|  |
| --- |
| JDK动态代理和CGLIB代理的区别  JDK动态代理只能对实现了接口的类生成代理，而不能针对类  CGLIB是针对类实现代理，主要是对指定的类生成一个子类，覆盖其中的方法（继承） |
| Spring MVC的执行流程     1. 一个请求匹配前端控制器DispatcherServlet的请求映射路径，容器将该请求转交给DispatcherServlet处理。 2. DispatcherServlet接收到请求后，将根据请求信息交给处理器映射器HandlerMapping 3. HandlerMapping根据用户的url请求，查找匹配该url的Handler，并返回一个执行链 4. DispatcherServlet再请求处理器适配器HandlerAdapter调用响应的Handler进行处理并返回ModelAndView给DispatcherServlet 5. DispatcherServlet将ModelAndView请求ViewResolver视图解析器解析，返回具体view 6. DispatherServlet将View进行渲染视图 7. DispathcherSevlet将页面响应给用户 |

|  |
| --- |
| 简述一下Spring的IOC 和 DI Aop 各是什么？  IOC 控制反转，这不是什么技术，而是一种思想。使用IOC的思想意味着你将设计好的对象交个容器管理，而不是传统的在你的对象内部直接控制。  谁在控制谁？——一般来讲，我们直接new一个对象，是我们运行的这个程序去主动的创建依赖对象，但是IOC会有一个IOC容器来负责这些对象的创建。这时候IOC容器控制了对象，控制了外部资源获取。  总结：IOC的基本思想，将对象的创建提取到外部，由外部容器提供所需要的组件  为什么使用IOC呢？  官方一点的说法是，高内聚，低耦合。解释：有了IOC容器后，把创建和查找依赖对象的控制权交给了容器，由容器进行注入组合对象，所以对象与对象之间的耦合是松散的，这样也方便测试，利于功能复用，更重要的是使得程序的整体体系结构变得非常灵活  IOC相当于是将应用程序赋予给了一个中心，IOC容器。  依赖：在A类里面创建了B类的实例，这样A依赖于B  DI依赖注入，各个组件的依赖关系是由容器来注入的。开发者依赖这个机制，只需要通过配置指定资源，完成业务逻辑即可，不用关心具体的资源来自何方，由谁实现（依然是面向对象的思想）  你说说这个DI和IOC他们有区别嘛？  有区别，但是其实DI和IOC说的是一个东西，IOC是我们想要做到事情，DI是我们采取的手段。  IOC的思想最核心的地方在于，资源不由使用资源的双发管理，而由不适用资源的第三方管理，这可以带来很多好处， 第一、资源集中管理，实现资源的可配置和易管理。  第二：降低了使用资源双方的依赖程度，也就是我们说的耦合度  AOP面向切面编程  在面向对象思想里面有三大法宝：封装、继承、多态，用这三大法宝建立了对象的层次，但是它是一个纵向的结构，有比较明确的上下级关系，但是在平行的层次中发挥不出太大的作用。  所以我们需要一种横向的结构来定义这些平行的关系，以日志记录为例，记录日志一般要在操作完毕之后，用OOP的思想，那我就需要去项目的每个操作数据库的方法的里面加上保存日志，这会导致代码重复，模块的重用会因此收到影响。  使用AOP，只需要自定义方法，在Spring配置文件中将该方法配置好，当每次调用完原先的save方法时，都会执行你写的保存日志方法。它将程序中的交叉业务逻辑，封装成一个切面，然后注入到目标对象中去。 |

|  |
| --- |
| Cookie和Session的区别，分布式怎么保证用户状态？  Cookie数据存放在客户的浏览器上，session数据放在服务器上  Cookie不是很安全，别人可以分析存放本地的Cookie并进行cookie欺骗，考虑到安全应当使用session。  Session会在一定时间内保存在服务器上，当访问增多，会比较占用服务器的性能  单个cookie保存的数据不能超过4k，  所以登陆信息等重要信息存放在session中，其他信息如需保留，可以放在cookie中。  Session共享问题，  SpringSession 将Session状态保存到外部Session存储中，如redis.设计一个Filter，利用HttpServletRequestWrapper实现自己的getSession（）方法，接管创建和管理session数据的工作。  配置一行spring.session.store-type = redis |

|  |
| --- |
| MySQL中有那些锁？怎么编写   1. 表级锁：开销小，加锁快；粒度（细分程度）大，并发度最低 2. 行级锁：开销大，加锁慢；粒度小，并发度也最高 3. 页面锁：开销中间，都是一般 |

|  |
| --- |
| Mybatis怎么处理结果集  使用association标签来解决一对一的关联查询，association标签可用的属性  Property：对象属性的名称  javaType: 对象属性的类型  column：所对应的外键字段名称  select 使用另一个查询封装结果  使用collection标签来解决一对多的关联查询，ofType属性指定集合中元素的对象类型。 |

|  |
| --- |
| Linux常用指令  Sudo：切换用户  Hostname：显示主机名  Uname：显示系统信息 -a 显示完整的系统信息  Ipconfig 查看或者配置网卡信息  Ping:测试与目标主机连接情况  Kill ：杀死进程  ls：显示文件或目录信息  mkdir：当前目录下创建一个文件夹  rm 删除 mv 移动目录 cat 查看目录  管道 | 压缩 tar -xzvf filename.tar.gz  Grep: 强大的搜索工具。  Vim：程序编辑器 |

|  |
| --- |
| SQL优化  锁有查询不能用 \* 号代替  所有查询必须有边界。  所有分页查询，必须有序  所有查询的条件和排序的条件，必须有索引  能用连接查询的，别用子查询  Or条件，尽量避免  能用where过滤的数据，绝对不要使用having过滤。 |

|  |
| --- |
| 常见的异常  ClassNotFoundException 加载类时，指定的路径找不到  illegalargumentException 非法参数，illegalStatusException非法状态（Openfeign调用没加注解）  SQLException 操作数据库异常。  StackOverflowExcetpion 堆栈溢出错误——递归调用构造函数。 |

|  |
| --- |
| Git 最强大的版本控制软件。     1. 系统中任意文件夹都可以作为本地仓库。本地仓库需要先初始化，生成git基本配置 2. 然后把本地的操作添加到缓冲区（add） 3. 最后把缓冲区操作提交到本地库（commit） 4. 最后把本地仓库数据推送到远程仓库。（push）   git 每个人都有分支，如果想要拉取别人的代码例如工具类该怎么办？？？？  Git fetch/ getcheckout 切换分支。 remote 远程。  docker中容器中使用另一个容器怎么办  docker outside of docker docker嵌套技术 可用于一个容器内调用另一个容器内程序。  什么是docker？  Docker 是基于Go语言开发的开源容器项目，他是内核级的虚拟化。  Docker的生命周期  5种状态：Created-初建状态 running-运行状态 stopped-停止状态 paused-暂停状态 deleted-删除状态 |

|  |
| --- |
| Eureka 相关内容  自我保护机制：当15分钟低于85%的Eureka Client 都不心跳，Eureka就会认为Server和Client之间出现了网络问题，就不销毁实例。通过eureka.serve.enable-self-preservation=false关闭。  优雅关机：通过配置shutdown功能即可。    Eureka server：提供服务注册和发现，多个Eureka Server 之间会同步数据，做到状态一致（最终一致性）  Service Provider：服务提供方，将自身服务注册到Eureka，从而使服务消费方能够找到  Service Consumer 服务消费方，从Eureka获取注册服务列表，从而能够消费服务。  CAP定理  C（强）一致性 就是在分布式中，什么时候数据也是相同的，也必须相同。比如银行的转账，  A（可用性）：不管是成功还是失败，在可容忍的范围内，必须有个返回结果  P（分区容错性）也叫最终一致性：就是最后的结果肯定是一样的。  Eureka是满足了AP。  @EnableEurekaServer 启动类上加的一个注解 |

|  |
| --- |
| Ribbon 负载均衡算法   1. 轮询策略（默认） RoundRobinRule 2. 权重轮询策略   配置方式 注入bean |

|  |
| --- |
| openFeign是嵌入applicationClient中的。  流程：ApplicationService 向EurekaServer注册服务  ApplicationClient从EurekaServer中发现服务信息  在ApplicationClient 中调用OpenFeign接口中的方法  ApplicationClient中OpenFeign通过应用程序名调用ApplicationService.  Openfeign接口要加@FeignClient注解  启动类要加@EnableFeignClients  接口中参数要加@RequestParam注解  集合就用@RequestBody Restful方式要用@PathVariable |

|  |
| --- |
| 灾难性雪崩效应：服务链中某一个服务不可用，导致一系列的服务不可用，最终导致真个服务逻辑崩溃。  Hystrix 网非开源的一款容错框架 启动类上添加@EnableCircuitBreaker  降级处理：fallback方法 实现类上：@HystrixCommand(fallbackMethod = ‘myFallback’)  熔断是降级的高级版本：错误请求比例达到一定比例（50%）后，将返送托底数据，并一段时间不访问applicationService，自我保护机制就是一段时间后，他将重试访问applicationService。 配置方式：在降级的基础上添加@HystrixCommand(fallbackMethod=’ myFallback’,commandProperties={ @HystrixProperty(name=’’ ,value = ‘’)})  请求缓存，就是把这个URL地址记录下来，但是Hystrix自带的缓存不好，集群下不能同步。可以利用Spring Cache。@Cache（key=‘’，cacheNames=‘’）  隔离——线程池隔离 信号量隔离  Spring gateWay 就是把各个服务对外提供的API汇聚起来，让外界看起来是一个统一的接口。 配置文件中配置路由即可 route id predicates谓词 path：--------  Gateway限流 令牌桶（解决突刺现象） 使用redis和Lua脚本实现。 |

|  |
| --- |
| 分布式事务存在两大理论依据：CAP定理，BASE理论  CAP定理 上面有， BASE理论就是对CAP定理中的AP进行细致的划分  BA基本可用 就是A 可用性 不管你速度怎么样，一定要有返回，不能宕机，可用的方式就是搭集群。  S 软状态 就是达到最终一致性前的状态，可以不一致  E 最终一致性 就是最后一定要一致。  Tx-LCN L:lock C:confirm N:notify  事务的发起者将会创建一个事务组，然后服务又去请求另一个请求，这样就意味着这个新服务也加入了事务组，都会有confirm的过程，最后事务组就会挨个通知是该回滚还是该提交。  三种模式：  LCN模式：通过代理来实现的，代理的是JDBC中的Connection。在执行SQL语句的时候都是挂起状态，占用这个线程，不能归还给数据库连接池。所以不适用于特别高并发的场景，和偶尔某功能很慢的场景。  优点就是简单，嵌入性低，通过注解或者配置就可以实现。  TCC模式：什么数据都可以 MySQL redis es fastDFS 都行 代码侵入性高，有三个方法try，confirm ，cancel  TXC 财大气粗性的 把提交之前的数据也保存一份，事务立马提交，错了也不怕，还有原数据，可以回到之前的数据。浪费资源。 |

|  |
| --- |
| TCP的三次握手和四次挥手  TCP会保证数据的正确性以及数据的顺序  使用TCP，主机A和主机B必须进行三次会话才能建立连接，1主机A想和B建立连接，会发送一个消息序列号，比如0. 2主机B收到后，会发送收到了ACK（acknowledge），并希望下次回复的序列号是1. 3主机A收到后，再次返送ACK，并告诉序列号1  丢包的话，会一直请求，知道收到正确的序列号。  1.A主动发起一个FIN的数据包要求停止连接，2.B收到后回复Ack 3. B也会发一个FIN的数据包 4A确认之后回复ACK 请求断开 |

|  |
| --- |
| Mybatis 引入  Mybatis的灵活性和可优化性超过了Hibernate。  Xml配置  Id 就是方法名  resultType 返回值类型  parameterType 参数类型  typeAliases 给包起别名  实体类中@Param 给变量起名。  如何获得主键自增的值？  方式一：useGenerateKeys：开启获得主键自增方式  KeyProperty：把主键自增值赋值给对象中哪一个属性  方式二：select @@identity 这种方式线程更加安全。  动态SQL  If 、 where 、 choose when 前面的when成立后面的就不判断了。、set  Foreach循环、bind模糊查询、  过程：构建SqlSessionFactory 通过XMLConfigBuilder解析全局配置文件，读出参数，并将读取的内容存入Configuration。然后使用Configation去创建SqlSessionFactory对象。  创建SqlSession，创建Mapper代理对象并执行数据库各种操作。 |

|  |
| --- |
| Spring Spring Boot 的区别  一开始讲什么是Spring ，什么是Spring Boot，  Spring框架为开发java应用程序提供了全面的基础框架支持。  SpringBoot基本上Spring框架的拓展，它消除了设置Spring应用程序所需的XML配置，为更快，更高效的开发生态铺平的道路。嵌入式tomcat容器。尽可能自动配置spring应用。SpringBoot的依赖只需要配置各种starter即可。  还有Spring MVC  简而言之 SpringBoot是Spring本身的拓展，使开发更加简单。 |

|  |
| --- |
| Spring Bean重复的话，默认是覆盖，  把allowBeanDefinitionOverriding改成false。 |

|  |
| --- |
| FastDFS    上传：先去连接跟踪器tracker，然后跟踪器去寻找一个相对一个较大的存储服务器，然后返回一个ip和port。客户端再去请求存储器服务器storage，返回一个结果，然后就会去请求跟踪器在那个服务器存储了什么，还有多大的空间。请求跟踪器做记录。跟踪器永远不可能请求存储器的。 下载和删除是一样的。  一个服务器是肯定不行的，最起码是集群。存储器是分卷的。那个卷存了什么。 |

|  |
| --- |
| Eureka和zookeeper的区别  Zookeeper常用做Dubbo(一站式解决方案)注册中心，zookeeper是存键值对的。  从CAP分析，Eureka是Ap，zookeeper是cp，为什么是CP是因为zookeeper的中心结构决定的，只要有一个服务器宕机，备用服务器就会投票选举，这是需要时间的。这段时间是不可用的。 |

|  |
| --- |
| MySQL  Mysql中MyISAM和InnoDB的区别？   1. InnoDB存储引擎支持事务，而MyISAM不支持事务 2. InnoDB支持行级锁，而MyISAM只支持表级锁   InnoDB行锁是通过给索引加锁实现的，即只有通过索引条件检索数据，InnoDB才使用行级锁，否则将使用表级锁！  MySQL表级锁有两种模式：表共享读锁和表独占写锁。就是说对MyISAM进行表进行读操作时，它不会阻塞其他用户对同一表的读请求，但会阻塞对同一表的写操作；而对Innodb表的写操作，会阻塞其他用户对同一表的读和写操作。   1. InnoDB支持外键，而MyISAM不支持外键 2. InnoDB不保存数据库表中表的具体行数，而MyISAM会保存；   对于select，update，insert，delete操作：  如果大量的select，MyISAM是更好的选择（因为MyISAM不支持事务，使得MySQL可以提供高速存储和检索，以及全文搜索能力）  如果执行大量的insert或update 出于性能方面的考虑，应该考虑InnoDB表。  InnoDB如何保证事务的四大特性？  MySQL的存储引擎InnoDB使用重做日志（redo log）保证一致性与持久性，  回滚日志（undo log）保证原子性，使用各种锁保证隔离性  Undo记录某数据被修改前的值，可以用来在事务失败时进行rollback  Redo记录某数据块被修改后的值，可以用来恢复未写入数据  共享锁与排它锁  共享锁和排他锁是具体的锁，是数据库机制上的锁  共享锁（读锁）：在同一个时间段内，多个用户可以读取同一个资源，读取的过程中数据不会发生任何变化。读锁之间相互不阻塞，多个用户可以同时读，但是不能允许有人修改  排它锁（写锁）：在任何时候只能有一个用户写入资源，当进行写锁时会阻塞其他的读锁或者写锁操作，只能有这一个用户来写，其他用户即不能读也不能写。  使用索引查询一定能提高查询性能吗？  通常，通过索引查询数据比全表扫描要快。但是我们也必须注意到它的代价  索引需要空间来存储，也需要定期维护，每当有记录在表中增减或索引列被修改时，索引本身也会被修改。这意味着每条记录的Insert、update、delete将为此付出多次磁盘的I/O。  因为索引需要额外的存储空间和处理，那些不必要的索引反而会使查询反应时间变慢。 |

|  |
| --- |
| MySQL慢查询如何优化？  检查是否走了索引，如果没有则优化SQL利用索引  检查所利用的索引，是否是最优索引  检查所查字段是否都是必须的，是否查询了过多字段，查出了多余数据  检查表中数据是否过多，是否应该进行分库分表了  检查数据库实例所在机器的性能配置，是否太低，是否可以适当增加资源 |

|  |
| --- |
| Redis和MySQL如何保证数据一致   1. 先更新MySQl，再更新Redis，如果更新Redis失败，可能仍然不一致。 2. 先删除Redis缓存数据，再更新MySQL，再次查询的时候在将数据添加到缓存中，这种方案能解决1方案的问题，但是在高并发下性能较低，而且仍然会出现数据不一致的问题，比如线程1删除了Redis缓存数据，正在更新MySQL，此时另外一个查询再查询，那么就会把MySQL中老数据又查到Redis中 3. 延时双删，步骤是：先删除Redis缓存数据，再更新MySQL，延迟几百毫秒再删除Redis缓存数据，这样就算再更新MySQL时，有其他线程读了MySQL，把老数据读到了Redis中，那么也会被删除掉，从而把数据保持一致。 |