1. **JavaIO-File类**

1.File类用来表示一个文件或目录:

访问文件或目录的属性(名字，大小，最后修改时间等)

操作文件或目录(创建，删除)

无法访问文件的数据

2.File类是对文件系统的映射，

new File("D\\A\\JAVA")只是在内存上创建的映射对象， 而不会在硬盘上创建文件

所以fi.getName()获取的是内存映射,硬盘无真实文件亦可。

File fi = new File("D:"+File.separator+"A"+File.separator+"F");"D:"中冒号必须存在

1)"."当前目录,在eclipse中运行时，表示当前项目的根目录，或者只写名字("xx")

2)分隔符不同系统:windows:"\",例如:d:\xxx.txt linux:"/"

File.separator替代分隔符，separator跨平台通用

3.获取属性信息的相关方法:

1)获取文件名:String getName()

2)获取文件/目录大小:long length() ；目录大小默认为0；文件大小（字节量）

3)查看文件是否真实存在:boolean **fir.exists()**

4)查看是否表示的是文件:boolean **fir.isFile()**

5)查看是否表示的是目录:boolean **fir.isDirectory()**

system.out.println(fi); //File类对象的路径可以直接输出，内部重写了toString方法

4.使用File创建一个新目录或文件

1)使用File删除一个目录/文件：**file.delete()**;

前提是该目录必须是一个空目录(目录中没有任何内容)，否则删不掉

2)使用File创建一个目录：**file.mkdir()**;

使用File创建一个文件：**file.createNewFile()**;

3)创建多级目录：**file.mkdirs();**

5.File操作：

1)File fir = new File("."+File.separator); 指定目录

File[] file = **fir.listFiles();** **遍历目录**，获取当前项目根目录下的所有子项

2)**FileFilter**:文件过滤器

使用文件过滤器来遍历一个目录下满足条件的部分子项。是一个接口，本身不能实例化，我们需要定义一个类【或匿名内部类】实现该接口，并**重写其定义的accept方法来定义过滤规则。**

accept方法用于定义过滤要求，当给定的file对象满足要求就返回true即可。

带过滤器filter的遍历

public static void listFilesFilter(File dir){

**FileFilter ft = new FileFilter()**{

public boolean **accept(**File dir){

return dir.getName().startsWith("."); //根据文件类型过滤

}

};

File[] fl = **dir.listFiles(ft);** //重载的listFiles方法:

for(File fisub : fl ){

System.out.println("子项 ：" + fisub.getName());

System.out.println("子项路径 ：" + fisub);

}

}

3)删除多级目录:

删除当前这个目录必须先将其所有子项删除才可以。

1. **JavaIO-RandomAccessFile类**

**1.面向开发者：**

RandomAccessFile java**提供的对文件数据的访问类，既可以读文件，也可以写文件。**

**支持随机访问文件，可以访问文件的任意位置**

RandomAccessFile raf = **new** RandomAccessFile(

**new** File(文件路径), "rw");

2.文件指针，raf.getFilePointer() 返回long

**raf.seek(0); 移动指针至首位**

**raf.seek(int x);是直接跳到文件的指定位置,与当前位置无关**

**raf.skipBytes(12)**;  是在当前位置的基础上再跳过指定字节数

指针的起始位置为0，每读写一个字节+1’

文件长度：raf.length()，.length返回值为long

1. 有两种模式"rw"(读写) "r"（只读)

从文件中当前指针位置处读写一个字节,并以int形式返回。该int值只有"低八位"有效。若返回的int值为-1,表示读取到文件末尾。

重复写入会覆盖原有字节数据，若比上一次少，只覆盖少的部分，其余仍在。若多全覆盖

EOFException的异常EOF（end of file)读取到文件末尾

File file = new File("."+File.separator+"JAVA.txt");//在内存上创建的映射对象

RandomAccessFile raf = new RandomAccessFile(file, "rw");

**//若文件不存在会自动创建，目录不存在报错**

void -raf.write(); 一次写一个字节

raf.writeInt(9);//一次写4字节

raf.writeLong(999L); //一次写8字节

raf.writeDouble(521.521);//一次写8字节

String str = “xxx”;

byte[] buf=str.getBytes("gbk"); //字符串转换为GBK编码的字节数组

raf.write(byte[] buf); //将str的byte数组批量写入raf对应的文件

raf.read()-int--->读一个字节

raf.readInt()-int---> 读4个字节的Int整数

raf.readLong()-long--->读8个字节的long

raf.readDouble()-double--->读8个字节的double

指针的起始位置为0，每读写一个字节位置+1

byte[] buf = new byte[(int)raf.length()]; .length返回值为long,所以强转int

raf.read(buf)-int //批量读取raf对应file文件的byte

一次性读取给定文件的字节数组总长度的字节量，并将这些字节顺序的存入到给定的字节数组中。返回值为实际读取到的字节量，若返回值为-1，则表示读取到文件末尾了。

**raf.close():**

使用RAF读写完文件后，一定要记得close,释放底层资源（Oracle官方说明）

**4.copy文件：**

**思路:创建两个RandomAccessFile，一个用来从原文件读取，另一个用来写到目标文件中。**

RandomAccessFile src= new RandomAccessFile("src.txt","r");

RandomAccessFile desc= new RandomAccessFile("desc.txt","rw");

**批量读写，创建一个byte数组作为缓冲区**

**byte**[] buf=**new** **byte**[1024]; //1K缓冲

**int** len =-1; // 保存的是读取字节的个数

**while**((len=src.read(buf))!=-1){

desc.write(buf, 0, len);

void write(byte[] d, int offset, int len)

将给定的字节数组中从offset指定位置开始的字节连续len个一起写出

}

desc.close();

src.close();

1. **JavaIO-InputStream/OutputStream**

**1.面向应用程序**

InputStream/OutputStream抽象类，应用程序读取/写出数据的方式

分类：字节流、字符流

**2.字节流:**

把一个任意文件作为字节流进行读写操作

**FileIOutputStream BufferedIOutputStream**

// fis.read(byte[] buf , int start, int size)

// fos.write(byte[] buf, int start, int size)

创建方法：

FileInputStream fis = new FileInputStream(src);

FileOutputStream fos = new FileOutputStream(dest，true);

**true追加操作，不修改原数据，直接在后面追加**

无true：若文件已存在，完全覆盖，删除原有重新写

BufferedInputStream bis = new BufferedInputStream(

new FileInputStream(srcFile));

BufferedOutputStream bos = new BufferedOutputStream(

new FileOutputStream(destFile));

**3.字符流:**

操作的是文本文件

字符流的底层依然是字节流FileInputStream

**InputStreamReader 完成byte流解析为char流,按照编码解析**

**OutputStreamWriter 提供char流到byte流，按照编码处理**

创建方法

InputStreamReader isr = new InputStreamReader(

new FileInputStream(src),"gbk");

OutputStreamWriter osw = new OutputStreamWriter(

new FileOutputStream(dest，true),"gbk")

要写文件本身的编码格式

**带缓冲字符流**

BufferedReader br = new BufferedReader(

new InputStreamReader(

new FileInputStream(src)，"gbk"));

BufferedWriter bw = new BufferedWriter(

new OutputStreamWriter(

new FileOutputStream(dest,true)，"gbk"));

**PrintWriter** pw = new PrintWriter(

new FileOutputStream(dest),true);

**PrintWriter内部嵌套了BufferedWriter，所以具有缓冲功能。**

**还提供了自动刷新缓冲区autoflush()功能，下方无需pw.flush()，每当调用println方法写出内容后，就会自动调用flush**

String line; 以行为单位读写字符串

**while**((line = br.readLine())!=**null**){

System.***out***.println(line);//一次读一行，并不能识别换行,使用ln换行

/\*需单独写出换行操作

bw.write(line);

bw.newLine();//换行操作

bw.flush();

pw.println(line);//一次读一行，并不能识别换行,使用ln换行的

}

br.close(); close方法内部会在关闭前自动调用flush一次。

//bw.close();

pw.close();

**4.键盘输入字符流**

**获取键盘输入，和scanner一样：InputStream in = System.in;**

BufferedReader br = **new** BufferedReader(

**new** InputStreamReader(System.***in***));

**while**(**true**){

String line= br.readLine();

System.***out***.println(line);

}

**5对象的反序列化和序列化**

**ObjectInputStream/ObjectOutputStream**

第一步：定义类要序列化的属性，和无需序列化的属性

**public** **class** Student **implements** Serializable{

**private** **static** **final** **long** ***serialVersionUID***=2L;//版本序列号

**private** String stuid; //string类实现了序列化接口

**private** String stuname;

**private** **transient** **int** stuage;

**该元素不会进行jvm默认的序列化。加transient原因：**

**1.并不是所有的属性都要进行序列化，特别是把对象在网络中传输时，无用的属性序列**化会占用资源，影响传输效率；

**第二步：1.创建对象，并序列化**

Student stu = **new** Student();

stu.setStuid("9");

stu.setStuname("Clear Cheng");

stu.setStuage(22);

//把内存上的数据写入磁盘中做长久保存的过程称为:持久化，无文件自动创建

FileOutputStream fos=**new** FileOutputStream("person.obj");

ObjectOutputStream oos = **new** ObjectOutputStream(fos);

oos.writeObject(stu);

oos.flush();

oos.close();

**第二步：对象的反序列化并输出信息至控制台**

ObjectInputStream ois = **new** ObjectInputStream(

**new** FileInputStream("person.obj"));

Student stu = (Student)ois.**readObject**();

System.***out***.println(stu); //类中重写了toString

ois.close();

子类的序列化

\* 一个类实现了序列化接口，那么其子类都可以实现序列化

**5.flush() 是把缓冲区的数据强行输出主要用在IO中，即清空缓冲区数据。**

一般在读写流(stream)的时候，数据是先被读到了内存中，再把数据写到文件中，

当数据读完的时候不代表你的数据已经写完了，因为还有一部分有可能会留在内存这个缓冲区中。这时如果调用了close()方法关闭了读写流，那么这部分数据就会丢失，所以应该在关闭读写流之前先flush()。

1. **线程**

**5种状态**：新建-就绪-运行-阻塞-死亡

**改变线程状态的三种方法**：sleep yield join

线程作用：使多任务并发工作

进程(动态)process：是程序的执行过程，持有资源和多个线程，当操作系统创建一个进程后，该进程会自动申请一个主线程

**进程是资源和线程的载体，是操作系统中最小的执行单元**

**同一进程中有多个线程，线程共享进程的资源**

使用Thread建立的线程默认情况下是用户线程，在进程中，只要有一个前台线程未退出，进程就不会终止，只要所有的前台线程都退出，进程就会自动终止

当执行程序时，首先操作系统会启动一个进程来运行jvm , jvm启动后会启动一个用户线程来执行main方法

程序运行过程中会创建其他线程，执行速度如果缓慢，main线程先执行完毕

1.线程有两种创建方式：

第一种：继承Thread类并重写run方法

**class** T1 **extends** Thread{

**public** **void** run(){

}

}

Thread t1 = **new** T1();

t1.start();

第二种：实现Runnable接口，并重写run方法【建议使用，单继承多实现】

**class** R1 **implements** Runnable{

**public** **void** run(){

}

}

Thread t1 = **new** Thread(**new** R1());

t1.start();

1. 使用匿名内部类来完成两种线程的创建方式

第一种：Thread t1 = **new** Thread(){

**public** **void** run(){

}

};

t1.start();

第二种：Thread t2 = **new** Thread(**new** Runnable(){

**public** **void** run(){

}

});

t2.start();

3.注意：

**1)启动线程不要调用run方法，而是应当调用start()方法。只有这样做该线程才会被**纳入线程调度，具有并发运行的能力。

4.相关方法

**Thread.currentThread():静态方法,该方法用于runnable接口定义的run（）方法中**获取运行当前代码片段的线程，thread可以直接调用

Thread t = Thread.*currentThread*();

long getId():返回该线程的标识符

String getName():返回该线程的名称

int getPriority():返回线程的优先级

Thread.state getState():获取线程的状态

boolean isDaemon():测试线程是否为守护线程

线程优先级：

**线程的优先级有10个等级，1最低，10最高，5默认，也可以使用Thread提供的常**量：

min.setPriority(Thread.***MIN\_PRIORITY***);

MIN\_PRIORITY:最低优先级，对应数字：1

MAX\_PRIORITY:最高优先级，对应数字：10

NORM\_PRIORITY:默认优先级，对应数字：5

原则上，优先级高的线程被分配CPU时间片的次数多

守护线程：

又叫做后台线程。使用上与前台线程没有区别，唯一在结束上有区别。

进程结束时，无论后台线程是否还在运行都要强制结束。

thread.**setDaemon**(true);//设置为守护线程

GC就是运行在一个守护线程上的。

**Thread 提供了一个静态阻塞方法sleep**

thread.*sleep*(1000);

用于让运行sleep的线程进入阻塞状态指定毫秒。当超时后，该线程会自动回到Runnable状态等待分配CPU时间片再次运行。

**Thread 提供了一个阻塞方法join**

Join方法的调用会使所有线程进入等待状态，直到调用join方法的线程执行完毕

**yield静态方法使当前线程主动让出当次CPU时间片回到Runnable状态，等待分配时间**片，让出之后执行哪一条线程不一定

**Java的一个关键字volatile修饰属性：**

保证了线程可以正确的读取其他线程写入的值

不添加会因为可见性问题出错，研究JMM内存模型，happen-before原则

**Object上定义了两个方法wait【等待阻塞】，notify**

当一个线程调用了一个对象的wait方法后，该线程就进入阻塞状态，直到这个对象

的notify方法被调用，该线程才会解除阻塞。

使用这两个方法完成线程同步操作的即时性更强，join必须等到线程工作完毕才可以，

即时性相对较差。

API文档上有说明：当我们调用某个对象的wait或者notify方法时，该对象需要使用synchronized块进行同步，而锁定对象就是wait与notify所属的对象。否则在实际运行过程中会发生异常。

**synchronized关键字是java中的同步锁来支持原子性【互斥性】**

原因：多线程并发读写同一个临界资源时会发生“线程并发安全问题”。

常见的临界资源：多线程共享实例变量，多线程共享静态公共变量

若解决线程安全问题，需要将异步操作变为同步操作

异步操作：多线程并发的操作，各干各的

同步操作：有先后顺序的操作，你干完我再干

synchronized关键字可以修饰方法

当修饰了方法，该方法就是“同步”方法，多个线程不能同时访问方法内部。这就解决了多个线程同时操作统一数据导致的混乱问题。

synchronized若修饰的是方法，上锁对象则是当前方法所属的对象

在方法上使用synchronized虽然可以起到同步的效果，但是由于整个方法所有代码变成同步，对于并发执行的效率来讲会有所降低。除非整个方法所有代码都需要这样做，否则我们应当使用synchronized块的形式，将需要同步的代码括起来减小同步范围，这样在保证安全的前提下也提高了并发执行的效率。

synchronized体现的互斥性

当一个对象的两个方法都被synchronized修饰，则两个方法就有了互斥性。

当synchronized块包含了两段不同的代码，而这两块锁对象相同，则这两块代码也有互斥性。

static静态方法上被synchronized修饰后，该静态方法变为同步的。由于静态方法与对象无关，和类相关，所以无论调用哪个对象的这个静态方法都具有同步性。

当然，静态方法应当使用类名.方法()的形式调用，而不应当使用对象调用。静态方法锁的是类，虚拟机在加载每个类时都会创建一个Class实例来描述它

将集合与Map转换为线程安全的

**ArrayList,LinkedList,HashSet,HashMap都不是线程安全的。多线程情况下操作他们**都会产生线程安全问题。

**可以使用Collections这个集合的工具类将上述的数据结构转换为线程安全的。**

1)List<String> list=new ArrayList<String>();

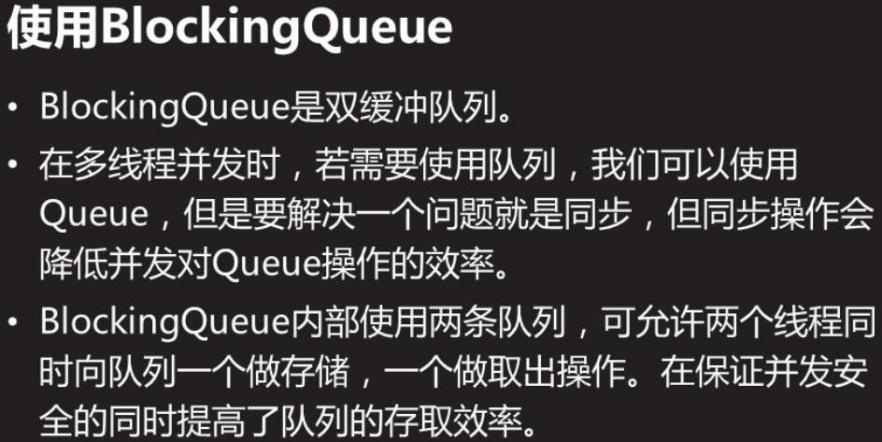
//该静态方法用于将给定的List集合转换为一个线程安全的List

**list=Collections.synchronizedList(list);//转过来为线程安全的**

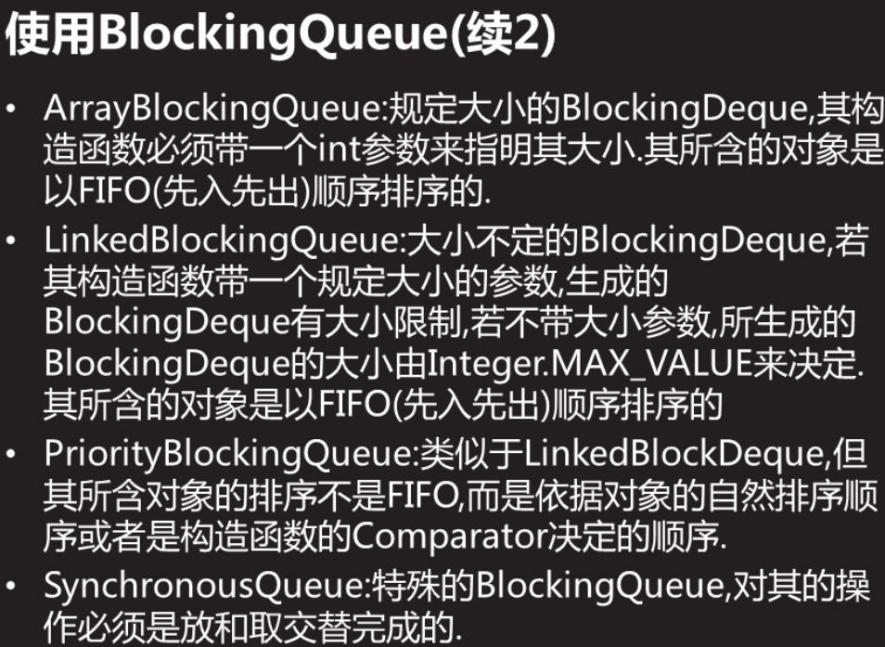
2)Map<String,Integer > map=new HashMap<String,Integer> ();

//将给定的Map转换为线程安全的

map=Collections.synchronizedMap(map);//转过来为线程安全的







1. **Socket**

**网络基础知识**

计算机之间的网络通信条件：

TCP/IP协议(通信语言) IP地址 端口号(不同应用程序的通信区分)

TCP/IP协议是以TPC和IP为基础的不同层次上多个协议的集合

TCP协议：传输控制协议 IP协议：互联网协议

**TCP/IP协议五层模型：**

1.物理层：网线，双绞线，网卡

2.数据链路层

3.网络层

4.传输层 TCP/IP协议实际所在

5.应用层 HTTP超文本传输协议 FTP文件传输协议 SMTP简单邮件传输协议

IP：计算机唯一标识

IPV4：32位的二进制 Talnet远程登录服务

端口号：区分不同的应用程序 0-65535， 0-1023系统保留

常用端口号：HTTP:80 FTP:21 Talnet:21

**InetAddress类**

用于标识网络上的硬件资源(主要用来标识IP地址的相关信息)

无构造方法，提供了静态方法返回一个InetAddress实例

InetAddress address = InetAddress.*getLocalHost*();

address.getHostName() 计算机名

address.getHostAddress() IP地址

**URL类(java.net)**

**统一资源定位符**，表示internet上某一资源的地址，是指向互联网资源的指针，通过 URL可以直接读取或写入网络上的数据

组成：协议名称 ： 资源名称

构造：字符串/协议 主机名 端口号 /

通过URL读取网页内容：

1.通过URL的openStream()方法可以得到指定资源的输入流

2.通过输入流可以读取访问网络上的数据

URL url=**new** URL("http://www.baidu.com");

InputStream is = url.openStream();

InputStreamReader isr = **new** InputStreamReader(is, "utf-8");

BufferedReader br = **new** BufferedReader(isr);

**socket（套接字）**

**IP地址+端口号组成，是网络运行的程序之间双向通信链路的终结点，**是TCP/UDP的基础

**Socket类**：

**Socket socket=new Socket("localhost",8088);**

重要方法：getinputstream/getoutputstream

shutdowninput/shutdownoutput

PrintWriter pw=**new** PrintWriter(

**new** OutputStreamWriter( socket.getOutputStream(),"gbk"),**true**);

BufferedReader br=**new** BufferedReader(

**new** InputStreamReader

(socket.getInputStream(),"gbk"));

**TCP协议**

**面向连接 可靠的 有序的，以字节流的方式发送数据**

服务端

**ServerSocket** serversocket=**new** ServerSocket(8088);

功能：

1)负责申请端口，以便客户端通过该端口链接到我们的服务器。

2)第二个功能就是接收客户端的连接，并生成与该客户端通讯的Socket。

方法：

**Socket accept():该方法是以个阻塞方法，会一直监听其打开的8088端口，等待一**个客户端连接，一旦连接，该方法会返回与该客户端交互的Socket。

客户端

**Socket socket=new Socket("localhost",8088);**

1：远程计算机IP地址

2：服务端应用程序申请的服务端口创建的过程就是连接的过程，只有与服务端建

立了连接才会创建成功，否则会抛出异常。

**UDP(用户数据报协议)**

**无连接 不可靠 无序的 ，速度相对较快，使用多 ，以数据报作为数据传输的载体**

Dategram:使用UDP协议，将数据保存在数据报中，通过网络进行通信

DategramPacket：表示数据报包

DategramSocket:进行端到端通信的类

InetAddress address=InetAddress.*getByName*("localhost");

**DatagramPacket** packet=**new** DatagramPacket(data, data.length, address, port);

**DatagramSocket** socket=**new** DatagramSocket();

socket.**send**(packet);

概要：以UDP进行数据传输时，会将数据封装成Dategram数据报，在数据报中指明要达到的socket

注意：对于同一个socket，如果关闭了输出流，与之相关联的socket也会被关闭

所以一般不关闭流，直接关闭socket

**七．XML**

用于简化数据的存储和共享。

存储:树形结构-倒着长得树(根节点-子节点-子节点)

XML解析概述:

基础方法：DOM SAX 官方提供

DOM 与平台无关的解析方式，各平台通用

SAX 基于事件驱动的解析方式

目的：获取节点名 节点值 属性名 属性值 ,java得到xml文件的所有数据

属性也是节点类型的一种，<></>开始到结束中的所有内容都会被看成子节点

扩展方法：JDOM DOM4J 只有java中可以使用的解析方法

**DOM：基于树，一次性将整个xml文件加载到内存，并形成DOM树，然后逐个解 析，**如果xml非常大，耗内存大影响性能

优点：形成了树结构，直观好理解，代码容易编写

解析过程中树结构保留在内存中，方便修改

**SAX：基于事件的一种解析，从上到下顺序执行，逐行扫描文档，一边扫描一边解析，** 判断触发Handler中的哪一个方法，需要重写Handler方法

优点：内存占用小，适用于只需要处理xml中数据时

可以在解析文档的任意时刻停止，可以立即开始，速度快，没有内存压力

缺点：不能对节点做修改，很难同时访问同一个xml中的多处不同数据

**JDOM：仅适用具体类而不适用接口，API大量使用了collection类**

**DOM4J：JDOM的一种智能分支，合并了许多超出基本xml文档表示的功能**

DOM4J 和 JDOM 都是基于底层API ，其实就是 DOM 和 SAX

优点：DOM4J使用接口和抽象基本类方法，是一个优秀的java xml api

性能优异，灵活性好，功能强大，易使用，开放源码

把xml文件在内存中构造树形结构，可以遍历和修改节点

三大框架Hibernate中也是使用DOM4J解析xml文件

缺点： 如果文件比较大，内存有压力，解析的时间会比较长