目录

[1 现有架构模式. 3](#_Toc17765)

[1.1 系统架构图... 客户端》nginx 》 3](#_Toc30190)

[1.2 ## 提供给 客户端的api接口 4](#_Toc29880)

[1.3 Cache 缓存 5](#_Toc18972)

[1.4 ## database 数据库 与访问接口 5](#_Toc3028)

[1.5 ## msa微服务与负载均衡 6](#_Toc7025)

[1.6 ## timer定时调度 6](#_Toc24316)

[1.7 ## 分库分表 7](#_Toc11044)

[2 架构改造总目标 8](#_Toc30852)

[3 功能系列增加 8](#_Toc22601)

[3.1 分布式的功能增加， 8](#_Toc2297)

[3.2 分布式文件存储 8](#_Toc20449)

[3.3 分布式缓存 8](#_Toc21874)

[3.4 监控系统 总控 8](#_Toc18002)

[3.5 sso联合登陆 9](#_Toc32168)

[3.6 数据同步触发机制 9](#_Toc19895)

[3.7 分布式日志日志 springcloud+elk 9](#_Toc32642)

[3.8 ## timer定时调度 9](#_Toc14255)

[3.9 消息推送mq 10](#_Toc19502)

[3.10 全文检索（im中可用） 11](#_Toc9867)

[4 架构系列 简化与提升扩展性系列 11](#_Toc8979)

[4.1 扩展性三件套DSL 通用设计 数据扩展性 12](#_Toc20753)

[4.2 频繁变动的业务配置化 脚本化 配置业务脚本 12](#_Toc18700)

[4.3 实现接口的通用性 多功能 尽可能业务无关性 12](#_Toc21801)

[4.4 数据扩展 schema free , schema less模式 半结构化数据表示 13](#_Toc9941)

[4.5 数据扩展性 数据库表结构变更同步到api接口 13](#_Toc6453)

[4.6 数据扩展性 数据库适当Json字段提升扩展性 13](#_Toc28566)

[4.7 同类目标使用简单技术实现 13](#_Toc30947)

[4.8 标准的协议，谨慎私有协议 13](#_Toc14559)

[4.9 使用现有的通用组件组合。。 14](#_Toc8623)

[4.10 简化设计 同步 vs 异步模式 14](#_Toc30630)

[4.11 分段隔离分别提升 14](#_Toc25723)

[4.12 All in one模式 容器等 14](#_Toc12796)

[4.13 工作流BPM与规则引擎 14](#_Toc11495)

[5 架构之 可读性提升 16](#_Toc29585)

[5.1 分类汇总，命名空间模式 16](#_Toc5188)

[5.2 链路层次简化，避免过多 17](#_Toc17222)

[5.3 附加本地语言提升可读性 17](#_Toc30049)

[5.4 同一业务代码尽可能集中在一处修改，避免跨越多个文件修改跳跃 17](#_Toc2650)

[6 架构改造 提升稳定性 17](#_Toc20355)

[6.1 稳定性top5 17](#_Toc8695)

[6.2 优选简单技术 使用更加简单稳定 17](#_Toc31297)

[6.3 看门狗模式哨兵模式 18](#_Toc8742)

[6.4 Try cartch指令冗余 18](#_Toc31003)

[6.5 隔离 分段隔离 18](#_Toc9595)

[6.6 定时gc防止资源泄漏（内存 连接 其他对象 线程等） 18](#_Toc29966)

[6.7 熔断器拦截器 19](#_Toc3824)

[6.8 减少单点 载均衡配置前移， 19](#_Toc21354)

[6.9 缩减调用链层次等，提升稳定性 19](#_Toc3018)

[6.10 多路冗余微服务 19](#_Toc23734)

[7 架构改造 提升性能系列 19](#_Toc6526)

[7.1 比较重要的性能提升三件套 简化（层次架构避免过多响应时间），多路分散，cache 19](#_Toc15214)

[7.2 层次简化 提升响应速度性能 19](#_Toc22516)

[7.3 ## msa微服务与负载均衡 20](#_Toc6610)

[7.4 Cache 缓存 21](#_Toc29332)

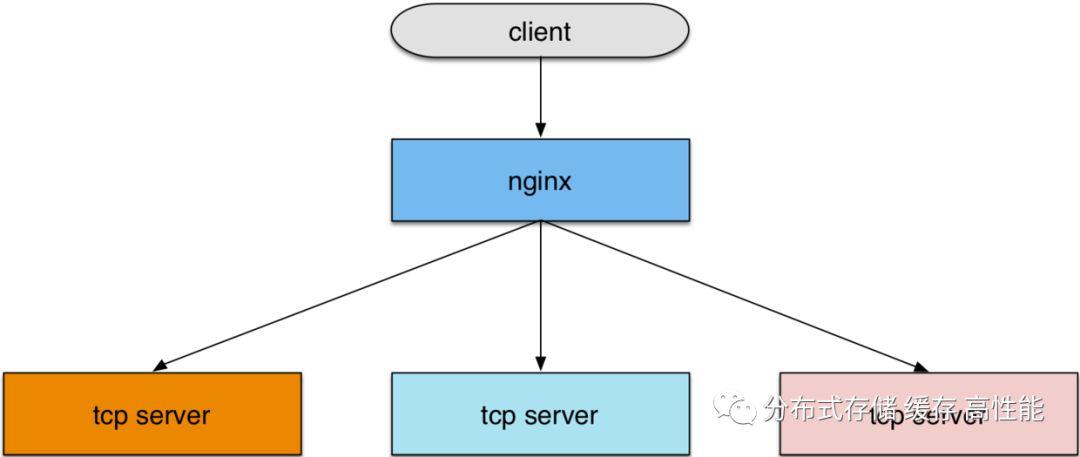
[7.5 ## database 数据库 +nosql 22](#_Toc7980)

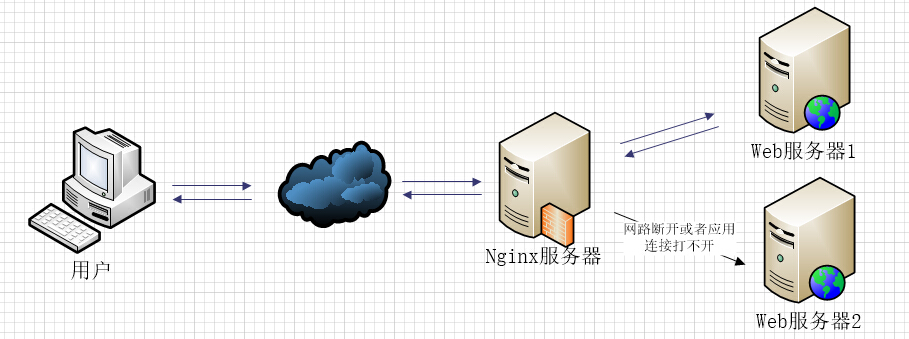
[7.6 ## 数据库负载均衡 23](#_Toc32509)

[8 架构综合 25](#_Toc20524)

# 现有架构模式.

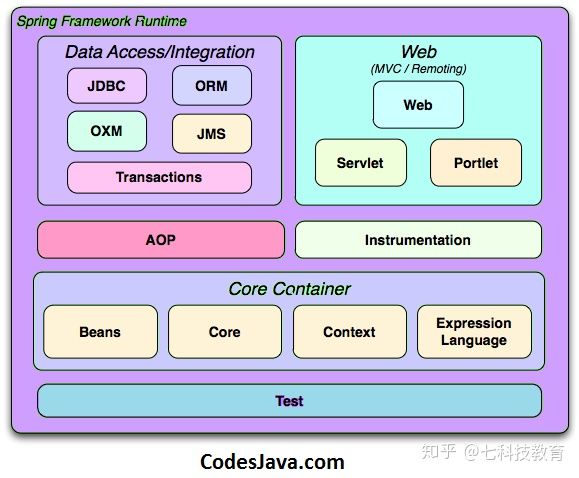
## 系统架构图... 客户端》nginx 》





## ## 提供给 客户端的api接口

主要是 rest 接口springboot



## Cache 缓存

Spring cache和redis

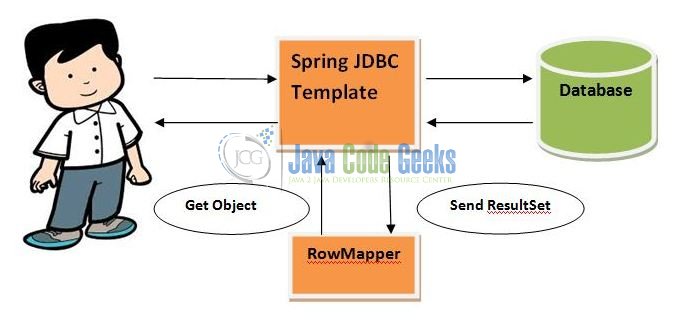
## ## database 数据库 与访问接口

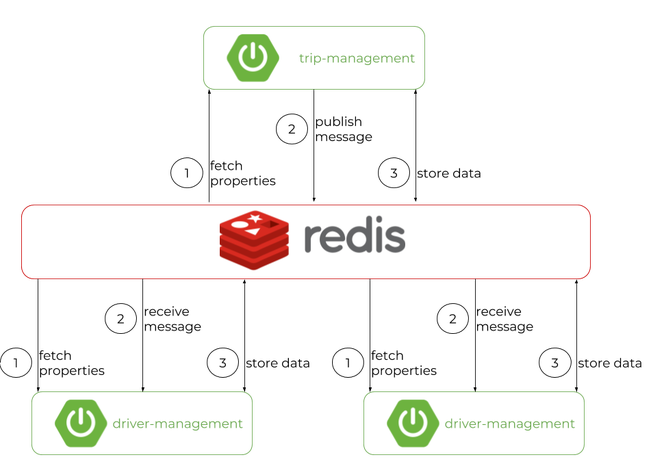
mybatis 5.7 以及以上

redis （nosql 数据库）

spirng jdbctemplete

mybatis ， redis驱动lib（socket接口）



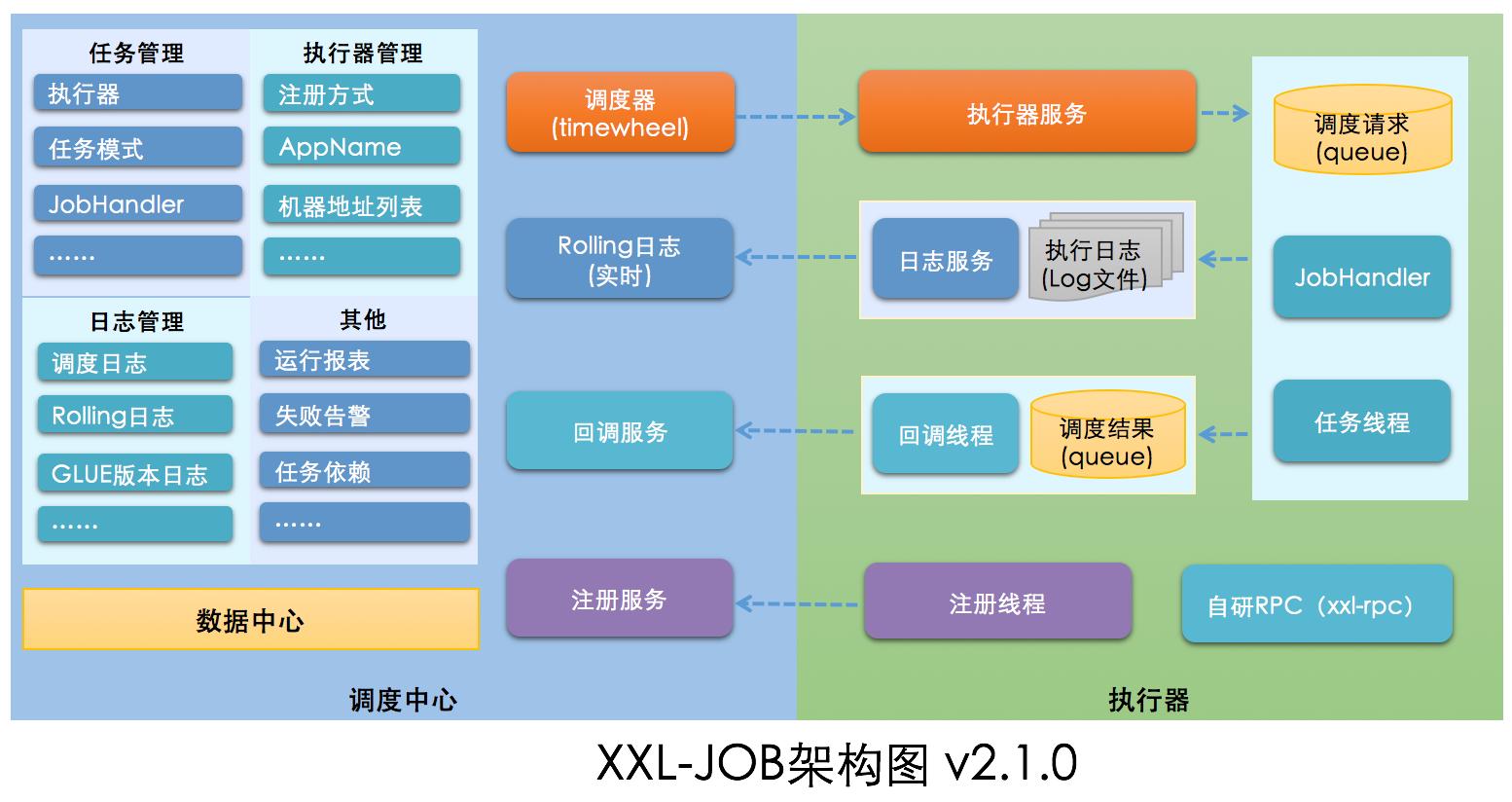


## ## msa微服务与负载均衡

nginx （ 目前）

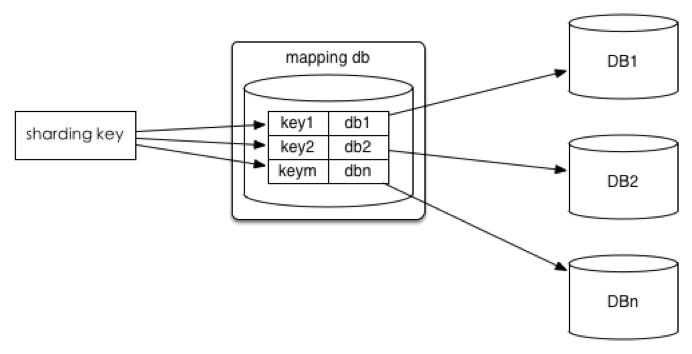
## ## timer定时调度

Xxljob



## ## 分库分表

自实现



# 架构改造总目标

调用总架构图



服务消费者

Web应用

服务提供者

服务提供者

Web应用



扩展性提升与简化

架构简化简单容易运维测试

简化组件数量方便一次启动

减少不必要的层次，一方面也提升了性能，减少了周转层次，减少代码跳转过多，以及一处修改跨越多个文件的问题

即时修改生效 热部署随时修改配置和代码即时生效。。

尽可能减少编译打包，配置化 脚本化

提升启动速度 尽可能控制在几秒内启动，方便修改测试

频繁变动业务配置化 方便修改 提升扩展性

项目启动解耦合，可以方便不同顺序方便启动不会互相影响，解除启动顺序依赖

类库版本强依赖解除 通过隔离不同版本实现，或换用其他类似非冲突组件实现

数据结构解耦扩展性提升，数据库表结构可容易增加修改而尽可能不改动程序代码，降低耦合性

某些流程繁琐可以简化

稳定性与性能提升

# 功能系列增加

## 分布式的功能增加，

## 分布式文件存储

## 分布式缓存

## 监控系统 总控

日志类监控

各类程序运行状态的监控，总控平台，ui集成

## sso联合登陆

## 数据同步触发机制

**不管什么数据库，实时数据同步工具，都是把自己模拟成一个从库，进行数据拉取和解析。** 具体来说，mysql是通过binlog进行同步；postgresql使用wal日志进行同步。

## 分布式日志日志 springcloud+elk

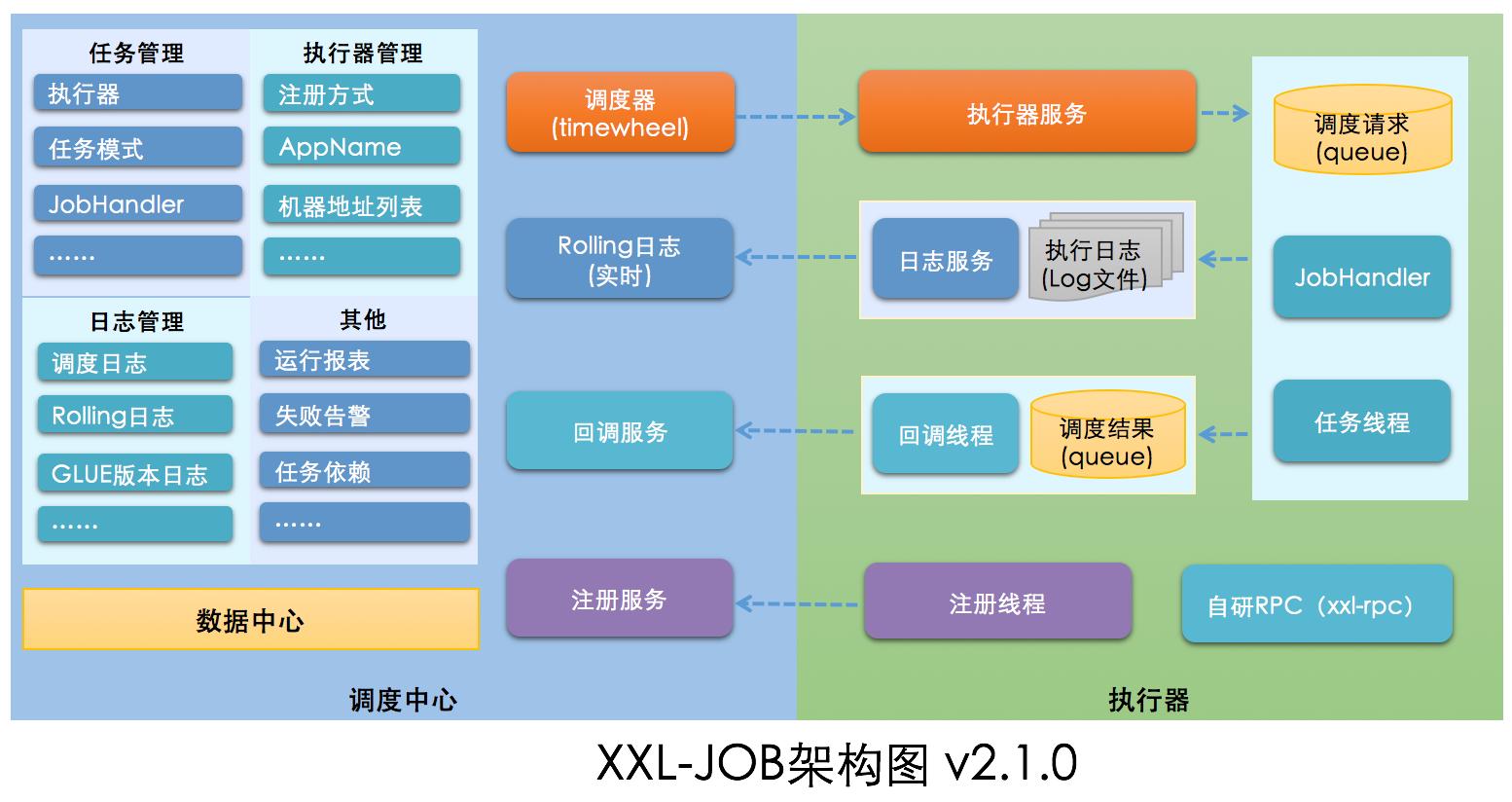
 由于微服务架构中每个服务可能分散在不同的服务器上，因此需要一套分布式日志的解决方案。spring-cloud提供了一个

用来trace服务的组件sleuth。它可以通过日志获得服务的依赖关系。基于sleuth，可以通过现有的日志工具实现分布式日志的采集。

这里使用的是ELK，也就是elasticsearch、logstash、kibana。

## ## timer定时调度

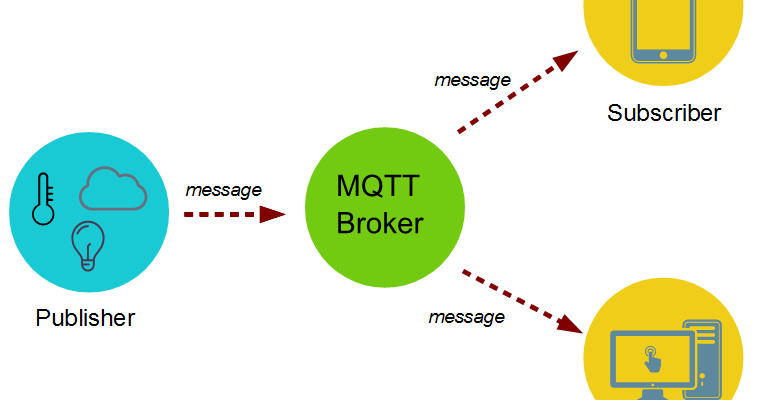
Xxljob。。。 后期根据需要可以附加linux crontab定时任务 +数据库定时任务

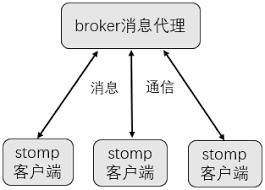


## 消息推送mq

支持Mqtt协议（常用）） stomp协议（常用）） xmpp协议（不常用）

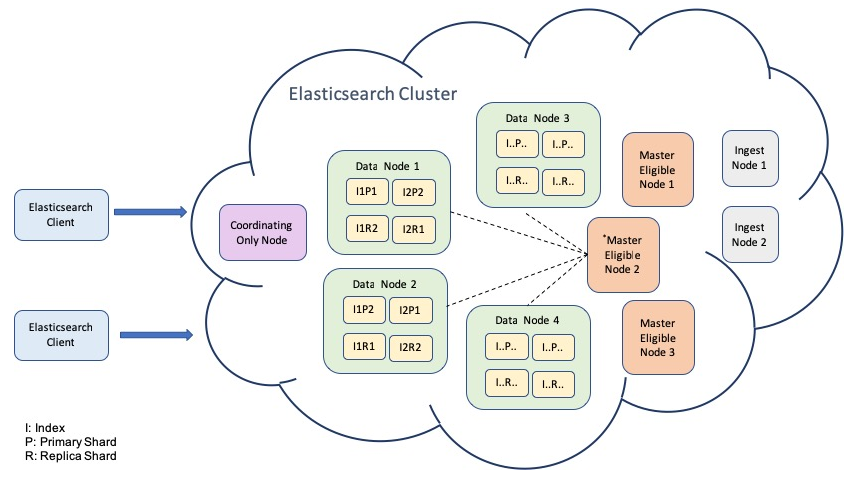
其他消息推送私有协议建议不使用，开放协议优先比较好





## 全文检索（im中可用）

可用es 或数据库自带的全文索引实现



# 架构系列 简化与提升扩展性系列

## 扩展性三件套DSL 通用设计 数据扩展性

## 频繁变动的业务配置化 脚本化 配置业务脚本



## 实现接口的通用性 多功能 尽可能业务无关性

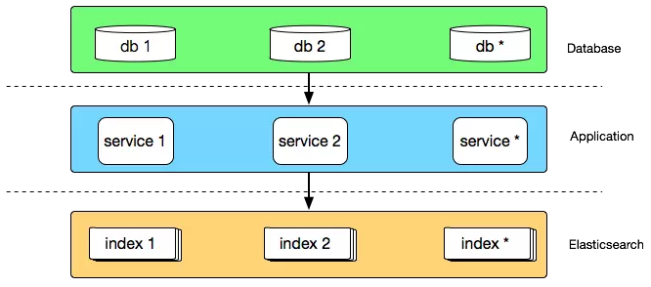
## 通用数据 访问接口与数据访问接口

尽可能参数化 提升通用性 +dsl模式实现通用接口

Sql接口

可以给nosql数据库（redis，es，mongodb）增加些简单sql方面api，更加简单的查询语句

以及用来简化查询较复杂的内存数据集合查询（list数组等）



程序api

可以增加些流行的mybatis和jpa等数据库api接口替换以前的 springtemplete数据库 api接口

Rest接口

增加通用的rest接口，特别是mysql，可将其默认的socket接口转换为rest，达到更加易用的目的。。es已经有默认的rest接口，

Redis可能也需要增加rest接口。。

## 数据扩展 schema free , schema less模式 半结构化数据表示

Redis mongodb这些nosql数据库可以适当使用，以及使用**RDBMS** 的json数据类型

## 数据扩展性 数据库表结构变更同步到api接口

带来的代码变更，做数据表字段自动化变更同步到api接口。

新增库表也可以可以通用接口查询，无需单独实现额外接口

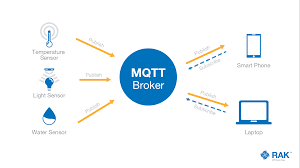
## 数据扩展性 数据库适当Json字段提升扩展性

数据结构可以随意扩展

## 同类目标使用简单技术实现

## 标准的协议，谨慎私有协议

标准协议往往文档众多，有通用组件



## 使用现有的通用组件组合。。

## 简化设计 同步 vs 异步模式

## 分段隔离分别提升

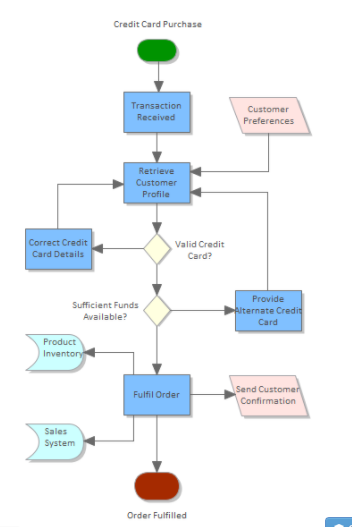
用于防止类库冲突等，，复杂业务 msa和mq可以提升扩展性，分段扩展，减少相互影响

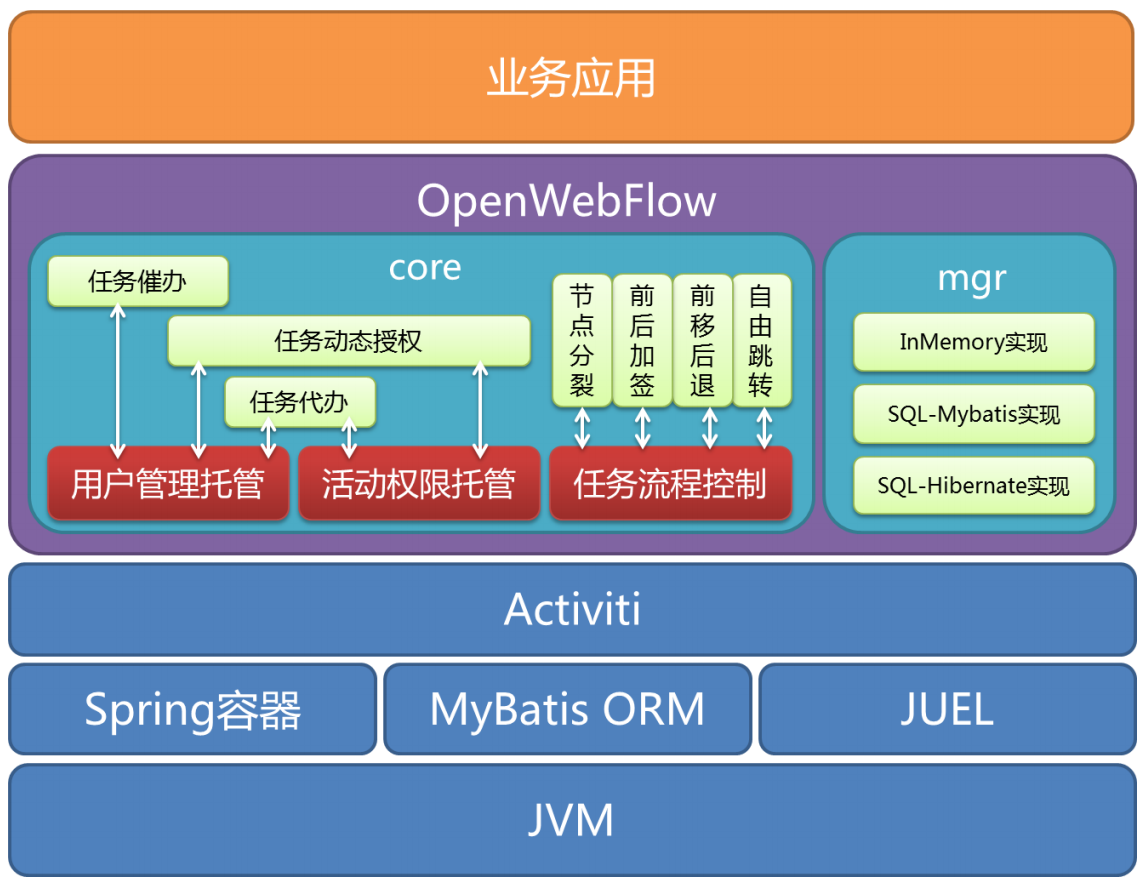
调用链太复杂的情况下，使用mq可以简化开发，分段处理

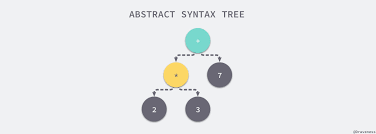
## All in one模式 容器等

## 工作流BPM与规则引擎

业务规则脚本，因为需求可能随着时间经常更改。开发团队的解决方案是广泛使用规则，以避免频繁的代码部署。







# 架构之 可读性提升

## 分类汇总，命名空间模式

Java类语言有成熟的命名空间机制

脚本类可以使用文件夹作为命名空间

数据库也可以适当分库，避免太多表积攒在同一库里面影响可读性，以及表名前缀模式来实现命名空间，单机可以按照实例》database库》table表 三层模式。。

Sql脚本的命名空间可以使用前缀模式实现或分库模式聚合

## 链路层次简化，避免过多

## 附加本地语言提升可读性

见名知意。。减少了翻译式注释

## 同一业务代码尽可能集中在一处修改，避免跨越多个文件修改跳跃

特别是脚本类业务代码，ide可能不好跳转。。编译类语言也可以减少跳转

# 架构改造 提升稳定性

## 稳定性top5

## 优选简单技术 使用更加简单稳定

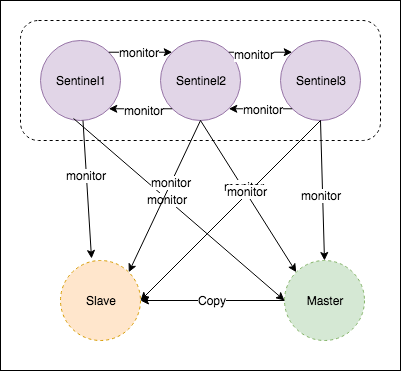
具体有 脚本化 配置化

越简单技术一般越稳定 使用实现功能与性能等

## 看门狗模式哨兵模式

Timer gc模式

T0,t1,t2三冗余环形看门狗模式

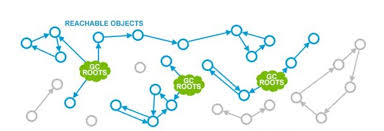


## Try cartch指令冗余

## 隔离 分段隔离

## 定时gc防止资源泄漏（内存 连接 其他对象 线程等）

资源包括cpu占用，内存，连接，磁盘空间等影响持续运行的东西



## 熔断器拦截器

拦截请求检查当前节点资源状况（cpu，内存，磁盘，连接等），如果超出负荷，可以拒绝防止崩溃

## 减少单点 载均衡配置前移，

## 缩减调用链层次等，提升稳定性

业务调整免编译，防止多点问题

## 多路冗余微服务

串行vs并行

# 架构改造 提升性能系列

## 比较重要的性能提升三件套 简化（层次架构避免过多响应时间），多路分散，cache

多路分散（分布式，**微服务msa微服务** 、分区，分库），

cache （mq cache等）协调高性能组件和低性能组件的不匹配。

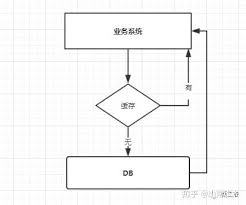
冷热数据分离 把热数据放入cache更快读写。。

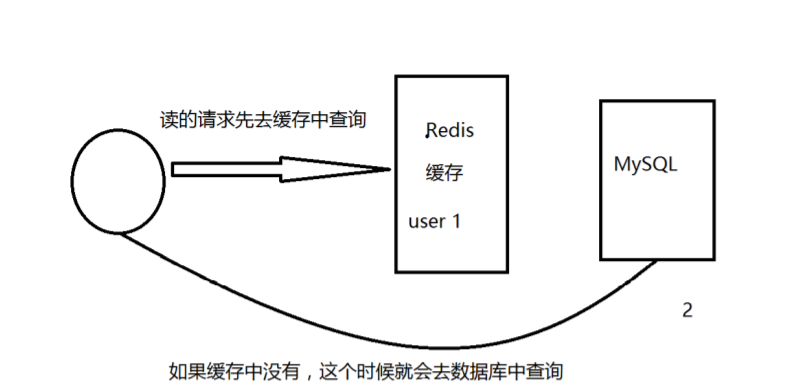
匹配效率，可以把低性能组件数据cache到一起，高性能组件一次处理。

时间分散削峰填谷，可以吧高性能组件产生数据等待低性能组件消费完毕，削峰填谷。。

## 层次简化 提升响应速度性能

如果架构本身层次过多，可以在某些模块功能直连模式，提升响应速度。。同样资源情况下，提升响应速度可能会影响吞吐量





同时多路直连读取数据会加快响应速度，先到先用，后到的重复数据丢弃

## ## msa微服务与负载均衡

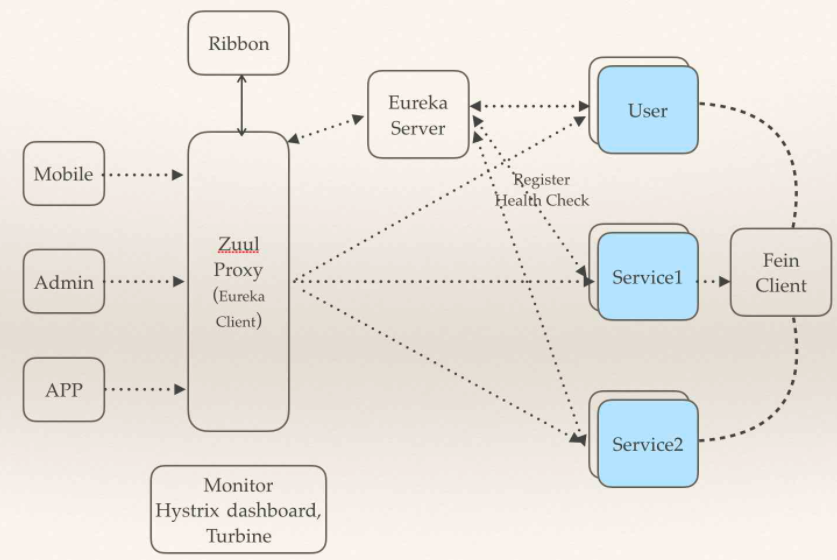
增加增加 springcloud来做微服务 经过讨论暂时不加dubbo这一类msa微服务框架

现有的nginx单点或也可以二次做负载均衡，进一步提升性能架构

##提供给 客户端的api接口改造

在nginx 和 web服务器之间增加msa微服务框架

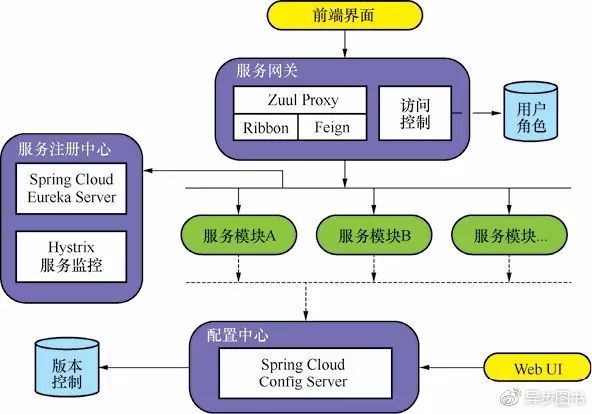
暂时不变，后期可以根据需要增加一些rpc方面的接口，不只是rest

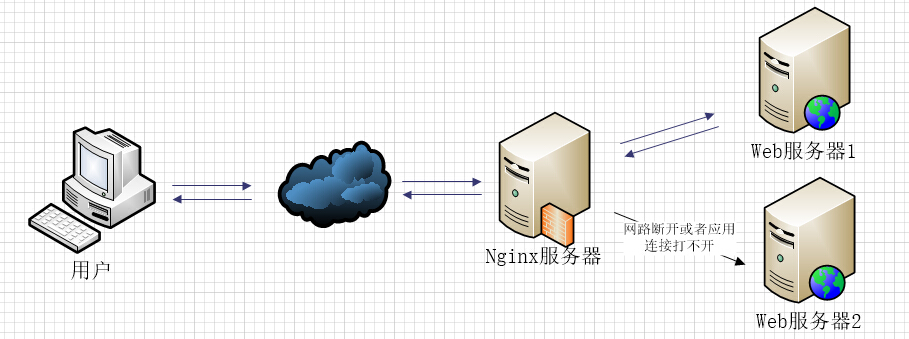


## Cache 缓存

目前是 Spring cache和redis

可以适当增加些mybatis缓存。Mq消息中间件。 三重缓存增加更好的性能，以及稳定性。。某个缓存失效还有下一层兜底





## ## database 数据库 +nosql

目前的mysql + nosqldb(redis)

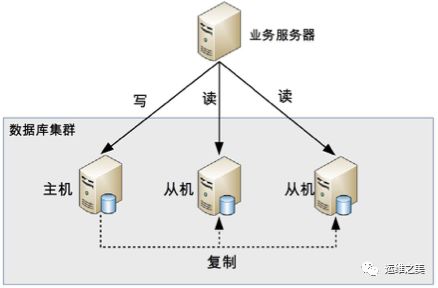
后期可能会加些全文检索 es 和json文档数据库 mongodb 的后备计划

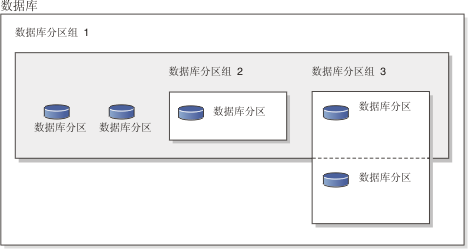
以后也可做数据库集群（读写分离 ，分区等，或者更大型数据库mssql oracle等备用选项 ）

## ## 数据库负载均衡

分库分表 自实现

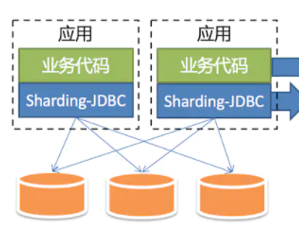
数据库集群读写分离 以及 分区（数据库自带功能）





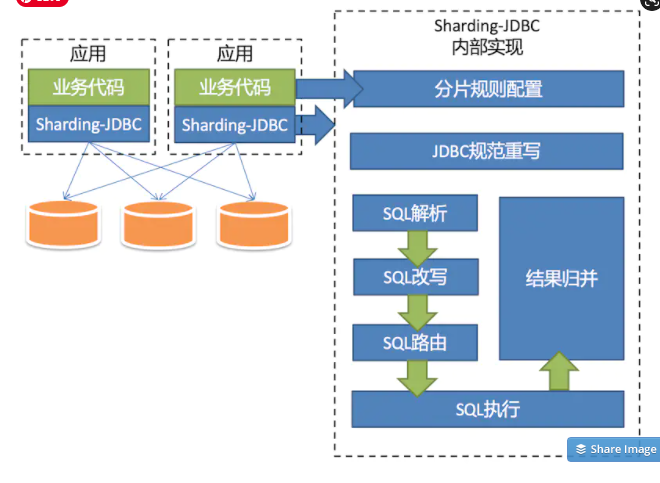
## 分库分表

自实现 动态数据源 （类似sharejdbc路由但不进行sql改写）



Mycat sharejdbc Sharding-Proxy可以备选

Sharding-JDBC系统架构图:



# 架构综合

各种模式都有所长，综合起来取长补短，逐步提升。。