亿级流量秒杀服务层架构

参考链接

系统架构知识图谱（一张价值10w的系统架构知识图谱）

https://www.processon.com/view/link/60fb9421637689719d246739

秒杀系统的架构

https://www.processon.com/view/link/61148c2b1e08536191d8f92f

服务层的服务治理

服务治理保护： 服务注册发现、分布式配置、服务网关、 服务路由、 服务监控 等

服务注册发现

对于微服务来讲，服务之间也是需要做服务发现的，常见的框架是 Dubbo 和 Spring Cloud，服务的注册中心可以是 ZooKeeper、Consul、etcd、Eureka 、nacos等。

分布式配置

测试环境的参考地址：

> http://cdh1:7788/crazymaker/redis/dev

Spring Cloud config

秒杀练习系统服务层模块

秒杀系统的实现会有多种多样的版本，本书从方便演示的角度出发，设计了一个相当简单的秒杀练习版本，具体分为三个主要的模块。

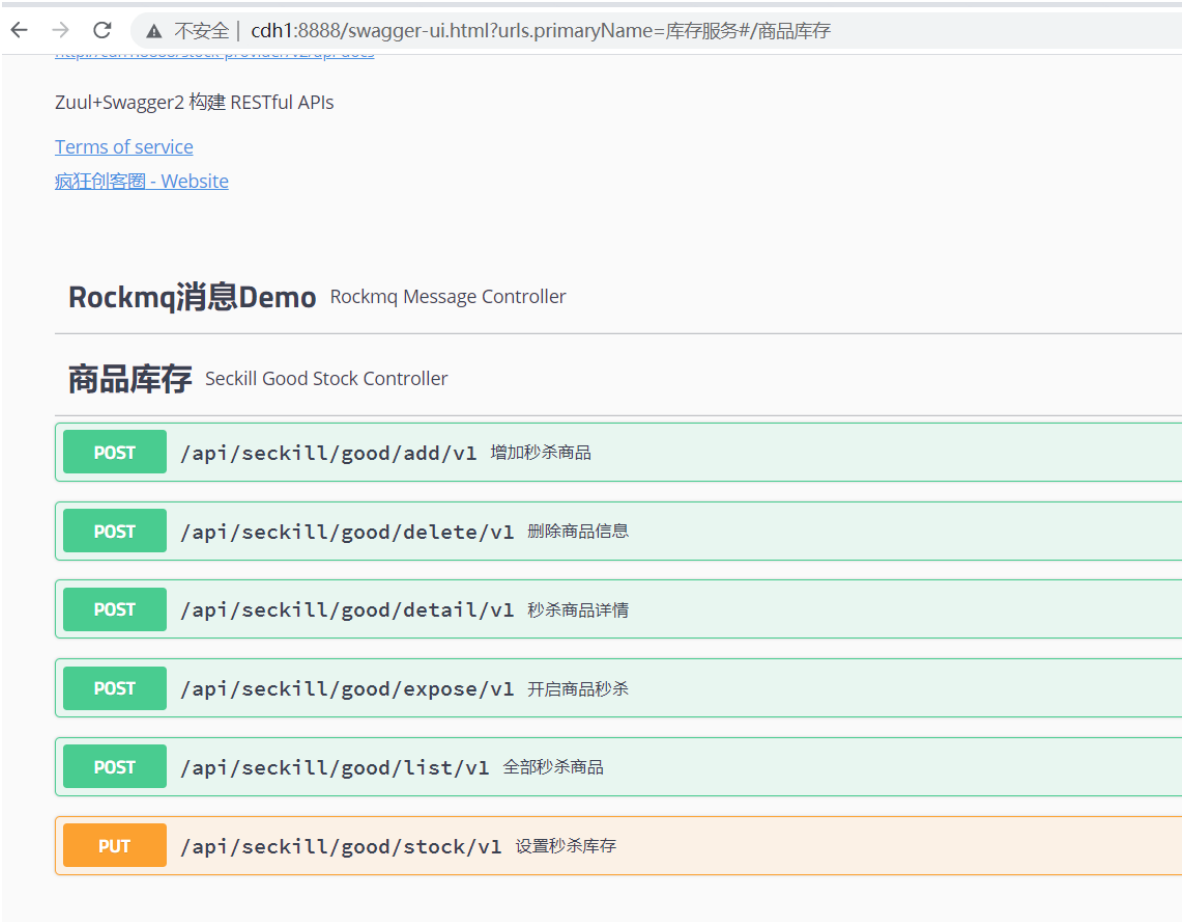
seckill-web模块

此模块是一个独立的Spring Boot程序，作为一个静态的Web服务器独立运行，主要运行秒杀的前端页面、脚本。

> 生产场景中，为了提高性能，可以将这个模块的所有静态资源，全部迁移到Nginx高性能Web服务器。相当于为了缩短访问距离，从 服务层移动到 接入层。

stock-provider模块

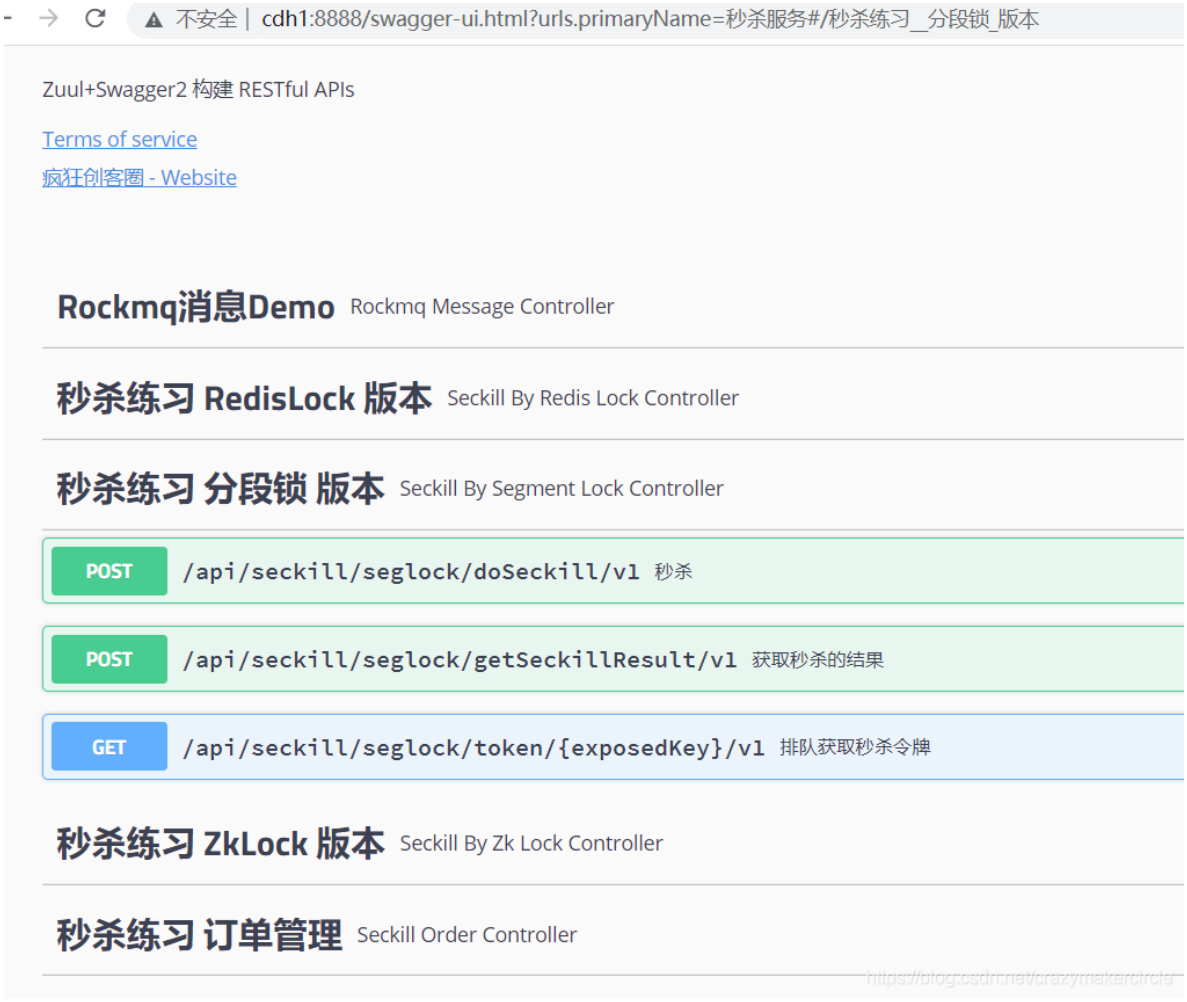
库存管理的后端Spring Cloud微服务提供者，主要功能为，秒杀商品sku管理，秒杀暴露等功能。



秒杀的后端Spring Cloud微服务提供者，秒杀的管理模块，主要的功能。

seckill-provider模块

秒杀的后端Spring Cloud微服务提供者，主要功能为，秒杀商品等功能。



uaa-provider模块

用户账号与认证（UAA）的后端Spring Cloud微服务提供者，主要运行用户认证、用户信息相关的后端接口。

> 本案例聚焦高并发实战，对此功能进行了弱化

以上三个模块的关系如下：

seckill-web模块作为的静态资源程序，会将秒杀的操作页面呈现给用户；

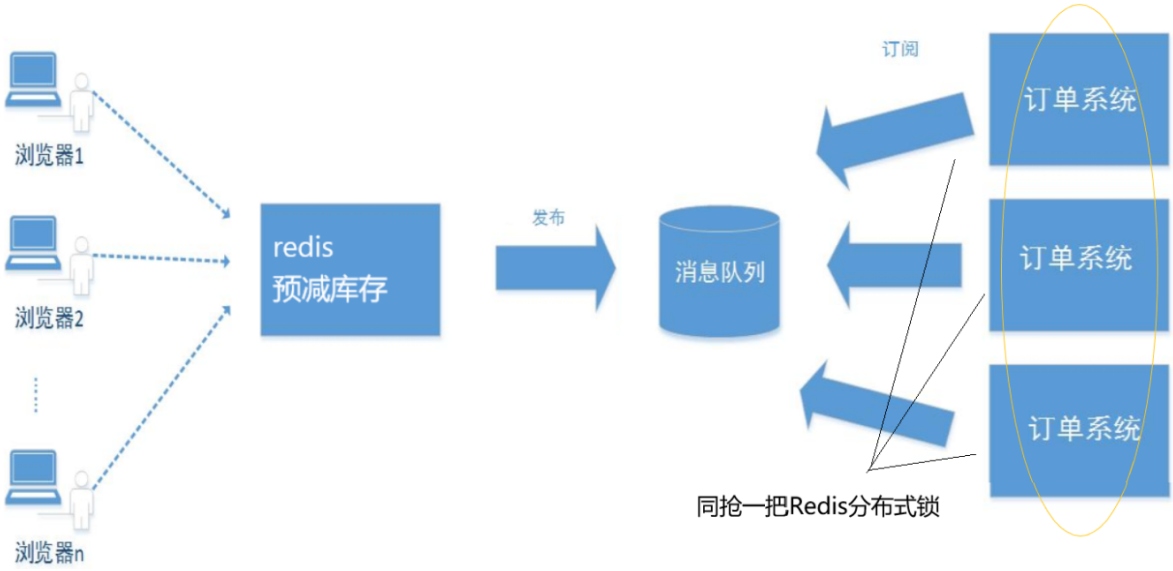
seckill-web的页面会根据用户的操作，通过网关Zuul发送给后端的stocker-provider和seckill-provider微服务提供者。

有效流量削峰架构

高并发操作迁移到并发量更高的nginx+redis+lua ，提交操作变成两段式：

- 第一阶段为有效流量锁定， 将有效的流量识别出来，过滤掉无效的流量 ，申请令牌，申请预减减库，申请成功之后，进入服务层
- 第二阶段为有效流量落地，进入服务层之后，进入消息队列，秒杀服务从消息队列拉取令牌，确认令牌，然后完成下单操作。 查库存 -> 创建订单 -> 扣减库存。通过分布式锁保障解决多个

provider实例并发下单产生的超卖问题。

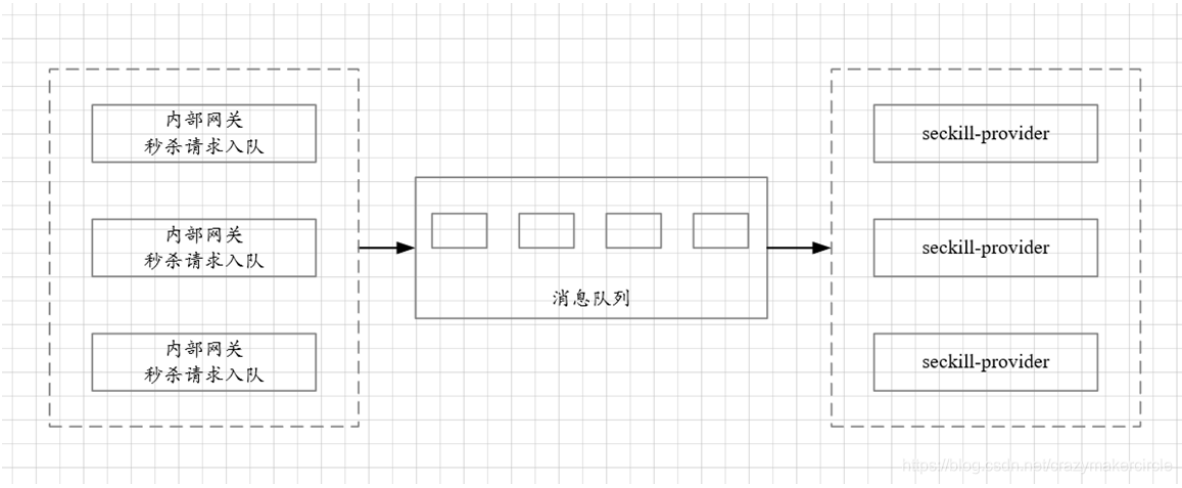


可以使用消息队列削峰

削峰从本质上来说就是更多地延缓用户请求，以及层层过滤用户的访问需求，遵从“最后落地到数据库的请求数要尽量少”的原则。通过消息队列可以大大的缓冲瞬时流量，把同步的直接调用转换成异步的间接推送，中间通过一个队列在入口承接瞬时的流量洪峰，在出口平滑地将消息推送出去。消息队列就像“水库”一样，拦蓄上游的洪水，削减进入下游河道的洪峰流量，从而达到减免洪水灾害的目的。使用消息队列对秒杀进行削峰，具体的架构如图10-6所示。

对于秒杀消息的入队，可以直接在内部网关完成。内部网关在完成用户的权限验证、秒杀令牌的有效性验证之后，将秒杀消息发往消息队列即可。秒杀服务通过消息队列的订阅，完成秒杀消息的消费。

图10-6 使用消息队列对秒杀进行削峰



常用消息队列系统：Kafka、RocketMQ、ActiveMQ、RabbitMQ、ZeroMQ、MetaMQ等。本书的内容主要聚焦在Spring Cloud和Nginx，对消息队列在这里不做过多的介绍，使用消息队列进行削峰的秒杀实现版本，请参见后续的疯狂创客圈社群博客。

两阶段提交之有效流量落地方案优点：

有效流量锁定、消息队列削峰，故同时解决性能问题。

两阶段提交之有效流量落地方案缺点：

- 数据不一致的问题：

由于异步写入DB，可能存在数据不一致，存在某一时刻DB和Redis中数据不一致的风险。

- 可能存在少买

可能存在少买，也就是如果拿到号的人不真正下订单，可能库存减为0，但是订单数并没有达到库存阀值。

少买的解决办法

一是秒杀，其实是亏本或者是不赚钱的买卖，存在微量的少买，不会对商家造成损失。

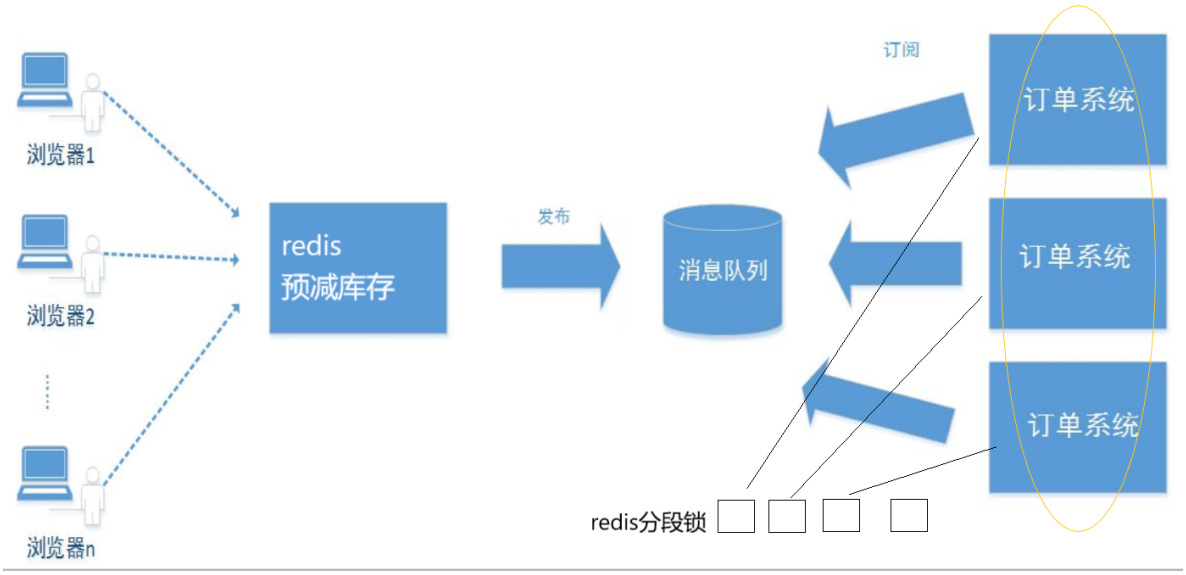
二可以通过定时检查和扫描的方式，对的缓存中的令牌数据进行订单检查，10分钟之内没有下单，可以删除令牌，恢复其售卖。

分布式锁的高并发性能提升

可以使用Redis 分段锁

假设下一个订单200ms，每秒3订单，600订单，200m

为了达到每秒600个订单，可以将锁分成 600 /5 =120 个段，每个段负责5个订单，600个订单，在第二个阶段1秒钟下单完成。



有关Redis分段锁的详细知识，请阅读下面的博文：

Redis分布式锁 （图解-秒懂-史上最全）

有效流量锁定之lua脚本执行

脚本加载、 发放令牌、校验令牌

ng调用lua，去访问redis

java调用lua脚本，去校验令牌

微服务网关

微服务集群的统一访问入口是微服务网关 gateway

微服务网关 gateway 就有责任自动的发现后端服务、当后端服务有所变化的时候，能够实现健康检查和动态的负载均衡。

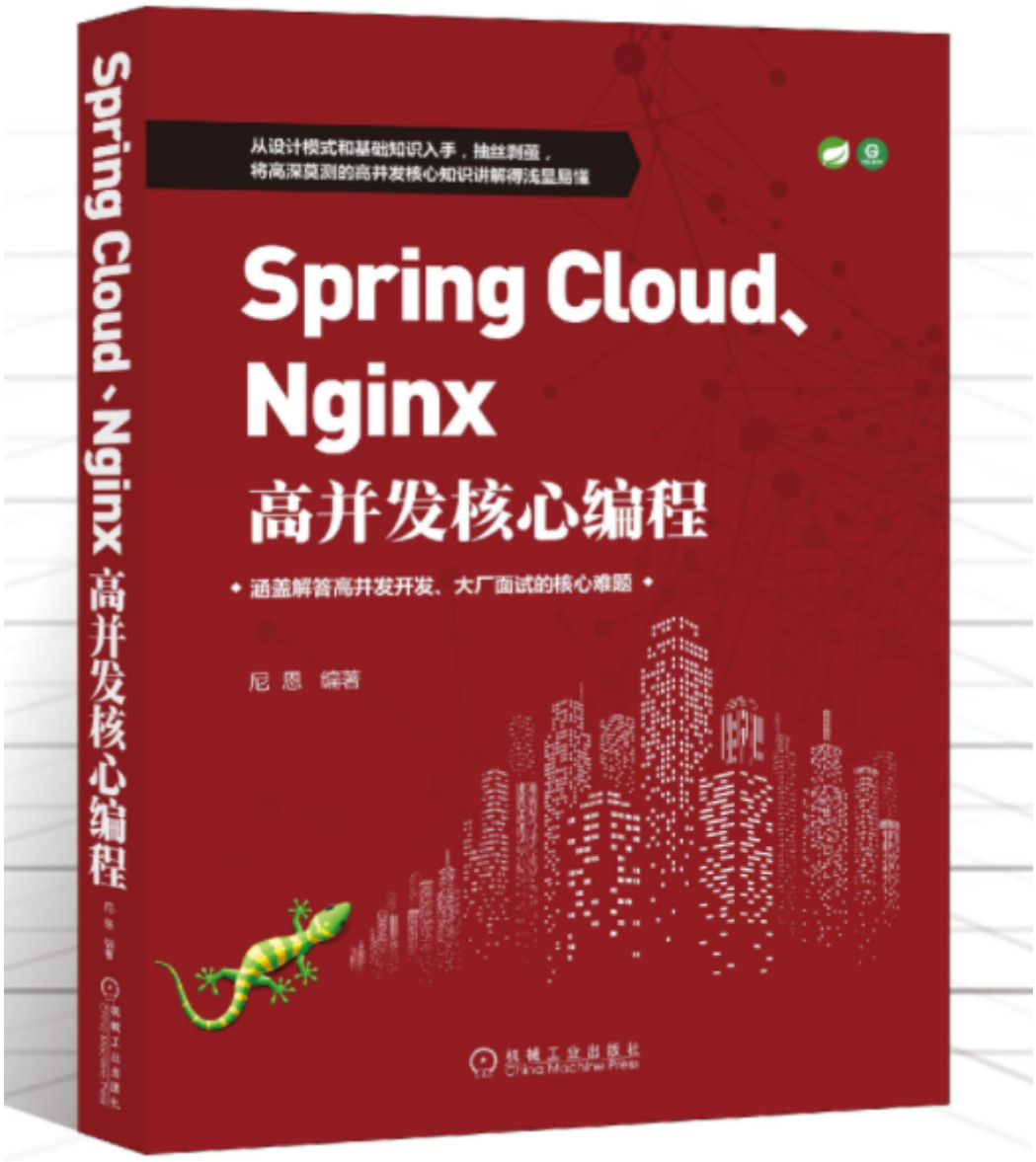
常见开源组件

SpringCloud gateway

> https://www.cnblogs.com/crazymakercircle/p/11704077.html

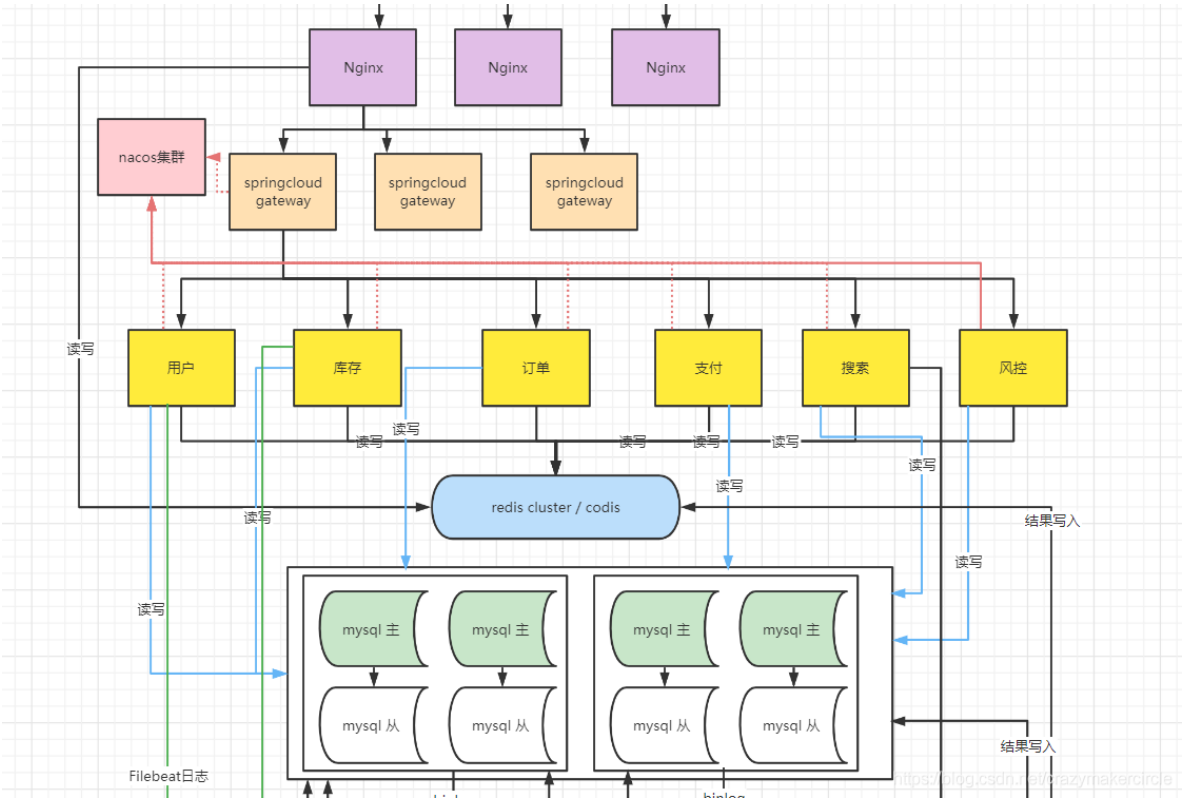
zuul

《SpringCloud、Nginx高并发核心编程》



微服务网关与微服务实例的关系

当 后端微服务成为瓶颈的时候，只要增加服务器数量，服务网关可以做到负载均衡，就能扩展站点层的性能，做到理论上的无限高并发。



网关身份的验证

另外，zuul网关提供身份的验证。**结合springsecurity jwt 一起提供**

为啥要跳过Zuul内部网关呢？内部网关需要对请求的URL做用户权限验证，如果请求header没有带token或者没有通过验证，请求会被拦截并返回未授权的错误。

> 这里，为了在练习时调试方便，建议直接跳过Zuul内部网关的权限验证功能。

网关的开发、部署、和使用

zuul的打包

fatjar（肥包）

生产环境，主要使用docker镜像的模式，推送到git仓库

zuul的部署

zuul的控制台界面（其端口为server.port配置项的值），大致如图2-2所示。

http://cdh1:8888/swagger-ui.html

Eureka Server的控制台界面（其端口为server.port配置项的值），大致如图2-2所示。

http://cdh1:7777/