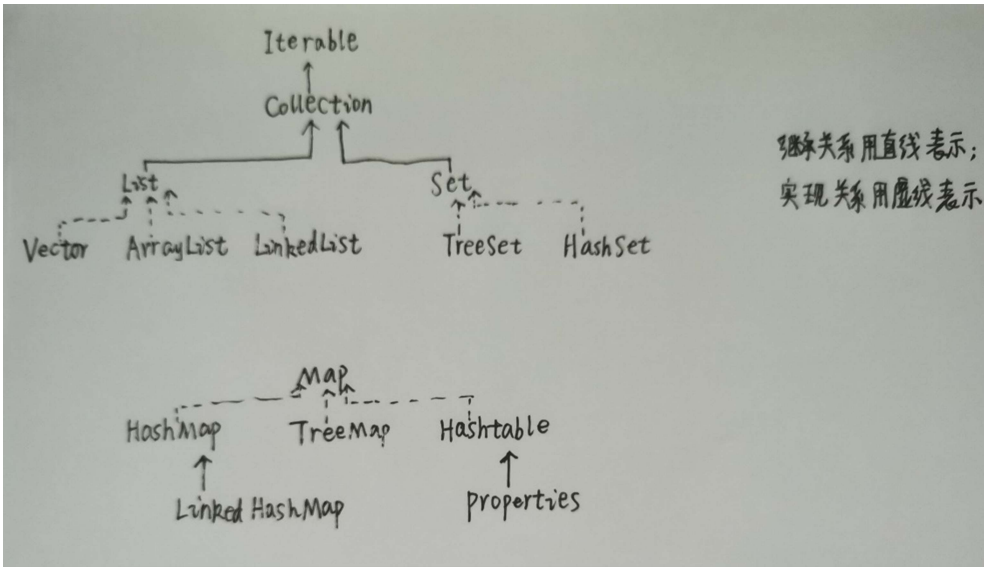
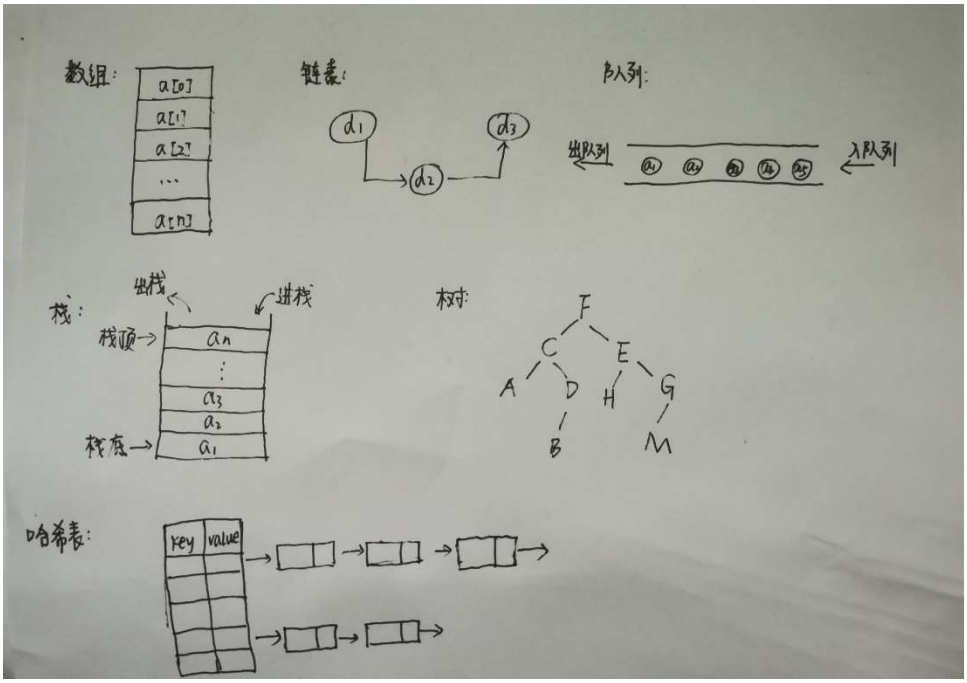
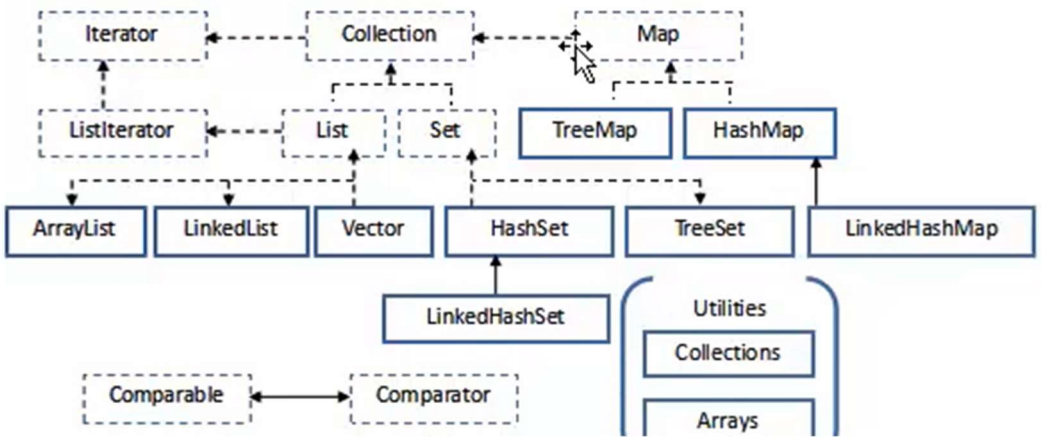
集合

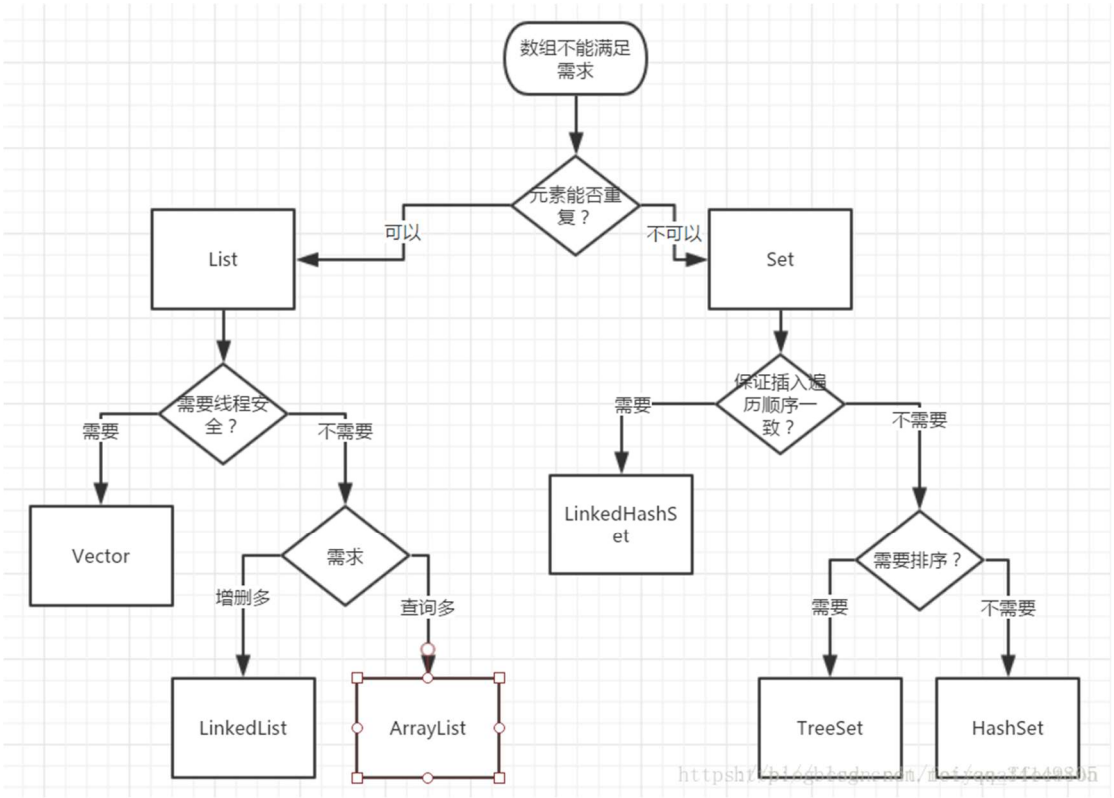

Iterable
Collection
List
Set
Vector
ArrayList
LinkedList
TreeSet
HashSet
继承关系用直线表示;
实现关系用虚线表示
Map
HashMap
TreeMap
Hashtable
LinkedHashMap
Properties


数组:
链表:
队列:
出队列
入队列
栈:
出栈
进栈
栈顶
栈底
树:
哈希表:
key
value



1. 链表和向量，ArrayList、LinkedList、Vector

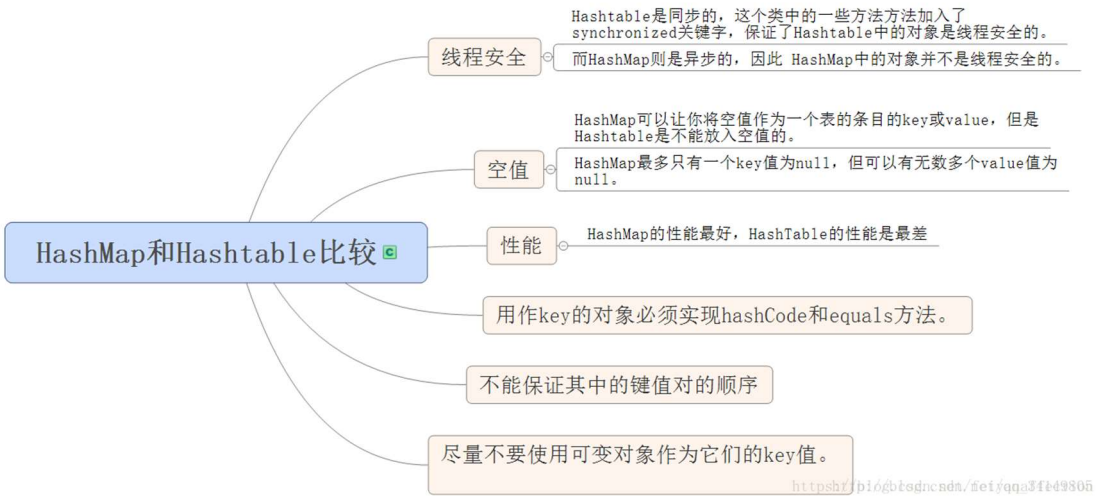
2. 集合，HashSet、TreeSet、LinkedHashSet



3. 栈和队列，Queue、stack

栈：先进后出　　队列：先进先出

4. 哈希表：hashmap、hashtable、treemap、ConcurrentHashMap



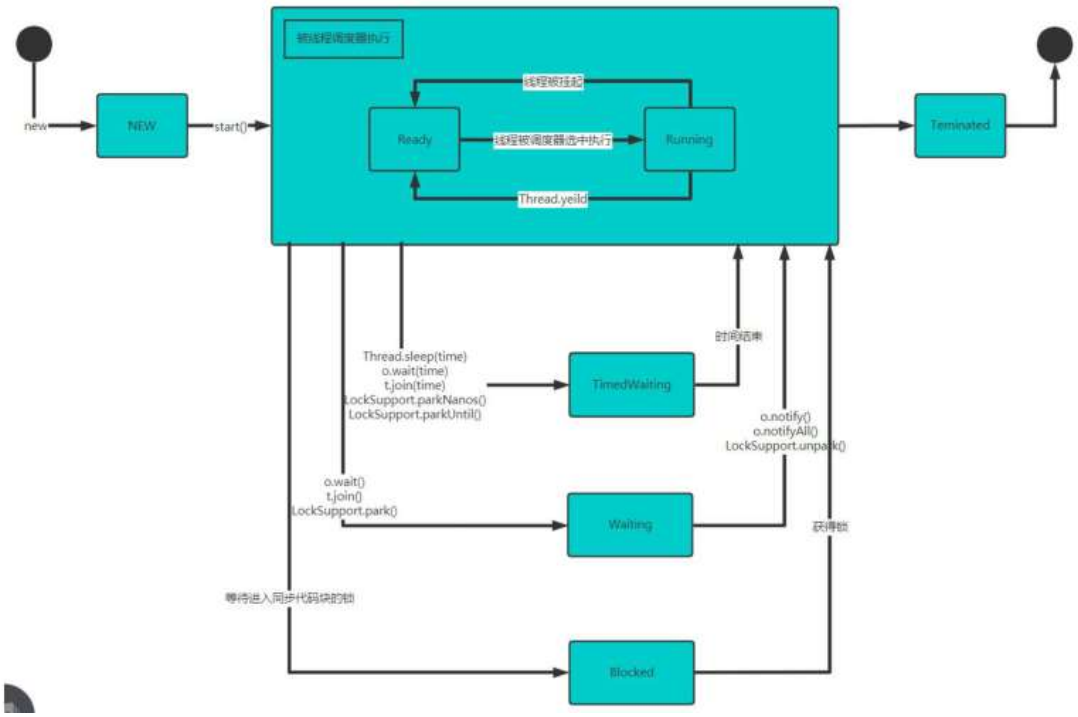
多线程与锁机制

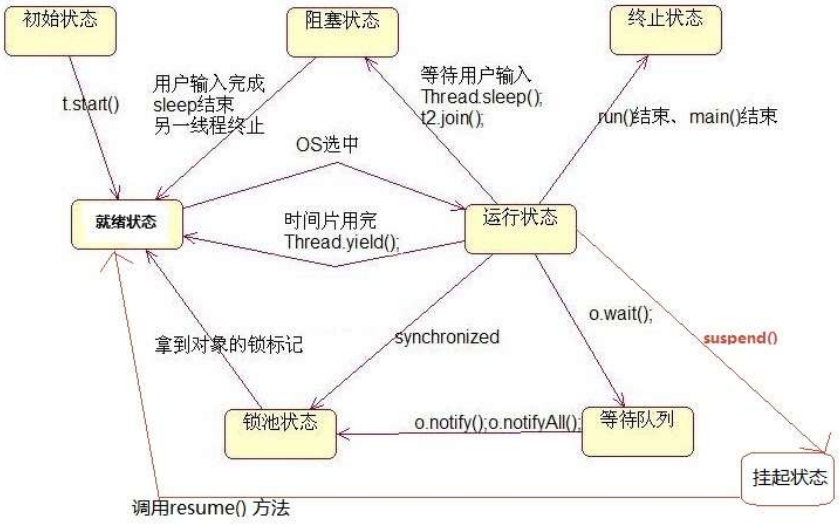
学习要求:

(1)掌握线程的不同的状态，各个状态之间的转换关系

线程状态。 线程可以处于以下状态之一：

- NEW
  尚未启动的线程处于此状态。
- RUNNABLE
  在Java虚拟机中执行的线程处于此状态。
- BLOCKED
  被阻塞等待监视器锁定的线程处于此状态。
- WAITING
  正在等待另一个线程执行特定动作的线程处于此状态。
- TIMED_WAITING
  正在等待另一个线程执行动作达到指定等待时间的线程处于此状态。
- TERMINATED
  已退出的线程处于此状态。





(2)如何创建一个线程?

1、继承 Thread 类，重写 run（）方法，然后直接 new 这个对象的实例，创建一个线程的实例，再调用 start（）方法启动线程。

2、实现 Runnable 接口，重写 run（）方法。然后调用 new Thread（runnable）的方式创建一个线程，再调用 start（）方法启动线程。

(3)掌握 volatile 关键字的原理

(4 如何中断线程?如何正确的关闭一个线程?

1、使用退出标志，使线程正常退出，也就是当 run 方法完成后线程终止。

当 run 方法执行完后，线程就会退出。但有时 run 方法是永远不会结束的。如在服务端程序中使用线程进行监听客户端请求，或是其他的需要循环处理的任务。在这种情况下，一般是将这些任务放在一个循环中，如 while 循环。如果想让循环永远运行下去，可以使用

while（true）{……}来处理。但要想使 while 循环在某一特定条件下退出，最直接的方法就是设一个 boolean 类型的标志，并通过设置这个标志为 true 或 false 来控制 while 循环是否退出。

2、使用 stop 方法强行终止线程（这个方法不推荐使用，因为 stop 和 suspend、resume 一样，也可能发生不可预料的结果）。

3、使用 interrupt 方法终止线程

使用 interrupt 方法来终端线程可分为两种情况：

（1）线程处于阻塞状态，如使用了 sleep 方法。

（2）使用 while（！isInterrupted（））{……}来判断线程是否被中断。

在第一种情况下使用 interrupt 方法，sleep 方法将抛出一个 InterruptedException 异常，而在第二种情况下线程将直接退出。

(5)如何利用 jdk 创建一个线程池?知道创建线程池的 5 大核心参数

(6)如何利用线程池实现定时任务?

(7)synchronized 和 ReentrantLock 的异同比较

相似点：都是加锁方式同步，且都是阻塞式的同步

功能区别：

这两种方式最大区别就是对于 Synchronized 来说，它是 java 语言的关键字，是原生语法层面的互斥，需要 jvm 实现。而 ReentrantLock 它是 JDK 1.5 之后提供的 API 层面的互斥锁，需要 lock()和 unlock()方法配合 try/finally 语句块来完成。

便利性：很明显 Synchronized 的使用比较方便简洁，并且由编译器去保证锁的加锁和释放，而 ReenTrantLock 需要手工声明来加锁和释放锁，为了避免忘记手工释放锁造成死锁，所以最好在 finally 中声明释放锁。

ReenTrantLock 可通过参数 true 设为公平锁，Synchronized 是非公平锁。

锁的细粒度和灵活度：很明显 ReenTrantLock 优于 Synchronized

性能的区别：

在 Synchronized 优化以前，synchronized 的性能是比 ReenTrantLock 差很多的，但是自从 Synchronized 引入了偏向锁，轻量级锁（自旋锁）后，两者的性能就差不多了，在两种方法都可用的情况下，官方甚至建议使用 synchronized，其实 synchronized 的优化我感觉就借鉴了 ReenTrantLock 中的 CAS 技术。都是试图在用户态就把加锁问题解决，避免进入内核态的线程阻塞。

(8)synchronized 修饰静态方法时获取的锁和修饰非静态方法时获取的锁有什么不同?

静态的锁为当前类本身

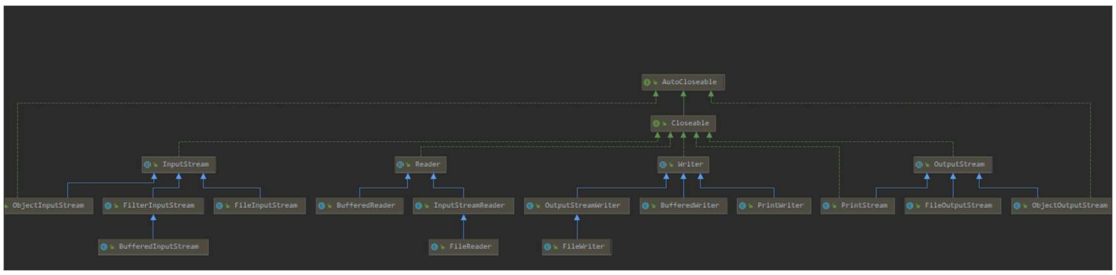
非静态的锁可以是 this，也可以是其他对象（必须是同一个对象）

(9)并发安全的集合类有哪些

Vector，LinkedList，Hashtable，ConCurrentHashMap

IO 相关

IO 框架体系图



(1)如何通过 java 代码创建文件和目录?

创建文件

```
//方式 1 new File(String pathname)
@Test
public void create01() {
    String filePath = "e:\\news1.txt";
    File file = new File(filePath);

    try {
        file.createNewFile();
        System.out.println("文件创建成功");
    } catch (IOException e) {
        e.printStackTrace();
    }

}
//方式 2 new File(File parent,String child) //根据父目录文件+子路径构建
//e:\\news2.txt
@Test
public    void create02() {
    File parentFile = new File("e:\\");
    String fileName = "news2.txt";
    //这里的 file 对象，在 java 程序中，只是一个对象
    //只有执行了 createNewFile  方法，才会真正的，在磁盘创建该文件
    File file = new File(parentFile, fileName);

    try {
        file.createNewFile();
        System.out.println("创建成功~");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
    //方式 3 new File(String parent,String child) //根据父目录+子路径构建
    @Test
    public void create03() {
        //String parentPath = "e:\\";
        String parentPath = "e:\\";
        String fileName = "news4.txt";
        File file = new File(parentPath, fileName);

        try {
            file.createNewFile();
            System.out.println("创建成功~");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

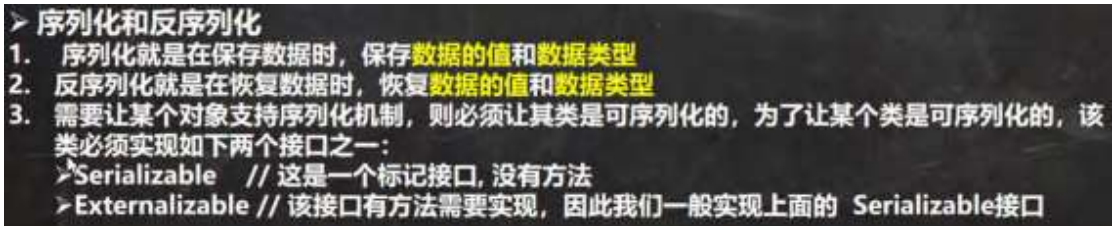
创建目录

```
//判断  D:\\demo\\a\\b\\c  目录是否存在，如果存在就提示已经存在，否则就创建
    @Test
    public void m3() {

        String directoryPath = "D:\\demo\\a\\b\\c";
        File file = new File(directoryPath);
        if (file.exists()) {
            System.out.println(directoryPath + "存在..");
        } else {
            if (file.mkdirs()) { //创建一级目录使用 mkdir() ，创建多级目录使用
mkdirs()
                System.out.println(directoryPath + "创建成功..");
            } else {
                System.out.println(directoryPath + "创建失败...");
            }
        }
```

(2)如何通过 java 代码读取文件内容、并为文件追加数据?
使用 inputStream、reader 的子类进行文件内容的读取。
是否追加数据：
a. new FileOutputStream(filePath) 创建方式，当写入内容时，会覆盖原来的内容
b. new FileOutputStream(filePath, true) 创建方式，当写入内容时，是追加到文件后面

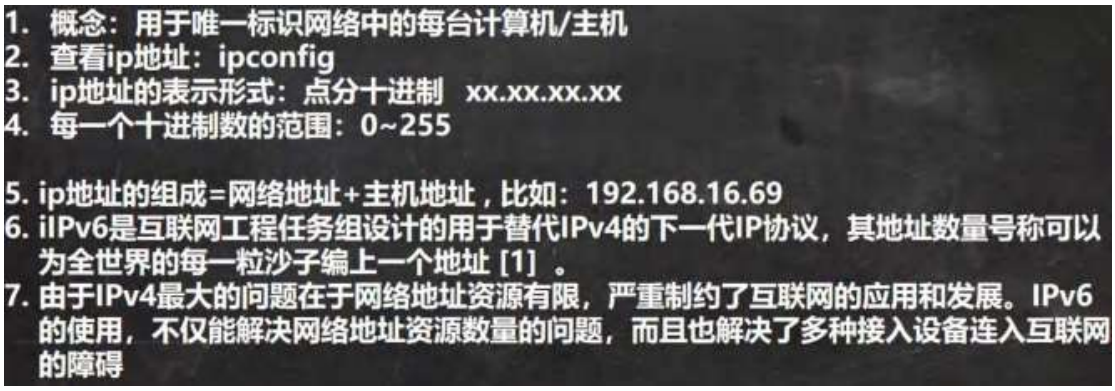
(3)什么是对象的序列化与反序列化?



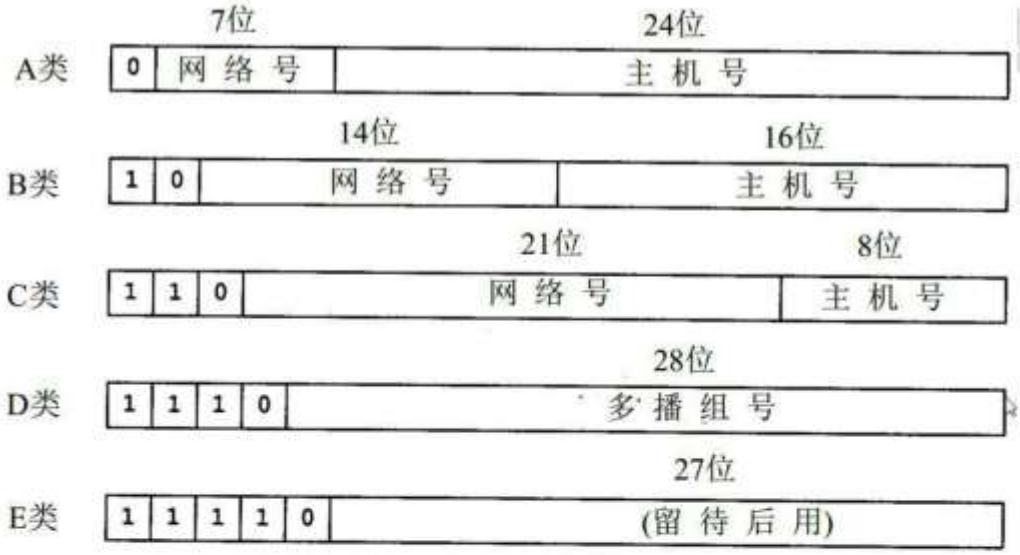
(4)什么是标准输入和标准输出?

网络编程

(1)什么是 ip 地址和 域名?
IP 地址



ipv4 地址分类



类型	范 围
A	0.0.0.0 到 127.255.255.255
B	128.0.0.0 到 191.255.255.255
C	192.0.0.0 到 223.255.255.255
D	224.0.0.0 到 239.255.255.255
E	240.0.0.0 到 247.255.255.255

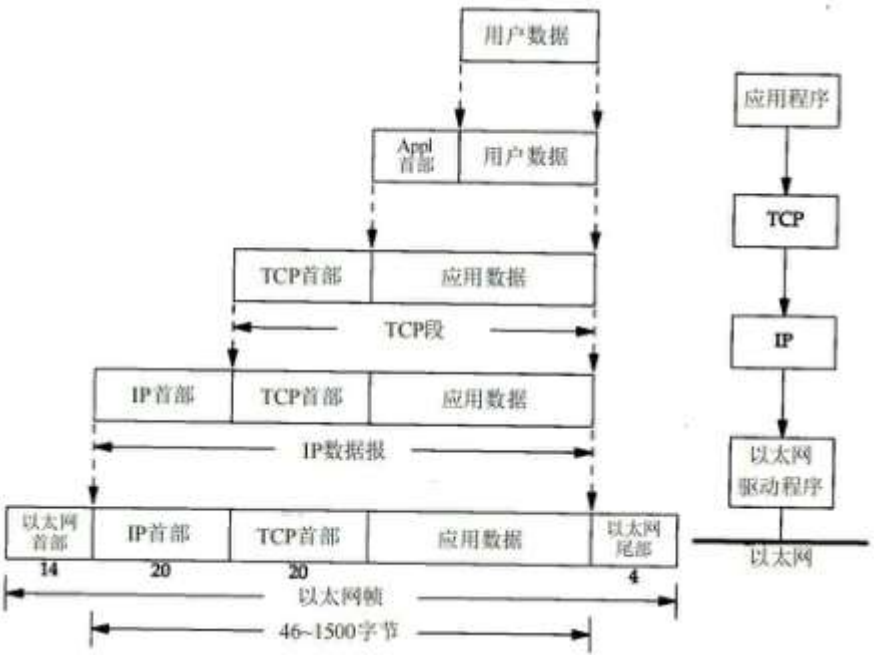
特殊的
127.0.0.1 表示本机地址

域名

1. www.baidu.com
2. 好处：为了方便记忆，解决记ip的困难
3. 概念：将ip地址映射成域名，这里怎么映射上，HTTP协议

● 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，端口范围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

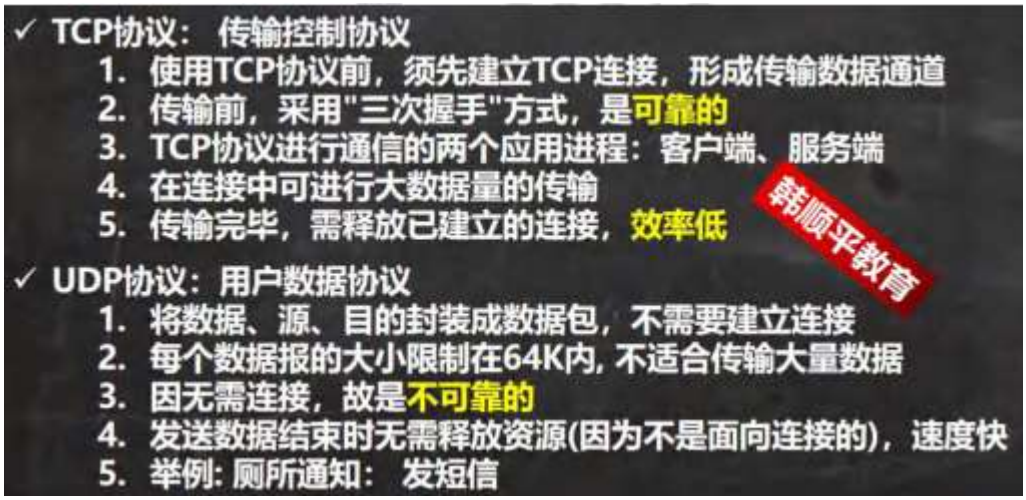
(2)了解网络 5 层协议(应用层、传输层、网络层、数据链路层、物理层)



OSI模型	TCP/IP模型	TCP/IP模型各层对应协议
应用层	应用层	HTTP、ftp、telnet、DNS...
表示层		
会话层		
传输层	传输层（TCP）	TCP、UDP、...
网络层	网络层（IP）	IP、ICMP、ARP...
数据链路层	物理+数据链路层	Link
物理层		

(3)掌握 tcp/udp 的原理和使用场景

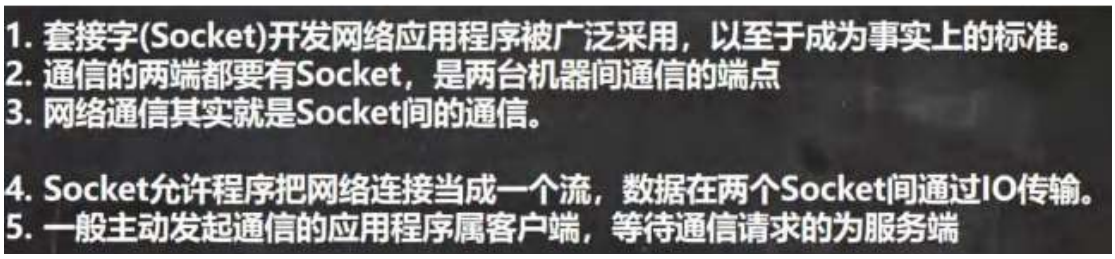
原理：

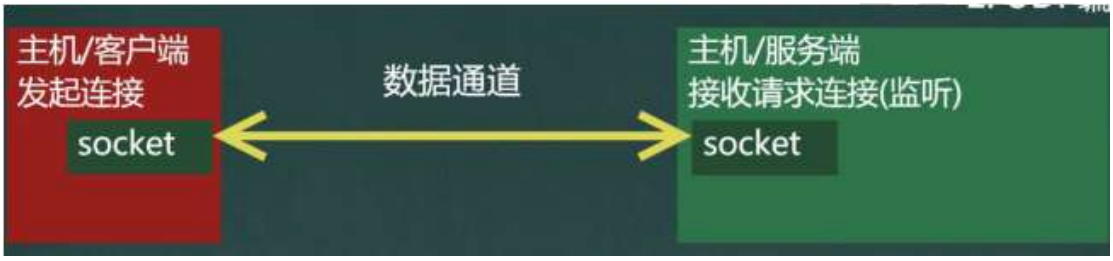


TCP 使用场景：文件传输、接受邮件、远程登录

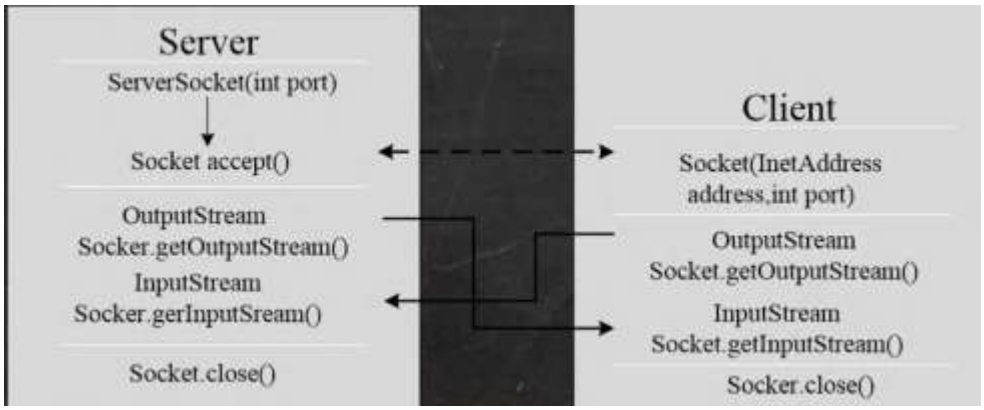
UDP 使用场景：QQ 聊天、在线视频、广播通信

(4)什么是 socket?





(5)如何编写一个 client/server 服务?



(6)如何进行文件下载?

Server：

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.InputStream;
import java.net.ServerSocket;
import java.net.Socket;
/**
 * @author 李春艳
 * @version 1.0
 * 先写文件下载的服务端
 */
public class Homework03Server {
    public static void main(String[] args) throws Exception {
        //1 监听 9999 端口
        ServerSocket serverSocket = new ServerSocket(9999);
        //2.等待客户端连接
        System.out.println("服务端，在 9999 端口监听，等待下载文件");
        Socket socket = serverSocket.accept();
```

```
            //3.读取  客户端发送要下载的文件名
            InputStream inputStream = socket.getInputStream();
            byte[] b = new byte[1024];
            int len = 0;
            String downLoadFileName = "";
            while ((len = inputStream.read(b)) != -1) {
                  downLoadFileName += new String(b, 0 , len);
            }
            System.out.println("客户端希望下载文件名=" + downLoadFileName);
            //如果客户下载的是  高山流水  我们就返回该文件，否则一律返回  无名.mp3
            String resFileName = "";
            if("高山流水".equals(downLoadFileName)) {
                  resFileName = "src\\高山流水.mp3";
            } else {
                  resFileName = "src\\无名.mp3";
            }
            //4. 创建一个输入流，读取文件
            BufferedInputStream bis =
                        new BufferedInputStream(new FileInputStream(resFileName));

            //5. 使用工具类 StreamUtils ， 读取文件到一个字节数组

            byte[] bytes = StreamUtils.streamToByteArray(bis);
            //6. 得到 Socket 关联的输出流
            BufferedOutputStream bos =
                        new BufferedOutputStream(socket.getOutputStream());
            //7. 写入到数据通道，返回给客户端
            bos.write(bytes);
            socket.shutdownOutput();//很关键.
            //8  关闭相关的资源
            bis.close();
            inputStream.close();
            socket.close();
            serverSocket.close();
            System.out.println("服务端退出...");
      }
}
```

Client：

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.Socket;
```

```
import java.util.Scanner;
/**
 * @author 李春艳
 * @version 1.0
 *  文件下载的客户端
 */
public class Homework03Client {
	public static void main(String[] args) throws Exception {
		//1. 接收用户输入，指定下载文件名
		Scanner scanner = new Scanner(System.in);
		System.out.println("请输入下载文件名");
		String downloadFileName = scanner.next();
		//2. 客户端连接服务端，准备发送
		Socket socket = new Socket(InetAddress.getLocalHost(), 9999);
		//3. 获取和 Socket 关联的输出流
		OutputStream outputStream = socket.getOutputStream();
		outputStream.write(downloadFileName.getBytes());
		//设置写入结束的标志
		socket.shutdownOutput();
		//4. 读取服务端返回的文件(字节数据)
		BufferedInputStream bis = new BufferedInputStream(socket.getInputStream());
		byte[] bytes = StreamUtils.streamToByteArray(bis);
		//5. 得到一个输出流，准备将 bytes 写入到磁盘文件
		String filePath = "e:\\" + downloadFileName + ".mp3";
		BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream(filePath));
		bos.write(bytes);
		//6. 关闭相关的资源
		bos.close();
		bis.close();
		outputStream.close();
		socket.close();
		System.out.println("客户端下载完毕，正确退出..");
	}
}
```