个人信息

- 王康 / 男 / 6年 工作经验 / 本科
- 电话：18610842638； Email：focusj.x@gmail.com
- 博客：http://www.jianshu.com/users/13542edebea3/

工作经历

2017年11月~今 摩拜 后端工程师

2016年03月~2017年11月 北京第三石信息科技有限公司 后端工程师

2014年10月~2015年11月 ThoughtWorks 全栈工程师

2012年04月~2014年09月 北京尤尼信息科技有限公司 全栈工程师

技术总结

- 熟练使用Java, Scala, Golang, 了解Golang,Python,JavaScript. 理解面向对象和函数式编程思想.
- 熟悉多线程编程和异步编程, 了解多种并发模型. 熟练使用Java多线程和Akka框架.
- 了解JVM; 了解常见垃圾回收算法及工作原理.
- 了解微服务架构, 及REST API设计.
- 熟悉缓存和消息队列的用法. 了解Redis，RabbitMQ，Kafka的使用.

项目经验

摩拜监控系统

职责: 负责设计摩拜监控系统和核心功能开发

技术选型：Java8，Golang，SpringBoot，gRPC，InfluxDB，Istio，Kubernetes

项目背景

摩拜监控系统主要分为两部分：框架级基础指标监控和业务监控。框架指标（包括jvm的信息，服务接口调用，线程数等等）借助telegraf定时拉取，然后写入InfluxDB; 业务指标，使用gRPC实现了主动上报系统，以确保实时性。同时我们还实现了报警策略分析，报警通道等服务。

1. 数据上报模块

- 数据上报服务基于gRPC和Spring Boot开发，对外提供gRPC和http两种协议接口。同时还封装了Java SDK和Spring Starter，方便业务开发者使用。另外服务端还实现了鉴权和流量控制功能。数据上报服务和SDK端保持定时的心跳。一方面可以用于检测应用的健康状况，另一方面还可以和服务端同步配置信息。实现动态的调整客户端指标聚合的时间维度。
- 服务端拿到数据后，push数据到Kafka，实现不同的Consumer来负责数据清洗，元数据存储等工作。第一个版本的报警元数据存储Consumer是这样实现的：判断元数据存在依赖于数据库，然后使用多线程实提升处理速度。但由于数据量太大这个实现导致了队列积压，引发了OOM，后来引入布隆过滤器对这块的逻辑进行了优化。

2. 策略模块使用Golang开发，借鉴了小米Falcon的思路。并做了以下优化：实现了同比功能；优化内存的使用；添加新策略可立即生效。
3. 基于Istio和Kubernetes的service mesh改造。

后端服务拆分

职责: 负责人

技术选型: Java8, Vert.x, JOOQ, MySQL, Redis

项目背景

北京第三石是成立了三年多的创业公司, 由于缺乏对后端服务重构和优化, 所有的业务都耦合在一个系统里, 导致系统臃肿复杂. 开发维护成本越来越高. 而且Python + Django也导致了严重的性能问题. 为了解决这些问题, 开始推进后端服务的拆分, 以及技术栈转型.

项目经历

- 确定服务拆分计划. 线上服务作拆首先要做到对客户端不感知, 而且还要确保线上的服务不受影响. 因此首要拆分的是用户和商品服务, 这两个服务对业务的依赖少. 与老系统整合时只需修改数据访问层, 相对简单, 影响较小. 然后再对客户端业务和管理后台业务逻辑进行切分.

- 技术调研和技术选型. 服务拆分之后, 原来对用户和商品的数据库调用会变成Http调用, 而且用户和商品服务会被大量的外围服务依赖, 因此这两个系统必须要具备低延迟, 高可用. 在技术选型的时候, 通过对比Spring Boot和Vert.x, 最终选择异步的, 基于事件的Vert.x作为基础框架. Load Balancer使用AWS的ELB. 缓存使用Redis.

- Redis缓存层基于消息实现. 应用程序与缓存管理程序之间通过EventBus进行通信, 所有缓存的操作都抽象为消息. 将缓存操作从业务逻辑中解耦.

- 完成用户服务拆分, 对外提供Rest API, 现在已在测试阶段.

订单系统

职责: 负责人

技术选型: Java 8, Spring Boot, Spring Statemachine, JOOQ, Akka, Flyway, RabbitMQ, Redis, MySQL

项目背景

北京第三石在美国市场做C2C业务, 主要是线下二手物品交易. 为了让用户交易更安全方便, 公司决定要做担保交易和物流.

项目经历:

- 引入Akka实现异步任务处理, 缩短API响应时间. 在一些业务场景中, 很多操作都可以做成异步. 比如, 向用户发送push, 邮件, 向消息队列发送消息等等. 第一版本的实现是基于Java的线程池, 但是这个方案可能在重启服务时造成任务丢失. 后改进为使用Akka Persistent Actor实现.

- 基于Redlock算法实现Redis分布式锁.

- 引入Spring Statemachine控制订单状态流转. 并且基于Spring Statemachine的回调功能实现订单操作历史的管理.

- Restful风格API的设计和实现.

名称：业务系统

职责：全栈开发

技术选型：.Net Web API, ReactJS, SQL Server

项目背景

该项目是ThoughtWorks为某美国公司开发的业务处理系统. 该公司主要负责为第三方公司提供财务审计, 国际报税等业务. 每项业务都由单独的系统处理. 为了方便第三方客户查看及使用, 需要作一个聚合的网站, 把现有的业务聚合到一个网站内操作.

项目经历:

- 用Pact确保系统间的集成. 我们构建的每一个系统都依赖于核心系统的API. 由于核心系统也在不断的开发新功能, 我们正在使用的API很有可能被别人修改, 导致系统挂掉. 为了解决这个问题, 我们引入了PactNet. PactNet是一个契约测试库, 我们作为消费端会递交给核心系统一份JSON格式的接口契约文件, 用以标记我们需要哪些API, API的参数, 以及响应结果. 这样核心系统开发人员在做API修改时, 会检查被修改过的API是否还满足我们的要求, 确保API的兼容性.

- 用ReactJS解决前端控件复用的问题. 该项目由多个子站点构成, 而且每个子站点的前端展示和交互是统一的. 如果每个站点都各自实现类似table, input, select等这些组件的话, 就会导致大量的重复代码. 并且目前有多个团队在并行开发, 每个团队的实现方式也不尽相同. 这是极不利于后期维护和升级的. 因此我们用ReactJS封装了一些通用的前端控件, 这样不仅解决了后期的维护问题,而且节省了很多开发工作.

- 使用redis做session管理. 我们的web站点使用WIF作登录认证, 每次用户登录时都会向WIF Server申请登录令牌(token). 这个token有一个存活时间, 在这段时间内token总是有效的, 只有当token过期时WIF Server才会重新生成新的token. 即使logout, 该token仍然可以使用. 由于token以cookie的形式存储在浏览器, 所以token存在被盗的可能, 一旦拿到该token我们所有依赖WIFServer认证的站点都受到威胁. 而且每个站点还提供了token续期的功能, 如果某个用户拿到了这个token, 并且知道了token续期的API, 那么这个用户就可以无限期的使用系统了.出于安全的考虑, 我们引入了redis来管理token. 当WIF生成一个新的token时都会把这个token存储到redis中, 用户访问网站时每次都会检查redis.

工作期望

Java/Scala工程师; 后端研发工程师;