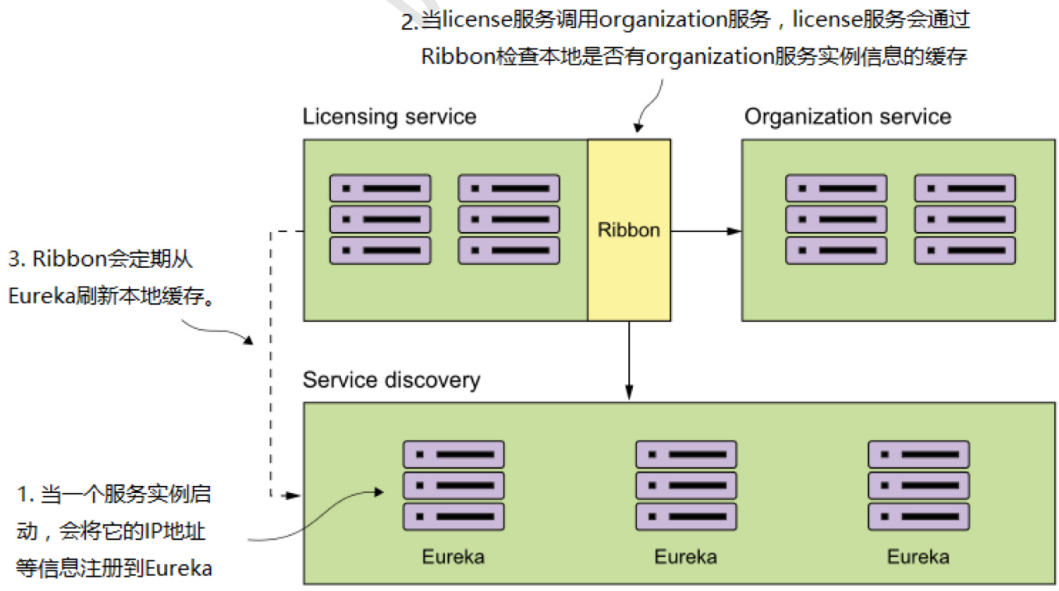
1. 说说你的工作经历?
2. 项目主要负责哪几个模块?
3. 画出你项目的结构图
4. Eureka是如何进行服务注册的?

a. 每30s发送心跳检测重新进行租约，如果客户端不能多次更新租约，它将在90s内从服务器注册中心移除。

a. 注册信息和更新会被复制到其他Eureka 节点，来自任何区域的客户端可以查找到注册中心信息，每30s发生一次复制来定位他们的服务，并进行远程调用。

b. 客户端还可以缓存一些服务实例信息，所以即使Eureka全挂掉，客户端也是可以定位到服务地址的。




5. 如果服务宕机或者无法访问了，我还去请求该服务，Eureka会怎么处理?会有什么现象?

Eureka服务注册中心的失效剔除与自我保护机制：https://www.jianshu.com/p/6cf86e0392a3

6. 谈谈Eureka的保护机制

Eureka Server的自我保护机制会检查最近15分钟内所有Eureka Client正常心跳的占比，如果低于85%就会被触发。

我们如果在Eureka Server的管理界面发现如下的红色内容，就说明

已经触发了自我保护机制。

```
EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.
```

当触发自我保护机制后Eureka Server就会锁定服务列表，不让服务列表内的服务过期，不过这样我们在访问服务时，得到的服务很有可能是已经失效的实例，如果是这样我们就会无法访问到期望的资源，会导致服务调用失败，所以这时我们就需要有对应的容错机制、熔断机制，我们在接下来的文章内会详细讲解这块知识点。

我们的服务如果是采用的公网IP地址，出现自我保护机制的几率就会大大增加，所以这时更要我们部署多个相同InstanId的服务或者建立一套完整的熔断机制解决方案。

### 自我保护开关

如果在本地测试环境，建议关掉自我保护机制，这样方便我们进行测试，也更准备的保证了服务实例的有效性！！！

关闭自我保护只需要修改application.yml配置文件内参数eureka.server.enable-self-preservation将值设置为false即可。

7. Ribbon的负载均衡是面向服务内部还是外部的?

   外部，https://blog.csdn.net/u013087513/article/details/79775306

8. Ribbon如何实现负载均衡的?

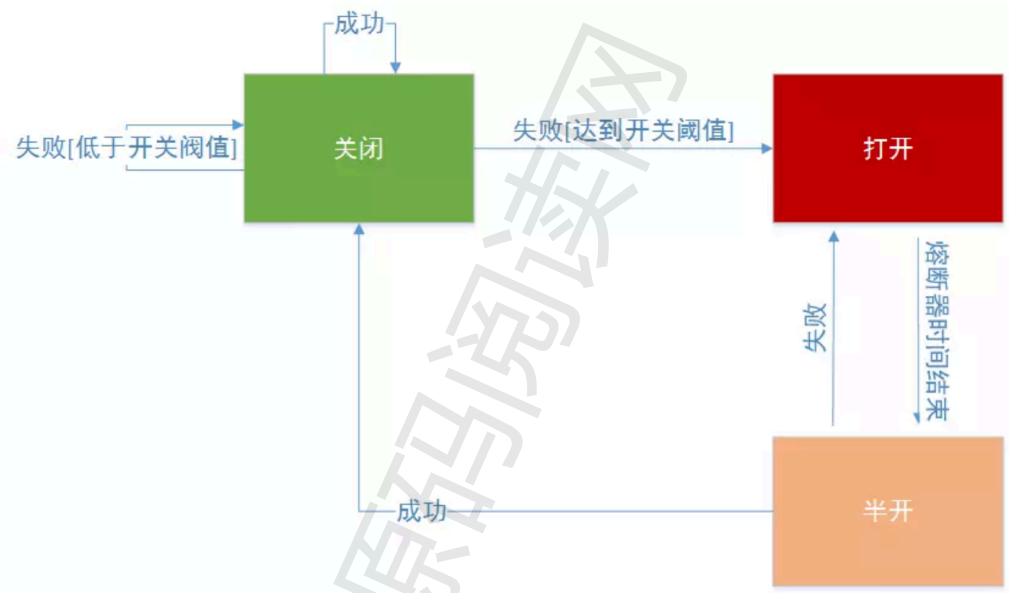
   这篇讲的非常好：https://www.jianshu.com/p/e459f43ef96d

9. 如果没有Eureka，我能直接通过Ribbon进行服务请求吗?

   可以的，需要导入 eureka-client-cat，https://blog.csdn.net/wangmx1993328/article/details/95513359

10. Hystrix如何实现熔断?

    Hystrix在运行过程中会向每个commandKey对应的熔断器报告**成功、失败、超时和拒绝**的状态，熔断器维护计算统计的数据，根据这些统计的信息来确定熔断器是否打开。如果打开，后续的请求都会被截断。然后会隔一段时间默认是5s，尝试半开，放入一部分流量请求进来，相当于对依赖服务进行一次健康检查，如果恢复，熔断器关闭，随后完全恢复调用。如下图：




11. 当服务无法访问时，是直接熔断还是降级?

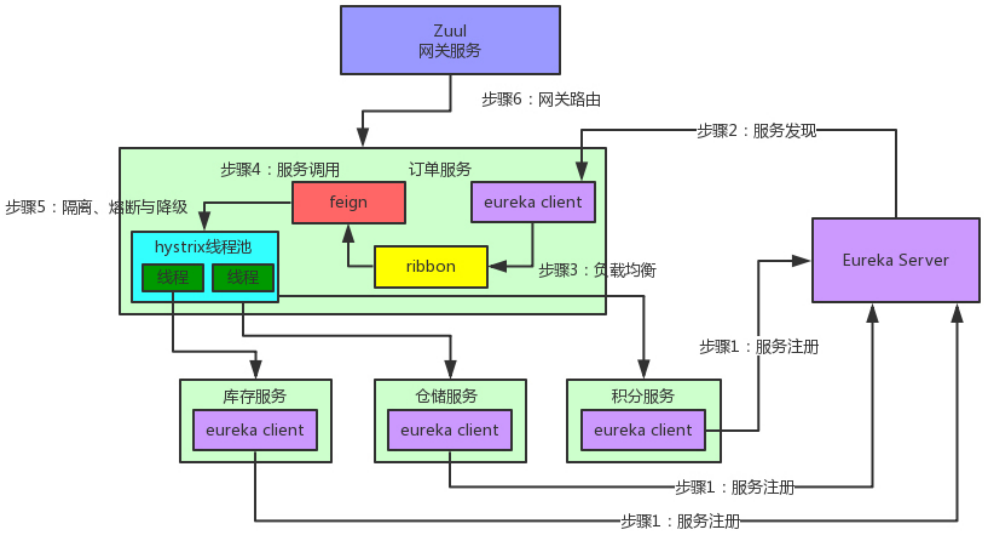
先降级，后熔断。

12. 怎么样才会出现熔断?

熔断就跟保险丝一样，当一个服务请求并发特别大，服务器已经招架不住了，调用错误率飙升，当错误率达到一定阈值后，就将这个服务熔断了。熔断之后，后续的请求就不会再请求服务器了，以减缓服务器的压力。

13. 在Springcloud中，消费者调用提供者的流程是如何的？请画图

springcloud的工作原理



springcloud由以下几个核心组件构成：

**Eureka**：各个服务启动时，Eureka Client都会将服务注册到Eureka Server，并且Eureka Client还可以反过来从Eureka Server拉取注册表，从而知道其他服务在哪里

**Ribbon**：服务间发起请求的时候，基于Ribbon做负载均衡，从一个服务的多台机器中选择一台

**Feign**：基于Feign的动态代理机制，根据注解和选择的机器，拼接请求URL地址，发起请求

**Hystrix**：发起请求是通过Hystrix的线程池来走的，不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务雪崩的问题

**Zuul**：如果前端、移动端要调用后端系统，统一从Zuul网关进入，由Zuul网关转发请求给对应的服务

14. Redis的集群方式有哪些?

主从、哨兵、redis-cluster：https://uule.iteye.com/blog/2438778

15. 如果你原来读取的Redis中的数据，它的数据的结构发生了变化，如何在不改变代码的情况下进行处理，使下游业务不受影响?

16. Redis的持久化方式有哪些？说说他们的具体实现、

Redis有两种持久化的方式：快照（RDB文件）和追加式文件（AOF文件）：

i. RDB持久化方式会在一个特定的间隔保存那个时间点的一个数据快照。

ii. AOF持久化方式则会记录每一个服务器收到的写操作。在服务启动时，这些记录的操作会逐条执行从而重建出原来的数据。写操作命令记录的格式跟Redis协议一致，以追加的方式进行保存。

iii. Redis的持久化是可以禁用的，就是说你可以让数据的生命周期只存在于服务器的运行时间里。

iv. 两种方式的持久化是可以同时存在的，但是当Redis重启时，AOF文件会被优先用于重建数据。

17. 使用Redis的过程中有没有遇到什么问题?

1、缓存和数据库双写一致性问题解决方案

前提是对数据有强一致性要求，不能放缓存；

只能降低不一致发生的概率，无法完全避免；

只能保证最终一致性。

1）采用正确的更新策略，先更新数据库，再删缓存

2）可能存在删除缓存失败的问题，提供一个补偿措施：如利用消息队列。

2、缓存雪崩解决方案

（大并发项目，流量在几百万）

-利用互斥锁：会使吞吐量下降

-给缓存加失效时间：随机值，避免集体失效

-双缓存

3、缓存穿透解决方案

（大并发项目，流量在几百万）

-利用互斥锁

-采用异步更新策略，无论Key是否取到值都直接返回

-提供一个能迅速判断请求是否有效的拦截机制（布隆过滤器）

4、缓存并发竞争解决方案

-不要求顺序时，准备一个分布式锁，同时去抢锁，然后在set操作。

-要求顺序时，在数据写入数据库时，需要保存一个时间戳。

-利用队列，将set操作变成串行访问。

18. Redis的内存回收机制有哪些?

了解：https://juejin.im/post/5d107ad851882576df7fba9e

19. Redis的过期策略有哪些？简单介绍下不同策略

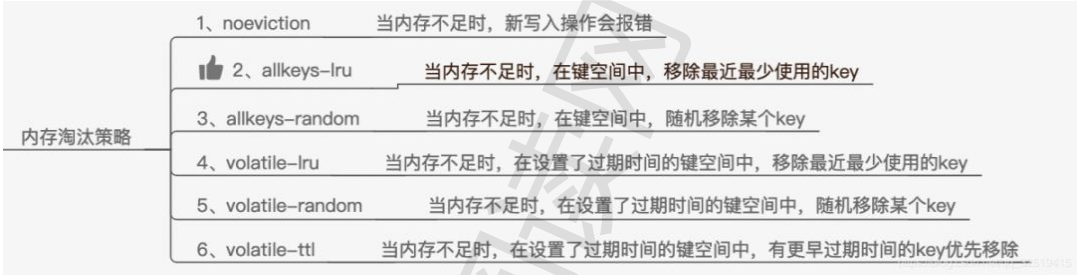
1. 定时过期：每个设置过期时间的key都需要创建一个定时器，到过期时间就会立即清除。该策略可以立即清除过期的数据，对内存很友好；但是会占用大量的CPU资源去处理过期的数据，从而影响缓存的响应时间和吞吐量。

2. 惰性过期：只有当访问一个key时，才会判断该key是否已过期，过期则清除。该策略可以最大化地节省CPU资源，却对内存非常不友好。极端情况可能出现大量的过期key没有再次被访问，从而不会被清除，占用大量内存。

3. 定期过期：每隔一定的时间，会扫描一定数量的数据库的expires字典中一定数量的key，并清除其中已过期的key。该策略是前两者的一个折中方案。通过调整定时扫描的时间间隔和每次扫描的限定耗时，可以在不同情况下使得CPU和内存资源达到最优的平衡效果。(expires字典会保存所有设置了过期时间的key的过期时间数据，其中，key是指向键空间中的某个键的指针，value是该键的毫秒精度的UNIX时间戳表示的过期时间。键空间是指该Redis集群中保存的所有键。)

20. 说说Redis的淘汰策略？淘汰的算法可以修改或者自己重写吗?

默认：noeviction

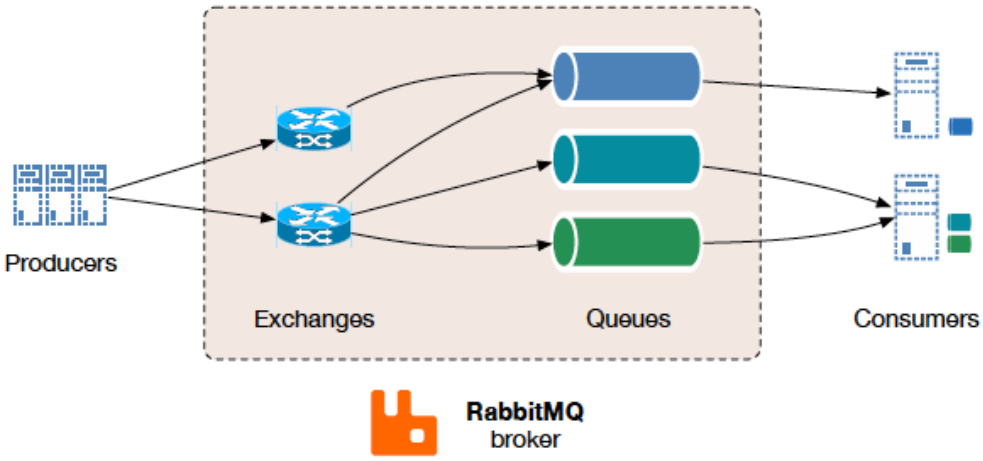



21. 你们的项目中消息中间件用的是什么?

    详见“面试题库/消息中间件”

22. 你们的Rabbit集群是怎么部署的?
23. Rabbit集群之间的数据是如何同步的？同步方式还是异步方式?
24. 一个队列中的数据你们是存放在一台机子上还是多台机子上？为什么?
25. RabbitMQ内部结构是怎么样的？请画出RabbitMQ的架构图

    RabbitMQ是一个高可用的消息中间件，支持多种协议和集群扩展。并且支持消息持久化和镜像队列，适用于对消息可靠性较高的场合，基本模型如下。https://fanchao01.github.io/blog/2018/02/09/rabbitmq-arch/




26. 你们公司的数据库有分库分表吗？如何实现的?

    详读https://www.cnblogs.com/butterfly100/p/9034281.html

27. Mysql的索引是基于什么?

    a. InnoDB：

        1. 支持事务处理
        2. 支持外键
        3. 支持行锁

4. 不支持FULLTEXT类型的索引（在Mysql5.6已引入）
5. 不保存表的具体行数，扫描表来计算有多少行
6. 对于AUTO_INCREMENT类型的字段，必须包含只有该字段的索引
7. DELETE 表时，是一行一行的删除
8. InnoDB 把数据和索引存放在表空间里面
9. 跨平台可直接拷贝使用
10. 表格很难被压缩

b. MyISAM：

1. 不支持事务，回滚将造成不完全回滚，不具有原子性
2. 不支持外键
3. 支持全文搜索
4. 保存表的具体行数,不带where时，直接返回保存的行数
5. DELETE 表时，先drop表，然后重建表
6. MyISAM 表被存放在三个文件 。frm 文件存放表格定义。 数据文件是MYD (MYData) 。 索引文件是MYI (MYIndex)引伸
7. 跨平台很难直接拷贝
8. AUTO_INCREMENT类型字段可以和其他字段一起建立联合索引
9. 表格可以被压缩

c. 选择：因为MyISAM相对简单所以在效率上要优于InnoDB.如果系统读多，写少。对原子性要求低。那么MyISAM最好的选择。且MyISAM恢复速度快。可直接用备份覆盖恢复。如果系统读少，写多的时候，尤其是并发写入高的时候。InnoDB就是首选了。两种类型都有自己优缺点，选择那个完全要看自己的实际类弄。

d. InnoDB引擎表是基于B+树的索引组织表

28. 说说B+树

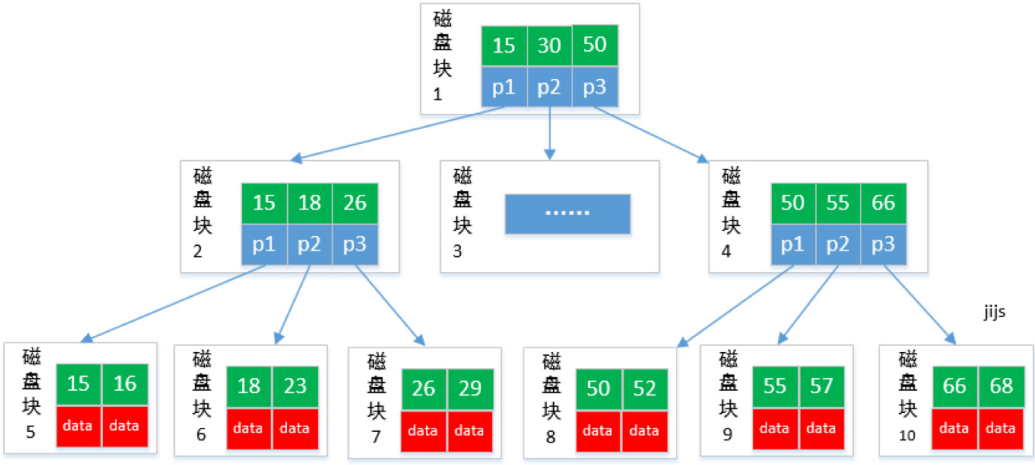
1. B-tree：


B-tree 利用了磁盘块的特性进行构建的树。每个磁盘块一个节点，每个节点包含了很关键字。把树的节点关键字增多后树的层级比原来的二叉树少了，减少数据查找的次数和复杂度。

B-tree巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页（每页为4K），这样每个节点只需要一次I/O就可以完全载入。

B-tree 的数据可以存在任何节点中。

2. B+tree：




B+tree 是 B-tree 的变种，B+tree 数据只存储在叶子节点中。这样在B树的基础上每个节点存储的关键字数更多，树的层级更少所以查询数据更快，所有指关键字指针都存在叶子节点，所以每次查找的次数都相同所以查询速度更稳定;

29. 使用自增ID和UUID作为主键有什么不同?

(1).如果InnoDB表的数据写入顺序能和B+树索引的叶子节点顺序一致的话，这时候存取效率是最高的。为了存储和查询性能应该使用自增长id做主键。

(2).对于InnoDB的主索引，数据会按照主键进行排序，由于UUID的无序性，InnoDB会产生巨大的IO压力，此时不适合使用UUID做物理主键，可以把它作为逻辑主键，物理主键依然使用自增ID。为了全局的唯一性，应该用uuid做索引关联其他表或做外键。

https://blog.csdn.net/XDSXHDYY/article/details/78994045

30. 说说数据库的事务隔离级别有哪些?

事务的四种隔离级别

| 隔离级别 | 脏读（Dirty Read） | 幻读（Phantom Read） |
| --- | --- | --- |
| 未提交读（Read uncommitted） | 可能 | 可能 |
| 已提交读（Read committed） | 不可能 | 可能 |
| 可重复读（Repeatable read） | 不可能 | 可能 |
| 可串行化（Serializable ） | 不可能 | 不可能 |

31. 在代码中，我们如何实现事务?

1.增加@EnableTransactionManagement注解

2.在service的实现类或者方法上添加下面这个注解即可.事务管理器的配置在多数据源那篇里已经配置过了.默认情况下只有RuntimeException才回滚,为了让Exception也回滚,增加了 rollbackFor = Exception.class.

32. 如果在一个事务中，代码业务流程很长，会有什么问题吗？为什么会出现这种问题?

可能出现事务超时：

@Transactional(timeout = 60)

如果用这个注解描述一个方法的话，线程已经跑到方法里面，如果已经过去60秒了还没跑完这个方法并且线程在这个方法中的后面还有涉及到对数据库的增删改查操作时会报事务超时错误（会回滚）。如果已经过去60秒了还没跑完但是后面已经没有涉及到对数据库的增删改查操作，那么这时不会报事务超时错误（不会回滚）。

33. 使用volatile关键字的时候有遇到过什么问题吗？为什么会出现这种问题?

34. 请说说volatile的底层实现原理

深入理解：https://crowhawk.github.io/2018/02/10/volatile/

35. 如何创建线程池？有什么参数？线程池的实现原理

实践操

作：https://blog.csdn.net/u011974987/article/details/51027795

36. 你有什么问题想问我们的吗？