面试准备

1. 技术
2. Java
   1. Java基础

1.1.1 Java常用集合

Collection

----List

--------ArrayList，基于数组，随机访问效率高，线程不安全

--------LinkedList，基于链表，头尾来增删的效率高

ArrayList的随机访问更高，基于数组实现的ArrayList可直接定位到目标对象，而LinkedList需要从头Node或尾Node开始向后/向前遍历若干次才能定位到目标对象

LinkedList在头/尾节点执行插入/删除操作的效率比ArrayList要高

由于ArrayList每次扩容的容量是当前的1.5倍，所以LinkedList所占的内存空间要更小一些

数组存储区间是连续的，占用内存严重。数组的特点是：寻址容易，插入和删除困难；链表存储区间离散，占用内存比较宽松。链表的特点是：寻址困难，插入和删除容易。

二者的遍历效率接近，但需要注意，遍历LinkedList时应用iterator方式，不要用get(int)方式，否则效率会很低

--------Vector，队列，线程安全的

------------stack

----Set，不含重复元素。相当于只存储Key的Map。

Map

----HashMap，线程不安全，轻量级HashTable。

HashMap是由数组+链表的一个结构组成。

----HashTable，线程安全，是HashMap的前身。

----TreeMap

1.1.2 异常

Throwable

----Error，是程序无法处理的错误，表示运行应用程序中较严重问题。

StackOutMemoryError – 内存溢出

----Exception

--------运行时异常（非检查异常），Java编译器不会检查它，编译会通过。

OutMemoryException - 内存不足

NullPointerException - 空指针引用异常  
ClassCastException - 类型强制转换异常。

IndexOutOfBoundsException - 下标越界异常

ArithmeticException - 算术运算异常  
NumberFormatException - 数字格式异常

--------非运行时异常（检查异常），IOException、SQLException等以及用户自定义的Exception异常

1.1.3 垃圾回收机制

解决的问题：程序员容易忘记回收不再使用的对象，从而造成内存泄漏。

解决方案：java语言设计了个垃圾回收机制，自动回收。

怎么判断需要回收： GC通过确定对象是否被活动对象引用来确定是否收集该对象。

主动回收：执行 system.gc()方法。

内存泄漏（Memory Leak）是指程序中已动态分配的堆内存由于某种原因程序未释放或无法释放，造成系统内存的浪费，导致程序运行速度减慢甚至系统崩溃等严重后果。

两个经典算法：引用计数法，和[可达](https://www.baidu.com/s?wd=%E5%8F%AF%E8%BE%BE&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)性分析算法。方法有System.gc()和finalize()。

1. 什么时候：GC时和老年代与永久代空间不足时。
2. 执行 system.gc()的时候
3. 老年代空间不足，一次Full GC 之后，然后不足 会触发 java.outofmemoryError:java heap space
4. 永久代空间不足    永生代或者永久代， java.outofMemory  PerGen Space
5. minor之后 survior放不下，放入老年代，老年代也放不下，触发FullGC, 或者新生代有对象放入老年代，老年代放不下，触发FullGC
6. 新生代晋升为老年代时候，老年代剩余空间低于新生代晋升为老年代的速率，会触发老年代回收
7. new 一个大对象，新生代放不下，直接到老年代，空间不够，触发

注：FullGCFull==MajorGC

Minor GC:新生代垃圾回收，非常频繁，一般速度比较快   
Major Gc：老年代垃圾回收，经常会伴随一次MinorGC，一般比MinorGC慢10倍以上

2）对什么东西

从root搜索不到，而且经过第一次标记、清理后，仍然没有复活的对象。

3）做了什么（即，垃圾收集算法）：清除和是否可恢复

1. 标记—清除算法：简单地标记且清除

两个阶段：标记，清除；

不足：效率问题；空间问题(会产生大量不连续的内存碎片)

1. 复制算法：备份且清除

将可用内存按容量分为大小相等的两块，每次都只使用其中一块；

不足：将内存缩小为了原来的一半

新生代

1. 标记—整理算法：留存活地标记且清除

标记，清除（让存活的对象都向一端移动）

1.1.4 Java四个基本特性

抽象、封装、继承、多态。

1.1.5 Redis五个数据类型

字符串类型（string），散列类型（hash），列表类型（list），集合类型（set），有序集合类型（zset）

1.1.5 StringBuffer和StringBuild的区别：同步锁

StringBuffer使用了同步锁，线程安全但效率低；StringBuild未使用同步锁效率高。

1.1.6 事务管理

1）四个基本特性：原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)、持久性(Durability)

原子性是指事务个体；一致性是指事务操作过程中数据使用上；隔离性是指事务个体之间；持久性是指数据的存储。

1. 七大传播特性
2. 英文：

Propagation：PROPAGATION，传播。

Required：REQUIRED，必须地。

Supports：SUPPORTS，支持。

Mandatory：MANDATORY，强制性的。

never： NEVER，绝不会

nested：NESTED，嵌套。

1. 须不需、该不该、有没有、新与此、在不在、挂与抛及需TransactionManager、嵌套。

| **传播行为** | **简述** | **含义** |
| --- | --- | --- |
| PROPAGATION\_REQUIRED | 必须地  须且无则在新 | 表示当前方法必须运行在事务中。如果当前事务存在，方法将会在该事务中运行。否则，会启动一个新的事务 |
| PROPAGATION\_SUPPORTS | 支持地  不需且有则在此 | 表示当前方法不需要事务上下文，但是如果存在当前事务的话，那么该方法会在这个事务中运行 |
| PROPAGATION\_MANDATORY | 强制使用  无则抛 | 表示该方法必须在事务中运行，如果当前事务不存在，则会抛出一个异常 |
| PROPAGATION\_REQUIRED\_NEW | 必须启新  1、须且新启而在  2、方法执行而事务挂起  3、TransactionManager来访问 | 表示当前方法必须运行在它自己的事务中。一个新的事务将被启动。如果存在当前事务，在该方法执行期间，当前事务会被挂起。如果使用JTATransactionManager的话，则需要访问TransactionManager |
| PROPAGATION\_NOT\_SUPPORTED | 不支持  1、不该且有则挂起  2、TransactionManager来访问 | 表示该方法不应该运行在事务中。如果存在当前事务，在该方法运行期间，当前事务将被挂起。如果使用JTATransactionManager的话，则需要访问TransactionManager |
| PROPAGATION\_NEVER | 绝不，未曾，不会发生。  不该且在则抛 | 表示当前方法不应该运行在事务上下文中。如果当前正有一个事务在运行，则会抛出异常 |
| PROPAGATION\_NESTED | 嵌套 | 表示如果当前已经存在一个事务，那么该方法将会在嵌套事务中运行。嵌套的事务可以独立于当前事务进行单独地提交或回滚。如果当前事务不存在，那么其行为与PROPAGATION\_REQUIRED一样。注意各厂商对这种传播行为的支持是有所差异的。可以参考资源管理器的文档来确认它们是否支持嵌套事务 |

3）五大隔离级别

|  |  |  |
| --- | --- | --- |
| ISOLATION\_DEFAULT | 默认 | 使用后端数据库默认的隔离级别 |
| ISOLATION\_READ\_UNCOMMITTED | 未提交  脏读（回滚） | 最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读 |
| ISOLATION\_READ\_COMMITTED | 已提交  不可重复读（数据行） | 允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生 |
| ISOLATION\_REPEATABLE\_READ | 重读  幻读（事务间并行，增删与条件） | 对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生 |
| ISOLATION\_SERIALIZABLE | 序列化  脏读、不可重复读和幻读全解决 | 最高的隔离级别，完全服从ACID的隔离级别，确保阻止脏读、不可重复读以及幻读，也是最慢的事务隔离级别，因为它通常是通过完全锁定事务相关的数据库表来实现的 |

1. 脏读、不可重复读和幻读
2. 脏读：行数据出问题，是回滚（虽然都是读取事务缓存，但不回滚则不会发生）过程行数据修改后被其他并行的事务执行过程中读取。
3. 不可重复读：行数据问题，是多次读取过程中行数据被修改，前后数据不一致。
4. 幻读：所有行数据，事务间并行造成的，一般发生在增删和条件被修改时。
5. 对应级别：默认使用数据库级别。
6. 未提交：发生脏读
7. 提交：不读取未提交的数据，一般数据库使用此级别
8. 只读：保证当前事务的原子性
9. 串行化：把并行改成串行

解析：隔离级别：

1. 脏读是回滚过程数据修改后被其他并行的事务执行过程中读取，不可重复读和幻读是多次读取过程中的数据被其他事务做修改造成前后数据不一致!

嵌套使用了一个单独的事务，这个事务 拥有多个可以回滚的保存点。

都是因为共享事务缓存造成的。

深入理解的关键点：缓存与数据库、提交与未提交（回滚）、事务并行与串行、行与表

1. 脏读、不可重复读和幻读
2. 脏读：行数据出问题，是回滚（虽然都是读取事务缓存，但不回滚则不会发生）过程行数据修改后被其他并行的事务执行过程中读取。
3. 不可重复读：行数据问题，是多次读取过程中行数据被修改，前后数据不一致。
4. 幻读：所有行数据，事务间并行造成的，一般发生在增删和条件被修改时。
5. 对应级别：默认使用数据库级别。
6. 未提交：发生脏读
7. 提交：不读取未提交的数据，一般数据库使用此级别
8. 只读：保证当前事务的原子性
9. 串行化：把并行改成串行
10. 一般描述
11. 脏读：读取来未提交的修改数据
12. 不可重复读：多次读取时，数据被修改造成前后读取不一致
13. 幻读：增删记录造成记录数的读取发生不一致情况。

1.1.7 常见注解：注解是IoC容器各种扫描并搜集的设定。

声明Bean类：

@Mapper @Repository（持久层）、@Service(业务层)、@Controller (控制层)和 @Componen[t](https://www.cnblogs.com/lxcmyf/p/5674399.html)（其它）

注入Bean类：@Autowired 和@Resource，

Java配置类：

@Configuration 声明当前类为配置类，相当于xml形式的Spring配置（类上）

@Bean 注解在方法上，声明当前方法的返回值为一个bean，替代xml中的方式（方法上）

@Configuration 声明当前类为配置类，其中内部组合了@Component注解，表明这个类是一个bean（类上）

@ComponentScan 用于对Component进行扫描，相当于xml中的（类上）

其他：

@EnableScheduling 在配置类上使用，开启计划任务的支持（类上）

@EnableTransactionManagement 开启注解式事务的支持

1.1.8 HTTP协议，GET和POST 的区别

Get：接收，挣得，主观地自己动手；Post：邮寄，投递，第三方代理。

答：Http定义了与服务器交互的不同方法，最基本的方法有4种，分别是GET，POST，PUT，DELETE。URL全称是资源描述符，我们可以这样认为：一个URL地址，它用于描述一个网络上的资源，而HTTP中的GET，POST，PUT，DELETE就对应着对这个资源的查，改，增，删4个操作。到这里，大家应该有个大概的了解了，GET一般用于获取/查询资源信息，而POST一般用于更新资源信息。

根据HTTP规范，GET用于信息获取，而且应该是安全的和幂等的。

1. .所谓安全的意味着该操作用于获取信息而非修改信息。
2. .幂等的意味着对同一URL的多个请求应该返回同样的结果。

根据HTTP规范，POST表示可能修改变服务器上的资源的请求。

说完原理性的问题，我们再从表面现像上面看看GET和POST的区别：

1.GET请求的数据会附在URL之后（就是把数据放置在HTTP协议头中），以?分割URL和传输数据，参数之间以&相连。POST把提交的数据则放置在是HTTP包的包体中。

2."GET方式提交的数据最多只能是1024（特定的浏览器及服务器对URL它的限制）字节，理论上POST没有限制（起限制作用的是服务器的处理程序的处理能力）。

3.POST的安全性要比GET的安全性高。

框架

1、Spring

1.1 原理：

IoC:  
概念：控制权由对象本身转向容器；由容器根据配置文件去创建实例并创建各个实例之间的依赖关系 核心：bean工厂；

控制权，是指对象的实例创建权与实例之间的依赖关系的控制权。

AOP的主要原理：Spring AOP使用的是动态代理。所谓的动态代理，就是说AOP框架不会去修改字节码，而是在内存中临时为方法生成一个AOP对象，这个AOP对象包含了目标对象的全部方法，并且在特定的切点做了增强处理，并回调原对象的方法。

内部最核心的就是IOC了， 动态注入，让一个对象的创建不用new了，可以自动的生产，这其实就是利用java里的反射 反射其实就是在运行时动态的去创建、调用对象，Spring就是在运行时，跟xml Spring的配置 文件来动态的创建对象，和调用对象里的方法的 。  
Spring还有一个核心就是AOP这个就是面向切面编程，可以为某一类对象 进行监督和控制（也就是 在调用这类对象的具体方法的前后去调用你指定的模块）  
从而达到对一个模块扩充的功能。这些都是通过 配置类达到的。  
Spring目的：就是让对象与对象（模块与模块）之间的关系没有通过代码来关联，都是通过配置类说明 管理的（Spring根据这些配置 内部通过反射去动态的组装对象）  
切记：Spring是一个容器，凡是在容器里的对象才会有Spring所提供的这些服务和功能。

2、SpringBoot

2.1、什么是 Spring Boot？

Spring Boot 是 Spring 开源组织下的子项目，是 Spring 组件一站式处理方案，主要是简化了使用 Spring 的难度，简省了繁重的配置，提供了各种启动器，开发者能快速上手。

2.2、为什么要用 Spring Boot？

Spring Boot 优点非常多，如：

独立运行

简化配置（项目中，手动的配置文件与配置类）

自动配置（springboot的约定，已约定好的配置）

无代码生成和XML配置

应用监控

上手容易

-快速创建独立运行的spring项目与主流框架集成   
-使用嵌入式的servlet容器，应用无需打包成war包   
-starters自动依赖与版本控制   
-大量的自动配置，简化开发，也可修改默认值   
-准生产环境的运行应用监控   
-与云计算的天然集成

Spring Boot 集这么多优点于一身，还有理由不使用它呢？

注：约定优于或大于配置，约定指的是“自动配置”，配置指的是“简化配置”

2.3、Spring Boot 的核心配置文件有哪几个？它们的区别是什么？

Spring Boot 的核心配置文件是 application 和 bootstrap 配置文件。

application 配置文件这个容易了解，主要用于 Spring Boot 项目的自动化配置。

bootstrap 配置文件有以下几个应用场景。

使用 Spring Cloud Config 配置中心时，这时需要在 bootstrap 配置文件中增加连接到配置中心的配置属性来加载外部配置中心的配置信息；

少量固定的不能被覆盖的属性；

少量加密/解密的场景；

2.4、Spring Boot 的配置文件有哪几种格式？它们有什么区别？

.properties 和 .yml，它们的区别主要是书写格式不同。

1).properties

IMG_256

2).yml



另外，.yml 格式不支持@PropertySource注解导入配置。

2.5、Spring Boot 的核心注解是哪个？它主要由哪几个注解组成的？

启动类上面的注解是@SpringBootApplication，它也是 Spring Boot 的核心注解，主要组合包含了以下 3 个注解：

1. @SpringBootConfiguration：组合了 @Configuration 注解，实现配置文件的功能。
2. @EnableAutoConfiguration：打开自动配置的功能，也可以关闭某个自动配置的选项，如关闭数据源自动配置功能： @SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })。
3. @ComponentScan：Spring组件扫描。

2.6、开启 Spring Boot 特性有哪几种方式？

1）继承spring-boot-starter-parent项目

2）导入spring-boot-dependencies项目依赖

2.7、Spring Boot 需要独立的容器运行吗？

可以不需要，内置了 Tomcat/ Jetty 等容器。

2.8、运行 Spring Boot 有哪几种方式？

1）打包用命令或者者放到容器中运行

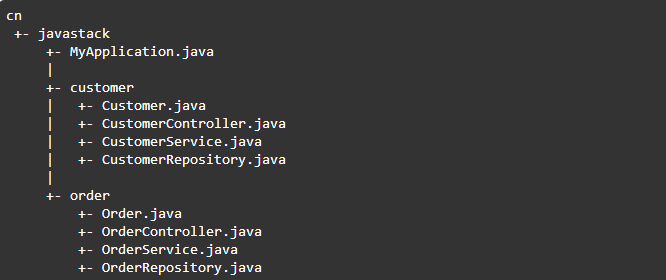
2）用 Maven/ Gradle 插件运行

3）直接执行 main 方法运行

2.9、Spring Boot 自动配置原理是什么？

注解 @EnableAutoConfiguration, @Configuration, @ConditionalOnClass 就是自动配置的核心，首先它得是一个配置文件，其次根据类路径下能否有这个类去自动配置。

2.10、Spring Boot 的目录结构是怎么的？



这个目录结构是主流及推荐的做法，而在主入口类上加上 @SpringBootApplication 注解来开启 Spring Boot 的各项能力，如自动配置、组件扫描等。

2.11、你如何了解 Spring Boot 中的 Starters？

以spring-boot-starter-XXX命名的，代表了一个特定的应用类型；且第三方的启动器不能以spring-boot开头命名，它们都被Spring Boot官方保留。

Starters可以了解为启动器，它包含了一系列可以集成到应用里面的依赖包，你可以一站式集成 Spring 及其余技术，而不需要四处找示例代码和依赖包。如你想使用 Spring JPA 访问数据库，只需加入 spring-boot-starter-data-jpa 启动器依赖就能使用了。

Starters包含了许多项目中需要用到的依赖，它们能快速持续的运行，都是一系列得到支持的管理传递性依赖。

2.12、如何在 Spring Boot 启动的时候运行少量特定的代码？

可以实现接口 ApplicationRunner 或者者 CommandLineRunner，这两个接口实现方式一样，它们都只提供了一个 run 方法。

SpringApplicationBuilder需要更多的代码。

2.13、Spring Boot 有哪几种读取配置的方式？

Spring Boot 可以通过 @PropertySource,@Value,@Environment, @ConfigurationProperties 来绑定变量。

2.14、Spring Boot 支持哪些日志框架？推荐和默认的日志框架是哪个？

Spring Boot 支持 Java Util Logging, Log4j2, Lockback 作为日志框架，假如你使用 Starters 启动器，Spring Boot 将使用 Logback 作为默认日志框架，

2.15、SpringBoot 实现热部署有哪几种方式？

主要有两种方式：

Spring Loaded

Spring-boot-devtools

Spring-boot-devtools?

2.16、你如何了解 Spring Boot 配置加载顺序？

在 Spring Boot 里面，可以使用以下几种方式来加载配置。

1）properties文件；

2）YAML文件；

3）系统环境变量；

4）命令行参数；

等等……

2.17、Spring Boot 如何定义多套不同环境配置？

提供多套配置文件，如：



运行时指定具体的配置文件

2.18、Spring Boot 可以兼容老 Spring 项目吗，如何做？

可以兼容，使用@ImportResource注解导入老 Spring 项目配置文件。

2.19、保护 Spring Boot 应用有哪些方法？

1. 在生产中使用HTTPS
2. 使用Snyk检查你的依赖关系
3. 更新到最新版本
4. 启用CSRF保护
5. 使用内容安全策略防止XSS攻击

2.20、springboot常用的starter有哪些   
spring-boot-starter-web 嵌入tomcat和web开发需要servlet与jsp支持   
spring-boot-starter-data-jpa 数据库支持   
spring-boot-starter-data-redis redis数据库支持   
spring-boot-starter-data-solr solr支持   
mybatis-spring-boot-starter 第三方的mybatis集成starter

1、springboot

敏捷开发

内置tomcat

无需xml配置

2、搭建

添加依赖包：利用了maven的继承和依赖特性！

parent，把基本包都添加了

web，添加了web应用所需包

3、创建服务

1）服务即是被作为请求的接口

2）@restController=@Controller+@RespondBody，返回josn字符串

3）添加@SpringbootApplication或者@SpringBootConfigeration+@EnableAutoConfigeration+@ComponentScan便可运行了

有此注解便可注入到springboot容器，添加到tomcat，加载运行springboot

4）写springcloud基本上是在写springmvc

* 1. 其它技术 高并发、MQ、Redis

1.2.1 Maven

常用命令：maven clean、maven war（jar）

Eclipse中的Update Project菜单来更新外部依赖的jar包。

1.2.2Linux常用命令

1. 查看已启动服务的命令: ps -ef|grep XXX

Ps: 进程状态显示，p, process , s，stat；e：均显示；f: flag； grep : 正则表达式，查找。

ps -ef|grep java 或者 ps -ef | grep 服务名称（wmsmem）

1. 通过查看日志验证启动是否正常：tail -fn 500 loginfo.log或startServer.log

Tail：尾部；f: follow; n: 行数；

找到ingfo.log或startServer.log文件(在/data/logs/目录下)，执行命令 tail -fn 500 loginfo.log或tail -fn500 startServer.log

1. Linux删除文件：rm –rf XXX；:sudo rm –rf XXX

rm: remove, 删除; r: recursive 递归删除; f: force 无需确认

直接rm就可以了，不过要加两个参数-rf 即：rm -rf 目录名字。若删除时出现 Permission denied 的提示，可以在命令前加sudo 即:sudo rm -rf 文件夹的名字，即可删除。

提醒：使用这个rm -rf的时候一定要格外小心，linux没有回收站的。rm还有更多的其他参数和用法，具体参数用法使用man rm查看

1. 启动或关闭服务：sudo ./startup.sh或./shutdown.sh

Sudo: 获取权限，即，以系统管理者的身份执行指令。

升级时只升级一个项目，则只重启此项目，不用把此服务器上的所有项目都重启了。启动命令为sudo ./startup.sh &（管理员后台执行，代替shell），关闭命令直接sudo ./shutdown.sh

1. 编辑文本

W：write；q: quit; x:

vi, 进入编辑模式；：wq强制写入并退出；：x：写入并退出; esc: 输入模式和命令模式切换。

q!: 强制退出。

1.2.3 高并发（多线程）

1．2.3.1常用锁：

（1）Synchronized，wait(),sleep(),notifyAll()方法（唤醒所有wait线程）或者notify()方法（只随机唤醒一个wait线程）。

Synchronized：同步，是java做修饰的内置特性，自动释放锁。

（2）Lock类：更多的功能

（3）volatile变量可以确保线性关系，即，为了解决多线程对同一个变量缓存一致性问题，写操作会发生在后续的读操作之前，但它不能保证原子性。

注：如果使用 synchronized ，如果A不释放，B将一直等下去，不能被中断。  
 如果使用ReentrantLock（可重入锁），如果A不释放，可以使B在等待了足够长的时间以后，中断等待，而干别的事情

1.2.3.2

1. 1、用Runnable还是用Thread

大家知道我们可以通过继承Thread类或者调用Runnable接口来实现线程，问题是，哪个方法更好呢？什么情况下使用哪种呢？如果你要继承其他的类，就实现Runnable接口

2、如何避免死锁？

java多线程中的死锁：是指两个或者两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。这是一个严重的问题，因此死锁会让你的程序挂起无法完成任务，死锁的发生必须满足一下四个条件

1. 互斥条件：一个资源每次只能被一个进程使用
2. 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放
3. 不剥夺条件：进程已获得的资源，在未使用完之前，不能强行剥夺
4. 循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系

避免死锁最简单的方法就是阻塞循环等待条件，将系统中所有资源设置标志位、排序，规定所有的进程申请资源必须以一定的顺序（升序或者降序）做操作来避免死锁

3、乐观锁和悲观锁的理解及如何实现，有哪些实现方式？

1. 乐观锁，每次操作时不加锁而是假设没有冲突而去完成某项操作，如果因为冲突失败就重试，直到成功为止

乐观锁可以使用volatile+CAS原语实现，带参数版本来避免ABA问题，在读取和替换的时候进行判定版本是否一致。

1. 悲观锁是会导致其它所有需要锁的线程挂起，等待持有锁的线程释放锁。

悲观锁可以使用synchronize的以及Lock。

4、同步和异步有何不同，在什么情况下分别使用它们？举例说明

1. 同步交互：如果数据将在线程间共享。

指发送一个请求，需要等待返回，然后才能发送下一个请求，有个等待的过程

例如：正在写的数据以后可能会被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就是共享数据，必须进行同步存取

1. 异步交互：指发送一个请求，不需要等待返回，随时可以再发送下一个请求，即不需要等待。

当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效。

1. 区别：是否需要共享；是否需要等待。

一个需要等待，一个不需要等待。

1.3.3 Java面试总结如何处理项目的高并发、大数据

1.HTML静态化：Freemarker、thymeleaf和Velocity

如果网站的请求量过大，我们可以将页面静态化提供访问来缓解服务器压力，能够缓解服务器压力加大以及降低数据库数据的频繁交换。适合于某些访问了过大，但是内容不经常改变的页面，如首页、新闻页等

注：Thymeleaf和vue不是一类事务。

模板引擎：Thymeleaf、freemarker、Velocity、JSP。

前端框架：vue、angularjs、react。

2.文件服务器

[顾名思义](https://www.baidu.com/s?wd=%E9%A1%BE%E5%90%8D%E6%80%9D%E4%B9%89&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)，文件服务器就是将文件系统单独拿出来提供专注于处理文件的存储访问系统，甚至于对个文件服务器。因为对于图片这种资源的访问存储是web服务最耗资源的地方，将文件服务器单独部署既可以将压力转移，交给专门的系统处理，又可以分担风险，如果图片服务器出现问题，那么主服务器能够保证正常，顶多就是文件请求不到。

3.负载均衡

负载均衡将是大型网站解决高负荷访问和大量并发请求采用的终极解决办法。

负载均衡建立在现有网络结构之上，它提供了一种廉价有效透明的方法扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。其原理就是将大量工作分摊到多个操作单元上进行执行，例如Web服务器、FTP服务器、企业关键应用服务器和其它关键任务服务器等，从而共同完成工作任务。

4、高并发情况下的解决方案

1. 应用程序和静态资源文件进行分离及HTML静态化
2. 页面数据缓存
3. 集群与分布式
4. 分批传送
5. 应用程序和静态资源文件进行分离

所谓的静态资源就是我们网站中用到的Html、Css、Js、Image、Video、Gif等静态资源。应用程序和静态资源文件进行分离也是常见的前后端分离的解决方案，应用服务只提供相应的数据服务，静态资源部署在指定的服务器上（Nginx服务器或者是CDN服务器上），前端界面通过Angular JS或者Node JS提供的路由技术访问应用服务器的具体服务获取相应的数据在前端游览器上进行渲染。这样可以在很大程度上减轻后端服务器的压力。例如，百度主页使用的图片就是单独的一个域名服务器上进行部署的

1. 页面数据缓存

页面缓存是将应用生成的很少发生数据变化的页面缓存起来，这样就不需要每次都重新生成页面数据了，从而节省大量CPU资源，如果将缓存的页面数据放到内存中速度就更快。

可以使用Nginx提供的缓存功能，或者可以使用专门的页面缓存服务器Squid。

1. 集群与分布式

 数据库集群

        当面对复杂的应用，用户大量访问的时候，一台数据很快无法满足需求，于是我们需要使用数据库集群或者库表散列。

我们在应用程序中安装业务和应用或者功能模块将数据进行分离，不同的模块对应不同的数据库或表，再按照一定的策略对某个页面或者功能进行更小的数据库散列。

1. 分批传送

        在做某项目的时候，一次传递的参数太多，而且数据库规定一次最多传递的参数最多是三万条，当时有五万条记录，那怎么传送呢？最终是分批传送，电梯里一次乘不下那么多的人，会报超重的bug，那就分批把人送上去。

        还有一次在考试系统中，如果那么多的考试人员同时提交到数据库中，数据库的压力增大，有时会被down掉，当时采用的方法是使用ajax异步传输，没有等待考生点击提交按钮的时候，就把考生的答案自动提交，这样也避免了突然断电考生前面做过的题出现丢失的现象。

5、线程相关概念：

1. 进程和线程概念：

进程，是正在运行的应用程序，是线程的集合！

线程，是程序执行流的最小单位！

1. 多线程，是为了提高执行效率，对进程做的分割集合！

多线程应用场景：ajax异步、上传下载、MQ、聊天工具等！

1. 同步和异步

同步，是按照顺序执行，有共享数据时，需要等待！

异步，是多个路径执行，无数据共享或数据操作不冲突时，不需要等待！

1. 最好使用接口实现来创建线程，因为java最好是面向接口编程，并且接口可以多个继承！
2. 守护线程（gc线程）会随主线程一起销毁；非守护线程（如，由主线程创建的自定义用户线程）与主线程互不影响而不会销毁！
3. join方法，是让出CPU执行权的功能！

6、线程的其他知识点

1. 线程安全问题

多个线程间共享一个全局变量，在写的时候被其他线程干扰造成的数据冲突问题！

1. This锁和字节码文件锁

非静态同步函数用的是this锁，静态同步函数用的是字节码文件锁！

1. 死锁：

多线程同步中，某一线程锁无法释放，其他线程一直等待，造成的现象叫做死锁！

1. 内存模型：

主内存和本地私有内存

1. DB优化

        a、在数据库设计的时候就要考虑到后期的维护，数据库三范式是我们设计数据库索要遵循的原则。

        b、索引的建立：经常被查询且很少被增改。

建立索引要适当，如果一个表经常用来被查询，对于增加和修改很少被用到，我们就可以为这个表建立索引，因为对于增加和修改和删除操作时，我们对索引的维护要大大超过索引给我们带来的效率。

        c、表字段的类型选择要恰当

包括字段的长度、类型等，要根据实际存储的数据进行选择，长度不要过长，否则会影响效率。

        d、外键要慎用，因为主键代表这一张表，而外键代表一群表，对表之间进行了关联，在删除修改等需要我们关联。

        e、在数据库操作上

                尽量使用prepareStatement，少用Statement，因为PrepareStatement是进行预编译的。

                connection设置为readOnly，Connection是对数据库连接，属于重量级，我们使用即可。

                连接池的使用，我们可以修改数据库默认的连接数。

1. 数据库
   1. 三大数据库
      1. MySQL的五大引擎

InnoDB 、MyIsam （ISAM索引方法–索引顺序存取方法）、Memory（也叫HEAP，堆内存）、Mrg\_Myisam：（分表的一种方式–水平分表）、Blackhole（黑洞引擎）

* 1. 性能调优

1. 大的调优：
2. 读写分离。
3. 增加缓存
4. 分库
5. 分区
6. 拆分表（垂直<拆分列，关联表>和水平<拆分行，表结构一样>）。
7. SQL语句调优：
8. 加索引
9. 避免用\*号
10. 尽量用大写
11. 选取最适用的字段属性
12. 使用连接（JOIN）来代替子查询(Sub-Queries)
13. 使用联合(UNION)来代替手动创建的临时表
14. 不鼓励使用like操作，非要用，尽量不加后%号。
15. 不要在列上进行运算
16. 不使用NOT IN和<>操作
    1. 使用技巧
17. 多线程（高并发）
18. 前端

3.1 视图

3.2 脚本语言

3.3 其它

1. 需求

药帮忙业务流程：

1、角色有：供应商、业务部门、收货员、资料员、验收员、上架员、质管员。

2、单据：采购订单、预约送货单、待处理单、收货单、拒收单、体积维护单、资料维护单、验收单、复查单、监管码扫描单、入库单、入库修改单。

3、业务部门制定采购订单，审核通过后发送给供应商，供应商接收到订单后准备送货，先发送预约送货号，收货员收到预约单号接收来货，收货员在商品提取或单据提取时，如果发现有异常便交给业务员联系供应商，处理不通过直接拒收，处理通过再重新发送预约单号再收货，如果没有发现异常，新建收货单，提取在途订单，记录批号、数量等，关联容器编号，确认收货结论进行收货或拒收，如果体积信息不完整生成体积维护单，如果基础信息不完整生成资料维护单，由资料员补充完信息后再生成验收单，如果都完整则直接生成验收单，验收环节中由验收员核对商品批号、数量、容器编号，评定结论为待处理、拒收、挂起、合格，待处理就生成复查单，拒收生成拒收单，挂起生成挂起单，合格时如果需要电子监管码扫描生成监管码扫描单，不需要则直接生成入库单；复查环节中由质管员审核是否有误，评定结论为拒收、合格和不合格，拒收生成拒收单，合格入合格库且也需要走是否需要电子监管码扫描，不合格入待处理库；入库环节，由上架员审核如果有误生成入库修改单再走验收环节，如果无误则做上架确认，然后库存更新，也就结束了入库流程。

1. 设计模式

4.1 什么叫做设计模式

是设计过程中科院反复使用、可以解决特定问题的设计方法。

1. 架构

1、架构、框架和设计模式的区别：

架构（整体结构和组件及设计方案）---🡪框架（组件和解决方案）---🡪设计模式（复用性的解决方案）！

软件或项目的程序代码和完成基于设计模式，设计模式基于框架，框架基于架构。

2、具体描述：

架构是关于软件如何设计的策略和抽象解决方案，框架是一种特殊的软件，是半成品。

设计模式：复用性解决方案。

是在某种特定上下文中针对一个软件生命周期中出现的问题而给出的多次适用的解决方案。

框架：复用的软件及软件结构。

框架是一组软件组件，它们互相协作提供了针对某个给定的问题领域中的应用程序所用到的一种可复用的体系结构。

架构：本软件（或项目）的整体描述，系统比较详细的设计。

又名软件架构，是有关软件整体结构与组件的抽象描述，用于指导大型软件系统各个方面的设计。简单的说架构就是一个蓝图，是一种设计方案，将客户的不同需求抽象成为抽象组件，并且能够描述这些抽象组件之间的通信和调用。

架构（动词）>框架>设计模式。

软件通过架构，可以设计出很多不同的框架。在一个框架中，也可以使用很多的设计模式。设计模式不是哪儿哪儿都可以用的，只有当出现了某一特定的问题时，才利用设计模式去解决。设计模式不是用的越多越好，在维护的时候，过多的设计模式会极大的增添维护成本。

1. 其它