> sorted(); //自然排序（需实现Comparable接口）
2. Stream<T> sorted(Comparator<? super T > comparator); //定制排序（需实现Comparable接口）



1. 常用方法：终止操作
2. long count(); //返回流中元素总个数
3. void forEach(Consumer<? super T> action); //遍历
4. boolean anyMatch(Predicate<? super T> predicate); //是否至少匹配一个
5. boolean allMatch(Predicate<? super T> predicate); //是否匹配所有元素
6. boolean noneMatch(Predicate<? super T> predicate); //是否没有匹配
7. Optional<T> findFirst(); //返回第一个
8. Optional<T> findAny(); //返回任意
9. Optional<T> min(Comparator<? super T> comparctor); //返回最小
10. Optional<T> max(Comparator<? super T> comparctor); //返回最大