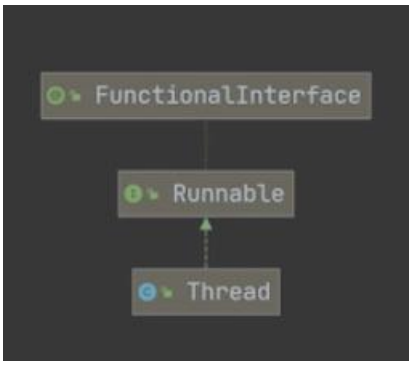
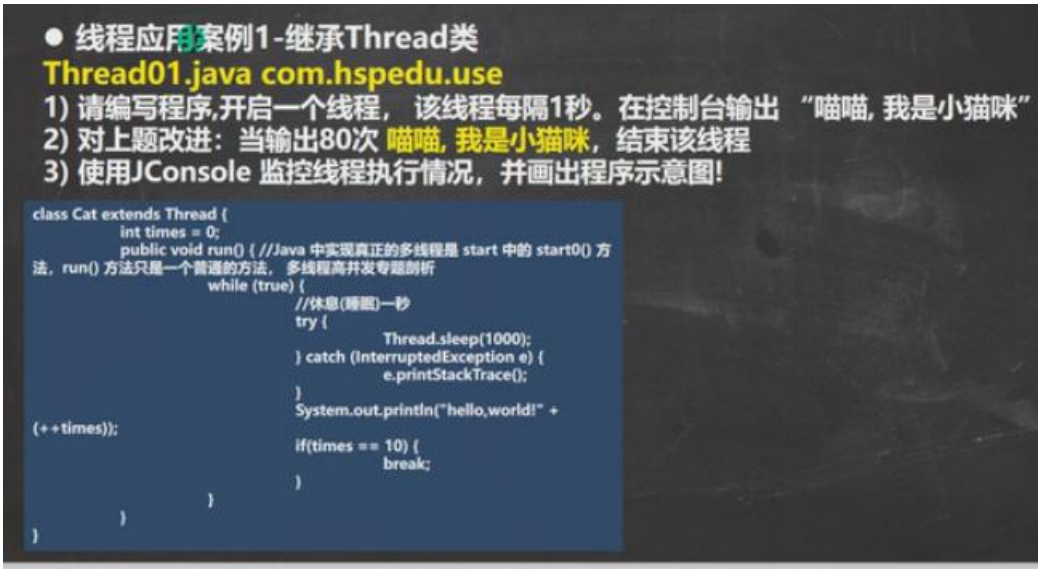
多线程基础

线程：
1. 线程由进程创建，是进程的一个实体；
2. 一个进程可以拥有多个线程
3. 单线程：同一个时刻，只允许执行一个线程
4. 多线程：同一个时刻，可以执行多个线程，比如：一个 QQ 进程，可以同时打开多个聊天窗口；一个迅雷进程，可以同时下载多个文件
5. 并发：同一时刻，多个任务交替执行，造成一种“貌似同时”的错觉，简而言之就是单核 CPU 实现的多任务就是并发
6. 并行：同一个时刻，多个任务同时执行，多核 CPU 可以实现并行

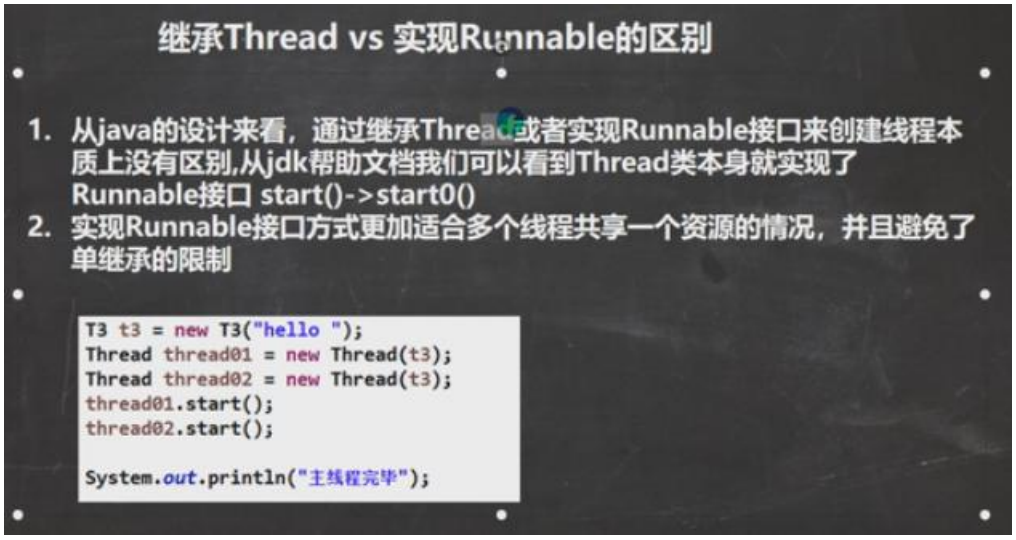
创建线程的两种方式：java 中线程使用的 2 种方式
1. 继承 Thread 类，重写 run 方法
2. 实现 Runnable 接口，重写 run 方法

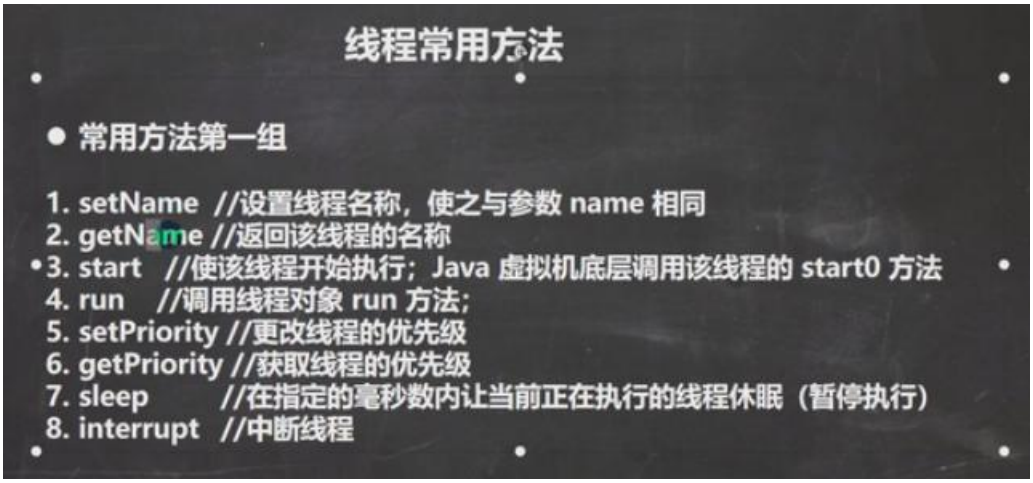




线程的基本使用 2---实现 Runnable 接口

1. Java 是单继承的，在某些情况下一个类可能已经继承了某个父类，这时再用继承 Thread 类的方法来创建线程是不太可能了
2. Java 设计者提供了通过实现 Runnable 接口来创建线程的方法





线程常用方法

- 注意事项和细节

1. start 底层会创建新的线程，调用run，run 就是一个简单的方法调用，不会启动新线程
2. 线程优先级的范围
3. interrupt，中断线程，但并没有真正的结束线程。所以一般用于中断正在休眠线程
4. sleep:线程的静态方法，使当前线程休眠

java.lang.Thread

线程常用方法

- 常用方法第二组

1. yield:线程的礼让。让出cpu，让其他线程执行，但礼让的时间不确定，所以也不一定礼让成功

2. join：线程的插队。插队的线程一旦插队成功，则肯定先执行完插入的线程所有的任务
案例：创建一个子线程，每隔1s 输出 hello, 输出 20 次, 主线程每隔1秒，输出 hi，输出 20次.要求: 两个线程同时执行，当主线程输出 5次后，就让子线程运行完毕，主线程再继续，

教育

Synchronized

- 同步具体方法-Synchronized

1. 同步代码块
```
synchronized (对象) {  // 得到对象的锁，才能操作同步代码
    // 需要被同步代码;
}
```
2. synchronized还可以放在方法声明中，表示整个方法-为同步方法

```
public synchronized void m (String name){
    //需要被同步的代码
}
```

3. 如何理解:
就好像 某小伙伴上厕所前先把门关上(上锁),完事后再出来(解锁),那么其它小伙伴就可在使用厕所了，如图:

4. 使用synchronized 解决售票问题

- 基本介绍

1. Java语言中，引入了对象互斥锁的概念，来保证共享数据操作的完整性。
2. 每个对象都对应于一个可称为“互斥锁”的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。
3. 关键字synchronized 来与对象的互斥锁联系。当某个对象用synchronized修饰时，表明该对象在任一时刻只能由一个线程访问
4. 同步的局限性：导致程序的执行效率要降低
5. 同步方法（非静态的）的锁可以是this，也可以是其他对象(要求是同一个对象)
6. 同步方法（静态的）的锁为当前类本身。

互斥锁

- 注意事项和细节

1. 同步方法如果没有使用static修饰：默认锁对象为this
2. 如果方法使用static修饰，默认锁对象:当前类.class

3. 实现的落地步骤：
- 需要先分析上锁的代码
- 选择同步代码块或同步方法
- 要求多个线程的锁对象为同一个即可！

线程的死锁

韩顺平教育

- 基本介绍

多个线程都占用了对方的锁资源，但不肯相让，导致了死锁，在编程是一定要避免死锁的发生.

- 应用案例

妈妈：你先完成作业，才让你玩手机
小明：你先让我玩手机，我才完成作业.

- 下面操作会释放锁

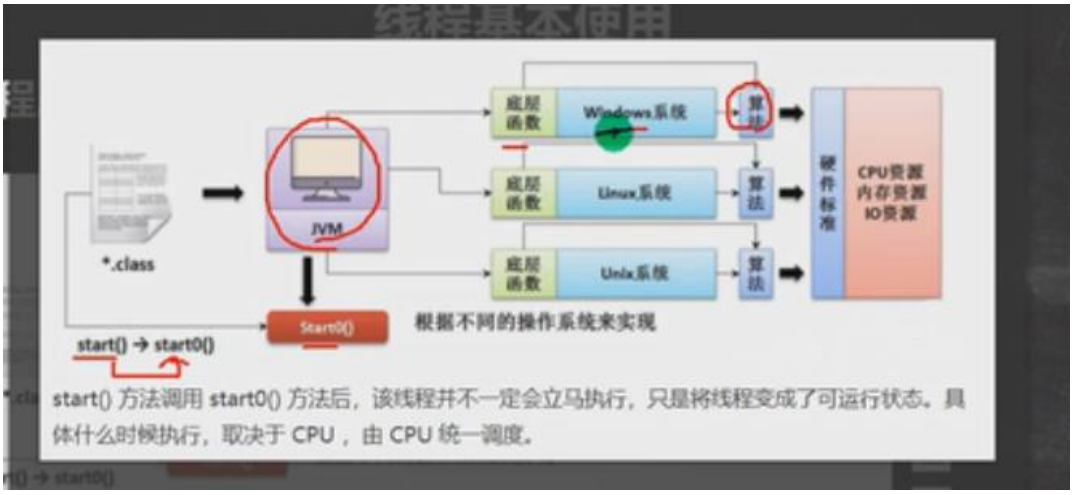
1. 当前线程的同步方法、同步代码块执行结束
   案例：上厕所，完事出来
2. 当前线程在同步代码块、同步方法中遇到break、return。
   案例：没有正常的完事，经理叫他修改bug，不得已出来

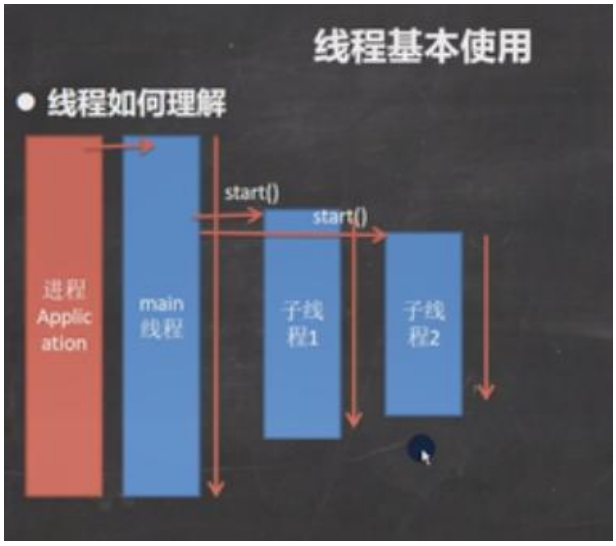
3. 当前线程在同步代码块、同步方法中出现了未处理的Error或Exception，导致异常结束
   案例：没有正常的完事，发现忘带纸，不得已出来
4. 当前线程在同步代码块、同步方法中执行了线程对象的wait()方法，当前线程暂停，并释放锁。
   案例：没有正常完事，觉得需要酝酿下，所以出来等会再进去

- 下面操作不会释放锁

1. 线程执行同步代码块或同步方法时，程序调用Thread.sleep()、Thread.yield()方法暂停当前线程的执行，不会释放锁
   案例：上厕所，太困了，在坑位上眯了一会

2. 线程执行同步代码块时，其他线程调用了该线程的suspend()方法将该线程挂起，该线程不会释放锁。
   提示：应尽量避免使用suspend()和resume()来控制线程， 方法不再推荐使用





继承Thread vs 实现Runnable的区别

1. 从java的设计来看，通过继承Thread或者实现Runnable接口来创建线程本质上没有区别,从jdk帮助文档我们可以看到Thread类本身就实现了Runnable接口
2. 实现Runnable接口方式更加适合多个线程共享一个资源的情况，并且避免了单继承的限制

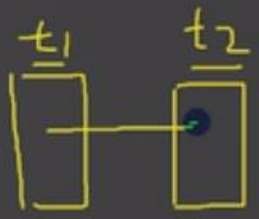
```
T3 t3 = new T3("hello ");
Thread thread01 = new Thread(t3);
Thread thread02 = new Thread(t3);
thread01.start();
thread02.start();

System.out.println("主线程完毕");
```

```
        //如果希望main线程去控制t1 线程的终止，必须可以修改 loop
        //让t1 退出run方法，从而终止 t1线程 -> 通知方式

        //让主线程休眠 10 秒，再通知 t1线程退出
        System.out.println("main线程休眠10s...");
        Thread.sleep(10 * 1000);
        t1.setLoop(false);
    }
}

class T extends Thread {
    private int count = 0;
    //设置一个控制变量
    private boolean loop = true;
    @Override
    public void run() {
```



Synchronized

- 同步具体方法-Synchronized

1. 同步代码块
   synchronized (对象) { // 得到对象的锁，才能操作同步代码
   // 需要被同步代码;
   }
2. synchronized还可以放在方法声明中，表示整个方法-为同步方法

```
public synchronized void m (String name){
        //需要被同步的代码
}
```

3. 如何理解:
   就好像 某小伙伴上厕所前先把门关上(上锁),完事后再出来(解锁),那么其它小伙伴就可在使用厕所了，如图:

4. 使用synchronized 解决售票问题

Synchronized

- 线程同步机制

1. 在多线程编程，一些敏感数据不允许被多个线程同时访问，此时就使用同步访问技术，保证数据在任何时刻，最多有一个线程访问，以保证数据的完整性。

2. 也可以这里理解：线程同步，即当有一个线程在对内存进行操作时，其他线程都不可以对这个内存地址进行操作，直到该线程完成操作， 其他线程才能对该内存地址进行操作.

互斥锁

- 基本介绍

1. Java在Java语言中，引入了对象互斥锁的概念，来保证共享数据操作的完整性。
2. 每个对象都对应于一个可称为“互斥锁”的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。
3. 关键字synchronized 来与对象的互斥锁联系。当某个对象用synchronized修饰时，表明该对象在任一时刻只能由一个线程访问
4. 同步的局限性：导致程序的执行效率要降低
5. 同步方法（非静态的）的锁可以是this，也可以是其他对象(要求是同一个对象)
6. 同步方法（静态的）的锁为当前类本身。

Synchronized

- 同步具体方法-Synchronized

1. 同步代码块

```
synchronized (对象) {  // 得到对象的锁，才能操作同步代码
    // 需要被同步代码;
}
```

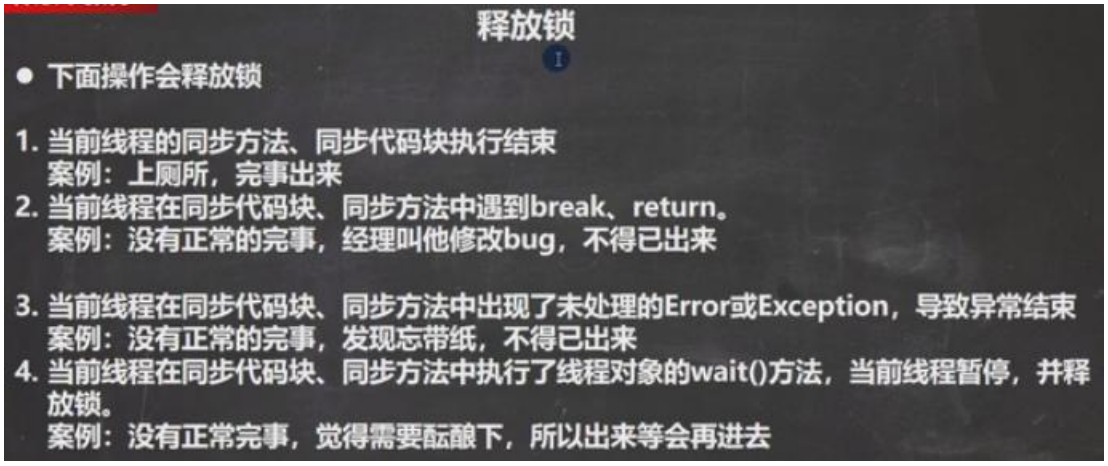
2. synchronized还可以放在方法声明中，表示整个方法-为同步方法

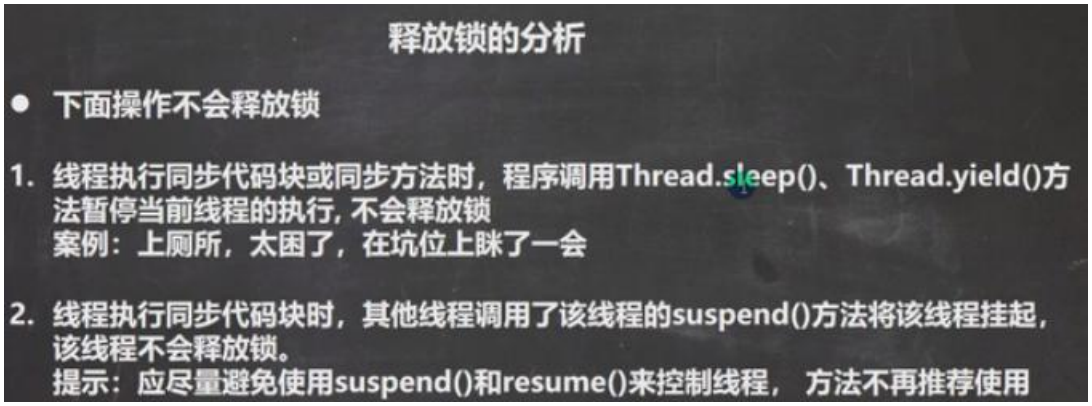
```
public synchronized void m (String name){
    //需要被同步的代码
}
```

3. 如何理解:
就好像 某小伙伴上厕所前先把门关上(上锁),完事后再出来(解锁),那么其它小伙伴就可在使用厕所了，如图:



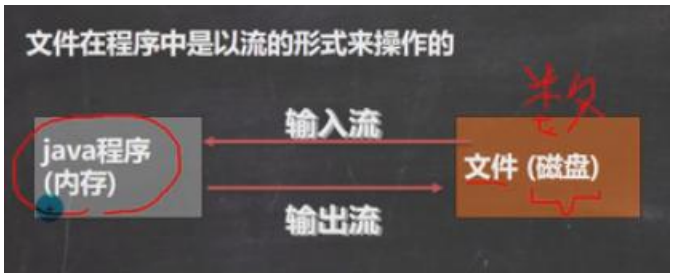
4. 使用synchronized 解决售票问题





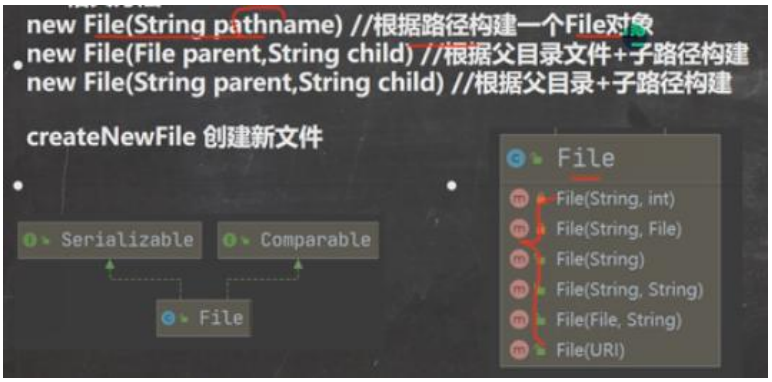
IO 流

文件：是保存数据的地方

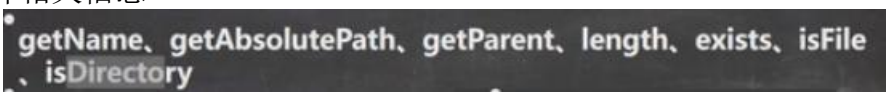


流：数据在数据源（文件）和程序（内存）之间经历的路径
输入流：数据从数据源（文件）和程序（内存）的路径
输出流：数据从程序（内存）到数据源（文件）的路径

常用的文件操作：
创建文件对象相关构造器和方法：

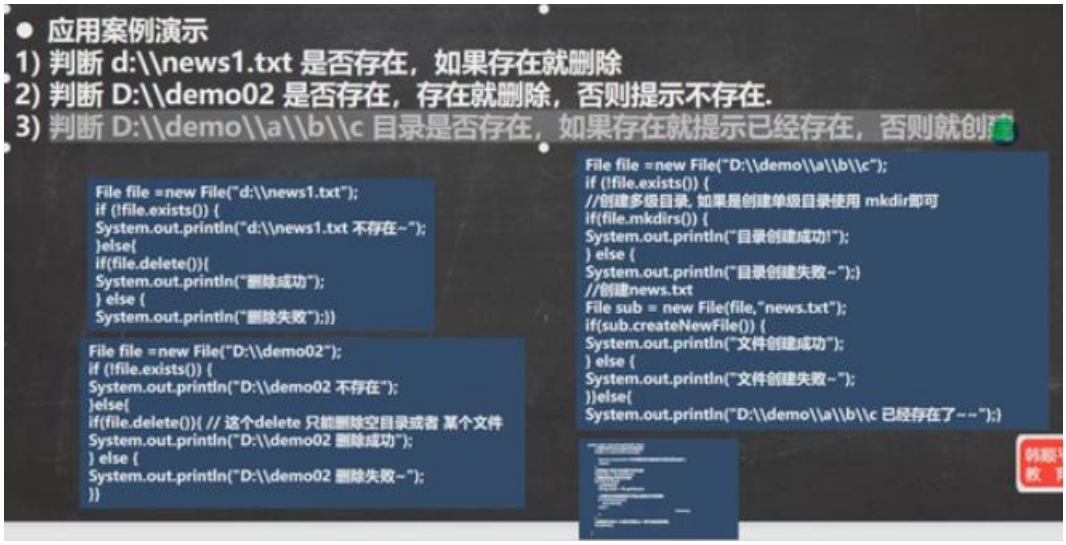


获取文件相关信息



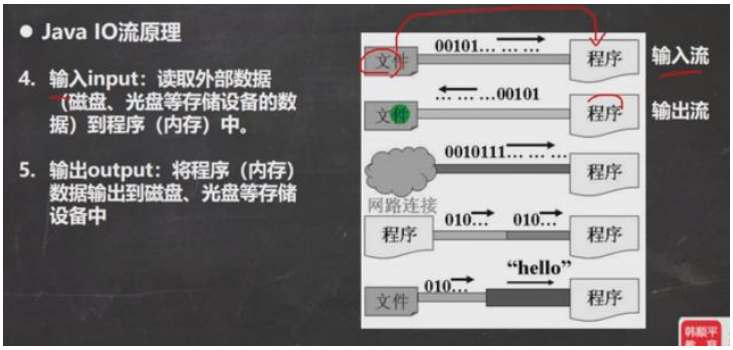
目录的操作和文件删除

mkdir()创建一级目录，mkdies()创建多级目录，delete()删除空目录或文件

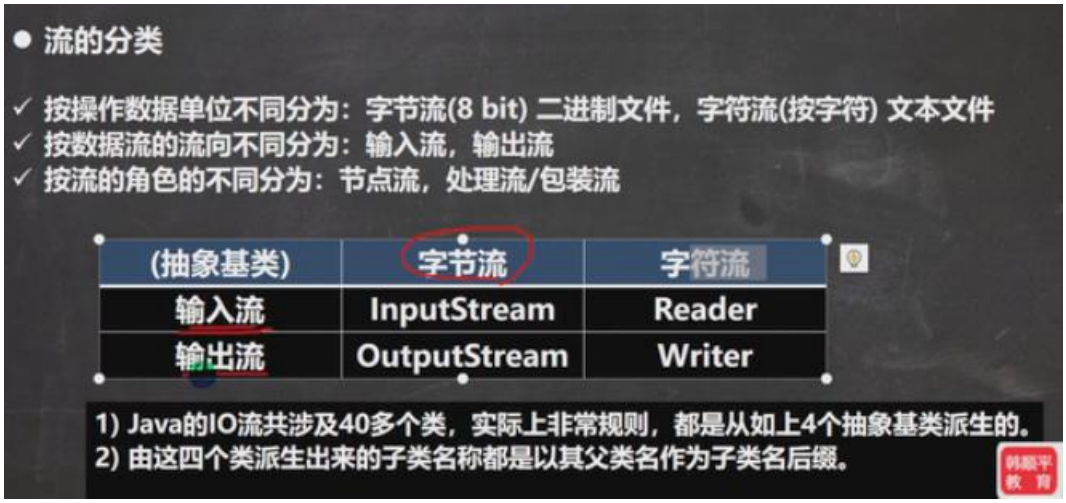


Java IO 流的原理：

1. I/O 是 Input/Output 的缩写，I/O 技术的非常实用的技术，用于处理数据传输，比如读写文件，网络通讯等；
2. Java 程序中，对于数据的输出/输出操作以“流（stream）”的方式进行
3. Java.io 包下提供了各种“流”类和接口，用以获取不同种类的数据，并通过方法输入或输出数据



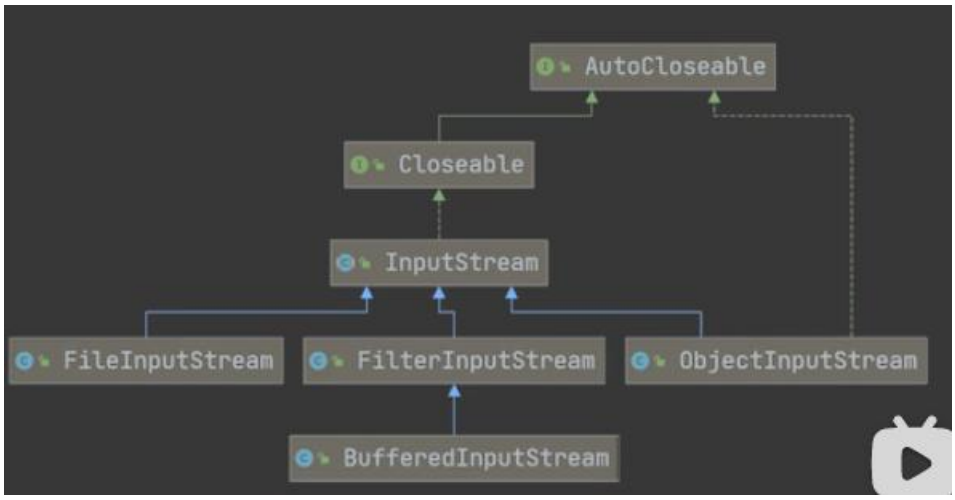
流的分类：



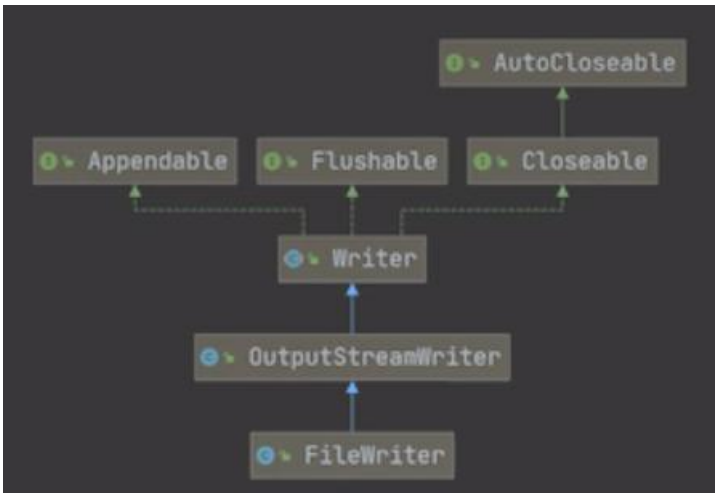
IO 流常用的类

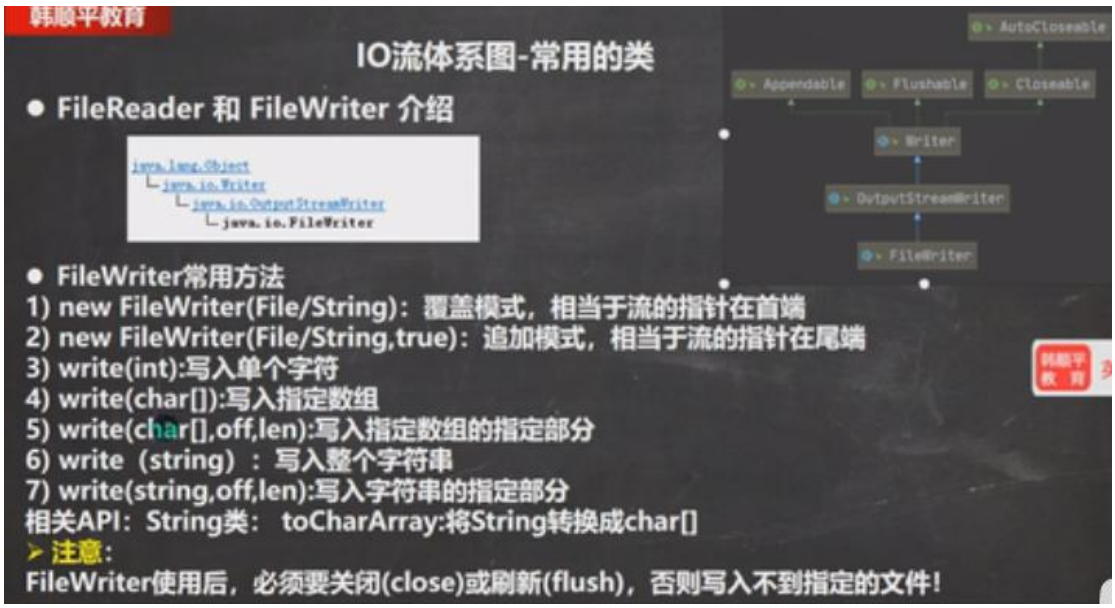
InputStream：字节输入流
InputStream 抽象类是所有类字节输入流的超类

InputStream 的常用子类：
1. FileInputStream：文件输入流
2. BufferedInputStream：缓冲字节输入流
3. ObjectInputStream：对象字节输入流



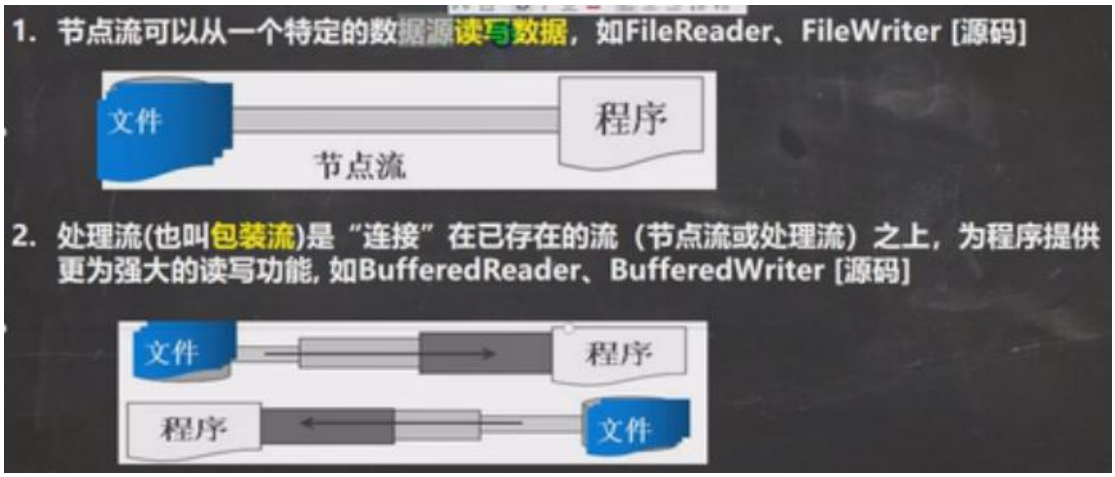
FileOutputStream 字节输出流

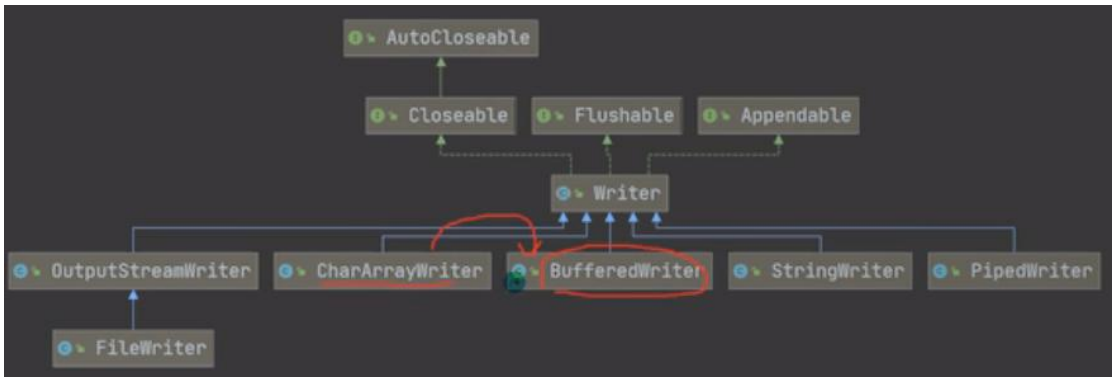




节点流和处理流

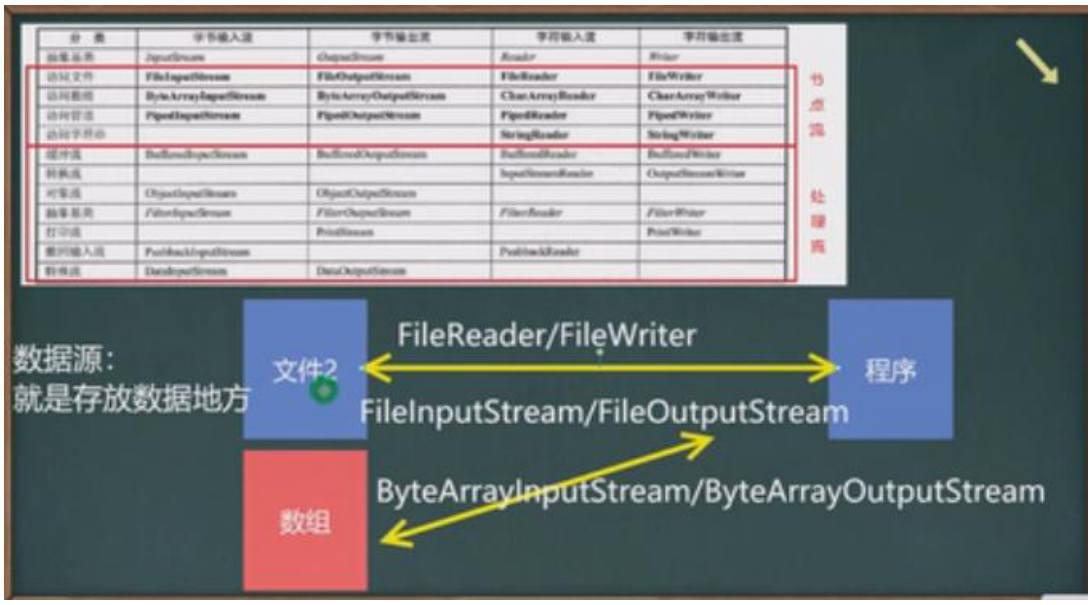
1. 节点流可以从一个特定的数据源读取数据，如：FileReader、FileWriter
2. 处理流（也叫包装流）是“连接”在已存在的流（节点流或处理流）之上，为程序提供更为强大的读写功能，如 BufferedReader、BufferedWriter



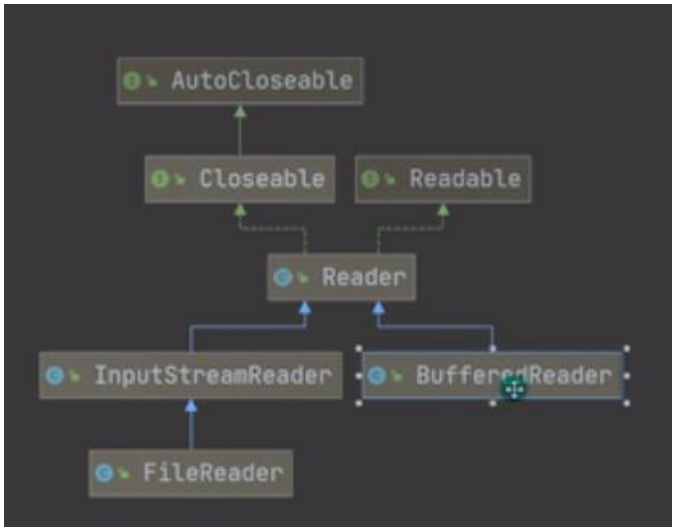


节点流和处理流的区别和联系：
（1）节点流是底层流/低级流，直接跟数据源相连接；
（2）处理流（包装流）包装节点流，既可以消除不同节点流的实现差异，也可以提供更方便的方法来完成输入和输出；
（3）处理流（也叫包装流）对节点流进行包装，使用了修饰器设计模式，不会直接与数据源相连接。

处理流的功能主要体现在以下两个方面：
（1）性能的提高：主要以增加缓冲的方式来提高输入输出的效率；
（2）操作的便捷：处理流可能提供了一系列便捷的方法来一次性输入输出大批量的数据，使用更加灵活方便。

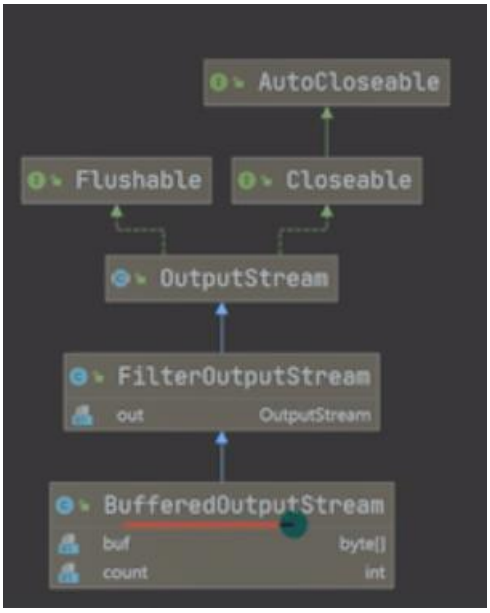


处理流----BufferedReader 和 BufferedWriter
BufferedReader 和 BufferedWriter 属于字符流，是按照字符来读取数据的；其关闭时，只需关闭外层流即可

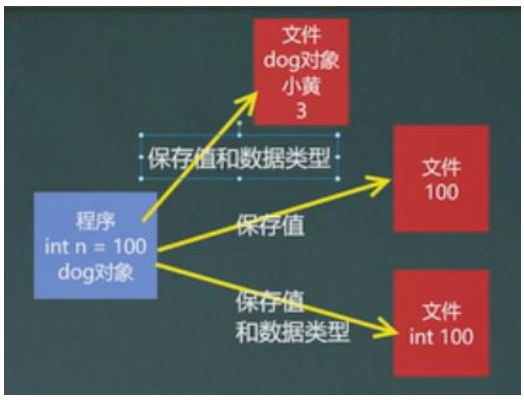


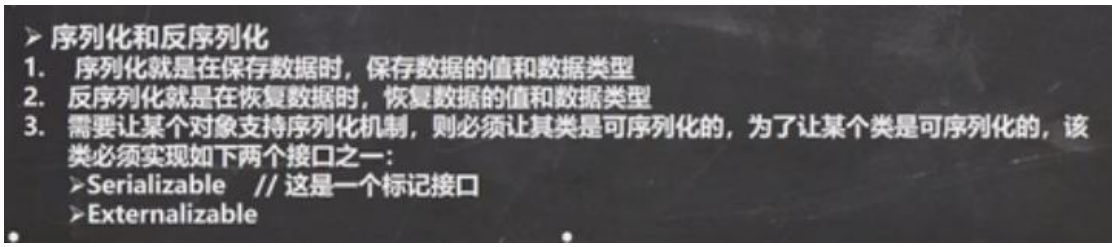
BufferedReader 和 BufferedWriter 是按照字符串操作的，不要去操作二进制文件（视频、图片等），可能会造成文件损坏

BufferedOutputStream 的类关系继承图



对象流----ObjectInputStream 和 ObjectOutputStream





序列化和反序列化的注意事项和细节：
（1）读写顺序要一致；
（2）要求序列化或反序列化的对象，需要实现 Serializable 接口；
（3）序列化的类中建议添加 SerialVersionUID，为了提高版本的兼容性；
（4）序列化对象时，默认将里面所有属性都进行了序列化，但除了 static 或 transient 修饰的成员；
（5）序列化对象时，要求里面属性的类型也要实现序列化接口；
（6）序列化具有可继承性，也就是如果某类已经实现了序列化，则它的所有子类也已经默认实现了序列化。

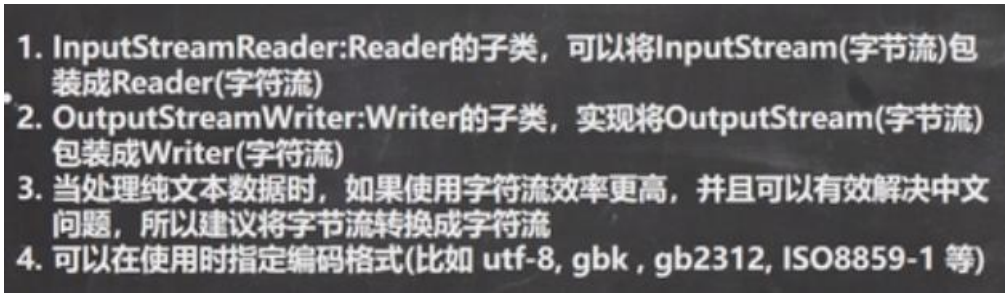
标准输入输出流：

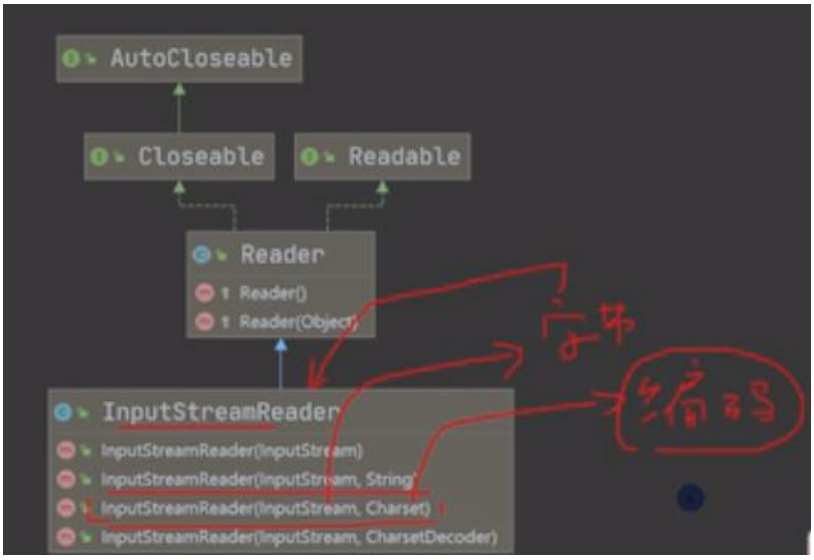
	类型	默认设备
System.in 标准输入	InputStream	键盘
System.out 标准输出	PrintStream	显示器

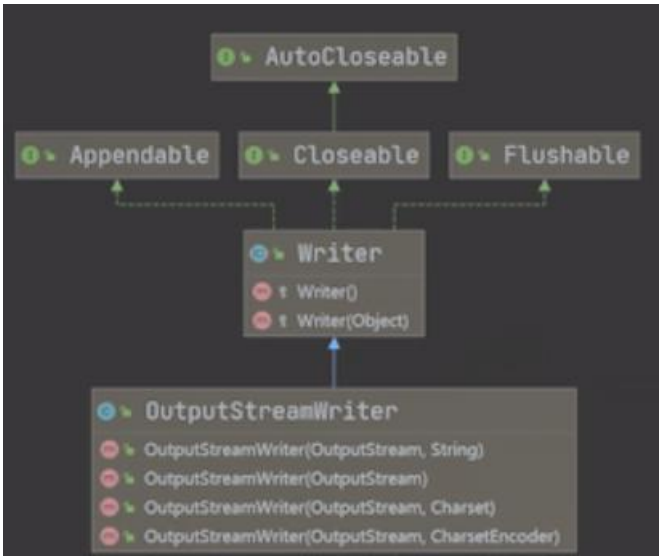
```
//System 类 的 public final static InputStream in = null;
// System.in 编译类型   InputStream
// System.in 运行类型   BufferedInputStream
// 表示的是标准输入 键盘
System.out.println(System.in.getClass());

//老韩解读
//1. System.out public final static PrintStream out = null;
//2. 编译类型 PrintStream
//3. 运行类型 PrintStream
//4. 表示标准输出 显示器
System.out.println(System.out.getClass());
```

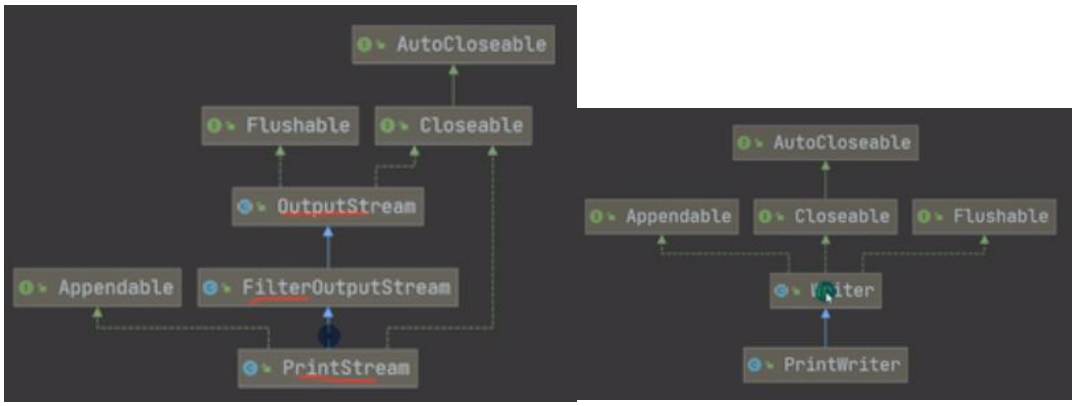
转换流 ---（字节流转换为字符流）InputStreamReader 和 OutputStreamWriter



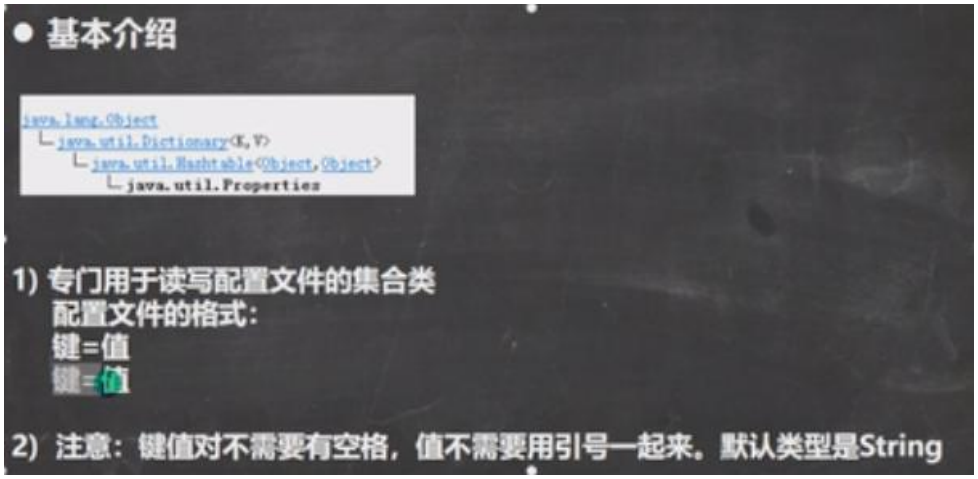


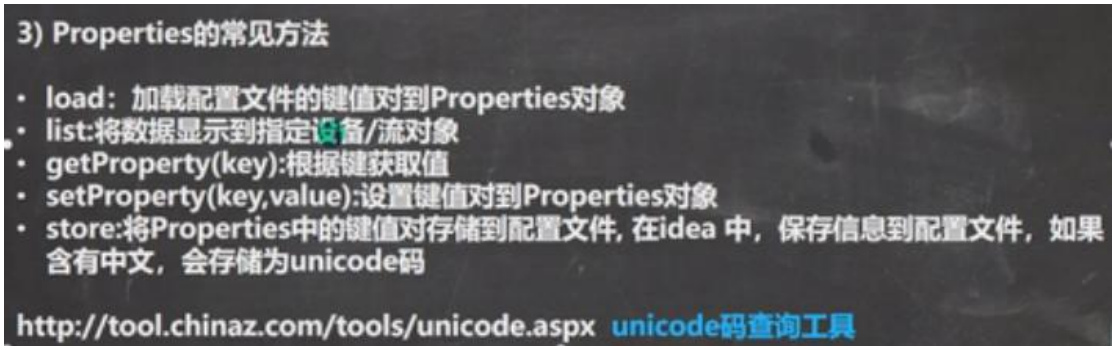


打印流----PrintStream 和 PrintWriter（打印流只有输出流，没有输入流）



Properties 类



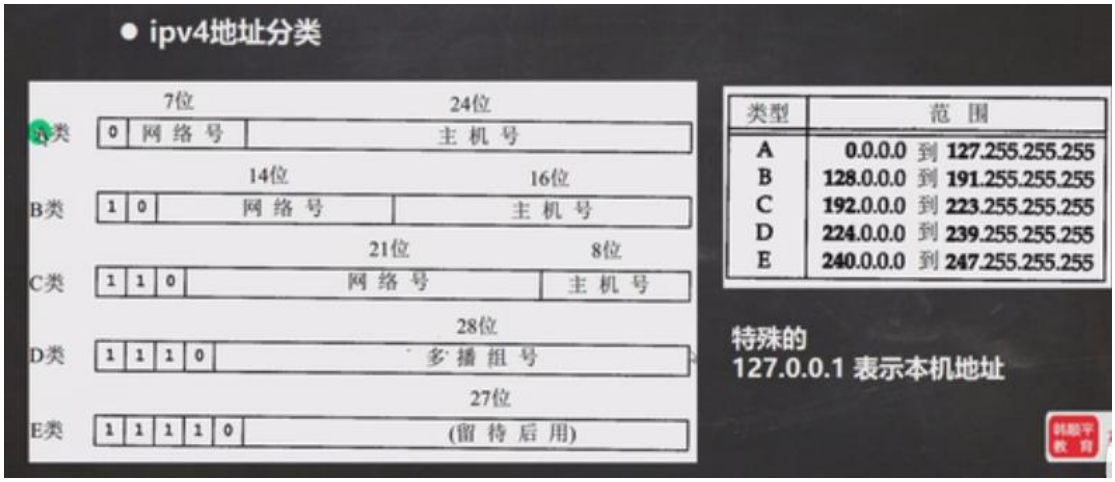


网络的相关概念
网络通信：
（1）概念：两台设备之间通过网络实现数据传输
（2）网络通信：将数据通过网络从一台设备传输到另一台设备
（3）Java.net 包下提供了一系列的类或接口，供使用，完成网络通信

网络：
（1）概念：两台或多台设备通过一定的物理设备连接起来构成了网络
（2）根据网络覆盖范围的不同，对网络进行分类
（3）局域网：覆盖范围最小，仅仅覆盖一个教室或一个机房
（4）城域网：覆盖范围较大，可以范围一个城市
（5）广域网：覆盖范围最大，可以覆盖全国乃至全球，万维网是广域网的代表

IP 地址：
（1）概念：用于唯一标识网络中的每台计算机/主机
（2）查看 IP 地址：ipconfig
（3）IP 地址的表示形式：点分十进制 xx.xx.xx.xx
（4）每个十进制数的范围：0～255
（5）ip 地址的组成 = 网络地址 + 主机地址，比如：192.168.16.69
（6）IPV6 是互联网工程任务组设计的用于替代 IPV4 的下一代 IP 协议，其地址数量号称是可以为全世界的每一粒沙子编上一个地址

（7）由于 IPV4 最大的问题在于网络地址资源有限，严重制约了互联网的应用和发展；IPV6 的使用不仅能解决网络地址资源数量的问题，而且也解决了多种接入设备连入互联网的障碍



域名：

（1）www.baidu.com

（2）好处：为了方便记忆，解决记 IP 困难

（3）概念：将 IP 地址映射成域名

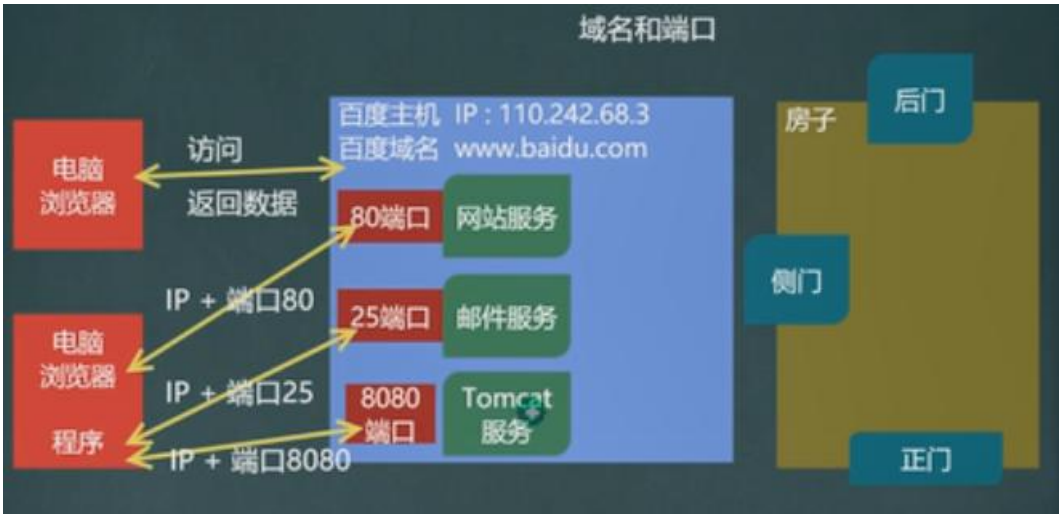
端口号：

（1）概念：用于标识计算机上某个特定的网络程序

（2）表示形式：以整数形式，范围 0~65535 [2 个字节表示端口 0~2^16-1]

（3）0~1024 已经被占用，比如 ssh 22，ftp 21，smtp 25，http 80

（4）常见的网络程序端口号：tomcat：8080；mysql：3306；oracle：1521；sqlserver：1433



网络通信协议：

协议（tcp/ip）：TCP/IP（Transmission Control Protocol/Internet Protocol）的简写中文译名为：传输控制协议/因特网互联协议，又叫网络通讯协议，这个协议是

Internet 最基本的协议、Internet 国际互联网的基础，简单来说是由网络层的 IP 协议和传输层的 TCP 协议组成的

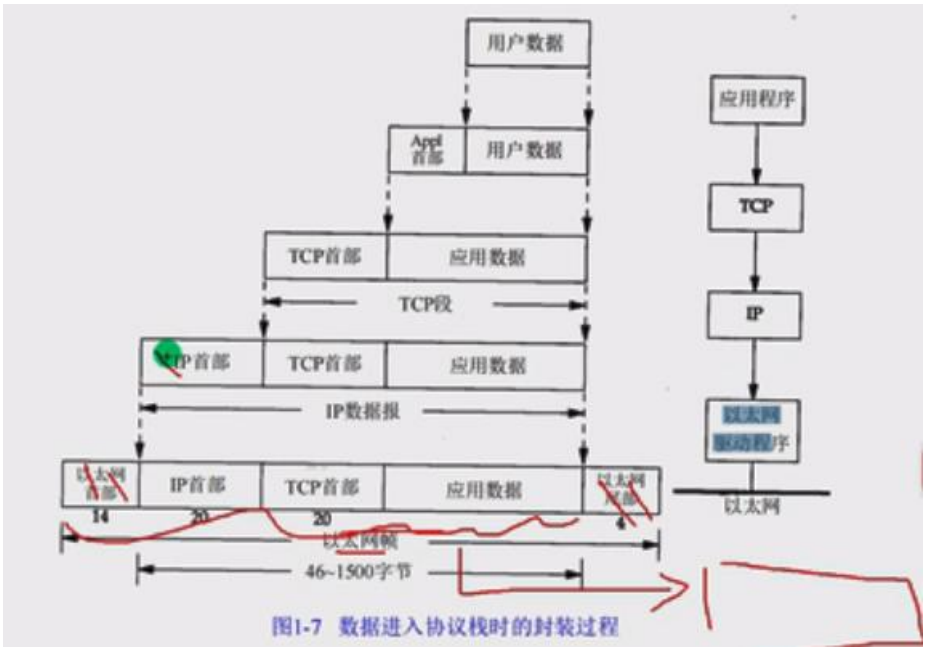


图1-7 数据进入协议栈时的封装过程

TCP 和 UDP：

TCP 协议：**传输控制协议**

（1）使用 TCP 协议签，需先建立 TCP 连接，形成传输数据通道

（2）传输前，采用“三次握手”方式，是可靠的

（3）TCP 协议进行通信的两个应用进程：**客户端、服务端**

（4）在连接中可以进行大数据量的传输

（5）传输完毕，需要释放已经建立的连接，效率低

UDP 协议：**用户数据协议**

（1）将数据、源、目的**封装成数据包**，不需要建立连接

（2）每个数据报的大小限制在 64K 以内

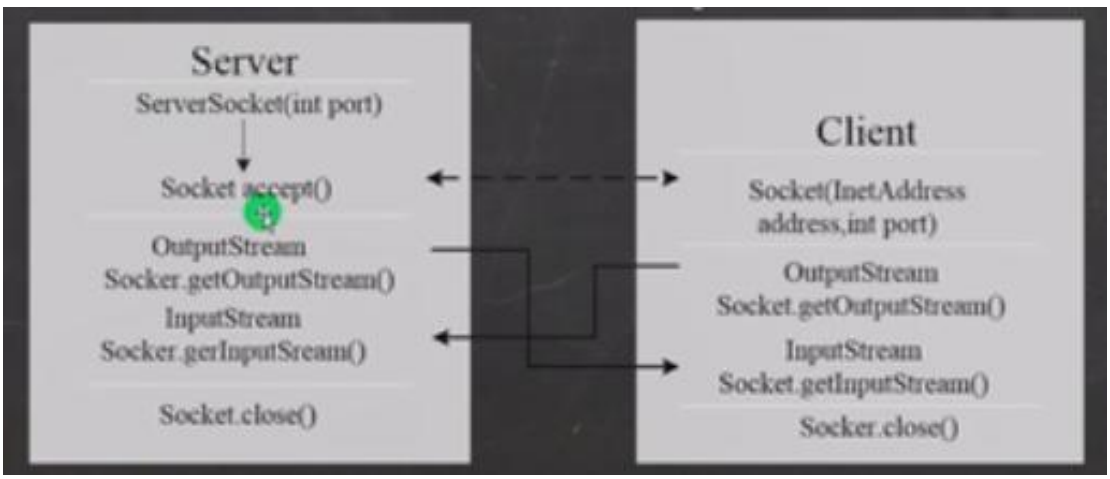
（3）因无需连接，故是不可靠的

（4）发送数据结束时无需释放资源（因为不是面向连接的），速度快

InetAddress 类
相关方法：
（1）获取本机 InetAddress 对象 getLocalHost
（2）根据指定主机名/域名获取 ip 地址对象 getByName
（3）获取 InetAddress 对象的主机名 getHostName
（4）获取 InetAddress 对象的地址 getHostAddress

Socket（套接字）
（1）套接字开发网络应用程序被广泛采用，以至于成为事实上的标准
（2）通信的**两端都要有 Socket**，是两台机器间通信的端点
（3）**网络通信其实就是 Socket 间的通信**
（4）Socket 允许程序把网络连接成一个流，数据在两个 Socket 间通过 IO 传输
（5）一般**主动发起通信的应用程序**属于**客户端，等待通信请求**的为**服务端**

TCP 网络编程
（1）基于**客户端---服务端**的网络通信
（2）底层使用的是 **TCP/IP** 协议
（3）应用场景举例：客户端发送数据，服务端接受并显示控制台
（4）基于 Socket 的 TCP 编程



TCP 网络通信编程

netstat 指令：

（1）netstat -an 可以查看当前主机网络情况，包括端口监听情况和网络连接情况

（2）netstat -an | more 可以分页显示

（3）要求在 dos 控制台下执行

说明：

（1）Listening 表示某个端口在监听

（2）如果有一个外部程序（客户端）连接到该端口，就会显示一条连接信息

UDP网络通信编程[了解]

- 基本介绍

1. 类 DatagramSocket 和 DatagramPacket[数据包/数据报] 实现了基于 UDP 协议网络程序。
2. UDP数据报通过数据报套接字 DatagramSocket 发送和接收，系统不保证UDP数据报一定能够安全送到目的地，也不能确定什么时候可以抵达。
3. DatagramPacket 对象封装了UDP数据报，在数据报中包含了发送端的IP地址和端口号以及接收端的IP地址和端口号。
4. UDP协议中每个数据报都给出了完整的地址信息，因此无须建立发送方和接收方的连接