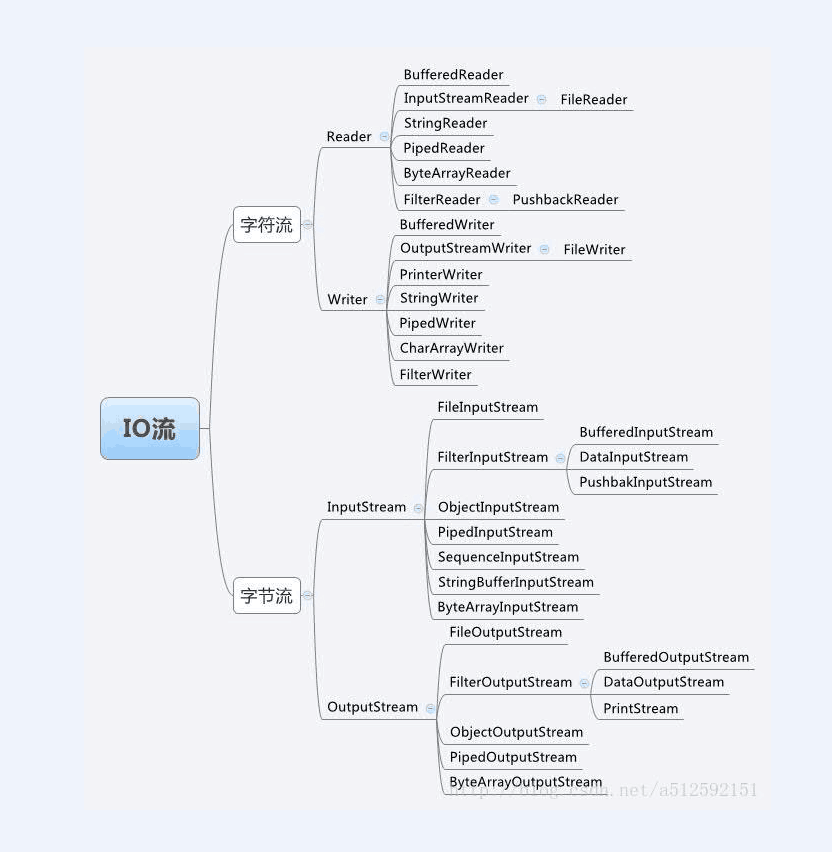
### IO流

* 1. IO流用来处理设备之间的数据传输
  2. 分类：字符流和字节流
     1. 字符流：字符流只能处理字符类型的数据；会用到缓存区
     2. 字节流：字节流能处理所有类型的数据（如图片、avi等）；不用缓存区
     3. 结论：优先选用字节流。首先因为硬盘上的所有文件都是以字节的形式进行传输或者保存的，包括图片等内容。但是字符只是在内存中才会形成的，所以在开发中，字节流使用广泛。



<https://www.cnblogs.com/QQ846300233/p/6046388.html>

* 1. File文件和目录路径名的抽象表示形式
  2. 转换流

OutputStreamWriter和InputStreamReader是字符和字节的桥梁：也可以称之为字符转换流。字符转换流原理：字节流+编码表。

FileWriter和FileReader：作为子类，仅作为操作字符文件的便捷类存在。当操作的字符文件，使用的是默认编码表时可以不用父类，而直接用子类就完成操作了，

注意：一旦要指定其他编码时，绝对不能用子类，必须使用字符转换流。什么时候用子类呢？

条件：

* + 1. 操作的是文件。
    2. 使用默认编码。
  1. 序列化和反序列化
     1. 用于从流中读取对象的操作流 ObjectInputStream 称为 反序列化流。支持 java.io.Serializable接口的对象才能从流读取
     2. 用于向流中写入对象的操作流 ObjectOutputStream 称为 序列化流。可以把 Java 对象的基本数据类型和图形写入 OutputStream。
     3. 特点：用于操作对象。可以将对象写入到文件中，也可以从文件中读取对象
     4. 当一个类的对象需要被序列化时，某些属性不需要被序列化，这时不需要序列化的属性可以使用关键字transient修饰。只要被transient修饰了，序列化时这个属性就不会被序列化了。
     5. 同时静态修饰也不会被序列化，因为序列化是把对象数据进行持久化存储，而静态的属于类加载时的数据，不会被序列化。

### 多线程

* 1. 进程和线程的概念
     1. 进程：**进程指正在运行的程序**。确切的来说，当一个程序进入内存运行，即变成一个进程，进程是处于运行过程中的程序，并且具有一定独立功能。
     2. 线程：**线程是进程中的一个执行单元**，负责当前进程中程序的执行，一个进程中至少有一个线程。一个进程中是可以有多个线程的，这个应用程序也可以称之为多线程程序。

**一个程序运行后至少有一个进程，一个进程中可以包含多个线程**

* + 1. 多线程执行时，在栈内存中，其实每一个执行线程都有一片自己所属的栈内存空间。进行方法的压栈和弹栈。
  1. 创建多线程的方式
     1. 继承Thread类
     2. 实现Runnable接口
     3. 实现Callable和Future创建线程
  2. sleep和yield的区别：
     1. sleep方法声明抛出InterruptedException，调用该方法需要捕获该异常。yield没有声明异常，也无需捕获。
     2. sleep方法暂停当前线程后，会进入阻塞状态，只有当睡眠时间到了，才会转入就绪状态。而yield方法调用后 ，是直接进入就绪状态。
  3. Thread类中提供了优先级的三个常量：

MAX\_PRIORITY =10 NORM\_PRIORITY =5 MIN\_PRIORITY =1

* 1. 同步的方式
     1. 同步函数：就是用synchronize关键字修饰的方法。因为每个java对象都有一个内置锁，当用synchronize关键字修饰方法时内置锁会保护整个方法，而在调用该方法之前，要先获得内置锁，否则就会处于阻塞状态。

public synchronized void run() {}

* + 1. 同步代码块：就是拥有synchronize关键字修饰的语句块，被该关键字修饰的语句块会自动被加上内置锁，从而实现同步。

public void run() {

while(true){ synchronized (this) { 　　　　　　　　　//同步代码块 if(tick>0){ try { Thread.sleep(10); //执行中让线程睡眠10毫秒，

} catch (InterruptedException e)

{ e.printStackTrace(); } System.out.println(Thread.currentThread().getName() + " " + tick--); } } } }

* + 1. 追加问题：如果同步函数被静态修饰之后，使用的锁是什么？静态方法中不能定义this！

　　静态内存是：内存中没有本类对象，但是一定有该类对应的字节码文件对象。 类名.class   该对象类型是Class。

　　所以静态的同步方法使用的锁是该方法所在类的字节码文件对象。 类名.class。代码如下：

public static mySyn(String name){

synchronized (Xxx.class) {

Xxx.name = name;

}

}

* + 1. start()方法和run()方法的区别

只有调用了start()方法，才会表现出多线程的特性，不同线程的run()方法里面的代码交替执行。如果只是调用run()方法，那么代码还是同步执行的，必须等待一个线程的run()方法里面的代码全部执行完毕之后，另外一个线程才可以执行其run()方法里面的代码。

* + 1. Runnable接口和Callable接口的区别

Runnable接口中的run()方法的返回值是void，它做的事情只是纯粹地去执行run()方法中的代码而已；Callable接口中的call()方法是有返回值的，是一个泛型，和Future、FutureTask配合可以用来获取异步执行的结果。

* + 1. 一个线程如果出现了运行时异常会怎么样

如果这个异常没有被捕获的话，这个线程就停止执行了。另外重要的一点是：如果这个线程持有某个某个对象的监视器，那么这个对象监视器会被立即释放

* + 1. 为什么wait()方法和notify()/notifyAll()方法要在同步块中被调用

这是JDK强制的，wait()方法和notify()/notifyAll()方法在调用前都必须先获得对象的锁。

* + 1. Linux环境下如何查找哪个线程使用CPU最长

获取项目的pid，jps或者ps -ef | grep java

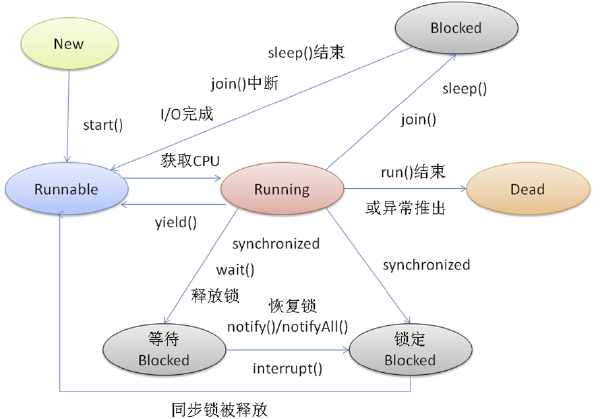
* + 1. Java中用到的线程调度算法是什么

抢占式。一个线程用完CPU之后，操作系统会根据线程优先级、线程饥饿情况等数据算出一个总的优先级并分配下一个时间片给某个线程执行。

* + 1. 单例模式的线程安全性
       1. 饿汉式单例模式的写法：线程安全
       2. 懒汉式单例模式的写法：非线程安全
       3. 双检锁单例模式的写法：线程安全
    2. 线程类的构造方法、静态块是被哪个线程调用的

这是一个非常刁钻和狡猾的问题。请记住：线程类的构造方法、静态块是被new这个线程类所在的线程所调用的，而run方法里面的代码才是被线程自身所调用的。

* 1. 线程的状态



* 1. Monitor监听器，在非多线程编码时该监视器不发挥作用，反之如果在synchronized 范围内，监视器发挥作用。wait/notify必须存在于synchronized块中。并且，这三个关键字针对的是同一个监视器（某对象的监视器）。这意味着wait之后，其他线程可以进入同步块执行。
  2. **容器类：**BlockingQueue（阻塞队列。该类是java.util.concurrent包下的重要类） ConcurrentHashMap
  3. **管理类：**ThreadPoolExecutor和 JMX框架下的系统级管理类 ThreadMXBean
  4. 线程池：线程池都是通过线程池工厂创建，再调用线程池中的方法获取线程，再通过线程去执行任务方法

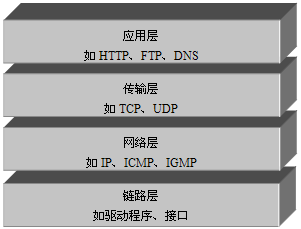
Executors：线程池创建工厂类

public static ExecutorService newFixedThreadPool(int nThreads)：返回线程池对象

ExecutorService：线程池类

### 网络编程

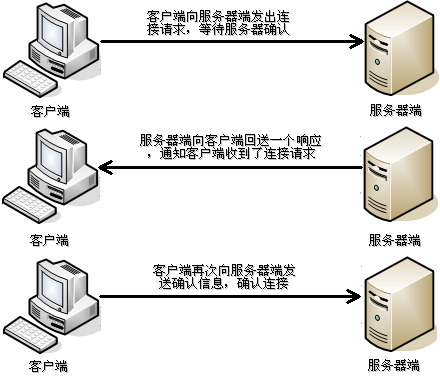
* 1. 概念：网络通信协议运用最广泛的是TCP/IP协议，包括TCP协议和IP协议，UDP（User Datagram Protocal）协议和其它一些协议的协议组



* + 1. 链路层：[链路层](http://zhidao.baidu.com/search?word=%CA%FD%BE%DD%C1%B4%C2%B7%B2%E3&fr=qb_search_exp&ie=gbk" \t "_blank)是用于定义物理传输通道，通常是对某些网络连接设备的驱动协议，例如针对**光纤、[网线](http://zhidao.baidu.com/search?word=%CB%AB%BD%CA%CF%DF&fr=qb_search_exp&ie=gbk" \t "_blank)提供的驱动**。
    2. 网络层：网络层是整个TCP/IP协议的核心，它主要用于将传输的数据进行分组，将分组数据发送到目标计算机或者网络。
    3. 运输层：主要使网络程序进行通信，在进行网络通信时，可以采用TCP协议，也可以采用UDP协议。
    4. 应用层：主要负责应用程序的协议，例如HTTP协议、FTP协议等。
  1. UDP是无连接通信协议，即在数据传输时，数据的发送端和接收端不建立逻辑连接。由于使用UDP协议消耗资源小，通信效率高，所以通常都会用于音频、视频和普通数据的传输例如视频会议都使用UDP协议.但是在使用UDP协议传送数据时，由于UDP的面向无连接性，不能保证数据的完整性，因此在传输重要数据时不建议使用UDP协议。UDP的交换过程如下图所示。



* + 1. 传输涉及两个关键字：DatagramSocket （码头）和DatagramPacket（集装县）
  1. TCP协议是面向连接的通信协议，即在传输数据前先在发送端和接收端建立逻辑连接，然后再传输数据，它提供了两台计算机之间可靠无差错的数据传输。在TCP连接中必须要明确客户端与服务器端，由客户端向服务端发出连接请求，每次连接的创建都需要经过“三次握手”。第一次握手，客户端向服务器端发出连接请求，等待服务器确认，第二次握手，服务器端向客户端回送一个响应，通知客户端收到了连接请求，第三次握手，客户端再次向服务器端发送确认信息，确认连接。整个交互过程如下图所示。



由于TCP协议的面向连接特性，它可以保证传输数据的安全性，所以是一个被广泛采用的协议，例如在下载文件时，如果数据接收不完整，将会导致文件数据丢失而不能被打开，因此，下载文件时必须采用TCP协议。

* + 1. 一个是**ServerSocket类**，用于表示**服务器端**，一个是**Socket类**，用于表示**客户端**

### 设计模式

* 1. **单例模式**：单例模式确保某个类只有一个实例，而且自行实例化并向整个系统提供这个实例。在计算机系统中，线程池、缓存、日志对象、对话框、打印机、显卡的驱动程序对象常被设计成单例。http://www.cnblogs.com/garryfu/p/7976546.html
     1. 饿汉式单例（立即加载方式）:饿汉式单例在类加载初始化时就创建好一个静态的对象供外部使用，除非系统重启，这个对象不会改变，所以本身就是线程安全的。
     2. 懒汉式单例（延迟加载方式）
     3. 静态内部类实现
     4. static静态代码块实现
     5. 内部枚举类实现
  2. **装饰者模式**：动态地给一个对象添加一些额外的职责。
     1. 涉及角色
        1. 抽象组件:定义一个抽象接口，来规范准备附加功能的类-----Component（被装饰对象的基类）
        2. 具体组件：将要被附加功能的类，实现抽象构件角色接口-----ConcreteComponent（具体被装饰对象）
        3. 抽象装饰者：持有对具体构件角色的引用并定义与抽象构件角色一致的接口-----Decorator（装饰者抽象类）
        4. 具体装饰：实现抽象装饰者角色，负责对具体构件添加额外功能-------ConcreteDecorator（具体装饰者）
     2. 应用：**IO流中：ObjectInputStream 和所有FilterInputStream的子类都是装饰流（装饰器模式的主角）。意思是FileInputStream类可以通过一个String路径名创建一个对象，FileInputStream(String name)。而DataInputStream必须装饰一个类才能返回一个对象，DataInputStream(InputStream in)**。
     3. **代理模式：**我们也是为一个类（委托类）创建一个代理类，来代表它来对外提供功能。们想对外开放某些功能，就可以将这些功能在代理类中被引用，如此一来，屏蔽了我们不想外露的功能，只将我们想开放的功能开放出来。
        1. 应用：spring的aop

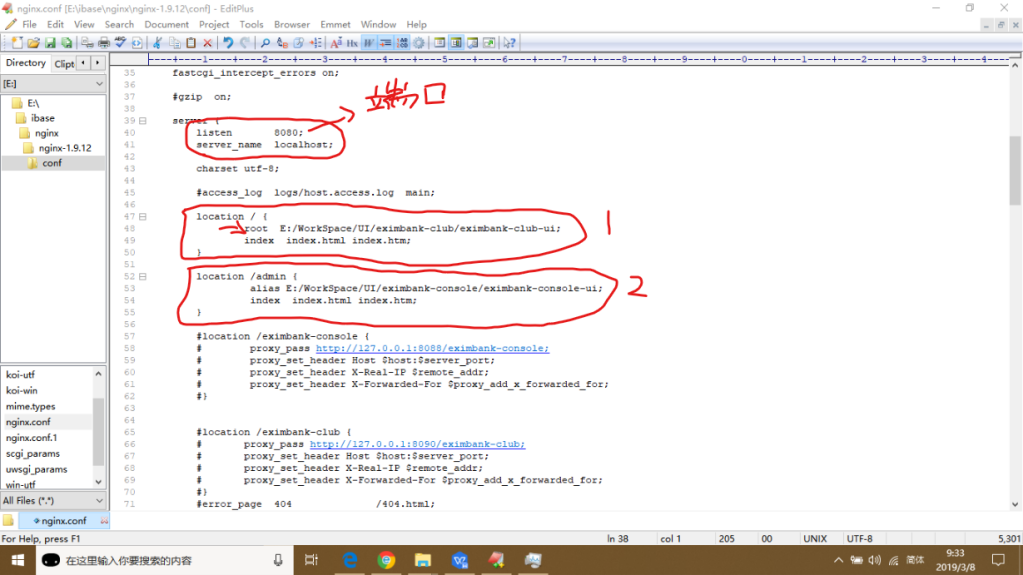
### Nginx

* 1. Nginx是一款高性能的http服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器。官方测试nginx能够支撑5万并发链接，并且cpu、内存等资源消耗缺非常低，运行非常稳定。
     1. **http服务器**。Nginx是一个http服务可以独立提供http服务。可以做网页静态服务器。
     2. **虚拟主机**。可以实现在一台服务器虚拟出多个网站。例如个人网站使用的虚拟主机。
     3. **反向代理**，**负载均衡**。当网站的访问量达到一定程度后，单台服务器不能满足用户的请求时，需要用多台服务器集群可以使用nginx做反向代理。并且多台服务器可以平均分担负载，不会因为某台服务器负载高宕机而某台服务器闲置的情况。
  2. nginx可以实现**虚拟主机**的配置，nginx主要使用三种虚拟主机配置：
     1. 基于ip的虚拟主机

添加ip地址

修改service\_name

* + 1. 基于端口的虚拟主机



* + 1. 基于域名的虚拟主机

一个域名只能绑定一个ip地址

一个ip可以被多个域名绑定。

如果要使用域名进行访问，则只需要配置域名和ip地址的映射即可，也就是DNS映射。

DNS映射分为两种：互联网映射和本地映射。

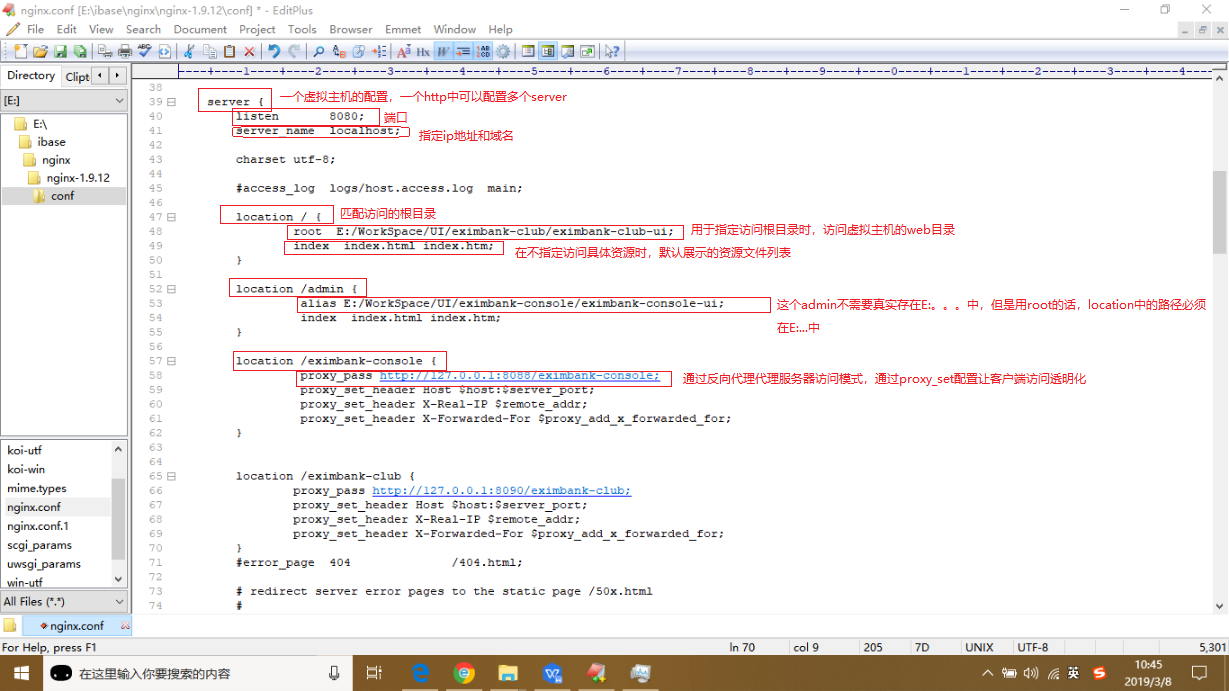
互联网映射：申请域名之后，由专门的DNS解析器解决该映射问题。

本地映射：修改hosts文件，实现ip地址和域名的映射配置：

Hosts文件的位置：C:\Windows\System32\drivers\etc\hosts

* 1. 反向代理

反向代理（Reverse Proxy）方式是指以代理服务器来接受internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给internet上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器。



### Linux

* 1. Cd ls tail -f rm cp vim find 80gg(gg GG)

### Dubbo,zookeper

* 1. Dubbo的配置文件

<dubbo:application name="anyname\_provider" />

<!-- 使用zookeeper注册中心暴露服务地址 -->

<dubbo:registry address="zookeeper://127.0.0.1:2181" />

<!-- 用dubbo协议在20880端口暴露服务 -->

<dubbo:protocol name="dubbo" port="20880" />

<!-- 声明需要暴露的服务接口 -->

<dubbo:service interface="com.shxz130.provider.Provider" ref="demoService" />

* 1. 启动流程
     1. 也就是说spring启动过程中，随着Spring在初始化过程中，碰到dubbo命名的标签，如（<dubbo:service>,<dubbo:registry>）等标签，会由DubboNamespaceHandler类处理，不同的标签会由不同的Parser处理。
     2. 当Spring容器处理完<dubbo:service>标签后，会在Spring容器中生成一个**ServiceBean** ，服务的发布也会在ServiceBean中完成。