# 云计算 1607C 李强

# JAVA工程师

1. 项目介绍(按照实训二做的项目写)

|  |
| --- |
| 汽车之家平台汇集汽车行业,汽车摄影,汽车改装,汽车行业观察,汽车维修等领域内的顶尖专家好手,独到的观点评论，类似微博的自媒体平台。项目整体采用了Spring Boot + Spring Cloud的框架。后端使用 Spring Data JPA 持久层框架与数据库进行交互。采用关系型数据库MySQL对一些用户的重要信息进行保存，用到的非关系型数据库有Redis、MongoDB。Redis主要用在用户根据Id查询时将返回的数据存到缓存中，或者是记录当前用户的状态，下次查询直接从缓存中返回，防止数据库压力过大。MongoDB主要用在储存回复、评论等具有层级关系的情况中。搜索系统通过使用 ElasticSearch 对问答信息进行搜索并展示。  用户可在平台上浏览最新有关新车商品，评价商品。汽车之家可实现用户对汽车相关的观看和评价,使信息发布方和用户都能够更快捷方便的评价有关汽车相关的最新信息，  提高用户的体验度 |

1. 功能模块介绍(按照自己得实际项目写)

|  |
| --- |
| **用户注册模块**  在登录页面时,有是否去登录或者是否去注册,如果去注册的话我们输入手机号进行短信验证,对于短信验证这一块我们利用的是阿里大于短信接口进行验证的,当前台点击验证码时后台随机生成一个6位数(我们利apache.commons.lang3)将随机数利用redis进行存储，另外发送到消息中间件rabbitMQ中，通过监听类进行监听，然后调用阿里大于短信接口实现短信发送 之后将验证码填入与之前存储的进行比对，这样就可以实现用户注册了  **搜索模块**  搜索微服务：这个模块主要是利用**Elasticsearch**来实现搜索功能，用户使用搜索功能可以搜索自己想要查看的信息，比如新兴汽车、汽车美容、销售排行等  项目中使用的是**Spring Data** **Elasticsearch**，他是对**Elasticsearch**的底层进行了封装，可以实现**Elasticsearch**同样的功能，**ElasticSearch**是一个基于Lucene的搜索服务器，这里我使用**Head**插件实现对**Elasticsearch**管理，**Head**插件是通过图形化界面来实现**Elasticsearch**的日常管理，使用**Head**插件可以实现对文档内容的删改和查询，还可以创建索引加快查询速度，在**Elasticsearch**中还使用到了**IK**中文分词器实现中文分词，**IK**提供了两个分词算法**ik\_smart** 和 i**k\_max\_word**，其中 **ik\_smart** 为最少切分，**ik\_max\_word**为最细粒度划分，实现了更全面的搜索，然后对于**Elasticsearch**与**MySQL**的数据同步是使用**Logstash**实现的。  **吐槽模块**  吐槽模块是实现让用户对发布的动态进行评论，分享自己的观点，吐槽模块主要是实现让用户可以对动态、问题、评论进行吐槽，抒发自己的看点，吐槽对于表结构设计时是有特点的,设计一个parentid 与主键\_id设置主外键关系,不是真正的主外键而是为了有树形结构这种体系。我们都知道在吐槽时是可以接着你吐槽的东西继续吐槽的，当然在相互点赞的同时我们也是利用了MongoDB里面的一个自增列,使用 $inc 运算符，对某列进行增加或减少。这里对评论和吐槽是使用**MongoDB**来进行存储的，是因为吐槽和评论这两个功能数据量大，写入操作频繁，价值较低，而**MongoDB**更适合来存储这类数据。  **登陆模块**  第一次登录，用户从浏览器输入用户名/密码，提交后到服务器的登录处理的Action层（Login Action）；Login Action调用认证服务进行用户名密码认证，如果认证通过，Login Action层调用用户信息服务获取用户信息（包括完整的用户信息及对应权限信息），返回用户信息后，Login Action从配置文件中获取Token签名生成的秘钥信息，进行Token的生成，生成Token的过程中可以调用第三方的JWT Lib生成签名后的JWT数据；完成JWT数据签名后，将其设置到COOKIE对象中，并重定向到首页，完成登录过程  **基础服务**  基础微服务：实现基本的增删改查，不用重复在每个微服务写这些代码，统一写完直接调用，简洁代码，提高效率，实现公共异常处理，是为了使我们的代码更容易维护。  **交友服务**  我们在用户模块里面写了一个更新好友粉丝和用户关注的方法时,我们让交友模块进行调用,利用的是SpringCloud里面的一个服务调用的组件Feign,通过在交友模块写了一个接口,利用@FeignClient进行方法的调用。    JSON Web Token（JWT）是一个非常轻巧的规范。这个规范允许我们使用JWT在用户和服务器之间传递安全可靠的信息。 **JWT的组成** 最后，我们将上面拼接完的字符串用HS256算法进行加密。在加密的时候，我们还需要提供一个密钥（secret）。如果我们用mystar作为密钥的话，那么就可以得到我们加密后的内容:  rSWamyAYwHCo7IFAgd1oRpSP7nzL7BF5t7ItqpKViM  最后将这一部分签名也拼接在被签名的字符串后面，我们就得到了完整的JWT:  第一次认证：第一次登录，用户从浏览器输入用户名/密码，提交后到服务器的登录处理的Action层（Login Action）；Login Action调用认证服务进行用户名密码认证，如果认证通过，Login Action层调用用户信息服务获取用户信息（包括完整的用户信息及对应权限信息），返回用户信息后，Login Action从配置文件中获取Token签名生成的秘钥信息，进行Token的生成，生成Token的过程中可以调用第三方的JWT Lib生成签名后的JWT数据；完成JWT数据签名后，将其设置到COOKIE对象中，并重定向到首页，完成登录过程      **请求认证**  客户端（APP客户端或浏览器）通过GET或POST请求访问资源（页面或调用API）；认证服务作为一个Middleware HOOK 对请求进行拦截，首先在cookie中查找Token信息，如果没有找到，则在HTTP Authorization Head中查找；  如果找到Token信息，则根据配置文件中的签名加密秘钥，调用JWT Lib对Token信息进行解密和解码；完成解码并验证签名通过后，对Token中的exp、nbf、aud等信息进行验证；全部通过后，根据获取的用户的角色权限信息，进行对请求的资源的权限逻辑判断；如果权限逻辑判断通过则通过Response对象返回；否则则返回HTTP 401； **对Token认证的五点认识** 对Token认证机制有5点直接注意的地方：  1一个Token就是一些信息的集合；  2在Token中包含足够多的信息，以便在后续请求中减少查询数据库的几率；  3服务端需要对cookie和HTTP Authrorization Header进行Token信息的检查；  基于上一点，你可以用一套token认证代码来面对浏览器类客户端和非浏览器类客户端；因为token是被签名的，所以我们可以认为一个可以解码认证通过的token是由我们系统发放的，其中带的信息是合法有效的；  **基于JWT的Token认证的安全问题**  **确保验证过程的安全性**  如何保证用户名/密码验证过程的安全性；因为在验证过程中，需要用户输入用户名和密码，在这一过程中，用户名、密码等敏感信息需要在网络中传输。因此，在这个过程中建议采用HTTPS，通过SSL加密传输，以确保通道的安全性。 **MongoDB与MySQL的区别** IMG_256  IMG_257IMG_258IMG_259  总体上讲：  由于MongoDB独特的数据处理方式，可以将热点数据加载到内存，故而对查询来讲，会非常快（当然也会非常消耗内存）；同时由于采用了BSON的方式存储数据，故而对JSON格式数据具有非常好的支持性以及友好的表结构修改性，文档式的存储方式，数据友好可见；数据库的分片集群负载具有非常好的扩展性以及非常不错的自动故障转移（大赞）。  不足：数据库的查询采用了特有的查询方式，有一定的学习成本（不高）；索引不咋滴；锁只能提供到collection级别，还做不到行级锁；没有事务机制（不能回滚啊） **MongoDB与Hadoop的区别** MongoDB侧重于对数据进行操作的应用系统，而Hadoop则侧重于对数据进行分析统计的应用。  MongoDB能够满足对数据库读写性能具有极高要求的应用场景（很消耗memory的），一般这些应用的响应延迟会要求控制在10ms以下，甚至更低。而Hadoop由于每一次的读写操作会包含大量数据（Hadoop更适合少次操作大批量数据的场景），通过聚集分析处理大量数据，这种分析一般都会走MapReduce，会造成很高的延迟（数分钟到数小时不等） **不适合MongoDB的场景** 如果业务中存在大量复杂的事务逻辑操作，则不要用MongoDB数据库 **MongoDB能为我解决哪些问题** 一般来讲，我会将MySQL中的部分表迁移到MongoDB中，主要是涉及到车辆历史轨迹以及温湿度数据等机器采集到的数据，而订单数据、客户数据等信息，仍然放到MySQL数据库中，主要是因为这两类数据实时采集，实时更新，会随着时间的推移，项目的扩大（PAAS服务），造成非常巨大的数据量，而一般MySQL在单表数据量超过500万后，性能就会下降的比较快，虽然可以通过分表的方式进行处理，但是随着时间的增长，仍然会给我带来比较大的麻烦（如查询等），这样，就不如将其放到MongoDB中存储，查询什么的都会比较方便，不过需要注意根据片键分片哦。  mongodb与关系型数据库相比的优缺点  与关系型数据库相比，MongoDB的优点：  ****①弱一致性（最终一致），更能保证用户的访问速度：**** 举例来说，在传统的关系型数据库中，一个COUNT类型的操作会锁定数据集，这样可以保证得到“当前”情况下的较精确值。这在某些情况下，例 如通过ATM查看账户信息的时候很重要，但对于Wordnik来说，数据是不断更新和增长的，这种“较精确”的保证几乎没有任何意义，反而会产生很大的延 迟。他们需要的是一个“大约”的数字以及更快的处理速度。 但某些情况下MongoDB会锁住数据库。如果此时正有数百个请求，则它们会堆积起来，造成许多问题。我们使用了下面的优化方式来避免锁定： 每次更新前，我们会先查询记录。查询操作会将对象放入内存，于是更新则会尽可能的迅速。在主/从部署方案中，从节点可以使用“-pretouch”参数运行，这也可以得到相同的效果。  使用多个mongod进程。我们根据访问模式将数据库拆分成多个进程。  ****②文档结构的存储方式，能够更便捷的获取数据。****  对于一个层级式的数据结构来说，如果要将这样的数据使用扁平式的，表状的结构来保存数据，这无论是在查询还是获取数据时都十分困难。  ****③内置GridFS，支持大容量的存储。****    GridFS是一个出色的分布式文件系统，可以支持海量的数据存储。    内置了GridFS了MongoDB，能够满足对大数据集的快速范围查询。  ****④内置Sharding。****  提供基于Range的Auto Sharding机制：一个collection可按照记录的范围，分成若干个段，切分到不同的Shard上。 Shards可以和复制结合，配合Replica sets能够实现Sharding+fail-over，不同的Shard之间可以负载均衡。查询是对 客户端是透明的。客户端执行查询，统计，MapReduce等操作，这些会被MongoDB自动路由到后端的数据节点。这让我们关注于自己的业务，适当的 时候可以无痛的升级。MongoDB的Sharding设计能力较大可支持约20 petabytes，足以支撑一般应用。  这可以保证MongoDB运行在便宜的PC服务器集群上。PC集群扩充起来非常方便并且成本很低，避免了“sharding”操作的复杂性和成本。  ****⑤第三方支持丰富。(这是与其他的NoSQL相比，MongoDB也具有的优势)****  现在网络上的很多NoSQL开源数据库完全属于社区型的，没有官方支持，给使用者带来了很大的风险。 而开源文档数据库MongoDB背后有商业公司10gen为其提供供商业培训和支持。  而且MongoDB社区非常活跃，很多开发框架都迅速提供了对MongDB的支持。不少知名大公司和网站也在生产环境中使用MongoDB，越来越多的创新型企业转而使用MongoDB作为和Django，RoR来搭配的技术方案。  ****⑥性能优越：****  在使用场合下，千万级别的文档对象，近10G的数据，对有索引的ID的查询不会比mysql慢，而对非索引字段的查询，则是全面胜出。 mysql实际无法胜任大数据量下任意字段的查询，而mongodb的查询性能实在让我惊讶。写入性能同样很令人满意，同样写入百万级别的数 据，mongodb比我以前试用过的couchdb要快得多，基本10分钟以下可以解决。补上一句，观察过程中mongodb都远算不上是CPU杀手。  ****与关系型数据库相比，MongoDB的缺点：****  ****①mongodb不支持事务操作。****    所以事务要求严格的系统（如果银行系统）肯定不能用它。(这点和优点①是对应的)  ****②mongodb占用空间过大。****    关于其原因，在官方的FAQ中，提到有如下几个方面： 1、空间的预分配：为避免形成过多的硬盘碎片，mongodb每次空间不足时都会申请生成一大块的硬盘空间，而且申请的量从64M、128M、256M那 样的指数递增，直到2G为单个文件的较大体积。随着数据量的增加，你可以在其数据目录里看到这些整块生成容量不断递增的文件。 2、字段名所占用的空间：为了保持每个记录内的结构信息用于查询，mongodb需要把每个字段的key-value都以BSON的形式存储，如果 value域相对于key域并不大，比如存放数值型的数据，则数据的overhead是较大的。一种减少空间占用的方法是把字段名尽量取短一些，这样占用 空间就小了，但这就要求在易读性与空间占用上作为权衡了。我曾建议作者把字段名作个index，每个字段名用一个字节表示，这样就不用担心字段名取多长 了。但作者的担忧也不无道理，这种索引方式需要每次查询得到结果后把索引值跟原值作一个替换，再发送到客户端，这个替换也是挺耗费时间的。现在的实现算是 拿空间来换取时间吧。 3、删除记录不释放空间：这很容易理解，为避免记录删除后的数据的大规模挪动，原记录空间不删除，只标记“已删除”即可，以后还可以重复利用。  4、可以定期运行db.repairDatabase()来整理记录，但这个过程会比较缓慢  ****③MongoDB没有如MySQL那样成熟的维护工具，这对于开发和IT运营都是个值得注意的地方。****  MongoDB适合存储一些关系简单、数据量又很大的数据，比如我们的平台上虚拟机的监控信息，包括内存、IO、CPU、网络等数据，每隔几秒就采集一次数据，每周、每月，量很大，而且旧的监控数据也不会保留太长时间，就使用的mongodb来存储这些数据； 另外mongodb的集群部署相对比较简单，易于扩展；比如主从复制，在mongo.conf配置几个参数就OK了；分片集群的配置也比较简单。还支持使用命令行来进行动态地添加和删除节点；  Mongodb的优点与不足 （1）Mongodb的不足之处 ****1、在集群分片中的数据分布不均匀 2、单机可靠性比较差 3、大数据量持续插入，写入性能有较大波动 4、磁盘空间占用比较大**** （2）Mongodb的过人之处 ****1、无模式 2、查询与索引方式灵活，是最像SQL的Nosql 3、支持复制集、主备、互为主备、自动分片等特性****   Mongodb与redis相比较：  mongoDB 源码语言是C++,redis也是C或C++,mongodb 文件存储是BSON格式类似JSON，或自定义的二进制格式。mongodb与redis性能都很依赖内存的大小，mongodb 有丰富的数据表达、索引；最类似于关系数据库，支持丰富的查询语言，redis数据丰富，较少的IO ，这方面mongodb优势明显mongodb不支持事物，靠客户端自身保证，redis支持事物，比较弱，仅能保证事物中的操作按顺序执行，这方面 redis优于mongodb。mongodb对海量数据的访问效率提升，redis 较小数据量的性能及运算,这方面 mongodb性能优于redis .monbgodb 有mapredurce功能，提供数据分析，redis 没有 ，这方面 mongodb优于redis 。  Elasticsearch 是一个兼有搜索引擎和NoSQL数据库功能的开源系统，基于Java/Lucene构建，可以用于全文搜索，结构化搜索以及近实时分析，Elasticsearch 是 面向文档型数据库，这意味着它存储的是整个对象或者 文档，它不但会存储它们，还会为他们建立索引  ****solr VS ES****  Solr是Apache Lucene项目的开源企业搜索平台。其主要功能包括全文检索、命中标示、分面搜索、动态聚类、数据库集成，以及富文本（如Word、PDF）的处理。  Solr是高度可扩展的，并提供了分布式搜索和索引复制。Solr是最流行的企业级搜索引擎，Solr4 还增加了NoSQL支持。  Solr是用Java编写、运行在Servlet容器（如 Apache Tomcat 或Jetty）的一个独立的全文搜索服务器。 Solr采用了 Lucene Java 搜索库为核心的全文索引和搜索，并具有类似REST的HTTP/XML和JSON的API。  Solr强大的外部配置功能使得无需进行Java编码，便可对 其进行调整以适应多种类型的应用程序。Solr有一个插件架构，以支持更多的高级定制  Elasticsearch 与 Solr 的比较总结  二者安装都很简单Solr 利用 Zookeeper 进行分布式管理，而 Elasticsearch 自身带有分布式协调管理功能  Solr 支持更多格式的数据，而 Elasticsearch 仅支持json文件格式  Solr 官方提供的功能更多，而 Elasticsearch 本身更注重于核心功能，高级功能多有第三方插件提供  Solr 在传统的搜索应用中表现好于 Elasticsearch，但在处理实时搜索应用时效率明显低于 Elasticsearch  Solr 是传统搜索应用的有力解决方案，但 Elasticsearch 更适用于新兴的实时搜索应用  ****核心概念****  ****集群（Cluster):****包含一个或多个具有相同 cluster.name 的节点.集群内节点协同工作，共享数据，并共同分担工作负荷。由于节点是从属集群的，集群会自我重组来均匀地分发数据. cluster Name是很重要的，因为每个节点只能是群集的一部分，当该节点被设置为相同的名称时，就会自动加入群集。集群中通过选举产生一个mater节点，它将负责管理集群范畴的变更，例如创建或删除索引，添加节点到集群或从集群删除节点。master 节点无需参与文档层面的变更和搜索，这意味着仅有一个 master 节点并不会因流量增长而成为瓶颈。任意一个节点都可以成为 master 节点。我们例举的集群只有一个节点，因此它会扮演 master 节点的角色。作为用户，我们可以访问包括 master 节点在内的集群中的任一节点。每个节点都知道各个文档的位置，并能够将我们的请求直接转发到拥有我们想要的数据的节点。无论我们访问的是哪个节点，它都会控制从拥有数据的节点收集响应的过程，并返回给客户端最终的结果。这一切都是由 Elasticsearch 透明管理的  ****节点(node):****一个节点是一个逻辑上独立的服务，可以存储数据，并参与集群的索引和搜索功能, 一个节点也有唯一的名字，群集通过节点名称进行管理和通信.  ****索引（Index)：**** 索引与关系型数据库实例(Database)相当。索引只是一个 逻辑命名空间，它指向一个或多个分片(shards)，内部用Apache Lucene实现索引中数据的读写  ****文档类型（Type）：****相当于数据库中的table概念。每个文档在ElasticSearch中都必须设定它的类型。文档类型使得同一个索引中在存储结构不同文档时，只需要依据文档类型就可以找到对应的参数映射(Mapping)信息，方便文档的存取  ****文档（Document) ：****相当于数据库中的row， 是可以被索引的基本单位。例如，你可以有一个的客户文档，有一个产品文档，还有一个订单的文档。文档是以JSON格式存储的。在一个索引中，您可以存储多个的文档。请注意，虽然在一个索引中有多分文档，但这些文档的结构是一致的，并在第一次存储的时候指定, 文档属于一种 类型(type)，各种各样的类型存在于一个 索引 中。你也可以通过类比传统的关系数据库得到一些大致的相似之处：  关系数据库 ⇒ 数据库 ⇒ 表 ⇒ 行 ⇒ 列(Columns)  Elasticsearch ⇒ 索引 ⇒ 类型 ⇒ 文档 ⇒ 字段(Fields) |

1. 开发中遇到的技术难点以及解决办法

|  |
| --- |
| 首先项目主要用到的是微服务架构和分布式架构,项目中就会遇到有关分布式场景的一些问题,例如分布式事务,分布式锁,跨域等问题.  分布式事务产生的原因是在多个系统中数据库垂直分割,数据库之间不能跨数据库进行访问,所以每个数据库的事务,都是在自己数据库独立进行管理的.这样就需要解决分布式事务的解决方案.而解决分布式事务的原理则是利用了Java的JTA,而java的JTA则是java操作XA接口,XA相当于协调者提供资源管理器与事务管理器之间进行通信的标准接口。而他们遵循2pc,3pc协议而我们绝大部分采用柔性事务都遵循CPA和Base理论,可以暂时不一致,但是最终要保证数据的一致性而我们公司最开始的探讨研究过分布式的解决方案第一种利用的是java Automatic框架,最开始感觉java Automatic有一定的缺点他遵循的是两段提交协议,他可能出现延时,当别人没有进行通知时可能进行卡死而我们公司发现Automatic用于多数据的情况下,用于传统的项目,而我们公司采用的是分布式项目所以不适合采用.在当时我们有采用另外一种方案,采用MQ解决分布式事务,但是由于我们的项目利用的是Springcloud,用MQ会比较麻烦所以最后又采用新的方案支付宝回调方式,就是类似于支付方式去做分布式事务管理但是又要自己写补偿机制和重拾机制还有要解决一些幂等性问题还是比较麻烦的,最后在github上找到一个新的框架tcc,lcn框架,发现lcn框架写的比较好,然后我们自己研究了一番,发现lcn底层原理的最大核心我不管理事务,而只是事务的搬用工lcn的原理是发起方调用接口,去xml创建事务分组,将事务分组的id传入,参与方接收到id时,操作一个假的关闭,不提交事务,当发起方代码执行完成以后,通知协调者,然后转发给参与方,看事务回滚还是提交.这是我们分布式事务的一个解决方案.  我们公司遇到的第二个难题就是分布式锁,什么是分布式锁,我们当时在项目当中遇到一种场景,需要做一个全局的id,但是全局id需要一个时间戳,如果在集群的情况下,时间戳因为服务器,可能在时间戳产生相同的情况下,当时想了好多的办法去解决全局id,之后利用分布式锁去解决.分布式锁相当于在多个服务器之间进行通信时,能够保证JVM性能的安全性问题,分布式锁跨Jvm,分布式锁在多个jvm进行通信时,能够保证多个JVM只有一个进行执行.在当时也有两种解决方式一个是利用reids,但是redis的有效期是比较麻烦的,有可能产生死锁,另外一种通过zookeeper的节点机制,首先zookeeper通过临时节点,当别人可以将临时节点创建成功时,就可以拿到我们这个锁了,因为节点是不允许重复的,当别人创建节点时,是不可创建的,需要去等待,等锁释放完成以后才可以拿到锁,那么zookeeper里面是怎样管理锁的释放了,zk有一个比较好的概念,就是有一个临时节点的特征.临时节点相当于session会话一关闭的情况下,整个连接将会断开,断开以后节点清空.所以锁就会被释放.  在前后端分离的情况下难免会出现跨域问题,在当时前端通过Ajax发请求时访问不到,跨域问题产生于浏览器方面,不是在服务端,而在当时我们就在服务器端解决跨域问题,同时也考虑过客户端解决,但是客户端解决有一些缺陷,通过jsonp,jsonp本身原理是通过标签进行转发,有这样一个缺陷,这个缺陷在当时我们解决方式是因为它只支持get请求,不支持post请求,所以我们当时就搭建网关的方式,网关方式相当于全部请求到网关,然后进行拦截,将部分分发到具体服务器上,从而保证域名相同,或者我们可以使用httpclient请求进行转发,但是这个比较占内存,或者使用设置请求头进行跨域  这是我在项目中要到的问题,分布式事务,分布式锁,和跨域问题,这几个问题的遇到让我有很大的提升. |

1. 项目技术亮点分析

|  |
| --- |
| Spring Cloud是怎么使用的？   1. 每个微服务使用Spring Boot开发，每个微服务工程包括了web、service、dao三层，这和开发一般的项目没有 区别： a、web层使用Spring MVC实现，对外暴露API接口给前端调用。 b、service层就是根据业务逻辑编写JavaBean，并使用Spring的声明式事务控制方式来控制事务。 c、dao层就是数据访问接口，来访问MySQL和Mongodb，访问MySQL使用Spring Data JPA和Mybatis，访问 mongodb使用Spring data mongodb。 2、微服务开发完成要向Eureka注册中心注册，以便被其它微服务查找和访问。 3、微服务与微服务之间使用feign来调用，feign Client具有负载均衡的作用。只需要在接口上声明@FeignClient注 解，Spring底层会产生动态代理对象，使用ribbon客户端完成调用。 4、前端访问微服务需要通过网关，网关使用Nginx和Zuul来实现，Nginx是最前边的负载均衡，通过Nginx之后便 到达了Zuul，项目中Zuul的功能是过虑用户请求，判断用户身份，对于一些对外公开的微服务则需要经过Zuul，直 接通过Nginx负载均衡即可访问。  **redis 如何才能做到高可用** 如果系统在 365 天内，有 99.99% 的时间，都是可以哗哗对外提供服务的，那么就说系统是高可用的。  一个 slave 挂掉了，是不会影响可用性的，还有其它的 slave 在提供相同数据下的相同的对外的查询服务。  但是，如果 master node 死掉了，会怎么样？没法写数据了，写缓存的时候，全部失效了。slave node 还有什么用呢，没有 master 给它们复制数据了，系统相当于不可用了。  redis 的高可用架构，叫做 failover ****故障转移****，也可以叫做主备切换。  master node 在故障时，自动检测，并且将某个 slave node 自动切换为 master node 的过程，叫做主备切换。这个过程，实现了 redis 的主从架构下的高可用。 **Redis 哨兵集群实现高可用:****哨兵的介绍** sentinel，中文名是哨兵。哨兵是 redis 集群机构中非常重要的一个组件，主要有以下功能：   * 集群监控：负责监控 redis master 和 slave 进程是否正常工作。 * 消息通知：如果某个 redis 实例有故障，那么哨兵负责发送消息作为报警通知给管理员。 * 故障转移：如果 master node 挂掉了，会自动转移到 slave node 上。 * 配置中心：如果故障转移发生了，通知 client 客户端新的 master 地址。   哨兵用于实现 redis 集群的高可用，本身也是分布式的，作为一个哨兵集群去运行，互相协同工作。   * 故障转移时，判断一个 master node 是否宕机了，需要大部分的哨兵都同意才行，涉及到了分布式选举的问题。 * 即使部分哨兵节点挂掉了，哨兵集群还是能正常工作的，因为如果一个作为高可用机制重要组成部分的故障转移系统本身是单点的，那就很坑爹了。  **哨兵的核心知识**  * 哨兵至少需要 3 个实例，来保证自己的健壮性。 * 哨兵 + redis 主从的部署架构，是****不保证数据零丢失****的，只能保证 redis 集群的高可用性。 * 对于哨兵 + redis 主从这种复杂的部署架构，尽量在测试环境和生产环境，都进行充足的测试和演练。   主备切换的过程，可能会导致数据丢失：   * 异步复制导致的数据丢失   因为 master->slave 的复制是异步的，所以可能有部分数据还没复制到 slave，master 就宕机了，此时这部分数据就丢失了   * 脑裂导致的数据丢失   脑裂，也就是说，某个 master 所在机器突然****脱离了正常的网络****，跟其他 slave 机器不能连接，但是实际上 master 还运行着。此时哨兵可能就会****认为**** master 宕机了，然后开启选举，将其他 slave 切换成了 master。这个时候，集群里就会有两个 master ，也就是所谓的****脑裂****。  此时虽然某个 slave 被切换成了 master，但是可能 client 还没来得及切换到新的 master，还继续向旧 master 写数据。因此旧 master 再次恢复的时候，会被作为一个 slave 挂到新的 master 上去，自己的数据会清空，重新从新的 master 复制数据。而新的 master 并没有后来 client 写入的数据，因此，这部分数据也就丢失了。 **用 Hystrix 构建高可用服务架构****1.Hystrix 是什么？** 在分布式系统中，每个服务都可能会调用很多其他服务，被调用的那些服务就是****依赖服务****，有的时候某些依赖服务出现故障也是很正常的。  Hystrix 可以让我们在分布式系统中对服务间的调用进行控制，加入一些****调用延迟****或者****依赖故障****的****容错机制****。  Hystrix 通过将依赖服务进行****资源隔离****，进而阻止某个依赖服务出现故障时在整个系统所有的依赖服务调用中进行蔓延；同时Hystrix 还提供故障时的 fallback 降级机制。  总而言之，Hystrix 通过这些方法帮助我们提升分布式系统的可用性和稳定性。 **Hystrix 的历史** Hystrix 是高可用性保障的一个框架。Netflix（可以认为是国外的优酷或者爱奇艺之类的视频网站）的 API 团队从 2011 年开始做一些提升系统可用性和稳定性的工作，Hystrix 就是从那时候开始发展出来的。  在 2012 年的时候，Hystrix 就变得比较成熟和稳定了，Netflix 中，除了 API 团队以外，很多其他的团队都开始使用 Hystrix。  时至今日，Netflix 中每天都有数十亿次的服务间调用，通过 Hystrix 框架在进行，而 Hystrix 也帮助 Netflix 网站提升了整体的可用性和稳定性。  [2018 年 11 月，Hystrix 在其 Github 主页宣布，不再开放新功能，推荐开发者使用其他仍然活跃的开源项目](https://github.com/Netflix/Hystrix/blob/master/README.md" \l "hystrix-status" \t "https://github.com/doocs/advanced-java/blob/master/docs/high-availability/_blank)。维护模式的转变绝不意味着 Hystrix 不再有价值。相反，Hystrix 激发了很多伟大的想法和项目，我们高可用的这一块知识还是会针对 Hystrix 进行讲解。 **Hystrix 的设计原则**  * 对依赖服务调用时出现的调用延迟和调用失败进行****控制和容错保护****。 * 在复杂的分布式系统中，阻止某一个依赖服务的故障在整个系统中蔓延。比如某一个服务故障了，导致其它服务也跟着故障。 * 提供 fail-fast（快速失败）和快速恢复的支持。 * 提供 fallback 优雅降级的支持。 * 支持近实时的监控、报警以及运维操作。   举个栗子。  有这样一个分布式系统，服务 A 依赖于服务 B，服务 B 依赖于服务 C/D/E。在这样一个成熟的系统内，比如说最多可能只有 100 个线程资源。正常情况下，40 个线程并发调用服务 C，各 30 个线程并发调用 D/E。  调用服务 C，只需要 20ms，现在因为服务 C 故障了，比如延迟，或者挂了，此时线程会 hang 住 2s 左右。40 个线程全部被卡住，由于请求不断涌入，其它的线程也用来调用服务 C，同样也会被卡住。这样导致服务 B 的线程资源被耗尽，无法接收新的请求，甚至可能因为大量线程不断的运转，导致自己宕机。服务 A 也挂。  [IMG_256](https://github.com/doocs/advanced-java/blob/master/images/service-invoke-road.png)  Hystrix 可以对其进行资源隔离，比如限制服务 B 只有 40 个线程调用服务 C。当此 40 个线程被 hang 住时，其它 60 个线程依然能正常调用工作。从而确保整个系统不会被拖垮。 **Hystrix 更加细节的设计原则**  * 阻止任何一个依赖服务耗尽所有的资源，比如 tomcat 中的所有线程资源。 * 避免请求排队和积压，采用限流和 fail fast 来控制故障。 * 提供 fallback 降级机制来应对故障。 * 使用资源隔离技术，比如 bulkhead（舱壁隔离技术）、swimlane（泳道技术）、circuit breaker（断路技术）来限制任何一个依赖服务的故障的影响。 * 通过近实时的统计/监控/报警功能，来提高故障发现的速度。 * 通过近实时的属性和配置****热修改****功能，来提高故障处理和恢复的速度。 * 保护依赖服务调用的所有故障情况，而不仅仅只是网络故障情况。 |
|  |

1. 项目答辩中亮点简历审核评定标准

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **考核内容** | **简历样式是否规范** | **项目内容是否完整** | **项目中是否有亮点** | **技术难点解决办法是否高效** | **存在问题** |
| **项目经理 （评定）** |  |  |  |  |  |
| **就业处干事（评定）** |  |  |  |  |  |
| 简历是否合格 | | | | 合格 | 不合格 |

1. 项目答辩成绩评定标准

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **考核内容** | **亮点简历与项目描述是否匹配** | **项目流程是否清晰** | **学生讲解是否流畅** | **项目介绍是否到位** | **技术亮点考核是否合格** | **存在问题** |
| **项目经理 （评定）** |  |  |  |  |  |  |
| **就业处干事（评定）** |  |  |  |  |  |  |
| 项目答辩是否合格 | | | | 合格 | 不合格 |  |