**Spring框架**

1. 什么是spring框架？

Spring框架是一个开源的框架，目的是为了简化企业级应用开发，是简单的JavaBean对象也拥有以前只有EJB才有的部分功能，spirng的核心是控制反转（IOC）和面向切面（AOP）;

1. IOC（控制反转） 和 AOP （面向切面）?

IOC （控制反转）: spring容器会直接为我们创建出所需要的对象，我们需要的时候就直接调用就可以了，实例化对象的方法有三种：类构造器，静态工厂方法，实例化工厂模式，我们在调用的时候不需要知道是哪种方法创建的，只需要直接调用；

AOP（面向切面）：在面向切面编程中，将一个个的Java对象某些类似的方面横向抽象成一个切面，对切面进行统一的操作如：权限验证、日志记录等公共操作的过程就是面向切面的编程；

DI（依赖注入）：spring为我们在创建所需对象时将其属性自动设置成所需要的值的 过程就是依赖注入的基本思想；

1. Spring中的设计模式？
2. 工厂模式：创建对象时不对客户端暴露创建逻辑，通过使用一个共同的接口来指向新创建的对象，spring中使用Bean Factory用来创建对象的实例
3. 代理模式：两种代理方式，若对象实现了接口，spring使用jdk的类代理，若没有实现接口，spring使用cglib库生成目标对象的子类
4. 单例模式：bean默认都是单例模式
5. 模板方法模式：解决代码重复问题
6. 前端控制器模式：spring提供了DispatcherServlet来对请求进行分发
7. 视图帮助：spring的jsp标签来辅助将分散的代码整合到视图中
8. 依赖注入：贯穿于BeanFactory / Application Context 接口的核心理念

4．spring的注解和xml注入先后？

Spring在2.5版本后支持注解的方式来配置依赖注入。注解注入在被容器在xml注入前处理，所以xml注入会覆盖掉注解注入对同一个属性的处理结果；

5．Spring的事务？

Spring支持两种事务管理：

1. 编程式事务：通过编程的方式管理事务，灵活性高，但难维护；
2. 声明式事务：将业务代码和事务管理分离，使用注解和xml配置来管理事务；

**Spring Mvc**

1. Spring Mvc的执行流程？
2. 客户端发请求到DispatcherServlet（前端控制器）
3. DispatcherServlet收到请求调用HandlerMapping（处理器映射器）
4. HandlerMapping调具体的处理器，生成处理器对象和处理器拦截器（如果没有不生成），返回给DispatcherServlet
5. DispatcherServlet调用HandlerAdapter（处理器适配器），处理器适配器经适配调用具体的Controller（后端控制器）执行并且返回ModelAndView对象给DispatcherServlet
6. DispatcherServlet将ModelAndView传给ViewReslover（视图解析器）解析返回具体的View
7. DispatcherServlet根据View进行渲染并响应客户端

**SpringBoot**

1. Springboot的简介、优点及策略？

简介：目的是Spring应用的搭建到开发的过程

优点：

1. 可以建立独立的spring应用程序，并且基于其Maven或Gradle插件，可以创建可执行的JARs和WARs；
2. 内嵌有Tomcat 或 Jetty 等servlet容器
3. 提供自动配置的“starter”项目对象模型（POMS）以简化Maven配置；
4. 自动配置spring容器
5. 减少代码生成，无需xml配置

策略：

SpringBoot框架中还有两个非常重要的策略：开箱即用和约定优于配置。

（1）开箱即用就是指在maven项目pom中添加需要的依赖jar，然后使用注解来代替繁琐的xml配置，这种策略使开发人员能够摆脱繁琐的配置和管理依赖，更加注重业务逻辑

（2）约定优于配置就是指springboot自己配置目标结构，开发者在结构中添加信息的设计范式，缺点是：降低灵活性，增加bug定位难度；优点是：减少xml配置，将代码编译，测试，打包自动化

2．Springboot的核心注解是哪个？如何组成的？

核心注解：@SpringBootApplication其中包含三个注解

@SpringBootConfiguration：组合@Configuration，实现配置文件的作用；

@ComponentScan：spring组件扫描；

@EnableAutoConfiguration：打开自动配置功能，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 })。

**Shiro**

1. 什么是shiro？作用？

Shiro是一个java安全框架

认证，授权，加密，会话管理，与web继承，缓存等

1. Shiro的核心组件？
2. Subject：当前用户的操作
3. SecurityManager：用于管理所有的Subject
4. Realms：用于进行权限信息的验证
5. Shiro的用户权限实现？（2种）
6. Authentication：验证用户身份的过程
7. Authentization：授权访问控制，对用户操作进行授权，是否允许进行当前操作；

4 .shiro的工作流程？

简单来讲就是：应用代码通过**Subject（当前用户的操作）**来进行**认证**和**授权**，而Subject又委托给**SecurityManager（用户操作管理者）**，我们需要给Shiro的SecurityManager注入**Realm（权限验证）**，从而让SecurityManager能得到合法的用户及其权限进行判断

**Linux系列集题**

1. **mybatis的缓存机制？**

一级缓存：SQL session

何时生效？

当mybatis.xml配置缓存开启之后，会默认生效

何时失效？

当执行了增删改操作之后，默认失效；

二级缓存：mapper.xml中的namespace

何时生效？

当mybatis.xml配置开启缓存之后，sqlsession缓存关闭之后，并且执行了查询语句，就会将数据存储到namespace中

执行增删改是否失效？何时失效？

不会失效，当tomcat重启之后缓存失效（一旦开启了二级缓存只能手动清理，mybatis会监听增删改操作，一旦发现namespace会自动清空）

一级缓存和二级缓存，数据在哪？

在tomcat内存空间中，一旦Tomcat重启会释放资源，所有缓存失效

缓存的存储流程？

当客户端发送查询请求的时候，首先会从缓存中进行查询，如果没有，就回去数据库中进行查询并且返回，把数据先存在缓存中，然后在返回给客户端

1. **Svn（项目版本控制工具）**

在项目提交的时候要注意的问题？

1. 在提交时，应先更新
2. 项目冲突，不更新直接提交造成项目冲突（revert可返回最后一次正确的版本 项目名---subversion---revert）
3. 如果是maven项目，只用提交src和pom.xml即可，不要提交 .idea文件

3**．Redis（键值对数据库/非关系型数据库）**

什么是集群？多台服务器之间处于联通状态，并且在做同一件事

为什么有集群？架的诞生是因为当前程序无法满足项目的需要

什么是缓存？经常使用，不经常更改的数据（地区/性别/身份证号）

* 1. 什么是redis？（以集群的形式存在，通常为6台，3主3从）
     1. 非关系型数据库（所有数据都是独立存在的，数据与数据之间没有关联，没有表的概念）
     2. 键值对数据库（存入的数据都是以key-value的形式存在/json）

Redis就是通过key值来获取value值

面试：Redis支持哪些数据格式？

Key值：数据格式永远是String类型

Value值：

String：最常用的

List：有序，值可重复

Set：无序，值不可重复

Zset：有序，值不可重复

Hash：1 . HashMap，2 . HashTable（线程安全）

为什么redis会速度快？

Redis实在内存中做的计算，并且把数据持久化到硬盘上

**4．Nginx（反向代理服务器）**

什么是代理？

处于真实服务器与真实客户端之间的中间件（带有某种作用）

代理分为两种：正向代理，反向代理

正向代理（翻墙）：与客户端在一起，所有配置在客户端完成，服务器不知道真实客户端是谁

反向代理（所有项目中都带有反向代理）：与真实服务器在一起，所有配置在服务器端完成，客户端不知道真实服务器是谁

什么是负载均衡？

负载：客户端所发送的请求就是服务器端的负载

负载均衡：客户端所发送的请求按照某种规则/算法分发给不同的服务器处理

负载均衡也是反向代理的一种体现！！！

负载均衡的算法规则：

1. 轮询（默认）
2. Weight（权重）：默认为1，正无穷，不为负
3. Down：配置down的服务器不参与负载均衡
4. Backup（后备服务器）：主服务器宕机，后备服务器启动，主服务器恢复，后备服务器关闭
5. **Ftp(文件传输)**
6. 什么是ftp？

ftp和http一样，都是一种协议，http是网络协议，ftp是文件传输协议

1. ftp的作用？

一般情况下实现文件的传输，在项目中一般实现了文件的上传和下载！

**6．Zookeeper（注册中心）**

1. 什么是zookeeper？特性？

Zookeeper是一个致力于开发和维护，用于协调和管理项目，并且是以集群形式存在的开源服务器。也就是注册中心（注册中心：eureka，consul），保证了CP性（数据一致性）

1. Zookeeper的选举机制什么？
2. 半数机制（paxos协议）：集群中半数以上机器存活，集群可用。所以Zookeeper适合安装奇数台服务器。
3. 当第一次进行选举时，所有节点各自依次发起投票，投票包含自己的服务器ID和最新事务，发现别人的myid比自己的大，就会投票目前已知myid最大的节点，每轮投票后，服务器统计当前是否存在票数过半的节点，如果有，该节点将会成为leader，其他节点变为follower
4. 一旦leader产生， leader的高32位就会变化(某种算法进行变化，会增大), 其他的不变但是所有服务器的低32位会自增，当leader宕机的时候，在使用半数选举规则的时候还要考虑zxid的情况其他的服务器高32位一样， -低32位不-样，低32位越小，成为leader的可能性就越大,被操作的可能也越大按照自然顺序进行递增(从小到大的顺序进行自增)
5. Zookeeper的设计模式是什么？

观察者模式：zookeeper会向注册进来的provider服务器索要当前的状态，当一台provider服务器的状态发生改变的时候，自动通知所有的provider服务器（观察者）做出相应的应急处理

**7．Dubbo（远程调用框架）**

使用Restful风格的调用弊端？

1. Consumer的Controller调用provider的controller，provider一宕机，整个服务崩溃
2. 基于http传输，对网络要求高，网络延迟就会发生异常
3. 违背了mvc风格的原则
4. 因为是http协议传输，安全性能差

RPC:为了解决restful的弊端（remote producer call:远程 过程 调用，也称之为远程调用框架）

本地调用？远程调用？

本地调用：在同一个项目/域，consumer发起调用请求，最终达到了自己的预期的数据

远程调用：在不同的项目/域，consumer发起调用请求，最终达到了自己的预期的数据

1．什么是dubbo？

Dubbo是阿里巴巴开发一个高性能、轻量级、基于Java语言开发的开源的RPC（远程调用框架）

2 .Dubbo的三大核心？

面向接口的方式

智能容错以及负载均衡

自动服务注册和发现

1. 使用Restful风格的调用弊端？
2. Consumer的Controller调用provider的controller，provider一宕机，整个服务崩溃
3. 基于http传输，对网络要求高，网络延迟就会发生异常
4. 违背了mvc风格的原则
5. 因为是http协议传输，安全性能差

**8．ElastiSearch（搜索引擎） kibana（es可视化操作平台）**

1．什么是es？

Es是基于lucene开发出来的搜索引擎架构，使用的是restful的风格

2 .Es的三大核心？

1. index：相当于关系型数据库中的库
2. Type：相当于关系型数据库中的表
3. Document：相当于关系型数据库中的表数据

3 . 什么是kibana？

kibana是一个可视化的开源平台，而且也是elastic公司的，是用于和ES合作，帮助ES实现可视化界面

面试题：

1 .模糊查询效率是否高？为什么？

不高，因为需要全表扫描

2 .模糊查询是怎么运行的？

模糊查询需要进行四次查询， like '%zhangsan' like 'zhangsan%' like '%zhangsan%' 'zhangsan'

3 . SQL优化

1．当只有一行数据的时候使用 limit 1

这种情况用于已知只有一条数据，或者只需要一条数据的时候使用，这样数据库引擎找到一条数据后就会停止搜索，增加性能

2．对操作符的优化，尽量不采用不利用索引的操作符

如：in，not in，is null，is not null， < , > 等

3 .尽量避免select \* 查询，使用具体字段名，减少 ‘\*’ 转换 ‘字段’ 的时间，避免多余的列

4 .慎用distinct（去重）关键字，执行 去重 操作时会对数据进行比较，过滤重复数据，大量占用资源,cpu时间

5 .不要使用select count（\*） 这样会全表扫描并且没有任何意义，可以用count（1）代替，避免字段的转换，或者用效率最高的count（主键）

4 .select count(\*) 和 select count(1) 和 select count(主键)哪一个效率高？为什么？

Count(\*) : \*是代表通配符，查询时，先将 \*转换为字段，再将字段编译二进制进行全表扫描，将数据量进行叠加

Conut（1）： 1代表通配符，先将字段转换为1，因为是数字不需要二进制编译，直接进行全表扫描，将数据量叠加

Conut（主键）：效率最高，，因为使用主键其实是使用主键的索引，所以速度会很快

5 . “select \* ” 和 “select 字段” 哪个效率高？为什么？

Select 字段的效率高，因为select 字段是直接对字段进行扫描，而select \* 需要先将\*转换为字段名，再进行扫描

6 .Es和Solr的区别？

Es和solr一样都是基于lucene开发的，实现效果都是一样的

|  |  |  |
| --- | --- | --- |
| 区别 | Elasticsearch | Solr |
| 配置 | 开箱即用 | 配置复杂 |
| 注册中心 | 自带的注册中心 | Zookeeper注册中心 |
| 数据格式 | Json | Json，csv，xml |
| 第三方支持程度 | 手动实现第三方插件 | 官方的功能多，自带配置 |
| 具体使用方向 | 建立索引块，查询慢，适合实时搜索：社交软件 | 查询速度快，建立索引慢（增删改）：电商 |
| 开发力度 | 新产品，维护更新快，使用少，学习时间成本高 | 成熟体系，社区完备，时常在发展完善 |

**9．Mycat（集群数据库管理中间件）**

1．什么是mycat？

Mycat是面向企业级开发的开源大数据库的集群中间件（客户端和服务器端之间或者服务器端和服务器端之间的一个项目/一套架构，称之为中间件），支持关系型数据库和非关系型数据库，支持事务（ACID），可替代MySQL的加强数据库以及昂贵的Oracle集群

2 .Mycat的核心思想（设计思想）

Mycat是HDFS大数据的新型sql server

HDFS（Hadoop的分布式文件系统思想）

Hadoop （大数据）

Distribute (分布式)

File (文件)

System （系统）

Mycat沿用了Hadoop的分布式文件系统思想

HDDS (Hadoop的分布式数据系统)

Hadoop （大数据）

Distribute (分布式)

Data （数据）

System （系统）

3 . mycat的功能？

1. 读写分离：在同一个系统(项目)中，将对数据库的读和写操作分别在不同的集群节点上处理，就称之为读写分离（读写分离配置需要重新配置mysql，因为读写分离有主从复制，mycat并不支持主从复制，所以必须要用mysql所自带的主从复制）
2. 分库分表：在同一个项目(系统)中，把数据量过大的库和表分开存放在不同集群节点中，称之为分库分表

4 . MapReduce:(Map Reduce)是什么？

MapReduce思想就是现在处理大批量数据的思想！

分为map和reduce。

Map负责自己节点上的数据计算；reduce是各个map的数据汇总。

5 . 如果将数据库从服务器端移走，单独放在一个新的数据库服务器中（好处：可以有效缓解服务器端的压力） 但在服务器端和数据库服务器之间进行交互时，会出现什么问题？

当数据量很大时，会出现网络阻塞；当连接请求超过最大连接数时，也会出现网络阻塞。若网络阻塞的时间超过了最大等待时间，则需要切换数据库（建两个库：主库和从库）

6 .什么是主从复制？

所有的数据都是主节点向从节点发送数据，从节点无论何时都不可能向主节点发送数据；一旦主节点宕机，从节点变成主节点。

情景：假设有一张user表中有100万条数据需要查询，主库和从库中所有数据都是一样的，无论是从主库还是从库中获取数据，均可能直接导致数据库的宕机，此时如何处理？---分库分表

7 .分库分表是否需要主从复制？

如果不是集群的分库分表，就不需要主从复制。

8 .读写分离：把写的库配置成主库，读的库配置成从库。

读写分离四种配置规则：

假设主库宕机：

1.当主节点宕机后，从节点既可以读又可以写。如果主节点没有宕机，从节点只能读。

2.无论主节点是否宕机，从节点都可以读和写。

3.无论主节点是否宕机，都只能读。

4.主节点也可以读

！！！mycat其实并不会从真实数据库中读数据，所做的是映射数据。（映射：实现了表复制）

**10．Docker（简便部署配置工具）**

LXC（Linux container(Linux容器)）：是一种新的虚拟化技术，这种技术就称之为docker

docker就是把所有需要配置全部都抽取出来做封装，所有的Linux只需要加载封装好的文件，也就是说所有的Linux加载都是同一套文件，解决了冗余问题

按需加载 : 根据需要的配置进行加载，在Linux启动的时候，只需要加载和启动相关的配置即可

分类存储 : 把Linux中配置按照不同的类型进行规划存放

1 .什么是docker？

Dockers是和运维有关的工具，简化并安全配置项目的部署运行，核心思想是：集装箱思想：方便传输，分类存储，独立存在，所谓的docker也就是集装箱和承载的集装箱系统

2 .docker的三大核心：

1. 镜像 : 类似iso文件(使用编程语言开发的系统)，只读状态，通过镜像可创建容器
2. 容器 : 由镜像所创建出的系统就称之为容器，一个镜像可以创建多个容器

---数据卷：把容器中所需要持久化的数据保存到本机的硬盘(文件夹)上，这个硬盘(文件夹)就称之为数据源

--把容器中的数据持久化到本机硬盘上，称之为数据卷的作用（数据共享）

1. 仓库（还没讲） ：类似与代码仓库，是Docker集中存放镜像文件的场所（有时候会看到有资料将Docker仓库和注册服务器（Registry）混为一谈，并不严格区分。实际上，注册服务器是存放仓库的地方，其上往往存放着多个仓库。每个仓库集中存放某一类镜像，往往包括多个镜像文件，通过不同的标签（tag）来进行区分）

--- 根据存储的镜像公开分享与否，Docker仓库分为**公开仓库（Public）**和**私有仓库（Private）**两种形式。

面试题：

1 .当把容器停止后，修改主机上的内容，那么容器再次启动的时候数据是否被修改了？

修改，因为容器会一直监听主机的内容，一旦发现自己不一样，会自动同步信息

2 .**带权限的数据持久化**（使用场景）：主机有读和写的权限，容器只有读的权限

读写分离：两个容器(mysql01(写),mysql02(读))，一旦容器损坏或者容器丢失，就会造成mysql中数据的丢失，所以需要把**数据持久化**到本机硬盘上，mysql01写入**权限**，mysql02只读**权限**

3 .主机只能读，容器即可以读又可以写？（不会问这个问题，主要是问**带权限的数据持久化使用场景**）

逻辑上没有问题，但是要注意主机和容器是包含关系的（容器在主机中），主机都没有写的权限，容器也不会有

**Spring cloud系列**

Spring cloud是一个生态圈/体系（每一个服务都可以叫一个框架，把这些服务整合才是一个spring cloud）

面向服务：是一种不同于面向对象的特殊思想，简单来说就是把项目 功能模块化，这些功能模块之间相互依赖。好处：模块功能独立，谁用到都可以直接集成使用。

微服务：是一种思想，没有具体的定义，是根据当前的项目业务所定义的实现

微服务架构：多个各不相关的项目组合起来的一个完整项目，这种架构风格就是微服务架构，实现微服务思想/架构的体现就是spring cloud

Spring cloud和dubbo的区别？（了解一下）

1 .Dubbo是二进制，传输速度快，占用宽带非常低

SpringCloud 是基于http的restFul，http（占用宽带高）

2 .Dubbo提供了多个开源注册中心（zookeeper，dubbo自带的）

SpringCloud 2019提供了zookeeper的支持，注册中心只有 eureka

3 .Dubbo 监控和日志不完善，需要手动配置并需要独立的服务器运行

SpringCloud 有非常完善的监控和日志系统

4 .Duboo 轻量级框架

springCloud 相对Dubbo非常沉

1. **. Eureka（注册中心）**
2. 什么是eureka？

Spring cloud生态圈的组件，是致力于开发和维护，用于协调和管理项目，并且是以集群形式存在的服务器。也就是注册中心，保证了AP性（可用性）

1. 无论是zookeeper还是eureka，特性只会在自己身上体现，跟provider和consumer都没有关系
2. Eureka中没有主从之分，服务器（节点）之间平等（平行），但是也是一个集群的形式存在，每一台节点都会在其他节点安装监听，监听其他节点的心跳（90s）

**面试题：**

1 . consumer请求调用provide时，eureka宕机，consumer是否可以调到provider？

可以调到，因为有缓存区的存在，缓存区存的是缓存信息（provider存活状态）

2 . eureka宕机，provider能否注册进eureka中？

可以注册，但是不会被发现，因为注册进来的时候显示的是不可用的状态，直到下一次发送心跳，更新状态信息

3 . eureka和zookeeper的区别？

1. zookeeper保证了数据的一致性，eureka保证了数据的可用性
2. Zookeeper中provide只向leader注册即可，eureka中provider需要向所有节点都注册信息
3. Zookeeper在选举leader时，节点都处于宕机状态，zookeeper处于瘫痪状态，eureka会持续接收信息，只是接受进来之后的状态为不可用
4. **. ribbon（客户端的负载均衡）**

客户端层面的负载均衡，所有配置都在客户端完成。可以结合eureka实现，也可以不结合

1. **. Feign（客户端的负载均衡）**

客户端层面的负载均衡，所有的配置都是在客户端完成，在eureka配合下实现

Ribbon和feign的区别？

1. 真实环境只会用feign，不会使用到ribbon
2. Feign的负载均衡是基于ribbon的，也就是feign是集成的有ribbon，但是Feign并不只有负载均衡，还有一个功能就是远程调用
3. Ribbon调用也是通过consumer的controller调用provider的controller实现的：RestTemplate中的注解@LoadBalance（开启负载均衡）--->如果不使用默认的轮询算法则还需要自己自定义算法--->在controller中通过restTemplate来模拟http的请求--->最后把请求发送给provider(controller调用controller) 其实违反了MVC的规则

Feign的实现：consumer的controller调用service层（定义接口）-->映射provider的controller，controller-->调用service-->调用mapper

1. **. Hystrix（断熔器）**
2. 熔断：服务器之间进行调用的时候如果出现了故障，就会进入熔断中，调用后备的方法，返回客户端可以接受的信息
3. 服务的降级：当服务器压力过大，为了核心业务的正常使用，就会将一些暂时无关紧要的服务给停掉，将资源全部供核心业务使用，等到核心业务稳定之后，重启其他服务，这就是服务的降级
4. 服务的隔离：
   1. 线程池隔离：使用线程池来存储请求，可以设置任务的超时时间，堆积的请求入线程池排队（优点：利用线程池，解决流量洪峰问题，并且可以异步调用，可以设置线程池大小，控制请求的并发量。缺点：在实际开发过程中，基本一个依赖创建一个线程池，线程之间切换、调度和排队，消耗系统资源。）
   2. 信号量隔离：利用原子操作的类型（如AtomicInteger等），记录请求个数，每次线程过来数量加1，线程结束之后，数量在减1（优点：控制请求的线程数量，没有线程之间的切换开销，提高了系统资源的利用率。缺点：需要监控线程结束状态，无法实现异步，采用计数器请求量达到峰顶，后续请求会直接丢弃，不会进入线程队列等待。）
5. 服务的限流（解决高并发）：当服务器负载过大的时候，就不再支持客户端的访问了，会直接走熔断进行处理后续的请求
6. 灰色发布（金丝雀发布）：保证整体系统的稳定，在初始灰度的时候就可以发现、调整问题，以保证其影响度。

灰度期：灰度发布开始到结束期间的这一段时间，称为灰度期。

例：

让一部分用户继续用产品特性A，一部分用户开始用产品特性B，如果用户对B没有什么反对意见，逐步扩大范围，所有用户都迁移到B上面来。

1. **. Zuul&Gateway（路由网关）**
2. 路由：将外部请求转发到具体的微服务实例上,是实现外部访问统一入口的基础
3. 过滤：负责对请求的处理过程进行干预,是实现请求校验、服务聚合等功能的基础.

gateway和zuul的区别是什么？

所实现的功能没有任何的区别，区别在于底层设计

gateway所使用的是netty(IO-->NIO)，针对于NIO做的多线程的封装，netty中的异步IO

zuul使用的是原始的servlet3

如果是低并发的情况下，gateway和zuul没有任何区别(速度，效率)，但是一旦涉及到高并发的时候，gateway要比zuul至少10倍，差距非常明显

1. **.分布式配置中心（config）**
2. 什么是分布式配置中心？

就是为微服务架构中的微服务提供集中化的外部配置支持，配置中心为各个微服务应用的所有环境提供了中心化的外部配置（可能比较难理解，想知道是什么意思就要知道为什么这么配置：这么配置就是为了解决微服务中很多个provider中的application.properties配置管理问题，以及配置冗余问题，把这些配置集中到一起进行存放，并且把重复的配置提取出来解决冗余）

1. 运行原理？

1 . git hub 上存放我们的配置文件

2 . config-server 远程连接到 git hub

3 . config-client 连接到config-server

运行：当我们启动config-client 服务的时候，client 会通过连接的 config-server 拿到远程git 上面的配置文件，然后通过 Spring 加载到对象中。

1. 在配置client端的时候，为什么要有两个配置文件（bootstrap和application）以及它们的加载顺序？

目的是为了查漏补缺，因为在配置的时候是有差异的，那些一样的配置是放在GitHub上的，有差异的配置是需要application中的

两者加载是先加载bootstrap的，将配置加载到application中，在一并进行加载，就是先从GitHub上把bootstrap加载下来，两者进行合并（当bootstrap和application配置一样的时候，先加载谁就会去覆盖另一个）

1. **.Zipkin（服务追踪）**
2. 什么是zipkin？

Zipkin是一款开源的实时数据追踪系统，主要功能是聚集来自各个异构系统的实时监控数据。

1. 为什么要用zipkin？

随着业务越来越复杂，系统也随之被拆分，看似简单的一个请求可能需要多个服务的调用才能完成，当请求变慢或者不可用时，这时候就需要解决如何快速定位服务故障点，Zipkin分布式跟踪系统就能很好的解决这样的问题。zipkin的服务器端(相当于一个监听，一直在监视项目是如何被调用的，以及项目整个调用过程和情况)

1. **.** **bus(config+bus(消息总线))**
2. 消息总线：实现云配置信息的动态实时更新
3. 运行原理：在github修改配置信息，正常情况下需要重新启动项目，这么一来整个微服务就会变的非常缓慢，但是经过bus之后，修改配置可以直接同步到项目中，也就是说在github上一旦修改配置信息，不再需要重新启动所有微服务，会自动加载进来，bus会通过各个服务实现refresh操作，让他们重新去加载新的配置信息。
4. **.consul（注册中心）**
5. 什么是Consul？

Consul是一个分布式，高可用且支持多数据中心的服务发现，配置和编排工具。Consul支持大规模部署，配置和维护面向服务的体系结构。也就是注册中心（注册中心：eureka，zookeeper，Etcd（没讲过，有兴趣自己百度）），保证了HCP性（高可用性）

1. Consul的主要功能？
   1. 服务发现：通过dns或http接口是的消费者发现服务，应用程序可以轻松找到所依赖的服务。
   2. 健康检查：防止将请求转发不健康的主机
   3. 键值存储：可以使用分层键/值存储，比如功能标记、动态配置等。
   4. 多数据中心：开箱即用，不需要复杂的配置。这就意味着不用建立抽象的逻辑思维来扩展多个地区。
2. 选举机制？
3. 其实consul的内部机制实现和zookeeper一样，当leader宕机以后，重新选举leader的时候整个集群仍然会处于瘫痪状态(因为选举机制的不同，导致整个集群瘫痪的时间不同，三台服务器，选举leader的时候只需要0.02ms)
4. 对比zookeeper，选举机制做了增强，算法的不同，导致选举机制不同：

zookeeper所使用的算法是Paxos(半数) consul所使用的算法Raft(也支持半数选举规则，只是一个条件)

1. 选举模式的时候每一台consul都有三个状态
   1. leader(选举成功的状态
   2. follower(选举成功的状态)
   3. condidate(候选人)
   4. 第一台consul启动-->候选人状态-->向自己投一票
   5. 第二台consul启动-->候选人状态-->第一台开始催眠(把第二台consul节点变成sleep)-->会把票投给第一台
   6. leader产生
   7. 第三台consul启动-->候选人状态-->第一台开始催眠(把第三台consul节点变成sleep)-->会把票投给第一台
   8. 如果第三台把票投给自己，leader会剔除第三台

(一：程序员可以直接指定leader，二：按照启动的顺序先后判定谁是leader)

如果第一轮选择无法选出leader，consul整个系统会随机sleep网路故障的机器-->然后把sleep的机器全部投票给苏醒的机器(哪一台先启动先投哪一台)

如果leader宕机，重新选举leader，并且所有的状态都为正常状态，直接使用的半数机制