Java基础

## 面向对象

1. **内部类**
2. **内部类分为几种**
3. **内部类可以引用它的包含类（外部类）的成员吗**
4. **请说一下 Java 中为什么要引入内部类？还有匿名内部类**
5. **构造方法**
6. **构造器链是什么**
7. **创建对象时构造器的调用顺序**
8. **抽象类与接口**
9. **Java中的23种设计模式：**

Factory（工厂模式），      Builder（建造模式），       FactoryMethod（工厂方法模式），

Prototype（原始模型模式），Singleton（单例模式），   Facade（门面模式），

Adapter（适配器模式），    Bridge（桥梁模式），        Composite（合成模式），

Decorator（装饰模式），    Flyweight（享元模式），     Proxy（代理模式），

Command（命令模式），      Interpreter（解释器模式）， Visitor（访问者模式），

Iterator（迭代子模式），   Mediator（调停者模式），    Memento（备忘录模式），

Observer（观察者模式），   State（状态模式），         Strategy（策略模式），

Template Method（模板方法模式）， Chain Of Responsibleity（责任链模式）

## 常用类

1. **Object**
2. **StringBuffer**
3. String、StringBuffer与StringBuilder之间区别

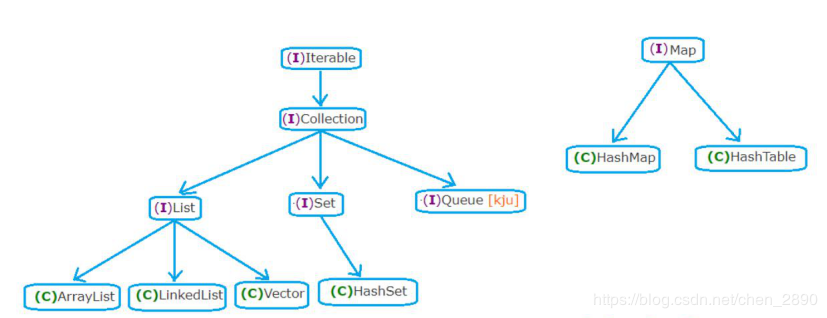
（1）.如果要操作少量的数据用= String 会不断创建对象

（2）.单线程操作字符串缓冲区 下操作大量数据=StringBuilder 不会创建对象，线程不安全

（3）.多线程操作字符串缓冲区 下操作大量数据=StringBuffer 不会创建对象，线程安全

1. **Date**
2. **Bigdecimal**
3. **Math**
4. **Object**

## 常用集合



1. **Map**
2. **HashMap的工作原理是什么, 内部的数据结构是什么**

**HashMap是基于哈希表的Map接口的非同步实现，允许key和value为null**

**HashMap内部是由数组和链表实现的，通过key的哈希值找到数组中的位置，并遍历该位置的链表，找到key相同的entry。**

**当调用hashmap的put方法放入元素的时候，先根据key的hash值得到这个元素在数组中的位置（即下标），然后把这个元素放在对应的位置。如果位置有其他元素则以链表的形式存放，新加入的在链头，最先加入的放在链尾。**

**当调用hashmap的get方法时，首先计算出key的hashcode，根据hash值找到在数组（table）中的索引，在该索引对应的单链表中查找是否有键值对的key与目标key相等，有就返回对应的value，没有则返回null。**

1. **HashMap和Hashtable的区别**

**Hashtable是线程安全的，也就是说是同步的，而HashMap是线程序不安全的，不是同步的**

**HashMap可以让你将空值作为一个表的条目的key或value**

1. **List**
2. **ArrayList 和 LinkedList 的区别**
3. **说出ArrayList,Vector, LinkedList的存储性能和特性**
4. **Collection**
5. **介绍Java中的Collection FrameWork。集合类框架的基本接口有哪些**
6. **Collections类是什么？Collection 和 Collections的区别？Collection、Map的实现**

## 异常

1. **基本概念**
2. **Error 和 Exception有什么区别**
3. **UnsupportedOperationException是什么**
4. **NullPointerException 和 ArrayIndexOutOfBoundException 之间有什么相同之处**
5. **什么是受检查的异常，什么是运行时异常**
6. **运行时异常与一般异常有何异同**
7. **简述一个你最常见到的runtime exception(运行时异常)**
8. **throw 和 throws 有什么区别？**
9. **OOM你遇到过哪些情况？你是怎么搞定的？**
10. **SOF你遇到过哪些情况？**
11. **既然我们可以用RuntimeException来处理错误，那么你认为为什么Java中还存在检查型异常**
12. **当自己创建异常类的时候应该注意什么**
13. **导致空指针异常的原因**
14. **异常处理 handle or declare 原则应该如何理解**
15. **怎么利用 JUnit 来测试一个方法的异常**
16. **catch块里别不写代码有什么问题**
17. **你曾经自定义实现过异常吗？怎么写的**
18. **什么是 异常链**
19. **在try块中可以抛出异常吗**
20. **finally**
21. **finally关键词在异常处理中如何使用**
22. **如果执行finally代码块之前方法返回了结果，或者JVM退出了，finally块中的代码还会执行吗**
23. **try里有return，finally还执行么？那么紧跟在这个try后的finally {}里的code会不会被执行，什么时候被执行，在return前还是后**
24. **在什么情况下，finally语句不会执行**
25. **final, finally 和 finalize 的区别**

final 用于声明属性，方法和类，表示属性不可变，方法不可被重写，类不可被继承。

finally 是异常处理语句结构的一部分，表示总是执行。

finalize 是 object 类的一个方法，在垃圾收集器执行的时候会调用这个对象回收的方法，工垃圾收集时其他资源的回收，比如关闭文件

## Jvm和GC

1. **GC**
2. **标记原理**

**标记两种方法：**引用计数和可达性分析

**引用计数器：**给对象添加一个引用计数器，每当有一个地方引用它时就加1，引用失效时就减1，当计数器为0的时候就标记为可回收。这种判断效率很高，但是很多主流的虚拟机并没有采用这种方法，主要是因为它很难解决几个对象之间循环引用的问题。

**可达性分析：**通过将一些称为"GC Roots"的对象作为起始点，从这些节点开始搜索，搜索和该节点发生直接或者间接引用关系的对象，将这些对象以链的形式组合起来，形成一张“关系网”，又叫做引用链。最后垃圾收集器就回收一些不在这张关系网上的对象连接GC Roots对象的object是确定还存活的对象，而右边的die obj由于和GCROOTS没有关系，所以会标记为可回收的对象。目前主流的商用虚拟机用的都是类似的方法。那什么对象才能作为“GC Roots”呢？在java中，有四种对象可以作为“GC Roots”

        1：栈帧（第一章的名词）中的引用对象。（栈中的）

        2：静态属性引用的对象。（方法区中的）

        3：常量引用的对象。（方法区中的）

        4：本地方法栈中JNI引用的对象。（本地方法栈中的）

1. **GC算法**
2. **标记-清除算法**

对等待被回收的对象进行标记，然后对标记对象进行清除

1. **复制算法**

解决标记-清除算法带来的内存碎片问题。先标记待回收的内存和不用回收的内存，下一步将不用回收的内存复制到新的内存区域，然后全部回收旧的内存区域，而新的内存区域是则是连续的。缺点是会损失掉部分系统内存，需要一部分内存进行复制。

1. **标记-压缩算法**

首先是标记，标记过后将不用回收的内存对象压缩到内存一端，此时即可直接清楚边界出的内存。避免复制带来的效率问题，同时也能避免碎片化

1. **Jvm**

概括地说来，JVM初始运行的时候都会分配好**Method Area（方法区）**和**Heap（堆）**，而JVM 每遇到一个线程，就为其分配一个**Program Counter Register（程序计数器）**, **VM Stack（虚拟机栈）和Native Method Stack （本地方法栈），**当线程终止时，三者（虚拟机栈，本地方法栈和程序计数器）所占用的内存空间也会被释放掉。这也是为什么我把内存区域分为线程共享和非线程共享的原因，非线程共享的那三个区域的生命周期与所属线程相同，而线程共享的区域与JAVA程序运行的生命周期相同，所以这也是系统垃圾回收的场所只发生在线程共享的区域（实际上对大部分虚拟机来说知发生在Heap上）的原因。

**参数**

1. **说说你知道的几种主要的jvm 参数**
2. **-XX:+UseCompressedOops 有什么作用**

**类加载器(ClassLoader)**

1. **Java 类加载器都有哪些**
2. **JVM如何加载顺序**

java源代码 ————> .class字节码 ————> 类加载器 ————> 字节校验器 ————> 解释器和即时编译器 ————> 执行引擎

**内存管理**

1. **JVM内存分哪几个区，每个区的作用是什么**
2. java栈内存

栈内存是指程序进入一个方法时，会为这个方法单独分配一块私属存储空间，用于存储这个方法内部的局部变量，当这个方法结束时，分配给这个方法的栈会释放，这个栈中的变量也将随之释放

1. Java堆

存放所有的对象实例都在这里分配内存

1. 序计数寄存器

程序计数器就是记录当前线程执行程序的位置，改变计数器的值来确定执行的下一条指令，线程恢复都是依赖程序计数器来完成。

1. 方法区域（Method Area）

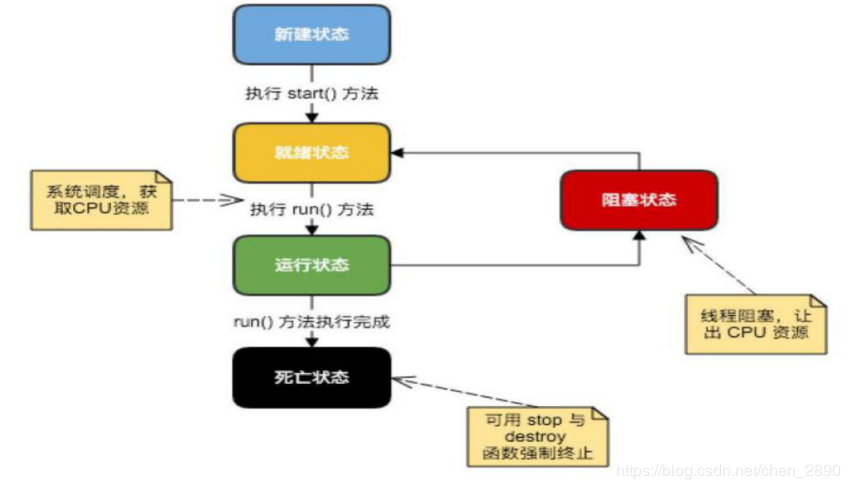
它用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据

1. 本地方法堆栈（Native Method Stacks）

  该区域与虚拟机栈所发挥的作用非常相似，只是虚拟机栈为虚拟机执行Java方法服务，而本地方法栈则为使用到的本地操作系统（Native）方法服务。

1. **一个对象从创建到销毁都是怎么在这些部分里存活和转移的**
2. **JVM中哪个参数是用来控制线程的栈堆栈小**
3. **简述内存分配与回收策略**
4. **简述重排序，内存屏障，happen-before，主内存，工作内存**
5. **Java中存在内存泄漏问题吗？请举例说明**
6. **简述 Java 中软引用（SoftReferenc）、弱引用（WeakReference）和虚引用**
7. **内存映射缓存区是什么**
8. **32 位 JVM 和 64 位 JVM 的最大堆内存分别是多数？32 位和 64 位的 JVM，int 类型变量的长度是多数？**
9. **怎样通过 Java 程序来判断 JVM 是 32 位 还是 64 位**
10. **JVM自身会维护缓存吗？是不是在堆中进行对象分配，操作系统的堆还是JVM自己管理堆**
11. **什么情况下会发生栈内存溢出**

## 多线程



1. **基本概念**
2. **什么是线程**
3. **多线程的优点**
4. **多线程的几种实现方式**
5. **用 Runnable 还是 Thread**
6. **什么是线程安全**
7. **Vector, SimpleDateFormat 是线程安全类吗**
8. **什么 Java 原型不是线程安全的**
9. **哪些集合类是线程安全的**
10. **多线程中的忙循环是什么**
11. **如何创建一个线程**
12. **编写多线程程序有几种实现方式**
13. **什么是线程局部变量**
14. **线程和进程有什么区别？进程间如何通讯，线程间如何通讯**
15. **什么是多线程环境下的伪共享（false sharing）**
16. **同步和异步有何异同，在什么情况下分别使用他们？举例说明**
17. **ThreadLocal和Synchonized区别**

ThreadLocal和Synchonized都用于解决多线程并发访问。

synchronized是利用锁的机制，使变量或代码块在某一时该只能被一个线程访问。

而ThreadLocal为每一个线程都提供了变量的副本，使得每个线程在某一时间访问到的并不是同一个对象，这样就隔离了多个线程对数据的数据共享。

优点：ThreadLocal为每个线程的中并发访问的数据提供一个副本，通过访问副本来运行业务，这样的结果是耗费了内存，单大大减少了线程同步所带来性能消耗，也减少了线程并发控制的复杂度。

1. **Thread**
2. **启动一个线程是调用 run() 还是 start() 方法？start() 和 run() 方法有什么区别**
3. **调用start()方法时会执行run()方法，为什么不能直接调用run()方法**
4. **sleep() 方法和对象的 wait() 方法都可以让线程暂停执行，它们有什么区别**
5. **yield方法有什么作用？sleep() 方法和 yield() 方法有什么区别**
6. **Java 中如何停止一个线程**
7. **stop() 和 suspend() 方法为何不推荐使用**
8. **如何在两个线程间共享数据**
9. **如何强制启动一个线程**
10. **如何让正在运行的线程暂停一段时间**
11. **什么是线程组，为什么在Java中不推荐使用**
12. **你是如何调用 wait（方法的）？使用 if 块还是循环？为什么**
13. **生命周期**
14. **有哪些不同的线程生命周期**
15. **线程状态，BLOCKED 和 WAITING 有什么区别**
16. **画一个线程的生命周期状态图**

**ThreadLocal 用途是什么，原理是什么，用的时候要注意什么**

1. **ThreadPool**
2. **线程池是什么？为什么要使用它**
3. **如何创建一个Java线程池**
4. **ThreadPool用法与优势**
5. **提交任务时，线程池队列已满时会发会生什么**
6. **newCache 和 newFixed 有什么区别？简述原理。构造函数的各个参数的含义是什么，比如 coreSize, maxsize 等**
7. **线程池的实现策略**
8. **线程池的关闭方式有几种，各自的区别是什么**
9. **线程池中submit() 和 execute()方法有什么区别？**
10. **线程调度**
11. **Java中用到的线程调度算法是什么**
12. **什么是多线程中的上下文切换**
13. **你对线程优先级的理解是什么**
14. **什么是线程调度器 (Thread Scheduler) 和时间分片 (Time Slicing)**
15. **线程同步**
16. **请说出你所知的线程同步的方法**
17. **synchronized 的原理是什么**
18. **synchronized 和 ReentrantLock 有什么不同**
19. **什么场景下可以使用 volatile 替换 synchronized**
20. **有T1，T2，T3三个线程，怎么确保它们按顺序执行？怎样保证T2在T1执行完后执行，T3在T2执行完后执行**
21. **同步块内的线程抛出异常会发生什么**
22. **当一个线程进入一个对象的 synchronized 方法A 之后，其它线程是否可进入此对象的 synchronized 方法B**
23. **使用 synchronized 修饰静态方法和非静态方法有什么区别**
24. **如何从给定集合那里创建一个 synchronized 的集合**
25. **锁**
26. **Java Concurrency API 中 的 Lock 接口是什么？对比同步它有什么优势**
27. **Lock 与 Synchronized 的区别？Lock 接口比 synchronized 块的优势是什么**
28. **ReadWriteLock是什么？**
29. **锁机制有什么用**
30. **什么是乐观锁（Optimistic Locking）？如何实现乐观锁？如何避免ABA问题**
31. **乐观锁和悲观锁的区别**

乐观锁：每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制

悲观锁：每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会block直到它拿到锁

## IO

## 网络编程

1. **HTTP协议**

常见错误码

1. **RPC协议**
2. **Socket协议**

## 反射注解

Javaweb

## Servlet

1. **servlet的生命周期**

servlet的生命周期通过javax.servlet.Servlet接口中的init()、service()和destroy()方法来表示

Servlet的生命周期包含了下面4个阶段：

1.加载和实例化

2.初始化

3.请求处理

4.服务终止

1. **Web服务器在与客户端交互时Servlet的工作过程是:**

1. 在客户端对web服务器发出请求

2. web服务器接收到请求后将其发送给Servlet

3. Servlet容器为此产生一个实例对象并调用ServletAPI中相应的方法来对客户端HTTP请求进行处理,然后将处理的响应结果返回给WEB服务器.

4. web服务器将从Servlet实例对象中收到的响应结构发送回客户端.

## Ajax

## Jdbc

1. **通过 JDBC 连接数据库有哪几种方式**
2. **阐述 JDBC 操作数据库的基本步骤**
3. **JDBC 中如何进行事务处理**
4. **什么是 JdbcTemplate**
5. **什么是 DAO 模块**
6. **使用 JDBC 操作数据库时，如何提升读取数据的性能？如何提升更新数据的性能**

## Tomcat

框架

## Spring(SpringBoot)

**Spring在代码中获取bean的几种方式**

方法一：在初始化时保存ApplicationContext对象 方法二：通过Spring提供的utils类获取ApplicationContext对象 方法三：继承自抽象类ApplicationObjectSupport 方法四：继承自抽象类WebApplicationObjectSupport 方法五：实现接口ApplicationContextAware 方法六：通过Spring提供的ContextLoader

**获取spring中bean的方式总结：**

##### 方法一：在初始化时保存ApplicationContext对象

1. ApplicationContext ac = new FileSystemXmlApplicationContext("applicationContext.xml");
2. ac.getBean("beanId");

说明：这种方式适用于采用Spring框架的独立应用程序，需要程序通过配置文件手工初始化Spring的情况。

##### 方法二：通过Spring提供的工具类获取ApplicationContext对象

1. ApplicationContext ac1 = WebApplicationContextUtils.getRequiredWebApplicationContext(ServletContext sc);
2. ApplicationContext ac2 = WebApplicationContextUtils.getWebApplicationContext(ServletContext sc);
3. ac1.getBean("beanId");
4. ac2.getBean("beanId");

说明：这种方式适合于采用Spring框架的B/S系统，通过ServletContext对象获取ApplicationContext对象，然后在通过它获取需要的类实例。上面两个工具方式的区别是，前者在获取失败时抛出异常，后者返回null。

##### 方法三：继承自抽象类ApplicationObjectSupport

说明：抽象类ApplicationObjectSupport提供getApplicationContext()方法，可以方便的获取ApplicationContext。

Spring初始化时，会通过该抽象类的setApplicationContext(ApplicationContext context)方法将ApplicationContext 对象注入。

##### 方法四：继承自抽象类WebApplicationObjectSupport

说明：类似上面方法，调用getWebApplicationContext()获取WebApplicationContext

##### 方法五：实现接口ApplicationContextAware

说明：实现该接口的setApplicationContext(ApplicationContext context)方法，并保存ApplicationContext 对象。Spring初始化时，会通过该方法将ApplicationContext对象注入。

以下是实现ApplicationContextAware接口方式的代码，前面两种方法类似：

1. public class SpringContextUtil implements ApplicationContextAware {
2. // Spring应用上下文环境
3. private static ApplicationContext applicationContext;
4. /\*\*
5. \* 实现ApplicationContextAware接口的回调方法，设置上下文环境
6. \*
7. \* @param applicationContext
8. \*/
9. public void setApplicationContext(ApplicationContext applicationContext) {
10. SpringContextUtil.applicationContext = applicationContext;
11. }
12. /\*\*
13. \* @return ApplicationContext
14. \*/
15. public static ApplicationContext getApplicationContext() {
16. return applicationContext;
17. }
18. /\*\*
19. \* 获取对象
20. \*
21. \* @param name
22. \* @return Object
23. \* @throws BeansException
24. \*/
25. public static Object getBean(String name) throws BeansException {
26. return applicationContext.getBean(name);
27. }
28. }

虽然，spring提供的后三种方法可以实现在普通的类中继承或实现相应的类或接口来获取spring 的ApplicationContext对象，但是在使用是一定要注意实现了这些类或接口的普通java类一定要在Spring 的配置文件applicationContext.xml文件中进行配置。否则获取的ApplicationContext对象将为null。

##### 方法六：通过Spring提供的ContextLoader

1. WebApplicationContext wac = ContextLoader.getCurrentWebApplicationContext();
2. wac.getBean(beanID);

最后提供一种不依赖于servlet,不需要注入的方式。但是需要注意一点，在服务器启动时，Spring容器初始化时，不能通过以下方法获取Spring 容器，细节可以查看spring源码org.springframework.web.context.ContextLoader。

**spring ioc原理**

IOC是控制反转，把对象创建和协调的管理权交给spring容器。以前的方式是通过new对象来的，现在把他交给spring容器来进行创建，这样就达到了降低代码耦合度，实现解耦。

DI（Dependency Injection，依赖注入）：一个重要特征是反射（reflection），它允许程序在运行的时候动态的生成对象、执行对象的方法、改变对象的属性，spring就是通过反射来实现注入的。

**Spring AOP原理**

Spring AOP中的动态代理主要有两种方式，JDK动态代理和CGLIB动态代理。JDK动态代理通过反射来接收被代理的类，并且要求被代理的类必须实现一个接口。JDK动态代理的核心是InvocationHandler接口和Proxy类。

**JDK动态代理(Dynamic Proxy)**

基于标准JDK的动态代理功能

只针对实现了接口的业务对象

**CGLIB**

通过动态地对目标对象进行子类化来实现AOP代理，

需要指定@EnableAspectJAutoProxy(proxyTargetClass = true)来强制使用

当业务对象没有实现任何接口的时候默认会选择CGLIB

## Mybatis

1. **事务相关**

4种事务特性：

原子性 （atomicity）:强调事务的不可分割.

一致性 （consistency）:事务的执行的前后数据的完整性保持一致.

隔离性 （isolation）:一个事务执行的过程中,不应该受到其他事务的干扰

持久性（durability） :事务一旦结束,数据就持久到数据库

5种隔离级别：

DEFAULT 这是一个PlatfromTransactionManager默认的隔离级别，使用数据库默认的事务隔离级别

未提交读（read uncommited） :脏读，不可重复读，虚读都有可能发生

已提交读 （read commited）:避免脏读。但是不可重复读和虚读有可能发生

可重复读 （repeatable read） :避免脏读和不可重复读.但是虚读有可能发生

串行化的 （serializable） :避免以上所有读问题

7种传播行为：    保证同一个事务中

PROPAGATION\_REQUIRED 支持当前事务，如果不存在 就新建一个(默认)

PROPAGATION\_SUPPORTS 支持当前事务，如果不存在，就不使用事务

PROPAGATION\_MANDATORY 支持当前事务，如果不存在，抛出异常

   保证没有在同一个事务中

PROPAGATION\_REQUIRES\_NEW 如果有事务存在，挂起当前事务，创建一个新的事务

PROPAGATION\_NOT\_SUPPORTED 以非事务方式运行，如果有事务存在，挂起当前事务

PROPAGATION\_NEVER 以非事务方式运行，如果有事务存在，抛出异常

PROPAGATION\_NESTED 如果当前事务存在，则嵌套事务执行

## Netty

netty原理和使用

Netty是一个高性能 事件驱动的异步的非堵塞的IO(NIO)框架，用于建立TCP等底层的连接，基于Netty可以建立高性能的Http服务器。

netty处理流程

Reactor分成两部分，mainReactor负责监听server socket，accept新连接，并将建立的socket分派给subReactor。subReactor负责多路分离已连接的socket，读写网络数据，对业务处理功能，其扔给worker线程池完成

BIO：同步阻塞的通信模型，每一个socket请求分配一个对应的线程，而这个线程执行中有各种等待情况。

NIO：同步非阻塞通信模型，mainReactor负责监听serversocket，accept新连接，并将建立的socket分派给subReactor。subReactor负责多路分离已连接的socket，读写网 络数据，对业务处理功能，其扔给worker线程池完成

## Redis(memcached)

redis原子性操作

redis原子性操作的实现原理在于redis底层使用单线程操作。

Redis存储数据类型：String、list、set、SortedSet、Pub/Sub

## Nginx

nginx

正向代理：针对客户端而言， 代理服务器代理客户端，转发请求，并将获得的内容返回给客户端。

反向代理：针对客户端而言， 代理服务器就像是原始服务器,代理集群的web节点服务器返回结果。

负载均衡策略：1、按照轮询（默认）方式进行负载 2、权重 3、ip\_hash

## Dubbo

## SpringCloud

数据库

## Mysql

数据库sql优化

全表扫描关键字：like、in、not in、!= <>、or 、null、is null

用 exists 代替 in

数据库索引。

1、#{}和${}的区别是什么？

#{}是预编译处理，${}是字符串替换。

Mybatis在处理#{}时，会将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值；

Mybatis在处理${}时，就是把${}替换成变量的值。

使用#{}可以有效的防止SQL注入，提高系统安全性。

2、当实体类中的属性名和表中的字段名不一样

第1种： 通过在查询的sql语句中定义字段名的别名，让字段名的别名和实体类的属性名一致

第2种： 通过<resultMap>来映射字段名和实体类属性名的一一对应的关系

3、如何执行批量插入

sqlsessionfactory.opensession

executortype.batch

4、索引什么情况下会失效

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)
2. like查询是以%开头
3. 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用起来,否则不使用索引
4. 如果mysql估计使用全表扫描要比使用索引快,则不使用索引

开发相关

## Git

## Gradle

## Maven

## Idea

## Linux