# 3 架构

## Mongodb

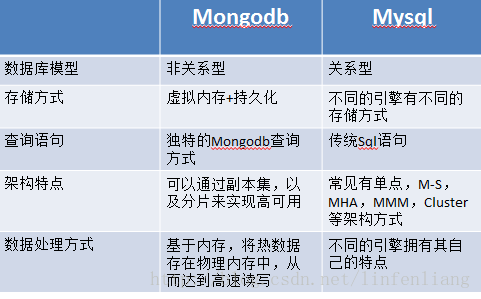
### 什么是mongodb

* 1. MongoDB 是由C++语言编写的，是一个基于分布式文件存储的开源数据库系统。
  2. MongoDB是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。它支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。

### MongoDB的特点

* 1. 文档数据类型
     1. SQL类型的数据库是正规化的，可以通过主键或者外键的约束保证数据的完整性与唯一性，所以SQL类型的数据库常用于对数据完整性较高的系统。MongoDB在这一方面是不如SQL类型的数据库，且MongoDB没有固定的Schema，正因为MongoDB少了一些这样的约束条件，可以让数据的存储数据结构更灵活，存储速度更加快。
  2. 即时查询能力
     1. MongoDB保留了关系型数据库即时查询的能力，保留了索引（底层是基于B tree）的能力。这一点汲取了关系型数据库的优点，相比于同类型的NoSQL redis 并没有上述的能力。
  3. 复制能力
     1. MongoDB自身提供了副本集能将数据分布在多台机器上实现冗余，目的是可以提供自动故障转移、扩展读能力。
  4. 速度与持久性
     1. MongoDB提供了Journaling日志的概念，实际上像mysql的bin-log日志，当需要插入的时候会先往日志里面写入记录，再完成实际的数据操作，这样如果出现停电，进程突然中断的情况，可以保障数据不会错误，可以通过修复功能读取Journaling日志进行修复。
  5. 数据扩展
     1. MongoDB使用分片技术对数据进行扩展，MongoDB能自动分片、自动转移分片里面的数据块，让每一个服务器里面存储的数据都是一样大小。

### ****MongoDB与MySQL的区别****



### Mongodb常用的命令

* 1. 切换数据库 sue dbs
  2. 插入语法
     1. db.users.insert({username:"smith"})
     2. db.users.save({username:"smith"})
     3. 区别若新增的数据中存在主键 ，insert() 会提示错误，而save() 则更改原来的内容为新内容。
  3. 查找语句
     1. db.users.find()
     2. db.users.count()
  4. 集合的相关操作

db.createCollection() 创建集合

db.集合名称.drop() 删除集合

show collections( tables) 查看所有集合

db.集合名.insert(文档) 集合添加数据 save insertone insermany

* 1. 文档的操作

db.集合名.insert(文档) 集合添加数据 save insertone insermany

db.集合名.remove

db.集合名.update

db.集合名.find()

## Linux

### Linux的常用命令

* 1. Ls查看当前文件夹下的内容
  2. pwd查看当前所在文件夹
  3. cd 切换目录
  4. mkdir[目录名] 创建文文件夹
  5. rm[文件名] 删除指定文件
  6. clear清空屏幕
  7. # 打包文件 tar -cvf 打包文件.tar 被打包的文件／路径...
  8. # 解包文件 tar -xvf 打包文件.tar
  9. # 解压缩文件 tar -zxvf 打包文件.tar.gz
  10. cp复制文件或者目录
  11. 查看某个服务是否已经启动 ps -ef |grep 服务名
  12. 启动redis ./redis-server ../redis.conf
  13. 杀死某个进程 kill -9 进程号
  14. 查看防火墙状态
      1. 查看防火墙状态 systemctl status firewalld
      2. 开启防火墙 systemctl start firewalld
      3. 关闭防火墙 systemctl stop firewalld
      4. 若遇到无法开启
      5. 先用：systemctl unmask firewalld.service
      6. 然后：systemctl start firewalld.service
  15. 对外开发端口
      1. 查看想开的端口是否已开：
      2. firewall-cmd --query-port=6379/tcp
      3. 添加指定需要开放的端口：
      4. firewall-cmd --add-port=123/tcp --permanent
      5. 重载入添加的端口：
      6. firewall-cmd --reload
      7. 查询指定端口是否开启成功：
      8. firewall-cmd --query-port=123/tcp

## ****Nginx****

### ****Nginx是什么****

* 1. 在服务器集群中，Nginx起到一个代理服务器的角色（即反向代理），为了避免单独一个服务器压力过大，将来自用户的请求转发给不同的服务器。

### Nginx负载策略

* 1. 轮询 这也是nginx的默认负载策略
  2. Weight 这是权重策略 根据权重进行分配 类似7,3开
  3. ip\_hash IP策略 就是所用户访问了一台服务之后就一直访问这个服务 可以保证session回话
  4. least\_conn 这是最少链接的策略 也就是说那台服务器连接数少就分配到那一台

## springboot

### 什么是springboot

* + 1. Springboot设计目的是用来简化新 Spring 应用的初始搭建以及开发过程，目的是为了让springboot项目更快的跑起来，尽可能的减少我们的配置文件

### SpringBoot的特点

（1）为所有 Spring 开发提供一个更快更广泛的入门体验。

（2）零配置。遵循“约定大于配置” 的原则。

（3）集成了大量常用的第三方库的配置， Spring Boot 应用为这些第三方库提供了几乎可以零配置的开箱即用的能力。

（4）提供一系列大型项目常用的非功能性特征，如嵌入式服务器、安全性、度量、运行状况检查、外部化配置等。

（5）Spring Boot 不是Spring 的替代者，它相当于是集成了spring，

### SpringBoot项目搭建过程

* + 1. 第一种 是通过springboot官方提供的方式进行一键式创建，但是前提是需要有外网
    2. 第二种是 是先创建一个maven项目 声明springboot的启动器，声明jdk版本，引入启动器

### 4 . Springboot的注解

1、@SpringBootApplication

这个注解是Spring Boot最核心的注解，用在 Spring Boot的主类上，标识这是一个 Spring Boot 应用，用来开启 Spring Boot 的各项能力。实际上这个注解是@Configuration,@EnableAutoConfiguration,@ComponentScan三个注解的组合。由于这些注解一般都是一起使用，所以Spring Boot提供了一个统一的注解@SpringBootApplication。

2、@EnableAutoConfiguration

允许 Spring Boot 自动配置注解，开启这个注解之后，Spring Boot 就能根据当前类路径下的包或者类来配置 Spring Bean。

3、@Configuration

用于定义配置类，指出该类是 Bean 配置的信息源，相当于传统的xml配置文件，一般加在主类上。如果有些第三方库需要用到xml文件，建议仍然通过@Configuration类作为项目的配置主类——可以使用@ImportResource注解加载xml配置文件。

4、@ComponentScan

组件扫描。让spring Boot扫描到Configuration类并把它加入到程序上下文。

@ComponentScan注解默认就会装配标识了@Controller，@Service，@Repository，@Component注解的类到spring容器中。

6、@Service

一般用于修饰service层的组件

7、@RestController

用于标注控制层组件(如struts中的action)，表示这是个控制器bean,并且是将函数的返回值直 接填入HTTP响应体中,是REST风格的控制器；它是@Controller和@ResponseBody的合集。

11、@AutoWired

byType方式。把配置好的Bean拿来用，完成属性、方法的组装，它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作。

14、@RequestMapping

RequestMapping是一个用来处理请求地址映射的注解；提供路由信息，负责URL到Controller中的具体函数的映射，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。

## Maven

刚刚有提到我们用maven来搭建我们的springboot 现在说一下maven吧

### 什么是maven

maven是用来构建项目和管理依赖的工具。构建项目就是创建项目，而且可以在项目编码完成之后,对项目进行编译、测试、打包、部署操作。管理依赖呢，就是对项目中所需要的jar包进行管理。

### **1.**   **为什么要使用Maven(Maven的好处)**

1.    使用maven创建出来的项目具有统一的结构

2.    规范jar包的下载

3.    节约项目的空间，让项目不是那么臃肿

4.    Maven可以把一个jar包所依赖的jar也一并下载下来

5.    通过maven可以对项目进行编译、测试、打包、部署操作

### **2.**   **Maven仓库分为哪几种?**

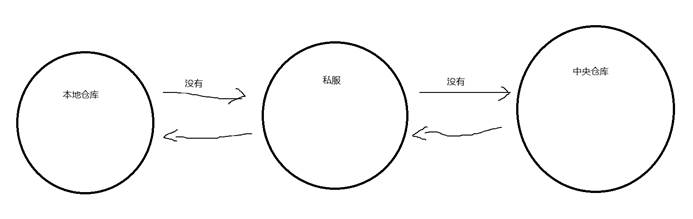
**本地仓库**:自己电脑上的一个文件夹，本地仓库默认设在了${user.home}/.m2/repository，本地仓库设在C盘有一点不好的地方，就是重装操作系统之后C盘被格式化了，之前仓库里的jar包就没了，所以咱们通常会修改本地仓库的位置，在maven安装目录下的conf/setting.xml中可以更改本地仓库的位置。

**中央仓库:**maven官方维护的一个仓库，里面包含了市面上绝大部分开源项目的jar包。但是中央仓库服务器所在遥远的美国，由于我们都懂的原因，访问速度比较慢，还有可能会中断。想要连接中央仓库必须得保证电脑联网了。

**第三方仓库(也就是私服)**:是架设在局域网的仓库，通常情况下公司会搭建自己的Maven私服，这样可以保证开发人员拿到的jar包都是统一的。而且我们班级之前也老师搭建的也有maven私服，大家都连接这个maven私服就可以下载jar包了。

### **3.**   **Maven获取jar包的顺序**

当本地仓库没有你需要的jar包时而且你也没有配置私服仓库地址时，maven会去中央仓库下载jar包。如果你的电脑上配置有私服仓库地址，则maven会去私服里下载jar包，如果私服也没有jar包，则私服会去中央仓库下载jar包。



### **4.**   **Setting(塞挺).xml和pom(炮木).xml**

settings.xml是maven的全局配置文件。Settings.xml中包含类似**本地仓库位置**、**修改远程仓库服务器**、**认证信息**等配置。而**pom.xml**主要描述了**项目的maven坐标**，**依赖关系**以及**其他所有的项目相关因素**，是项目级别的配置文件。

### **5.**   **Maven常用命令(掌握5个)**

mvn clean(科林)    清空产生的项目( target里)

mvn compile(凯木排欧) 编译源代码

mvn package(排K继)     打包

mvn test           运行测试

mvn test-compile   编译测试代码

mvn -Dtest package  只打包不测试

mvn  source(扫斯).jar   源码打包

## springCloud概念整理

### 什么是springcloud

Springcloud是一个分布式微服务架构的一站式解决方案，可以理解为是一个技术栈，包含了很多组件，其中最核心的几个就是Eureka（犹瑞卡）,Ribbon,Feign（芬）,hystrix（黑丝特瑞）,zuul（如）这几个组件。

SpringBoot可以离开SpringCloud独立使用开发项目，但是SpringCloud离不开SpringBoot，属于依赖的关系.

SpringBoot专注于快速、方便的开发单个微服务个体，SpringCloud关注全局的服务治理框架。

### SpringCloud项目的核心组件：

Spring Cloud Eureka：服务注册与发现

Spring Cloud Ribbon：客户端负载均衡

Spring Cloud Feign：声明性的Web服务客户端

Spring Cloud Hystrix：断路器

Spring Cloud Zuul：服务网关

### eureka

eureka三个核心组件：eureka服务、服务提供方、服务消费方；

<!-- eureka服务端的启动器-->

        <dependency>

            <groupId>org.springframework.cloud</groupId>

            <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

        </dependency>

配置文件

eureka:

  client:

    register-with-eureka: false # 是否注册自己的信息到EurekaServer，默认是true

    fetch-registry: false # 是否拉取其它服务的信息，默认是true

    service-url: # EurekaServer的地址，现在是自己的地址，如果是集群，需要加上其它Server的地址。

      defaultZone: http://127.0.0.1:8050/eureka # http://本机ip:端口号/eureka

4、在启动类上加@EnableEurekaServer注解（说明吧这个项目注册到eureka）

springboot使用RestTemplate类中的方法进行接口访问；

#### **Eureka工作原理**

服务启动向Eureka注册中心注册服务，注册中心将注册信息同步给其他注册中心，当服务消费者需要调用服务提供者，则向服务注册中心获取服务提供者地址，然后会将获取的地址缓存在本地，下次调用的则直接从缓存中取，当服务注册中心检测到服务提供者宕机、网络不通等服务不可用时，则在注册中心将服务置为down状态，并向订阅者发布当前服务提供者状态，订阅过的服务消费者更新本地缓存。

服务提供者启动后周期性（30秒）向服务注册中心发送心跳，以证明当前服务可用，服务中心在一定时间（默认90秒）未收到心跳，则认为该服务宕机，注销该实例

#### Eureka自我保护机制:

默认情况下，如果Eureka在一定时间内没有接受到某个微服务实例的心跳，Eureka注册中心将会将该实例注销。但是当网络分区故障发生时，微服务与Eureka Server之间无法正常通信，以上行为可能变得非常危险了——因为微服务本身其实是健康的，此时本不应该注销这个微服务。Eureka通过“自我保护模式”来解决这个问题——当Eureka Server节点在**短时间内丢失过多客户端时**（可能发生了网络分区故障），那么这个节点就会进入自我保护模式。一旦进入该模式，Eureka Server就会保护服务注册表中的信息，不再删除服务注册表中的数据（也就是不会注销任何微服务）。当网络故障恢复后，该Eureka Server节点会自动退出自我保护模式。

在自我保护模式中，Eureka Server会保护服务注册表中的信息，不再注销任何服务实例。当它收到的心跳数重新恢复到阈值以上时，该Eureka Server节点就会自动退出自我保护模式。它的设计哲学就是宁可保留错误的服务注册信息，也不盲目注销任何可能健康的服务实例。一句话讲解：**好死不如赖活着**。

### **Feign的理论知识**

Feign是一个声明式的Web服务客户端，使得**编写Web服务客户端**变得非常容易，

**只需要创建一个接口，然后在上面添加注解即可**。**声明式调用就像调用本地方法一样调用远程方法**

#### **Feign的工作原理**

主程序入口添加了@EnableFeignClients注解开启对FeignClient扫描加载处理。根据Feign Client的开发规范，定义接口并加@FeignClient注解。

当程序启动时，会进行包扫描，扫描所有@FeignClients的注解的类，并且将这些信息注入Spring IOC容器中，当定义的的Feign接口中的方法被调用时，通过JDK的代理方式，来生成具体的RequestTemplate.当生成代理时，Feign会为每个接口方法创建一个RequestTemplate。当生成代理时，Feign会为每个接口方法创建一个RequestTemplate对象，该对象封装HTTP请求需要的全部信息，如请求参数名，请求方法等信息都是在这个过程中确定的。

然后RequestTemplate生成Request,然后把Request交给Client去处理，这里指的是Client可以是JDK原生的URLConnection,Apache的HttpClient,也可以是OKhttp，最后Client被封装到LoadBalanceClient类，这个类结合Ribbon负载均衡发送服务器之间的调用。

## Hystrix理论知识

在讲Hystrix之前先说一下服务雪崩，在springcloud**可以用Feign+Ribbon来进行服务之间的调用**。为了保证其高可用，单个服务通常会集群部署。由于网络原因或者自身的原因，服务并不能保证100%可用。如客户端访问A服务，而A服务需要调用B服务，B服务需要调用C服务，由于网络原因或者自身的原因，如果B服务或者C服务不能及时响应，A服务将处于阻塞状态，直到B服务C服务响应。此时若有大量的请求涌入，容器的线程资源会被消耗完毕，导致服务瘫痪。**服务与服务之间的依赖性，故障会传播，造成连锁反应，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应**

### 服务雪崩的解决方案

**l 熔断模式**：这种模式主要是参考电路熔断，如果一条线路电压过高，保险丝会熔断，防止火灾。放到我们的系统中，如果某个目标服务调用慢或者有大量超时，此时，熔断该服务的调用，对于后续调用请求，不在继续调用目标服务，直接返回，快速释放资源。如果目标服务情况好转则恢复调用。

### 服务熔断

当下游的服务因为某种原因突然变得不可用或响应过慢，上游服务为了保证自己整体服务的可用性，不再继续调用目标服务，直接返回，快速释放资源。如果目标服务情况好转则恢复调用。

### 服务降级

l 当下游的服务因为某种原因响应过慢，下游服务主动停掉一些不太重要的业务，释放出服务器资源，增加响应速度！(**弃车保帅，壮士断腕，刮骨疗伤**)

当下游的服务因为某种原因不可用，上游主动调用本地的一些降级逻辑，避免卡顿，迅速返回给用户

### 服务熔断和服务降级的区别和联系

服务降级有很多种降级方式！如开关降级、限流降级、熔断降级!**服务熔断属于降级方式的一种**！**因为从实现上来说，熔断和降级必定是一起出现**。因为当发生下游服务不可用的情况，这个时候为了对最终用户负责，就需要进入上游的降级逻辑了。因此，**将熔断降级视为降级方式的一种**，也是可以说的通的！

### **Hystrix能干啥**

**l 服务熔断**

当Hystrix Command请求后端服务失败数量超过一定比例(默认50%), 断路器会切换到开路状态(Open). 这时所有请求会直接失败而不会发送到后端服务. 断路器保持在开路状态一段时间后(默认5秒), 自动切换到半开路状态(HALF-OPEN).

**l 服务降级**

Fallback相当于是降级操作, 我们可以实现一个fallback方法, 当请求后端服务出现异常或者服务熔断后的时候, 直接调用本地fallback方法，告知后面的请求服务不可用了，不要再来了**。**

## Zuul的理论知识

### 什么是zull

Zuul是一个微服务网关，**微服务网关**，**介于客户端与服务器之间的中间层**，所有的外部请求都会先经过微服务网关。

### 为什么需要外服务网关呢

不同的微服务一般会有不同的网络地址，客户端在访问这些微服务时必须记住几十甚至几百个地址，这对于客户端方来说太复杂也难以维护。

而且可能会出现很多问题。

客户端会请求多个不同的服务，需要维护不同的请求地址，增加开发难度

后端每个微服务可能是由不同语言编写的、采用了不同的协议，比如HTTP、Dubbo、GRPC等，但是你不可能要求客户端去适配这么多种协议，这是一项非常有挑战的工作，项目会变的非常复杂且很难维护。

### 服务网关两个核心

路由转发和**过滤器**

#### **路由转发**：接收一切外界(客户端)请求，转发到后端的微服务上去；

#### **过滤器**：在服务网关中可以完成一系列的横切功能，例如权限校验、限流以及监控等，这些都可以通过过滤器完成（其实路由转发也是通过过滤器实现的）。

过滤器的四中类型

**l PRE**：这种过滤器在请求被路由之前调用。我们可利用这种过滤器实现身份验证、在集群中选择请求的微服务、记录调试信息等。

**l ROUTING**：这种过滤器将请求路由到微服务。这种过滤器用于构建发送给微服务的请求，并使用Apache HttpClient或Netfilx Ribbon请求微服务。

**l POST**：这种过滤器在路由到微服务以后执行。这种过滤器可用来为响应添加标准的HTTPHeader、收集统计信息和指标、将响应从微服务发送给客户端等。

**ERROR**：在其他阶段发生错误时执行该过滤器。

我们可以通过将Zuul和Eureka进行整合，将Zuul自身注册为Eureka服务治理下的应用，同时从Eureka中获得其他微服务的消息，也即**以后通过客户端(浏览器，APP)访问微服务都是通过Zuul跳转后获得**。

## Ribbon

### 1什么是ribbon

Spring cloud Ribbon是一个基于HTTP和TCP的客户端负载均衡工具，Ribbon本身提供了不同负载均 衡策略使用不同的应用场景。

### Ribbon负载策略

1轮询策略

有一点需要注意，轮询策略有一个上限，当轮询了10个服务端节点还没有找到可用服务的话，轮询结束。

1. 随机策略（RandomRule）
   1. 随机策略：使用jdk自带的随机数生成工具，生成一个随机数，然后去可用服务列表中拉取服务节点Server。如果当前节点不可用，则进入下一轮随机策略，直到选到可用服务节点为止。
2. 可用过滤策略
   1. 策略描述：过滤掉连接失败的服务节点，并且过滤掉高并发的服务节点，然后从健康的服务节点中，使用轮询策略选出一个节点返回。
3. 响应时间权重策略
   1. 策略描述：根据响应时间，分配一个权重weight，响应时间越长，weight越小，被选中的可能性越低。
4. 轮询失败重试策略（RetryRule）
   1. 轮询失败重试策略（RetryRule）是这样工作的，首先使用轮询策略进行负载均衡，如果轮询失败，则再使用轮询策略进行一次重试，相当于重试下一个节点，看下一个节点是否可用，如果再失败，则直接返回失败。
   2. 这里还有一个点要注意，重试的时间间隔，默认是500毫秒，我们可以自定义这个重试时间间隔。
5. 并发量最小可用策略（BestAvailableRule）
   1. 策略描述：选择一个并发量最小的server返回。如何判断并发量最小呢？ServerStats有个属性activeRequestCount，这个属性记录的就是server的并发量。轮询所有的server，选择其中activeRequestCount最小的那个server，就是并发量最小的服务节点。
   2. 并发量最小可用策略（BestAvailableRule）的优点是：可以充分考虑每台服务节点的负载，把请求打到负载压力最小的服务节点上。但是缺点是：因为需要轮询所有的服务节点，如果集群数量太大，那么就会比较耗时。当然一般来说，几十台，几百台的集群数量是不用考虑这个问题的。因此对于大部分的项目而言，是一个不错的选择。
6. ZoneAvoidanceRule
   1. 策略描述：复合判断server所在区域的性能和server的可用性，来选择server返回