# SpringMVC的工作流程?

1）用户发送请求至前端控制器DispatcherServlet

2）DispatcherServlet收到请求调用HandlerMapping处理器映射器

3）处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet

4）DispatcherServlet通过HandlerAdapter处理器适配器调用处理器

5）执行处理器(Controller，也叫后端控制器)

6）Controller执行完成返回ModelAndView

7）HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet

8）DispatcherServlet将ModelAndView传给ViewReslover视图解析器

9）ViewReslover解析后返回具体View

10）DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）

11）DispatcherServlet响应用户

# 2，谈谈你对mvc的理解

MVC是Model—View—Controller的简称，即模型—视图—控制器，它强制性的把应用程序的输入、处理和输出分开。  
    MVC的处理过程：首先控制器接受用户的请求，调用相应的模型来进行业务处理，并返回数据给控制器，控制器调用相应的视图来显示处理的结果

# 配置文件中都有哪些内容？

applicationContext-mvc.xml：

<!--开启MVC注解 -->

<!--放行静态资源 css js html -->

<!--配置内部资源视图解析器

配置前缀和后缀

-->

---------------------------------------------------------------------------

web.xml：

<!--配置前端控制器 -->

<!--配置POST请求过滤器 -->

---------------------------------------------------------------------------

applicationContext.xml：

<!--开启包扫描 -->

<!--引入外部配置文件 -->

<!--配置数据源 -->

<!--配置事务处理-->

---------------------------------------------------------------------------

mybatis-config.xml：

<!-- 开启驼峰自动映射 -->

---------------------------------------------------------------------------

applicationContext-mybatis.xml：

<!--spring整合mybatis

1。整合SqlSessionFactoryBean

2。整合mapper接口文件

-->

# 什么是事务？怎么理解ACID？

事务是一组操作数据库的动作集合。

ACID是事务的四大特性：

A：原子性体现在对于一个事务来讲，要么一起执行成功要么一起失败，执行的过程中是不能被打断或者执行其他操作的。

C：一致性表现为事务进行过后和执行前，整体系统都是稳定的，比如对于入账出账操作是不会有总资金的变化的。

I：隔离性表示各个事务之间不会互相影响，数据库一般会提供多种级别的隔离。实际上多个事务是并发执行的，但是他们之间不会互相影响

D：持久性表示一旦一个事务成功了，那么他的改变是永久性的被记录和操作。

# 怎么理解代理模式？

代理模式一般用于：

当一个对象不想或者不能直接引用另一个对象时，代理对象可以在两者之间起到中介的作用

通过对真实对象的封装，来实现扩展性

假设目标对象的实现类实现了接口，Spring AOP 将会采用 JDK 动态代理来生成 AOP 代理类

假设目标对象的实现类没有实现接口，Spring AOP 将会采用 CGLIB 来生成 AOP 代理类

# 数据库优化策略

1）优化sql语句

原则：

1.1)尽可能根据主键查询；

1.2)任何地方都不要使用 select \* ，用具体的字段列表代替“\*”

1.3) 应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描

1.4) 应尽量避免在 where 子句中对字段进行 null 值判断

1.5) 应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描

1.6) 能用 between 就不要用 in

2）创建索引(对经常查询的数据创建索引)

3）添加缓存(Redis/MemCache)

4）定期进行数据转储(将一些查询较少的数据保存到历史表，让当前表维护可控的数据量)

5）分库分表(需要大量的数据库服务器)

# 7，抽象类与接口的区别

|  |  |  |
| --- | --- | --- |
| **参数** | **抽象类** | **接口** |
| 默认的方法实现 | 它可以有默认的方法实现 | 接口完全是抽象的。它根本不存在方法的实现 |
| 实现 | 子类使用**extends**关键字来继承抽象类。如果子类不是抽象类的话，它需要提供抽象类中所有声明的方法的实现。 | 子类使用关键字**implements**来实现接口。它需要提供接口中所有声明的方法的实现 |
| 构造器 | 抽象类可以有构造器 | 接口不能有构造器 |
| 实例化 | 不能直接实例化，可以在实例化子类时被间接实例化 | 不能实例化 |
| 访问修饰符 | 抽象方法可以有**public**、**protected**和**default**这些修饰符 | 接口方法默认修饰符是**public**。你不可以使用其它修饰符。 |
| main方法 | 抽象方法可以有main方法并且我们可以运行它 | 接口没有main方法，因此我们不能运行它。 |
| 多继承 | 抽象方法可以继承一个类和实现多个接口 | 接口只可以继承一个或多个其它接口 |
| 速度 | 它比接口速度要快 | 接口是稍微有点慢的，因为它需要时间去寻找在类中实现的方法。 |
| 添加新方法 | 如果你往抽象类中添加新的方法，你可以给它提供默认的实现。因此你不需要改变你现在的代码。 | 如果你往接口中添加方法，那么你必须改变实现该接口的类。 |

# 8，java中的类修饰符、成员变量修饰符、方法修饰符

1）类修饰符：

public（访问控制符），将一个类声明为公共类，他可以被任何对象访问，一个程序的主类必须是公共类。

abstract，将一个类声明为抽象类，没有实现的方法，需要子类提供方法实现

final，将一个类声明为最终（即非继承类），表示他不能被其他类继承。

friendly，默认的修饰符，只有在相同包中的对象才能使用这样的类。

2）成员变量修饰符：

public（公共访问控制符），指定该变量为公共的，他可以被任何对象的方法访问。

protected（保护访问控制符）指定该变量可以别被自己的类和子类访问。在子类中可以覆盖此变量。

friendly ，在同一个包中的类可以访问，其他包中的类不能访问。

private（私有访问控制符）指定该变量只允许自己的类的方法访问，其他任何类（包括子类）中的方法均不能访问。

final，最终修饰符，指定此变量的值不能变。

static（静态修饰符）指定变量被所有对象共享，即所有实例都可以使用该变量。变量属于这个类。

transient（过度修饰符）指定该变量是系统保留，暂无特别作用的临时性变量。

volatile（易失修饰符）指定该变量可以同时被几个线程控制和修改。

3）方法修饰符：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 类内部 | 本包 | 子类 | 外部包 |
| public | √ | √ | √ | √ |
| protected | √ | √ | √ | × |
| default | √ | √ | × | × |
| private | √ | × | × | × |

# 9，ArrayList和LinkedList区别

1) ArrayList是基于动态数组的数据结构，LinkedList是基于链表的数据

构。

2)对于随机访问，ArrayList优于LinkedList，因为LinkedList要移动指针。

3）对于新增和删除操作，LinedList比较占优势，因为ArrayList要移动数据。

# 10，什么是原子操作？

所谓原子操作是指不会被线程调度机制打断的操作；这种操作一旦开始，就一直运行到结束，中间不会有任何线程转换。

# 11，JSONP是什么？有什么作用？

JSONP(JSON with Padding)是JSON的一种“使用模式”，可用于解决主流浏览器跨域数据访问的问题。由于同源策略，一般来说一个网页无法与不是该网页的服务器沟通，而 HTML 的<script> 元素是一个例外。利用 <script> 元素的这个开放策略，网页可以得到从其他来源动态产生的 JSON 数据，而这种使用模式就是所谓的 JSONP。

# HttpClient是什么？有什么作用？

HttpClient 是用来提供支持 HTTP 协议的客户端编程工具包。

作用：在java代码中发起Http请求

# 什么是Restful风格？

restful风格，就是一种面向资源服务的API设计方式。

 非REST的url：http://…/queryItems.action?id=001&type=T01   
REST风格的url：[http://…/id/001/type/T01](http://…./id/001/type/T01)

# 什么是redis?使用redis有哪些好处？

Redis 是一个基于内存的高性能key-value数据库。

1)速度快，因为数据存在内存中

2)支持丰富数据类型，支持string，list，set，sorted set，hash

3)支持事务，操作都是原子性  
4)丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除

# 15，RabbitMQ的原理

MQ，消息队列的意思，是一种通信方法，它是消费-生产者模型的一个典型的代表，生产者往消息队列中不断写入消息，而另一端消费者则可以读取队列中的消息。

1）生产者生产消息，发送给服务器端的交换机

2）交换机收到消息，根据路由key，将消息转发给匹配的队列

3）队列收到消息，将消息发送给订阅者

4）订阅者收到消息，发送ACK给队列确认收到消息

5）队列收到ACK，删除队列中缓存的此条消息

# 16，单点登录系统SSO

通过单点登录系统，用户只需要登录一次就可以访问所有相互信任的应用系统。

1）当用户输入用户名和密码点击登录时，数据会发送给SSO单点登录系统。

2）如果用户名和密码正确，则为用户生成唯一的一个秘钥TOKEN，并将用户对象转化userJSON数据。之后将token和userJSON保存到redis集群中，token充当key，userJSON充当value。

3）SSO将token信息返回并保存到客户端的Cookie中。

4）当用户访问需要权限的系统时，通过request对象获取Cookie信息。之后校验token数据是否正确，如果正确则放行，如果不正确则跳转到用户登陆页面重新登陆。

该业务的实现需要借助拦截器实现。

# 17，shiro原理

shiro是一个安全框架，可以实现用户认证，用户授权。

subject：主体，可以是用户也可以是程序，主体要访问系统，系统需要对主体进行认证、授权。

securityManager：安全管理器，主体进行认证和授权都是通过securityManager进行。

authenticator：认证器，主体进行认证最终通过authenticator进行的。

authorizer：授权器，主体进行授权最终通过authorizer进行的。

sessionManager：web应用中一般是用web容器对session进行管理，shiro也提供一套session管理的方式。

SessionDao：  通过SessionDao管理session数据，针对个性化的session数据存储需要使用sessionDao。

cache Manager：缓存管理器，主要对session和授权数据进行缓存，比如将授权数据通过cacheManager进行缓存管理，和ehcache整合对缓存数据进行管理。

realm：域，领域，相当于数据源，通过realm存取认证、授权相关数据。

 注意：在realm中存储授权和认证的逻辑。

cryptography：密码管理，提供了一套加密/解密的组件，方便开发。比如提供常用的散列、加/解密等功能。

比如 md5散列算法。

认证流程：

1）subject提交用户输入的身份信息token

2）安全管理器调用认证器进行认证

3）认证器调用realm，并将token传给realm去查询用户信息

4）如果查询不到，则提示用户不存在；

如果查到用户信息，就将数据库中加密后的密码和用来加密的盐返回给认证器

5）认证器将用户输入的密码和盐进行加密运算后，如果和数据库中的加密密码一致，则用户可以成功登录，否则密码错误，登录失败

授权流程：

1）对subject进行授权，shiro会调用方法isPermitted获取所需的权限字符串

2、安全管理器调用授权器执行授权

3、授权器调用realm从数据库查询权限数据

5、授权器进行权限字符串比对

6、如果比对后，查询到的权限数据中有被需要的权限，说明用户有访问权限，否则，没有权限，抛出异常。

# 18，[solr全文检索原理](https://www.cnblogs.com/renzherushe/p/4782396.html)

先建立索引，再对索引进行搜索的过程就叫全文检索。

索引创建：将数据提取信息，创建索引的过程。

搜索索引：就是得到用户的查询请求，搜索创建的索引，然后返回结果的过程。

# 19，Dubbo调用流程

1）dubbo调用原理实质将微服务思想和SOA思想进行融合。

2）当提供者启动时，将自身的信息写入到注册中心。

3）当消费者启动时，首先通过注册中心获取全部的服务列表信息，之后保存到本地。

4）当消费者需要完成业务时，调用第三方中立接口中的方法即可。如果有参数则直接传参。（RPC协议）

5）提供者接收到请求时，调用接口方法的最终实现类完成业务操作，之后将结果通过RPC返回给消费者即可。

6）当消费者接收到返回数据后，进行后续业务操作。

# 字符串拼接的方法

1）加号 “+”

2）String concat() 方法

3）StringUtils.join() 方法

4）StringBuffer append() 方法

5）StringBuilder append() 方法

# 21，什么是mysql内连接，左外连接，右外连接

1）内连接指的是把表连接时表与表之间匹配的数据行查询出来

2）左连接的意思是，无论是否符合连接条件都会把左边那张表的记录全部查询出来，右边的那张表只匹配符合条件的数据行。右连接则与之相反

3）自连接查询就是当前表与自身的连接查询，自连接查询一般用作表中的某个字段的值是引用另一个字段的值

如果是 1 对 多，关系维护方在 多 的一方；

如果是 多 对 多，关系维护需要建立一张中间表；

# 22，什么是sql注入攻击？怎么防止？

所谓SQL注入攻击，就是攻击者欺骗服务器执行恶意的SQL命令。

当登录页面中输入的内容直接用来构造动态SQL命令的时候，就容易受到注入攻击。

防范方法：使用PreparedStatement，因为其有预编译效果，动态传参时，即使参数中包含敏感字符，也只会被当做一个字段的属性值来处理，而不会作为一个sql指令

# 23，消息队列：死信，死信队列

消息变成死信有以下几种情况：

1）消息被拒绝并且requeue=false

2）消息过期（参考：RabbitMQ之TTL（Time-To-Live 过期时间））

3）队列达到最大长度

当一个消息在队列中变成死信（DeadMessage）之后，它能被重新发送到DLX死信交换器中，与DLX绑定的队列就是死信队列，死信队列有如下几种处理策略：

1）直接抛弃死信队列

2）定时抛弃死信队列

# 24，java开发环境，测试环境，生产环境

1）开发环境就是每个开发人员电脑上的开发环境

2）测试环境是测试人员用来进行测试的环境

3）生产环境是正式使用的系统环境（Linux）

从开发到提交，测试到生产环境是怎么个过程？过程如下：

1）windows下开发

2）提交war到svn或者ftp服务器，测试人员下载，部署，搭建测试环境

3）测试环境：windows下测试和linux下测试，测试分功能测试和性能测试，比如用loadrunner或者jmeter等。测试的目的是查漏补缺，让产品更健壮

4）解决完测试人员提出的bug后，重新打包，进入2

5）备份功能完好的代码和war，提交给PM（产品经理），确定后让测试人员部署到正式环境，然后进入3

6）写书册、使用说明等

软件开发过程一般涉及“开发 -> 测试 -> 部署上线”多个阶段，每个阶段的环境的配置参数会有不同，如数据源，文件路径等。为避免每次切换环境时都要进行参数配置等繁琐的操作，可以通过spring的profile功能来进行配置参数的切换。

首先可以在resources文件夹下分别为每个环境建立单独的文件夹（也可以额外建立一个common文件夹，用于存放公共的参数配置文件），每个文件夹下面存放对应的环境所需的配置文件。

在resources文件夹下建立applicationContext-profile.xml文件，用来定义不同的profile：

这样就实现了通过profile标记不同的环境，接下来就可以通过设置spring.profiles.default和spring.profiles.active这两个属性来激活和使用对应的配置文件。default为默认，如果没有通过active来指定，那么就默认使用default定义的环境。

这两个属性可以通过多种方法来设置：

在web.xml中作为web应用的上下文参数context-param；

在web.xml中作为DispatcherServlet的初始化参数；

作为JNDI条目；

作为环境变量；

作为JVM的系统属性；

在集成测试类上，使用@ActiveProfiles注解配置。

# 25，如何解决多服务器部署定时器重复执行的问题

1）加锁，拿到这个锁的节点执行相应的任务

2）可以在数据库对定时任务进行状态标记，根据状态来调节节点间定时任务的执行

3）代码分离，如果有可能，将定时任务独立出来，成为一个单独的项目工程，部署单节点，避免集群部署

# 26，什么是gc

gc是java的垃圾回收机制，jvm会自动回收无任何引用的对象占据的内存空间。

垃圾回收可以有效的防止内存泄露，有效的使用空闲的内存。

强制系统垃圾回收有两种方式：

1）调用System类的gc()静态方法：System.gc()

2）调用Runtime对象的gc()实例：Runtime.getRuntime().gc()

# 27，什么是JVM？

JVM是java虚拟机，是抽象化的计算机，jvm负责将Java字节码文件翻译成不同操作平台的机器码

# 28，什么是线程安全？

如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他变量值也和预期是一样的，那么就是线程安全的。

# 29，String，StringBuilder，StringBuffer三者的区别

这三个类之间的区别主要是在两个方面，运行速度和线程安全。

1）运行速度 StringBuilder > StringBuffer > String

2）在线程安全上，StringBuilder是线程不安全的，而StringBuffer是线程安全的

# 30，java中的sleep()和wait()的区别

1）sleep()方法是Thread类中的，sleep()方法会让程序暂停执行指定的时间，当指定的时间到了又会自动恢复运行状态，线程不会释放对象锁。

2）wait()方法是Object类中的，当调用wait()方法的时候，线程会释放对象锁。

# 31，Nginx反向代理的原理

1)当用户通过网络地址访问资源时，如果访问的地址刚好是Nginx反向代理的地址，则该请求访问Nginx

2)Nginx接收到用户请求后，按照内部的配置文件，动态转换请求路径

# 32，如何遍历map

Map<String，String> map =

new HashMap<String， String>();

map.put("1"，"java");

map.put("2"，"c");

map.put("3"，"go");

1)利用keySet()方法取值遍历

for(String key : map.keySet()){

System.out.println(

"key=" + key +

"and value=" + map.get(key));

}

2)利用values()方法直接遍历所有的value值

该方式取不到key值，直接遍历map中存放的value值

for(String v : map.values()){

System.out.println("value= " + v);

}

3）利用entrySet().iterator()迭代器遍历

Iterator<Map.Entry<String，String>> it = map.entrySet().iterator();

while(it.hasNext()){

Map.Entry<String，String> entry =

it.next();

System.out.println(

"key=" + entry.getKey() +

" and value=" + entry.getValue());

}

4)直接利用entrySet()方法遍历

for(Map.Entry<String，String > entry : map.entrySet()){

System.out.println(

"key=" + entry.getKey() +

" and value=" + entry.getValue());

}

# 33，cookie怎么跨域

跨二级域名

我们知道cookie是可以跨二级域名来访问，这个很好理解，例如你 www.test1.com 在的web应用程序创建了一个cookie，要想在bbs.test1.com这样的二级域名对应的应用程序中访问，就必须你在创建cookie的时候设置domain参数domain=test1.com。

# 34，maven的作用

1）Maven可以自动下载jar包

2）Maven可以自动下载jar包依赖的其它jar包

3）借助Maven，可将jar包仅仅保存在仓库中，当需要该文件时，就引用该文件接口，不需要复制文件过来占用空间

4）项目比较大时，可借助Maven将一个项目拆分成多个子项目

5）借助maven可直接将项目打成war包

6）maven项目中可以使用tomcat插件

7）借助maven可以对项目进行热部署。

热部署的意思就是在不妨碍已经运行的服务器中的项目情况下，可以直接把修改好的，新版本的项目部署到配置好的服务器上的一种部署措施；

# 35，谈谈你对spring的理解

Spring是一个IOC和AOP容器框架

1)Spring的一个核心是IOC(控制反转)：传统模式中，当需要一个对象时，我们会自己创建一个对象，而在Spring开发模式中，Spring容器（使用了工厂模式）为我们创建了所需要的对象，我们使用时不需要自己去创建，直接调用Spring为我们提供的对象即可，这就是控制反转的思想。

实例化一个java对象有三种方式：使用类构造器，使用静态工厂方法，使用实例工厂方法，当使用spring时我们就不需要关心通过何种方式实例化一个对象，spring通过控制反转机制自动为我们实例化一个对象。ApplicationContext是spring框架的IOC容器。

2）IOC中包含了依赖注入（DI），依赖注入的意思就是，Spring使用javaBean对象的Set方法或者带参数的构造方法为我们创建所需对象时将其属性自动设置为所需要的值。

3）Spring的另一个核心是AOP（面向切面编程），我们将一个个对象某些类似的方面横向抽象成一个切面，对这个切面进行一些如权限验证，事物管理，日志记录等公用操作处理，这个过程，就是面向切面编程。

# 36，动态sql中如何条件判断

使用if test

# 37，消息队列的使用场景

1）异步处理

场景说明：用户注册后，需要发注册邮件和注册短信，传统的做法有两种1.串行的方式；2.并行方式。

1.1）串行方式：将注册信息写入数据库成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后，返回给客户端。

1.2）并行方式：将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。

引入消息队列，将不是必须的业务逻辑，异步处理。

用户的响应时间相当于是注册信息写入数据库的时间。注册邮件，发送短信写入消息队列后，直接返回，因为写入消息队列的速度很快，基本可以忽略，因此用户的响应时间基本就是是注册信息写入数据库的时间。

2）应用解耦

场景说明：用户下单后，订单系统需要通知库存系统。传统的做法是，订单系统调用库存系统的接口。

引入消息队列后：

订单系统：用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功。

库存系统：订阅下单的消息，获取下单信息，库存系统根据下单信息，进行库存操作。

即使在下单时库存系统不能正常使用，也不影响正常下单，因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。实现订单系统与库存系统的应用解耦。

3）流量削锋

应用场景：秒杀活动，一般会因为流量过大，导致流量暴增，应用挂掉。为解决这个问题，一般需要在应用前端加入消息队列。

可以控制活动的人数；

可以缓解短时间内高流量压垮应用；

3.1）用户的请求，服务器接收后，首先写入消息队列。假如消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面；

3.2）秒杀业务根据消息队列中的请求信息，再做后续处理。

4）日志处理

比如Kafka的应用，解决大量日志传输的问题。架构简化如下：

日志采集客户端，负责日志数据采集，定时写受写入Kafka队列；

Kafka消息队列，负责日志数据的接收，存储和转发；

日志处理应用：订阅并消费kafka队列中的日志数据；

5）消息通讯

消息通讯是指，消息队列一般都内置了高效的通信机制，因此也可以用在纯的消息通讯。比如实现点对点消息队列，或者聊天室等。

# 38，redis有哪些持久化机制，各有什么优缺点

RDB和AOF

1）RDB的优点：

1.1）只有一个文件dump.rdb，方便持久化。

1.2）容灾性好，一个文件可以保存到安全的磁盘。

1.3）使用单独子进程来进行持久化，主进程不会进行任何IO操作，保证了redis的高性能

1.4）数据集大时，比AOF的启动效率更高。

2）RDB的缺点：

2.1）数据安全性低。(RDB是间隔一段时间进行持久化，如果持久化之间redis发生故障，会发生数据丢失。所以这种方式更适合数据要求不严谨的时候)

3）AOF的优点

3.1）数据安全，aof持久化可以配置appendfsync属性，有always，每进行一次命令操作就记录到aof文件中一次。

3.2）通过append模式写文件，即使中途服务器宕机，可以通过redis-check-aof工具解决数据一致性问题。

3.3）AOF机制的rewrite模式。(AOF文件没被rewrite之前（文件过大时会对命令进行合并重写），可以删除其中的某些命令（比如误操作的flushall）)

4）AOF的缺点

4.1）AOF文件比RDB文件大，且恢复速度慢。

4.2）数据集大的时候，比rdb启动效率低。

# 39，redis模糊查询key

在redis里,允许模糊查询key

有3个通配符 \*, ? ,[]

\*: 通配任意多个字符

?: 通配单个字符

[]: 通配括号内的某1个字符

示例：keys \*

# 40，如何处理高并发

高并发：同时并行大量请求

1）硬件方面：升级服务器，增加服务器

2）软件方面：

2.1）加锁

2.2）使用消息队列

2.3）使用缓存

2.4）HTML静态化

2.6）代码优化和数据库sql优化

2.5）单独部署文件服务器

2.6）搭建集群

# 41，有其它跨域方式吗？JSONP的缺点？

iframe。

JSONP的缺点：

1）jsonp只支持get请求而不支持post请求

2）jsonp存在安全性问题

# 42，Linux命令

1）按条件搜索查看日志

cat XX.log | grep 条件关键字

more 和 less 分页查看

2）修改文件权限

chmod 可读 可写 可执行

3）怎么找出cpu占用率高的进程

命令行输入top回车，然后按下大写M按照memory排序，按下大写P按照CPU排序。

# 43，如何实现多线程

1）继承Thread

2）实现Runnable接口

3）实现Callable接口

# 44，索引什么时候会失效

索引并不是时时都会生效的，比如以下几种情况，将导致索引失效：

1）如果条件中有or，即使其中有条件带索引也不会使用(这也是为什么尽量少用or的原因)

注意：要想使用or，又想让索引生效，只能将or条件中的每个列都加上索引

2）对于多列索引，不是使用的第一部分，则不会使用索引

3）like查询是以%开头

4）如果列类型是字符串，那一定要在条件中将数据使用引号引 起来,否则不使用索引

5）如果mysql估计使用全表扫描要比使用索引快,则不使用索引

# 45，mysql的日期类型

1）Date格式 YYYY-MM-DD

2）Datetime格式 YYYY-MM-DD HH:MM:SS

3）TimeStamp格式 时间戳，从1970-01-01 00:00:00到当前的时间差值

# 46，HashMap与HashTable有什么区别

1）HashMap中键值 允许为空 并且是非同步的

2）Hashtable中键值 不允许为空 是同步的

3）继承不同，但都实现了Map接口

HashMap继承了AbstractMap

HashTable继承了Dictiionary

# 47，RabbitMQ的5种模式

1）简单模式

功能：一个生产者P发送消息到队列Q,一个消费者C接收

2）工作队列模式Work Queue

功能：一个生产者，一个队列，多个消费者，每个消费者获取到的消息唯一

3）发布/订阅模式Publish/Subscribe

功能：一个生产者发送的消息会被多个消费者获取。一个生产者、一个交换机、多个队列、多个消费者

4）路由模式Routing

说明：生产者发送消息到交换机并且要指定路由key，消费者将队列绑定到交换机时需要指定路由key

5）通配符模式Topics

说明：生产者P发送消息到交换机X，type=topic，交换机根据绑定队列的routing key的值进行通配符匹配。

# 匹配一个或者多个词

\* 只能匹配一个词

# 48，mysql数据类型

主要包括以下五大类：

整数类型：BIT、BOOL、TINY INT、SMALL INT、MEDIUM INT、 INT、 BIG INT

浮点数类型：FLOAT、DOUBLE、DECIMAL

字符串类型：CHAR、VARCHAR、TINY TEXT、TEXT、MEDIUM TEXT、LONGTEXT、TINY BLOB、BLOB、MEDIUM BLOB、LONG BLOB

日期类型：Date、DateTime、TimeStamp、Time、Year

其他数据类型：BINARY、VARBINARY、ENUM、SET、Geometry、Point、MultiPoint、LineString、MultiLineString、Polygon、GeometryCollection等