## 一、线程复习大纲

1.什么叫做线程，和进程的区别

线程就是一个正在运行的软件（迅雷），下载一个东西就会分配出来一个线程

进程就是CUP、内存等分配的单位，执行时的一个实例。一个运行的软件就是进程

1. 如何创建线程（3种方式）

继承Thread类进行覆盖Run方法

实现Runnable接口

实现Callable接口

1. 实现Callable接口和实现Runnable接口有什么区别？

Callable接口中的call方法返回值是一个泛型，和Future配合可以获取异步执行结果

Runnable接口中的run方法的返回值是void它只是纯粹的去执行run方法中的代码，

1. 实现Callable接口如何获取线程的返回值

Future接口和FutureTask类

1. 三种新建的线程如何启动
   * + 1. 继承Thread类 重写run方法 new MyThread().start();
       2. 实现Runnable接口 重写run方法 new Thread（X,’名子’）.start（）；
       3. 实现Callable接口 重写call方法
2. 线程的状态有哪些？
   1. 新建
   2. 就绪
   3. 运行
   4. 堵塞
   5. 死亡
3. 每种状态之间是如何转换的

**初始化状态(New)**：新创建了一个线程对象。

**就绪状态(Runnable)**：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于“**可运行线程池**”中，变得可运行，只**等待获取CPU的使用权**。**即在就绪状态的进程除CPU之外，其它的运行所需资源都已全部获得。**进入就绪状态后，当该对象被操作系统选中，获得CPU时间片就会进入运行状态

**3、运行状态(Running)：**就绪状态的线程获取了CPU，执行程序代码。运行状态时如果线程调用了yield()方法，意思是放弃当前获得的CPU时间片，回到就绪状态，这时与其他进程处于同等竞争状态，OS有可能会接着又让这个进程进入运行状态

**4、阻塞状态(Blocked)：**阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。当线程调用了自身的sleep()方法或其他线程的join()方法就会进入阻塞状态(该状态既停止当前线程，但并不释放所占有的资源，不释放”锁标志”)。当sleep()结束或join()结束后，该线程进入可运行状态，继续等待OS分配时间片;

**5、锁池状态：**当线程刚进入就绪状态(注意，还没运行)，发现将要调用的资源被synchroniza(同步)，获取不到锁标记，将会立即进入锁池状态，等待获取锁标记(这时的锁池里也许已经有了其他线程在等待获取锁标记，这时它们处于队列状态，既先到先得)，一旦线程获得锁标记后，就转入就绪状态，等待OS分配CPU时间片

**6、队列状态：**当线程调用wait()方法后，会释放掉它所占有的“锁标志”并进入等待队列(进入这个状态会释放所占有的所有资源，与阻塞状态不同)，进入这个状态后，是不能自动唤醒的，必须依靠其他线程调用notify()或notifyAll()方法才能被唤醒(由于notify()只是唤醒一个线程，但我们由不能确定具体唤醒的是哪一个线程，也许我们需要唤醒的线程不能够被唤醒，因此在实际使用时，一般都用notifyAll()方法，唤醒有所线程)，线程被唤醒后会进入锁池，等待获取锁标记。

**7、终止状态(Dead)：**线程执行完了(即main()方法结束后)或者因异常退出了run()方法，该线程结束生命周期。

1. 四种常用的线程池，包括如何使用线程池

定长线程

可缓存线程

单线程化

1. newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。

public class ThreadPoolDemo {

public static void main (String[] args) {

ExecutorService cachedThreadPool = Executors.newCachedThreadPool();

for(int i = 0; i < 10; i++) {

final int index = i;

try {

Thread.sleep(index\*1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

cachedThreadPool.execute(new Runnable () {

public void run () {

System.out.println(index);

}

});

}

}

}

1. newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。

public class ThreadPoolDemo {

public static void main (String[] args) {

ExecutorService fixedThreadPool = Executors.newFixedThreadPool(3);

for(int i = 0; i < 10; i++) {

final int index = i;

fixedThreadPool.execute(new Runnable () {

public void run () {

System.out.println(index);

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

});

}

}

}

1. newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。

public class ThreadPoolDemo {

public static void main (String[] args) {

ScheduledExecutorService scheduleThreadPool = Executors.newScheduledThreadPool(5);

scheduleThreadPool.schedule(new Runnable() {

public void run() {

System.out.println("delay 3 seconds");

}

}, 3, TimeUnit.SECONDS);

}

}

     （4）newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

public class ThreadPoolDemo {

public static void main (String[] args) {

ExecutorService singleThreadExecutor = Executors.newSingleThreadExecutor();

for(int i = 0; i < 10; i++) {

final int index = i;

singleThreadExecutor.execute(new Runnable() {

public void run() {

try {

System.out.println(index);

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

});

}

}

}

1. 什么叫做线程安全
   * 1. 确保在多线程访问的时候，我们的程序还能按照我们的预期的行为去执行
2. 实现线程安全的方式有哪些（锁，原子类）

synchronized锁（偏向锁，轻量级锁，重量级锁）

volatile锁，只能保证线程之间的可见性，但不能保证数据的原子性

jdk1.5并发包中提供的Atomic原子类

Lock锁

1. 锁都有哪些
   * 1. 乐观锁/悲观锁
     2. 公平锁/非公平锁
     3. 独享锁/共享锁
     4. 互斥锁/读写锁
     5. 自旋锁
     6. 偏向锁
     7. 轻量级所/重量级锁
     8. 可重入锁
     9. 分段锁
2. synchronized代码块和方法的锁对象是谁

对象锁：synchronized(this)以及非static的synchronized方法。锁的是这个对象。

* 全局锁：synchronized(类名.class)和static 的synchronized方法。锁的是那个写了synchronized关键字的方法或者代码块。

1. Lock的实现类有哪些
   * 1. AQS(AbstractQueuedSynchronizer)、ReentrantLock、ReentrantReadWriteLock,CountDownLatch,Semphore。
2. 关键方法的用法： join() wait() notify() yield()
   1. if（i==10）{t.join();}
   2. 当前线程放弃同步锁进入等待状态，直到其他线程进入此同步锁notify（）或notifyAll（）唤醒线程为止。
   3. 唤醒锁上等待的第一个调用wait（）方法的线程
   4. if（i==10）{Thread.yield();}
3. wait()和sleep（）的区别
   * 1. Sleep它是让线程暂时睡眠，可以放在任何位置
     2. Wait是让当线程暂时等待，必需放在同步方法或者同步块里
4. 线程安全的集合有哪些？

Vector：就比Arraylist多了个同步化机制（线程安全）。

Hashtable：就比Hashmap多了个线程安全。

ConcurrentHashMap:是一种高效但是线程安全的集合。

Stack：栈，也是线程安全的，继承于Vector。

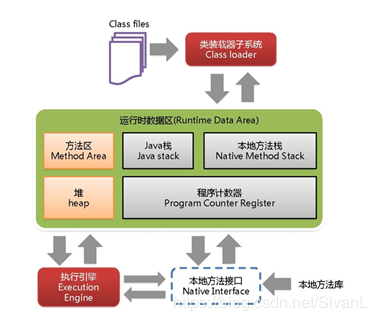
## 二、JVM复习

（1）基础问题

1.jvm的功能时什么

将写好的源程序编译成.class文件，再解析成计算机所识别的二进制码

1. 什么是JRE, JDK?
   1. JRE就是java 的运行环境
   2. JDK就是java 的开发工具
2. JVM的结构有哪些？



1. JVM的内存结构和每块内存的存储对象。
   1. 类加载器：负责加载class文件，将calss文件字节码内容加载到内存中，并将这些内容转换成方法区中的运行时数据结构
   2. 本地方法栈：线程私有的，为虚拟机执行Native方法服务。
   3. Java堆：线程共享，在虚拟机启动时创建，存放对象实例和数组
   4. 方法区：存放虚拟机加载的类信息，常量，静态变量
   5. 程序计数器：作为当前线程所执行的字节码的行动指示器，主要记录当前线程在执行到哪里
2. JVM内存结构中堆内存的分区
   * 1. 新生代 老生代
     2. Eden空间
     3. From Survivor空间
     4. To Survivor空间
3. Java类加载的几个过程。
   1. 类的加载把.class文件中的数据读取到内存中，通常创建一个字节数组读入.class文件，然后产生与所加载类对应的class对象，加载完成后，进入连接阶段，包括验证、准备和解析，最后jvm对类进行初始化
4. 一个类的实例化顺序。（比如：父类静态数据，构造函数，字段，子类静态数据，构造函数，字段）
   * 1. **父类静态成员和静态代码块->子类静态成员和静态代码块->父类非静态成员和非静态代码块->父类构造方法->子类非静态成员和非静态代码块->子类构造方法**
5. GC的两种判定方法是什么
   1. 引用计数：象再被创建时，对象头里会存储引用计数器，对象被引用，计数器+1；引用失效，计数器 －1；GC时会回收计数器为0的对象。但是JVM没有用这种方式，因为无法判定相互循环引用（A引用B,B引用A）的情况，无法解决对象互相循环引用。
   2. 引用链：  程序把所有的引用看作图（类似树结构的图），选定一个对象作为GC Root根节点，从该节点开始寻找对应的引用节点并标记，找到这个节点之后，继续寻找这个节点的引用节点，当所有的引用节点寻找完毕之后，剩余的节点认为是不可达的无用节点，会被回收
6. 垃圾回收算法有几种类型？
   * 1. 标记清除法
     2. 复制算法
     3. 标记整理法
     4. 分带收集算法
7. GC如何判断一个类是“无用的类，需要回收。

1.在JAVA堆中不存在该类的实例。

2.加载该类的 ClassLoader 已经被回收。

3.该类对应的 java.lang.Class 对象没有在任何地方被引用，无法在任何地方通过反射访问该类的方法。

（2）扩展知识

11. **说一下强引用、软引用、弱引用、虚引用以及他们之间和gc的关系**

**强引用：只要程序痛过new关键字创建对象，那么垃圾回收期永远不会进行对象回收，除非系统内存不够**

**软引用：非必须引用，当内存资源不够即将溢出时，这个时候GC将会对对象的内存空间进行回收，从而避免内存溢错误的发生**

**弱引用：第二次垃圾回收时回收，不管当前内存资源是否充足。**

**虚引用：虚引用主要用于检测对象是否已经从内存中闪出，只要垃圾回收开始工作，那么虚引用就会被回收。**

1. 与垃圾回收相关的JVM参数有哪些？

-Xms / -Xmx — 堆的初始大小 / 堆的最大大小  
        -Xmn — 堆中年轻代的大小  
        -XX:-DisableExplicitGC — 让System.gc()不产生任何作用  
        -XX:+PrintGCDetails — 打印GC的细节  
        -XX:+PrintGCDateStamps — 打印GC操作的时间戳  
        -XX:NewSize / XX:MaxNewSize — 设置新生代大小/新生代最大大小  
        -XX:NewRatio — 可以设置老生代和新生代的比例  
        -XX:PrintTenuringDistribution — 设置每次新生代GC后输出幸存者乐园中对象年龄的分布  
        -XX:InitialTenuringThreshold / -XX:MaxTenuringThreshold：设置老年代阀值的初始值和最大值  
        -XX:TargetSurvivorRatio：设置幸存区的目标使用率

1. 你知道哪些JVM性能调优。
   1. 调优可以依赖、参考的数据有系统运行日志、堆栈错误信息、gc日志、线程快照、堆转储快照等
   2. 系统运行日志：系统运行日志就是在程序代码中打印出的日志，描述了代码级别的系统运行轨迹（执行的方法、入参、返回值等），一般系统出现问题，系统运行日志是首先要查看的日志。
   3. 堆栈错误信息：当系统出现异常后，可以根据堆栈信息初步定位问题所在，比如根据“java.lang.OutOfMemoryError: Java heap space”可以判断是堆内存溢出；根据“java.lang.StackOverflowError”可以判断是栈溢出；根据“java.lang.OutOfMemoryError: PermGen space”可以判断是方法区溢出等。
   4. GC日志：程序启动时用 -XX:+PrintGCDetails 和 -Xloggc:/data/jvm/gc.log 可以在程序运行时把gc的详细过程记录下来，或者直接配置“-verbose:gc”参数把gc日志打印到控制台，通过记录的gc日志可以分析每块内存区域gc的频率、时间等，从而发现问题，进行有针对性的优化。
   5. 用 jps（JVM process Status）可以查看虚拟机启动的所有进程、执行主类的全名、JVM启动参数，比如当执行了JPSTest类中的main方法后（main方法持续执行），执行 jps -l可看到下面的JPSTest类的pid为31354，加上-v参数还可以看到JVM启动参数。
   6. 用jstat（JVM Statistics Monitoring Tool）监视虚拟机信息

14.有没有遇到过OutOfMemeory问题，你是怎么处理这个问题的。

permgen space、heap space 错误。

　　　　　　常见的原因

* + - 内存加载的数据量太大：一次性从数据库取太多数据；
    - 集合类中有对对象的引用，使用后未清空，GC不能进行回收；
    - 代码中存在循环产生过多的重复对象；
    - 启动参数堆内存值小。

15.JDK1.8后Perm Space有哪些变动？

JDK 1.8后用元空间替代了 Perm Space；字符串常量存放到堆内存中。

16. MetaSpace默认是有限的吗？

MetaSpace大小默认没有限制，一般根据系统内存的大小。JVM会动态改变此值。

-XX:MetaspaceSize：分配给类元数据空间（以字节计）的初始大小（Oracle逻辑存储上的初始高水位，the initial high-water-mark）。此值为估计值，MetaspaceSize的值设置的过大会延长垃圾回收时间。垃圾回收过后，引起下一次垃圾回收的类元数据空间的大小可能会变大。

-XX:MaxMetaspaceSize：分配给类元数据空间的最大值，超过此值就会触发Full GC，此值默认没有限制，但应取决于系统内存的大小。JVM会动态地改变此值。

|  |
| --- |
|  |

16.StackOverflow异常有没有遇到过，你猜测会在什么情况下触发。

栈内存溢出，一般由栈内存的局部变量过爆了，导致内存溢出。出现在递归方法，参数个数过多，递归过深，递归没有出口。

## 三、IO流

1.操作Path

创建一个Path， File和Path之间的转换，File和URI之间的转换

File和Path ：String path = file.getPath()

Uri转file：file = new File(new URI(uri.toString()));

2.操作File类

判断文件是否存在？创建文件？创建文件夹？删除文件？删除文件？复制文件？遍历文件夹？

判断文件是否存在：

public static boolean existsFile(String path) {

File file = new File(path);

return file.exists() && file.isFile();

}

创建：

public static void main(String[] args) {

*// 创建File对象*

File file = new File("a.txt");

try {

*//创建*

file.createNewFile();

} catch (IOException e) {

*// TODO Auto-generated catch block*

e.printStackTrace();

}

创建文件夹：

public static void main(String[] args) {

*// 创建File对象*

File file = new File("liuguilin");

file.mkdir();

}

删除文件：

public void delFile(String filePathAndName) {

try {

String filePath = filePathAndName;

filePath = filePath.toString();

java.io.File myDelFile = new java.io.File(filePath);

myDelFile.delete();

}

catch (Exception e) {

System.out.println("删除文件操作出错");

e.printStackTrace();

}

}

删除文件夹：

public void delFolder(String folderPath) {

try {

delAllFile(folderPath); //删除完里面所有内容

String filePath = folderPath;

filePath = filePath.toString();

java.io.File myFilePath = new java.io.File(filePath);

myFilePath.delete(); //删除空文件夹

}

catch (Exception e) {

System.out.println("删除文件夹操作出错");

e.printStackTrace();

}

}

复制文件：

\*/

public void copyFile(String oldPath, String newPath) {

try {

int bytesum = 0;

int byteread = 0;

File oldfile = new File(oldPath);

if (oldfile.exists()) { //文件存在时

InputStream inStream = new FileInputStream(oldPath);

//读入原文件

FileOutputStream fs = new FileOutputStream(newPath);

byte[] buffer = new byte[1444];

int length;

while ( (byteread = inStream.read(buffer)) != -1) {

bytesum += byteread; //字节数 文件大小

System.out.println(bytesum);

fs.write(buffer, 0, byteread);

}

inStream.close();

}

}

catch (Exception e) {

System.out.println("复制单个文件操作出错");

e.printStackTrace();

}

}

遍历文件夹：

public class FileTest {

public static void main(String[] args) {

File dir = new File("F:/documents/example"); listDirectory(dir);

}

public static void listDirectory(File dir) {

if (!dir.exists()) {

throw new IllegalArgumentException("目录" + dir + "不存在");

} if (!dir.isDirectory()) {

throw new IllegalArgumentException(dir + "不是目录");

}

File[] files = dir.listFiles();

if (files != null && files.length > 0) {

for (File file : files) {

if (file.isDirectory()) { listDirectory(file); } else {

System.out.println(file);

}

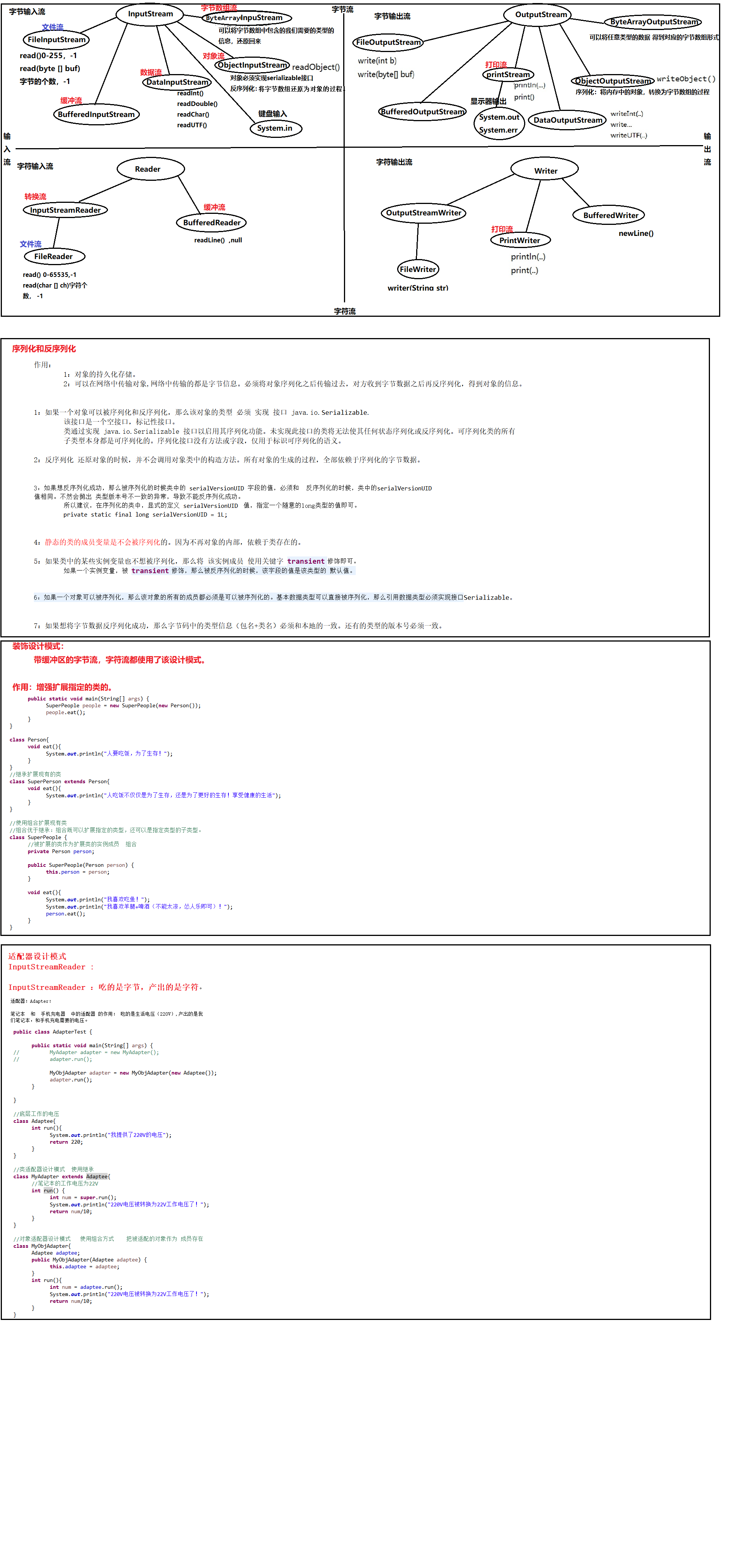
}

}

}

}

1. IO架构



**File常用 API**

Exists 判断文件是否存在

createNewFile 创建文件

Mkdir/s 创建问津/多级目录

Delete 删除目录

1. IO中直接操作文件的类有哪些？缓冲操作的类有哪些？对象序列化操作，转换控制类，打印控制类有哪些？
2. 通过IO实现一个文件的复制；

public class Demo4 {

    public static void main(String[] args) throws IOException {

        //1.首先创建一个文件字节输入流来读取数据

        FileInputStream in=new FileInputStream("D:\\sjk.txt");

        //2.创建一个文件字节输出流来写数据

        FileOutputStream out=new FileOutputStream("D:\\a.txt");

        //3.创建一个字节数组作为缓冲区，通过定义长度来规定每次读取的大小

        byte[]b=new byte[10];

        //4.定义变量记录总共读取的字节个数

        int len;

        //5.读取sjk.txt中的数据

        while((len=in.read(b))!=-1){

            //6.将数据写入a.txt

           out.write(b,0,len);

        }

     //7.关流

     in.close();

     out.close();

   }

}

1. IO和NIO的区别
   * 1. IO是阻塞IO
     2. NIO是非阻塞IO

IO：面向字节流 阻塞

INO：面向缓冲区 基于选择器的非阻塞

1. NIO的组件有哪些

Allocate 分配

Read 读

Write 写

Clear 清空

Flip 读写切换

* 1. Selector(选择器)
  2. Channel(管道) fileChannel
  3. Buffer(缓冲区) capacity，position，limit

1. Buffer介绍，（capacity，position，limit的介绍）； Buffer的API方法。
   * 1. Capacity代表buffer总容量 如果buffer满了就要清空 clear。
     2. position表示当前的位置初始化是0 ，position最大可为capacity – 1
     3. Limit表示你能往buffer里面存多少数据能读到之前写入的所有数据。
   1. Buffer 便是用于读取或操作二进制数据的对象。
   2. 作为一个内存块，Buffer有一个固定的大小值，也叫“capacity”.你只能往里写capacity个byte、long，char等类型。一旦Buffer满了，需要将其清空（通过读数据或者清除数据）才能继续写数据往里写数据。
   3. position表示当前的位置。初始的position值为0.当一个byte、long等数据写到Buffer后， position会向前移动到下一个可插入数据的Buffer单元。position最大可为capacity – 1。 当读取数据时，也是从某个特定位置读。当将Buffer从写模式切换到读模式，position会被重置为0。当从Buffer的position处读取数据时，position向前移动到下一个可读的位置。
   4. 在写模式下，Buffer的limit表示你最多能往Buffer里写多少数据。 写模式下，limit等于Buffer的capacity。 当切换Buffer到读模式时， limit表示你最多能读到多少数据。因此，当切换Buffer到读模式时，limit会被设置成写模式下的position值。换句话说，你能读到之前写入的所有数据
2. Channel的介绍和API方法

主要有三大基本的组件：Buffer、Channel和Selector，在Java NIO的世界中，Selector是中央控制器，Buffer是承载数据的容器，而Channel可以说是最基础的门面，它是本地I/O设备、网络I/O的通信桥梁，只有搭建了这座桥梁，数据才能被写入Buffer，连接才能被Selector控制

1. 通过NIO实现一个文件的复制。

@Test

public void copyFileByNoi() throws IOException {

//获取输入流和输出流

FileInputStream in = new FileInputStream("a.txt");

FileOutputStream out = new FileOutputStream("b.txt");

//获取nio中的通道

FileChannel inChannel = in.getChannel();

FileChannel outChannel = out.getChannel();

//指定缓冲区大小

ByteBuffer buffer = ByteBuffer.allocate(1024);

//将通道的数据写入缓冲区中

while (inChannel.read(buffer)!=-1){

//切换为读取数据的模式

buffer.flip();

//将缓冲区中的数据写入到输出流中

outChannel.write(buffer);

//清空缓冲区

buffer.clear();

}

//关闭资源

out.close();

in.close();

outChannel.close();

inChannel.close();

}

1. **字节流与字符流有什么区别？**

字节流操作的基本单元为字节；字符流操作的基本单元为Unicode码元。

字节流默认不使用缓冲区；字符流使用缓冲区。

字节流通常用于处理二进制数据 但它不支持直接写入或读取Unicode码元；字符流通常处理文本数据，它支持写入及读取Unicode码元。

1. **字节流和字符流哪个好？怎么选择？**
   1. 如果是文本文件通常使用字符流，而像视频，图片，音频等文件都是二进制数据使用字节流。当然文本文件也可以使用字节流来操作，字节流更通用。
2. 知道其他的IO模型

阻塞IO

非阻塞IO

IO复用（select和poll）

信号驱动

异步IO（Posix.1 的 aio...系列函数

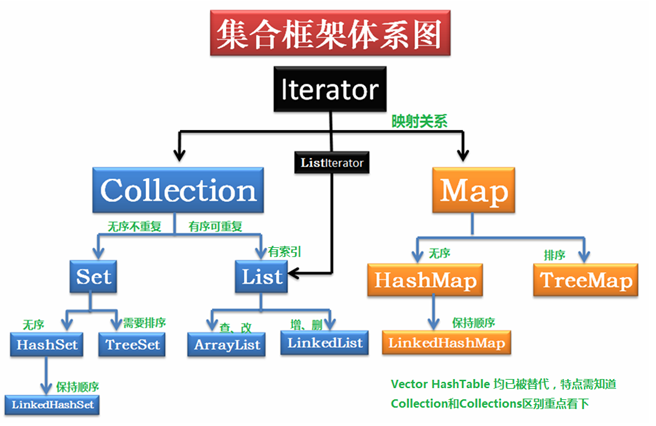
Bio 同步阻塞io

Jdk1.4后 nio 同步非阻塞io

Jdk1.7 aio异步aio nio.2

## 四、集合

1.集合的架构



1. HashMap和HashTable有什么不同？
2. HashMap是新版JDK提供的接口 Hashtable是早期提供的接口
3. HashMap实现Map接口 Hashtable继承Dictionary类，。
4. HashMap线程不安全 Hashtable线程安全，。
5. HashMap允许null值 Hashtable不允许null值，。

3.ArrayList和Vecotor有什么不同？

ArrayList线程不同步，Vecotor线程同步

ArrayList比Vecotor快

ArrayList更加通用

1. ArrayList和LinkedList有什么不同？
   1. ArrayList线程安全，查询快增删慢
   2. ListedList线程不安全 增删快，查询慢
   3. Vecotor线程安全，查询增删都慢。

5.Array和ArrayList有什么不同？

Array可以容纳基本类型和对象，ArrayList只容纳对象

Array可以指定大小 而ArrayList大小固定

1. ArrayList的初始大小是多少？扩容机制是什么？
   1. 初始大小是10
   2. 1.5倍
2. HashMap和HashSet的初始大小是多少？扩容机制是什么？
   1. HashMap初始大小是16 HashSet初始16
   2. HashMap扩容0.75可以指定初始化大小和扩容 HashSet第一次扩容为32
3. 哪些集合类是线程安全的？
   1. ArrayList
   2. Vecotor
   3. HashTable
4. 并发集合类是什么？
   1. ConcurrentHashMap
5. 我们能否使用任何类作为Map的key?
6. 遍历集合的方式有几种
   1. Map遍历
   2. List遍历
   3. For循环
   4. 迭代器Iterator
   5. Foreach
7. 在遍历集合时如何避免ConcurrentModificationException异常
   1. Iterator.remove()*从迭代器中移除集合元素*
8. **Comparable和Comparator是什么？**
   1. **Comparable是排序接口**
   2. **Comparator是比较器接口**
9. **了解**Collections.unmodifiableCollection(Collection c)和Collections.synchronizedCollection(Collection c)的作用。

15.集合框架的泛型有什么优点

16.Iterator是什么

是一个迭代器是一个设计模式，她是一个对象。

## 五．异常

1.java异常体系



1. Exception和Error的区别。
   1. Error是程序无法处理的错误，它是由JVM产生和抛出的，  
          Exception是程序本身可以处理的异常，这种异常分两大类运行时异常和非运行时异常。程序中应当尽可能去处理这些异常。
2. 描述运行时异常（非检查型异常）和编译期异常（检查型异常），并举例说明。
   1. 非检查异常:  
      ClassCastException  
      IllegalThreadStateException(两次启动线程)  
        
      检查异常:  
      InterruptedException(Thread.sleep()会抛出此异常)
3. 异常的处理方式
   1. 捕获和抛出

5.异常处理的一般原则

****尽量不要捕获类似Exception这样的通用异常，而是应该捕获特定异常****。

****不要生吞异常****

1. throw和throws两个关键字的不同。
   1. Throw用于抛出异常对象，后面跟的是异常对象，throw用于函数内
   2. Throws用于抛出异常类，后面跟的是异常类名，可跟多个 用逗号隔开
2. java中final,finalize,finally关键字的区别。

final可以修饰类,不能被继承  
   \* 修饰方法,不能被重写  
   \* 修饰变量,只能赋值一次  
   \*   
   \* finally是try语句中的一个语句体,不能单独使用,用来释放资源  
   \*   
   \* finalize是一个方法,当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。

1. finally和finalize方法有什么区别。
   1. 无论是否抛出异常，finally代码块bai都会执行，它主du要是用来释放应用占zhi用的资源。finalize()方法是Object类的一个protected方法，它是在对象被垃圾回收之前由Java虚拟机来调用的。
2. 如果try里面已经使用了return返回结果，finally还会执行吗？如果try里system.exit()， JVM退出了，finally代码块还会执行吗？
   1. 会继续执行 finally是在return后面的表达式运算之后执行的，
3. 如何自定义异常
   1. ****新建异常类（继承自Exception或RuntimeException）****
   2. ****重写HandlerExceptionResolver接口的resolveException方法做一个全局的异常处理（基于spring mvc****
   3. ****配置异常类****

## 六．反射和代理

1.什么是反射，为什么要用反射，哪里用到了反射。

1在运行中对于任意一个类，都能够知道这个类的所有属性和方法。

2反射可以赋予jvm动态编译能力

3通过baijava反射获得加载类，获得类的实例

还有可以获得类中bai的私有成员属性

1. 获取clazz对象的三种方法

是通过类对象来获取class的

通过类直接调class

通过Class.forName();

1. 如何使用java的反射获取属性，方法，构造器。
   1. https://blog.csdn.net/u013453970/article/details/48377687

4.如何通过反射调用一个方法，通过反射得到一个实例对象。

1.什么是代理

2.通过实现同一个接口实现静态代理，列出代码和类图。（类似于JDK的方式）

3.通过继承的方式实现静态代理，列出代码和类图。（类似于**CGLIB的方式**）

4.动态代理比静态代理的优势。

5.通过Proxy创建动态代理类，写一个实例。

6.spring AOP底层使用的是动态代理的方式，通过AOP实现操作日志的功能。

## 七、比较异同

1.抽象类和接口的异同

接口是接口，抽象类是一个类

接口中不可以存在构造器，抽象类可以存在构造器

接口中的属性是静态常量，抽象类中可以有普通方法

接口用来实现，抽象类用于继承

1. overwrite（重写） 与 overload（重载）
   1. 子类继承父类的方法，方法名，参数列表，返回值类型都一样，对方法体进行修改和重写
   2. 在一个类中相同的方法名，参数不同，重载对返回值类型没有要求

3、throw 与 throws

Throw用于抛出异常对象，后面跟异常类对象，用在函数内

Throws用于抛出异常类，后面跟异常类名，可以跟多个用在函数上

1. final 、finally 与 finalize
   1. ****final****是修饰符，修饰类、方法和变量，意为不可修改的。
   2. ****finally****是在异常处理时提供finally块来执行任何清除操作的
   3. ****finalize****是方法名，在垃圾回收器将对象从内存中清理出去之前做的必要工作

5、Collection 与 Collections

Collection是单列集合的接口，有子类接口list和set

Collections是针对集合的操作工具类

6、String 、StringBuffer、StringBuilder

String是一个不可变的类，一旦被创建是不可改变，直到销毁

Stringbuffer安全可变的字符串

Stringbuilder jdk1.5与stringbuffer基本相同，线程不安全创建字符串优先使用stringbuildr

1. ArrayList 与 LinkedList、Vector
   1. Arraylist底层是数组，查询快 增删慢
   2. LinkedList底层是离链表，增删快查询慢
   3. Vector底层是数组，查询增删都慢。

8.HashMap、TreeMap、LinkedHashMap、Hashtable、ConcurrentHashMap

9、http 、 https

http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

http的连接很简单，是无状态的；；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输

10、sleep（）、wait（）

Sleep是让线程睡眠一段时间，不会立马释放锁，等待执行时间到才会释放

Wait让线程等待一段时间 ，调用时会立马释放锁，进入等待状态，等待其他线程唤醒

11、== / equals（）

==比较基本类型比较的是内容，比较引用类型比较的是地址值

Equles不管是比较基本数据类型还是引用类型都比较地址值

1. cookie / session
   1. Session安全 cookie不安全
   2. Session是存放在服务器上，cookie是存放浏览器上
2. stack / queue
   1. 1.删除数据元素的位置不同，栈的删除操作在表尾进行，队列的删除操作在表头进行。
   2. **队列先进先出，栈先进后出。**
   3. 顺序栈能够实现多栈空间共享，而顺序队列不能。

## 八、jdk8的新特性

一、增加了新语法：Lambda 表达式使代码更简洁

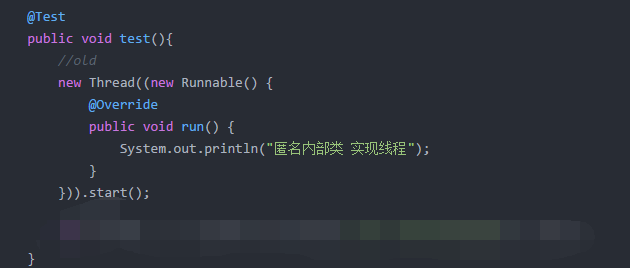
1.Lambda语法

**(parameters) -> expression**或  **(parameters) ->{ statements; }**

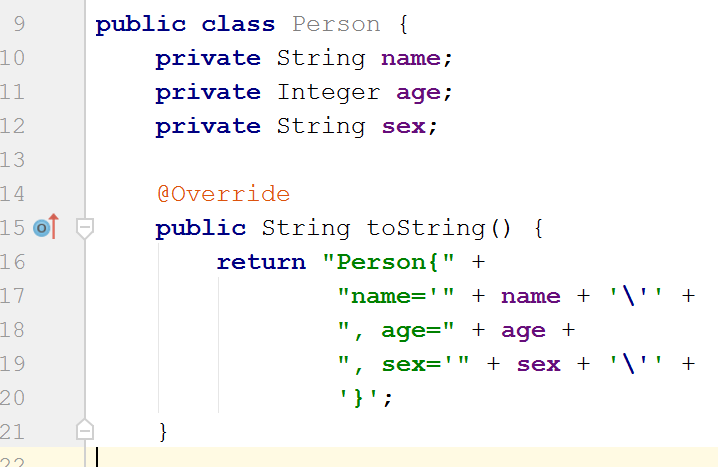
根据这个语法分别写两个简单的例子，带参数和不带参数的。

2.实例1：用() -> {}代码块替代了整个匿名类

要求：用Lambda表达式改写下面的代码



3.强大的 Stream API（集合数据的操作）



如上一个Person类的list集合。要求如下：

1）通过forEach方法用Lambda表达式遍历

2）通过filter 方法过滤年龄大于20的并打印。

3）通过limit方法返回年龄>20的前两条数据

4）通过sorted方法对年龄进行排序

5）通过map方法把每个人的年龄增加3岁

6）通过reduce方法计算所有人年龄的综合

7）获取年龄最小的Person

8 )获取年龄最大的Person

4.强大的collect API，生成新的集合对象

1）通过toMap方法把list集合转为map集合，name为key,Person对象为value.

2）将name属性用“，”连接成字符串.

3）将所有的name属性放到一个新的set集合中

4）通过groupingBy方法按年龄进行分组

5.利用IntSummaryStatistics、LongSummaryStatistics 或者 DoubleSummaryStatistics分别计算Int, Long, Double格式的list集合的最大值，最小值，平均值，总和和个数。

int集合为：{4,6,8,3,6,23,4,56,7,34,22,55}

二.最大化的减少空指针 异常：Optional 类 的使用

Optional.ofNullable(T t):若 t 不为 null,创建 Optional 实例,否则创建空实例

isPresent() : 判断是否包含值

orElse(T t) : 如果调用对象包含值，返回该值，否则返回t

map(Function f): 如果有值对其处理，并返回处理后的Optional，否则返回Optional.empty()

flatMap(Function mapper):与 map 类似，要求返回值必须是Optional

三、 接口的新特性

要求：新建一个接口类在类里添加静态方法和默认方法。

四、新日期时间的 api

jdk8之前我们用的时间API包括Date, Calendar, date和字符串之间的转换用SimpleDateFormat。JDK8后，在java.time包下有LocalDate, LocalTime, LocalDateTime, Instant等表示时间的类。用DateTimeFormatter进行时间和字符串之间的转换。

1）获取当前时间的时间信息

1.使用LocalDate获取今天的年月日星期；

2.通过LocalTime获取当前的时分秒

**3.通过LocalDateTime获取当前时间的年月日时分秒**

2）给定时间(分别用两种形式)

1.获取给定（2020年5月1日）时间的LocalDate

**2.获取给定（**12点6分3秒**）时间的LocalTime**

**3.获取给定(**2020年5月1日12点6分3秒**)时间的LocalDateTime**

3)日期的比较

1.LocalDate的isBefore(), isAfter(), equals()方法用来比较时间，举例说明，当前时间和2020-05-01日进行比较。

4）对时间的修改

1.用LocalDate为例来获取当前日期，去年，上个月，上周，昨天的日期，并获取明年，下个月，下周和明天的日期。

2.以LocalTime为例获取5个小时前，15分钟前和50秒前的时间，和5个小时后，15分钟后和50秒后的时间。

3.以LocalDateTime为例通过minus和plus两个方法完成1和2的要求；

minus和plus对应LocalDate, LocalTime的用法都是一样的。

5）日期时间格式化

以LocalDateTime为例，把当前时间转为yyyyMMdd HHmmss的格式，然后再把字符串转为LocalDateTime;

6)时间戳

1.获取当前的时间戳对象Instant,

把Instant转换为LocalDateTime和Date,

把LocalDateTime和Date对象转为时间戳

7)日期之间的时间差

1.用Period求LocalDate格式的2000-01-01到今天相差几年几个月天

2.用Duration计算LocalDateTime格式的2020-05-01 15:05:03相差的天数和小时数。

3.以LocalDateTime为例用ChronoUnit计算2020-05-01 15:05:03距今相差的年数，月数，天数，小时数，分钟数。

五、其它的新特性

1.集合的默认长度

2.jvm内存

## 九、web

1)tomcat服务器面试题

1.说一下session和cookie的区别

Session安全 cookie不安全

Session是存放在服务器上，cookie是存放浏览器上

2.说一说SessionStorage和localStorage还有cookie

3.tomcat的四种部署办法

4.tomcat调优

5.Nginx是什么及作用

2）网络协议

1.陈述输入url然后回车后的场景

2.陈述http，

3.http和https的区别

4.什么是无状态，怎么解决HTTP的无状态

5.简述传输层Tcp和UDP的区别

6.简述OSI七层协议模型和TCP/IP四层协议模型

7.你知道的应用层协议有哪些？

8.什么是socket？简述基于tcp协议的套接字通信流程。

9.什么是DNS

3）jsp面试题

1.jsp和servlet的区别，共同点和各自应用的范围

2.page，requet，session，application的作用范围和应用场景

3.写出5种以上jstl使用标签

4.jsp乱码如何解决，几种解决方案。

一、JSP页面显示乱码

<%@ page contentType=”text/html; charset=gb2312″%>

二、表单提交中文时出现乱码

request.seCharacterEncoding(rgb2312″)

//对请求进行统一编码

三、数据库连接出现乱码

要涉及中文的地方全部是乱码，解决办法：在数据库的数据库URL中加上useUnicode=true&characterEncoding=GBK就OK了。

四、通过过滤器完成

五、在server.xml中的设置编码格式

5.页面间对象传递的方法

request，session，application，cookie等

request.setAttribute(key，value)

session.setAttribute(key，value)

application.setAttribute(key，value)

6.有几种方式将页面的值传到后台

7.如何防止form表单的重复提交

4）servlet面试题

1.servlet的生命周期

2.过滤器Filter的作用及配置

3.servlet中获取页面传过来的值的方法有几种

5)其它面试问题

1.接口开发需要注意的问题

## 十、数据库

1. **drop、truncate、 delete区别**

**2.varchar和char的使用场景?**

3.什么是内连接，外链接，交叉链接和笛卡尔积。

4.什么是索引，索引的作用，和缺点。什么时候用索引，什么时候不用索引。

5.索引有B+索引和hash索引索引及其应用。



6什么是SQL注入，如何防止SQL注入

7.什么是事务，事务的特点。

8.事务的隔离级别，每个隔离级别会引发的问题，MYSQL默认是哪个隔离级别。

9.事务的传播行为

10.mysql常见的三种存储引擎及其区别。

11.sql的优化策略有哪些。

12.mysql都有什么锁

13.mysql单机的QPS访问量为2000，高并发的解决方案有哪些。

<https://segmentfault.com/a/1190000013517914>

## 十一、SSM

1.spring在SSM框架中的作用是什么？

2.spring框架的优点是什么？

3.BeanFactory和ApplicationContext有什么区别？

4.spring中的哪个类的哪个方法可用于获取bean

5.Spring 支持的事务管理类型有哪些？你在项目中使用哪种方式？

 6.什么是Spring 的AOP,AOP的工作原理是什么？

7.什么是IOC/DI，

8.简述Spring中常用注解？

9.Spring Bean的作用域之间有什么区别？

10.简述spring的三种装配方式是如何把对象装配到IOC容器中的？

11.springDI的三种方式。

12.说出spring框架使用的至少4种设计模式，分别举例如何使用的。

1.简述对mybatis的理解

2.什么是orm和jdbc的区别是什么？

3.mybatis的编程步骤是什么？

4.mybatis的mapper接口调用时有哪些要求？

5.通常一个mapper映射文件一个dao接口，请问这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法可以重载吗？为啥？

6.mybatis中${value}与#{} 的区别是什么？

7.获取上一次自动生成的主键值？

8.mybatis实现一对一有几种方式，分别是如何实现的；

9.mybatis实现一对多有几种方式，分别是如何实现的。

10.mybatis的动态SQL的理解，都有哪些动态SQL标签，如何使用。

11.Mybatis是否支持延迟加载？如果支持，它的实现原理是什么

12.Mybatis能否实现批处理，如何实现？（BatchExecutor）

1.springMVC的请求流程？

2.什么是MVC，传统MVC框架（jsp+javaBean+servlet）存在的问题？

3.如何解决post和get请求的乱码问题。

4.说出SpringMVC常用的5个注解？

5.如何使用 SpringMVC完成JSON操作？

6.SpringMvc的控制器是不是单例模式,如果是,有什么问题,

7.@RequestMapping注解在类上和方法上的作用，简述如何把一个url（/usercontroller/adduser）请求映射到特定的方法中

8.@RequestMapping注解的参数有哪些？分别有什么作用？

9.从jsp传递参数到java后台，有哪些方式？

10.springMVC的返回值有哪些类型？分别代表什么意思？

11.springMVC返回数据到jsp页面的方法有哪些？

## 十二、springBoot

1.什么是springBoot， springBoot的优点有哪些？

2.springboot的核心注解有哪些？分别是什么功能？

3.springboot的自动配置原理

4.spring boot提供了哪些starter project options？

5.我们为什么需要spring-boot-maven-plugin，他提供了什么功能？

6.springboot的配置文件的形式和位置。

7.springboot集成mybatis的过程

8.@RequestMapping和@GetMappping的区别

9.Spring Boot 有哪几种读取配置的方式？

10.Spring Boot 支持哪些日志框架？推荐和默认的日志框架是哪个？

11.如何理解springboot中的starters？

12.springboot如何实现异常处理？

13.如何重新加载springboot上面的更改，而无需重新启动服务器？

14.springboot中的监视器是什么，有什么作用。

15.运行springboot有哪几种方式，至少选择两种？

16. 如何让一个springboot的项目兼容公司已前的老spring项目？