mycat+dubbo分析

chenke@dumpcache.com

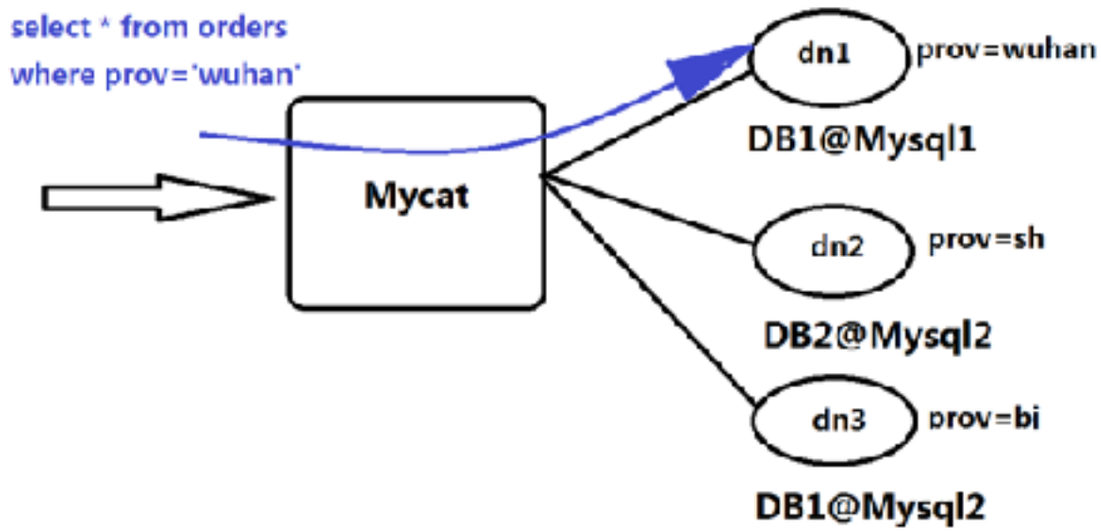
menu

- mycat相关
- dubbo相关
- 建议

mycat相关

- 原理介绍
- 和tddl的区别
- 部署架构
- 核心概念
- 分表分库相关策略

mycat原理介绍



特点：

1.支持HA.

2.高性能，非阻塞

3.支持读写分离

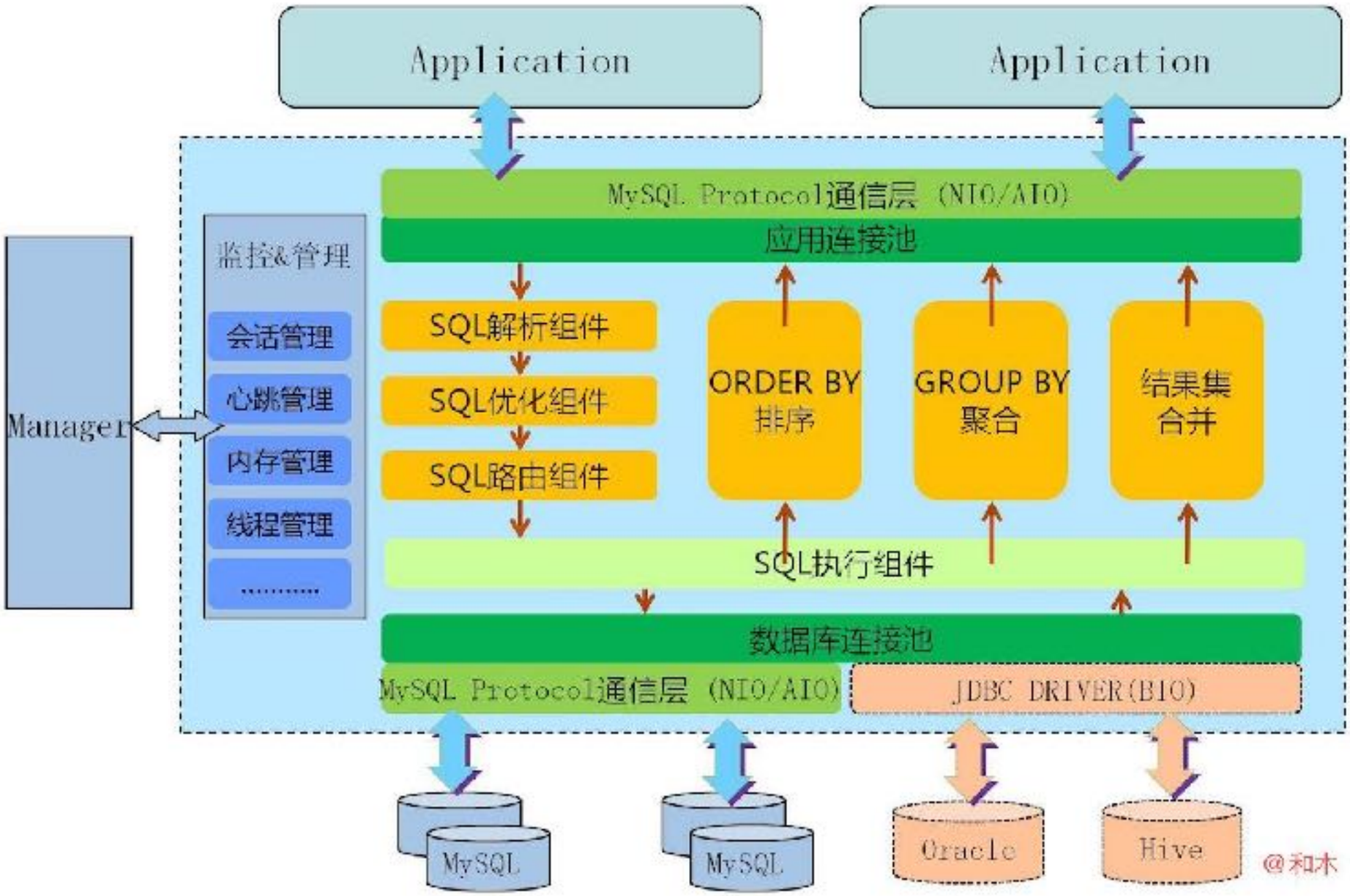
4.支持多种分库策略

5.通过mysql协议暴露，对应用透明

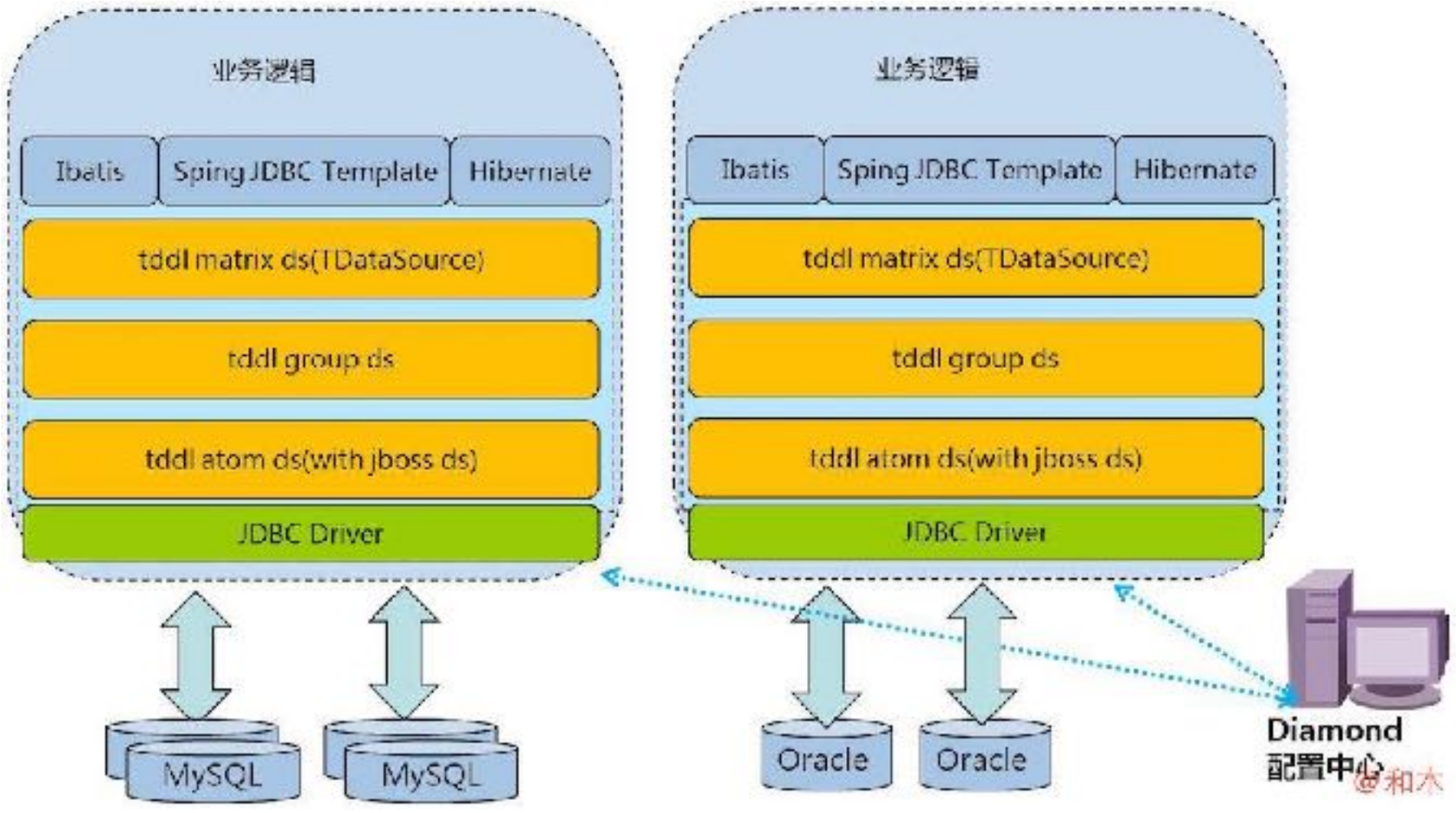
和tddl的区别

- 都是java编写，陈思儒的amoeba是用c写的。
- 前身为cobar，但最新版本已经完全重写。
- mycat与amoeba和cobar一样，都是服务端的mysql代理中间件，跟应用透明。
- tddl是一个客户端的解决方案，更像是一个framework

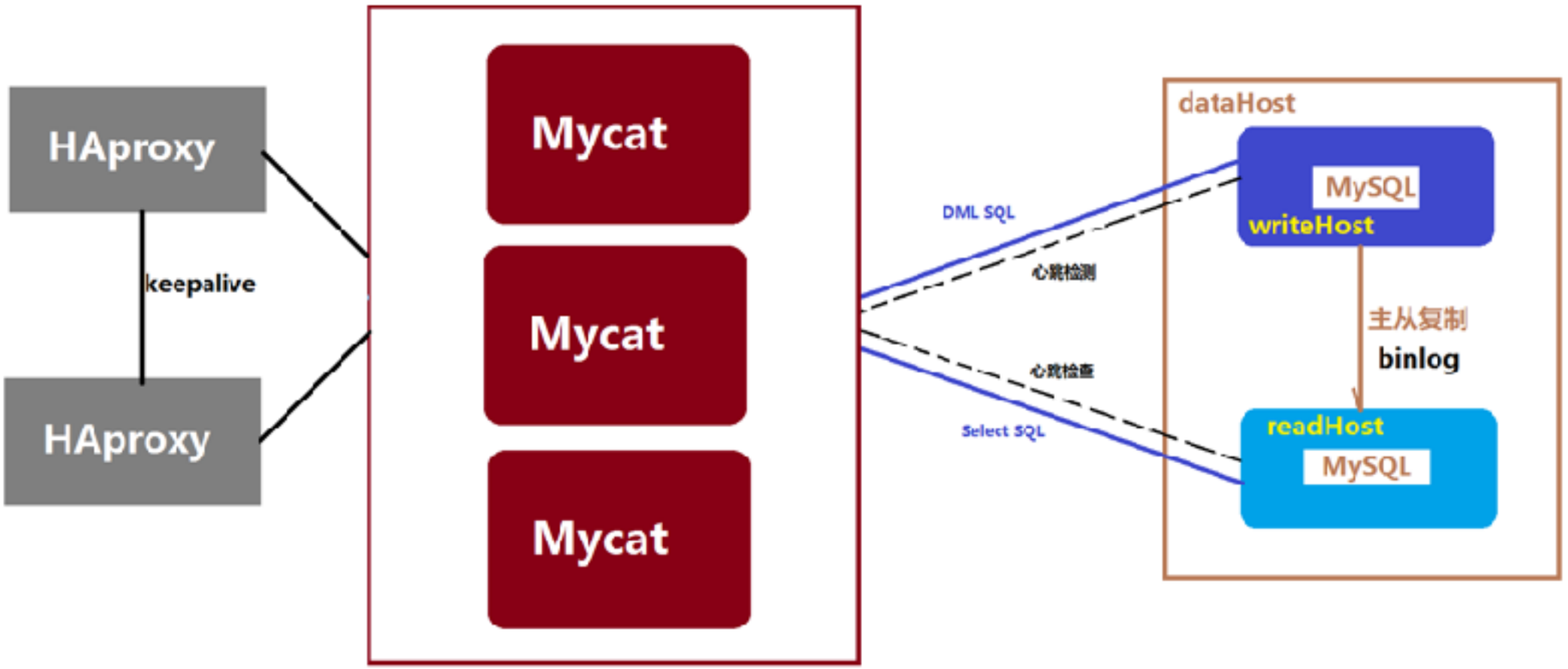
mycat的架构


Application
Application
Manager
监控&管理
会话管理
心跳管理
内存管理
线程管理
.....
MySQL Protocol通信层 (NIO/AIO)
应用连接池
SQL解析组件
SQL优化组件
SQL路由组件
ORDER BY
排序
GROUP BY
聚合
结果集
合并
SQL执行组件
数据库连接池
MySQL Protocol通信层 (NIO/AIO)
JDBC DRIVER(BIO)
MySQL
MySQL
Oracle
Hive
@和木

tddl的架构


业务逻辑
Ibatis
Sping JDBC Template
Hibernate
tddl matrix ds(TDataSource)
tddl group ds
tddl atom ds(with jboss ds)
JDBC Driver
业务逻辑
Ibatis
Sping JDBC Template
Hibernate
tddl matrix ds(TDataSource)
tddl group ds
tddl atom ds(with jboss ds)
JDBC Driver
MySQL
MySQL
Oracle
Oracle
Diamond
配置中心

mycat部署架构


HAproxy
keepalive
HAproxy
Mycat
Mycat
Mycat
DML SQL
心跳检测
心跳检查
Select SQL
dataHost
MySQL
writeHost
主从复制
binlog
readHost
MySQL

mycat核心概念

- dataHost，对应database
- dataNode,对应一个mysql实例
- schema，可与基于多租户进行逻辑隔离
- table:分片表，ER表，全局表
- sequence
- rule

mycat分表分库策略

- 读写分离
- 全局表
- ER Join
- Share join(分表后，可以关联多张表查询结果)
- 其他（Spark/Storm join等）

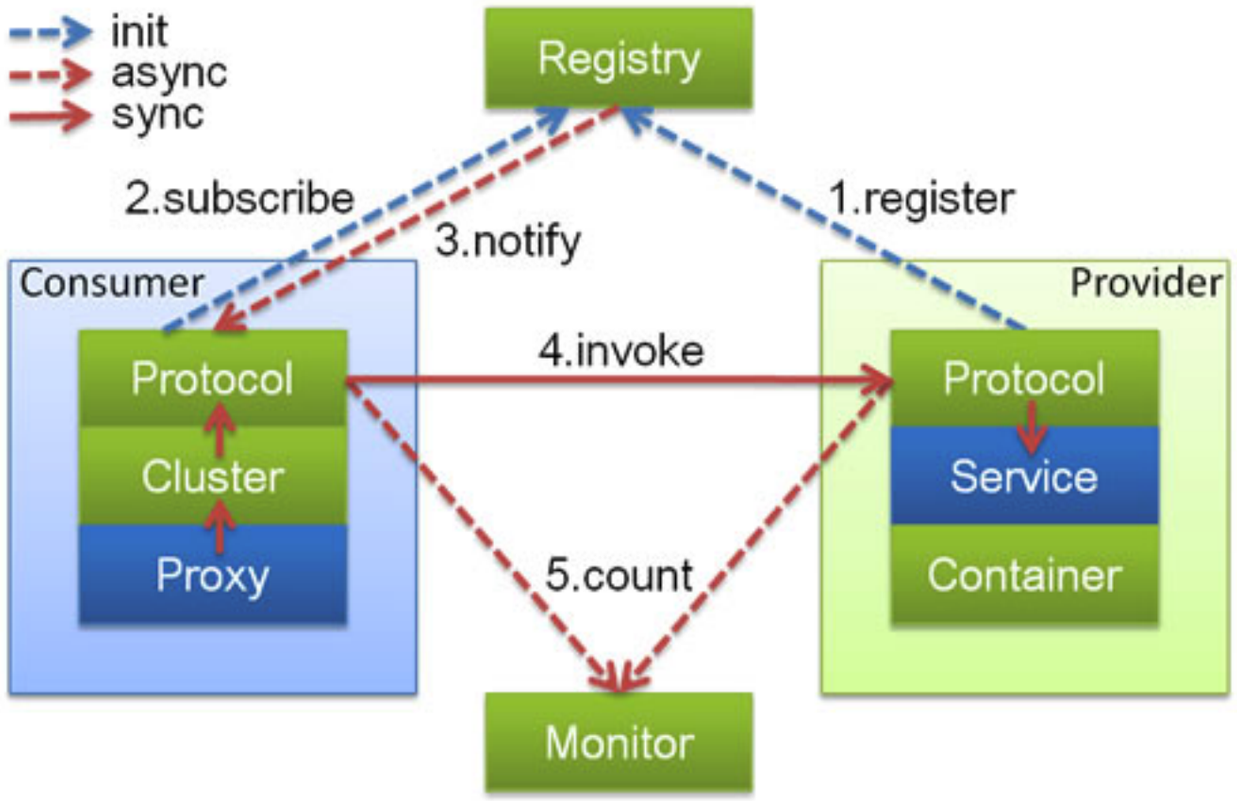
分片规则

- 9.5 Mycat常用的分片规则
  - 9.5.1 分片枚举
  - 9.5.2 固定分片hash算法
  - 9.5.3 范围约定
  - 9.5.4 取模
  - 9.5.5 按日期（天）分片
  - 9.5.6 取模范围约束
  - 9.5.7 截取数字做hash求模范围约束
  - 9.5.8 应用指定
  - 9.5.9 截取数字hash解析
  - 9.5.10 一致性hash
  - 9.5.11 按单月小时拆分
  - 9.5.12 范围求模分片
  - 9.5.13 日期范围hash分片
  - 9.5.14 冷热数据分片
  - 9.5.15 自然月分片

dubbo相关

- 架构&数据流程
- 协议扩展方法
- php相关

dubbo架构


init
async
sync
Registry
2.subscribe
3.notify
1.register
Consumer
Provider
4.invoke
Protocol
Cluster
Proxy
Protocol
Service
Container
5.count
Monitor

dubbo协议扩展方法

- public interface Protocol {
-
- 	/**
- 	 * 暴露远程服务：
- 	 * 1. 协议在接收请求时，应记录请求来源方地址信息：RpcContext.getContext().setRemoteAddress();
- 	 * 2. export()必须是幂等的，也就是暴露同一个URL的Invoker两次，和暴露一次没有区别。
- 	 * 3. export()传入的Invoker由框架实现并传入，协议不需要关心。
- 	 *
- 	 * @param <T> 服务的类型
- 	 * @param invoker 服务的执行体
- 	 * @return exporter 暴露服务的引用，用于取消暴露
- 	 * @throws RpcException 当暴露服务出错时抛出，比如端口已占用
- 	 */
- 	<T> Exporter<T> export(Invoker<T> invoker) throws RpcException;

dubbo目前支持的协议

dubbo://
rmi://
hessian://
http://
webservice://
thrift://
memcached://
redis://

php调用dubbo

- hessian&json等都有支持
- 核心问题是从zk中获取服务提供者进行负载均衡和高可用的实现

建议

- 架构设计的目的是为了更好的支撑未来业务发展
- 技术并不能解决所有的业务问题（任何技术都有副作用）
- 业务角度梳理清楚领域模型对未来相对重要
- 统一的规范比技术的细节相对更加重要
- 性能相差不大的情况下，实现速度优于选型花销

- **Q&A**