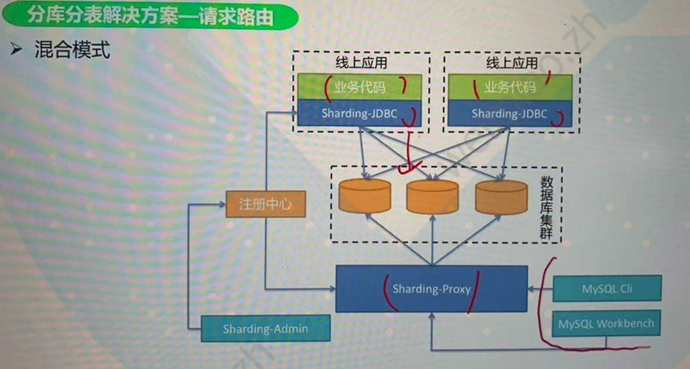
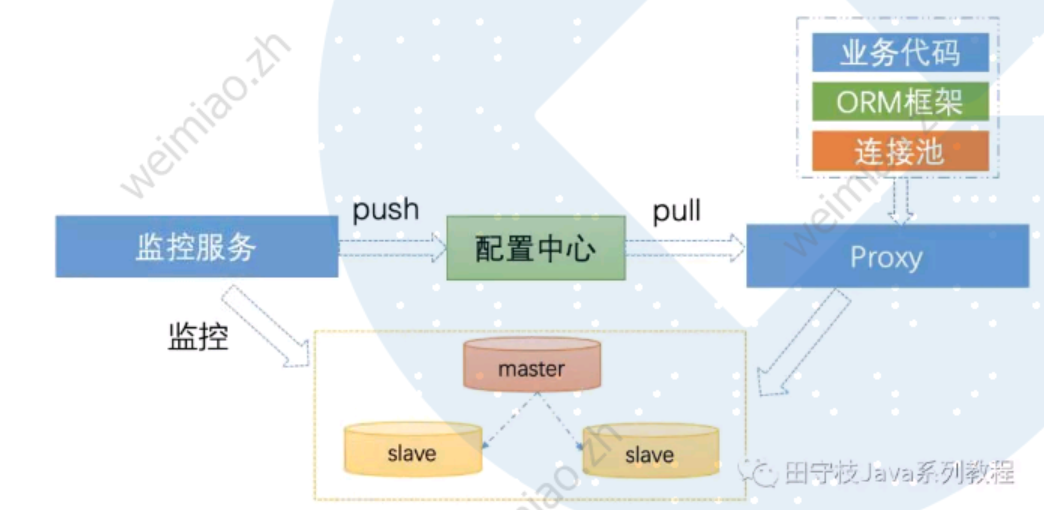
**理想开发方案：**

**=========================**





结合各自优缺点，SharingJDBC业务开发+ShardingProxy后台运维。

分库分表中间件实现中需要考虑的重点：

Etcd/appollo

Druid

1. SQL路由

解析，路由，改写，执行，结果集合并

分页，聚合，子查询，join

小表广播（canal）

2)二级索引

同步组件

3）分布式id生成器

在分库分表的情况下，数据库的自增主键已经无法使用。所以要使用一个分布式的id生成器。分布式事务id生成器要满足以下条件：唯一、趋势递增(减少落库时的索引开销)、高性能、高可用。

twitter的snowflake

12位日期+10位IP+6位序列ID+4位数据库扩展位

4) 分布式事务

XA，但是性能很差，对数据库的版本也有要求，例如必须使用mysql 5.7，官方还建议将事务隔离级别设置为串行化，这是无法容忍的。

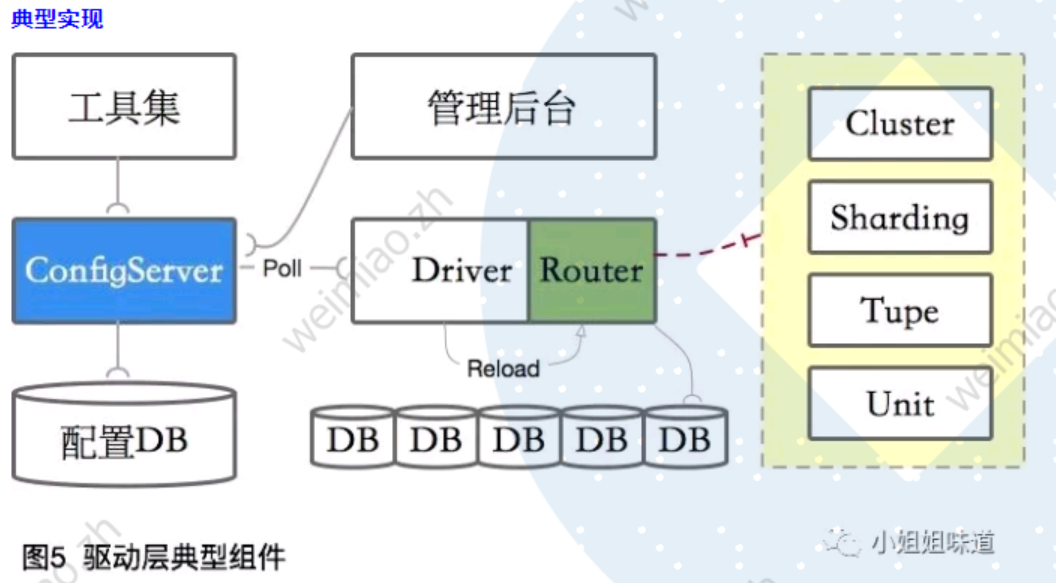
sharding-jdbc就使用了华为开源的一套微服务架构解决方案service comb中的saga组件，来实现分布式事务最终一致性

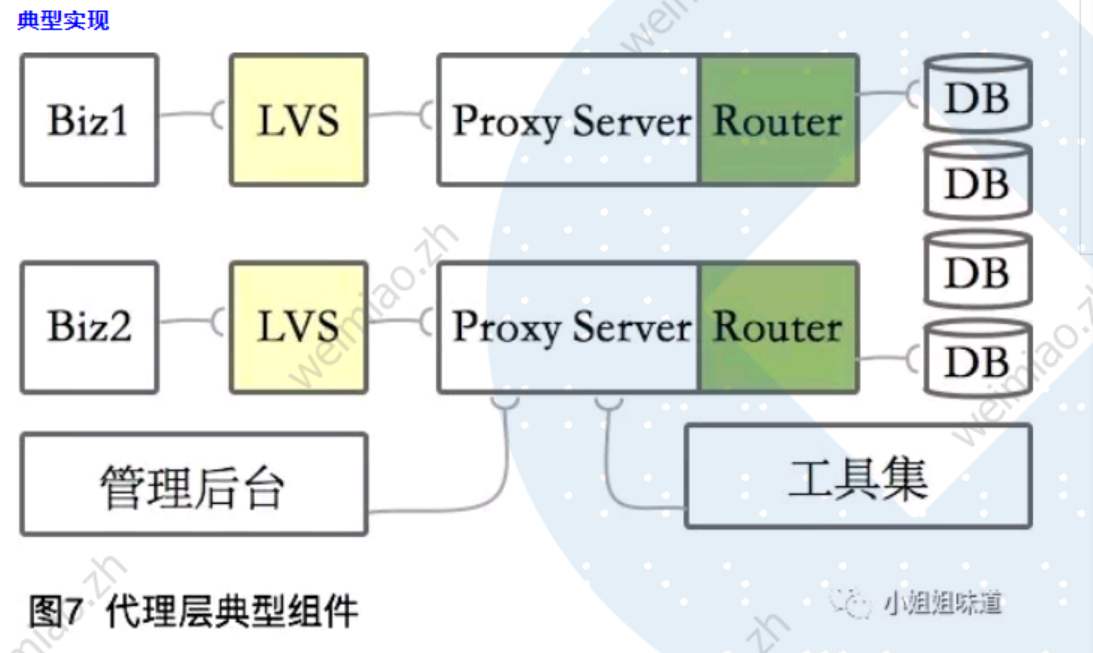
BASE柔性事务管理器（SEATA-AT配置）

Saga模式是SEATA提供的长事务解决方案，在Saga模式中，业务流程中每个参与者都提交本地事务，当出现某一个参与者失败则补偿前面已经成功的参与者，一阶段正向服务和二阶段补偿服务都由业务开发实现。AT：自动模式，通过我们记录运行sql的undolog，来完成事务失败时的自动重做。

**现有典型实现模式对比**

**=========================**



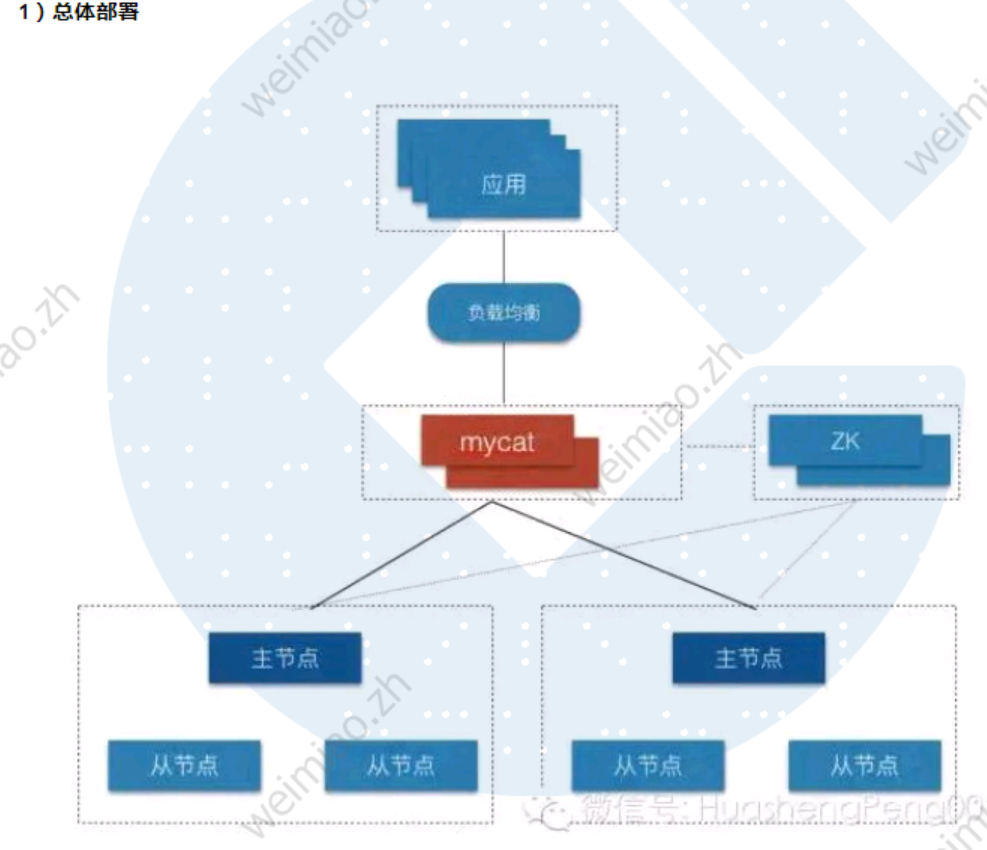


对比：

|  |  |  |
| --- | --- | --- |
| 对比项 | ShardingJDBC | ShardingJDBC/MYCAT |
| 数据库 | 任意 | MYSQL/前端MySQL |
| 异构语言 | Java | 任意 |
| 性能 | 损耗低 | 链路长，损耗高 |
| 去中心化 | 是 | 否 |
| 静态入口 | 无 | 有 |

**1)代理层MYCAT**

**========================**



Mycat2不像1.6有主从概念,也就说Mycat2之间的地位是平等的,Zookeeper的作用是作为配置中心,存储mycat的配置,以及提供元数据锁的功能

1.部署zk,并记录zk服务地址zk\_address

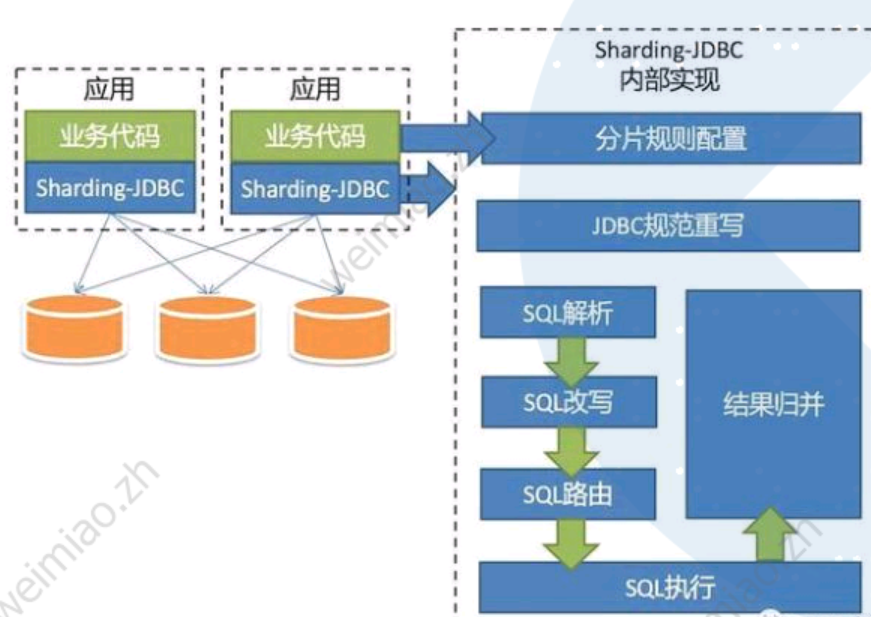
2.更改mycat配置



 Mycat里的数据库事务Mycat 目前没有出来跨分片的事务强一致性支持，目前单库内部可以保证事务的完整性，**如果跨库事务， 在执行的时候任何分片出错，可以保证所有分片回滚，但是一旦应用发起commit指令，无法保证所有分片都成功，考虑到某个分片挂的可能性不大所以称为弱xa。**

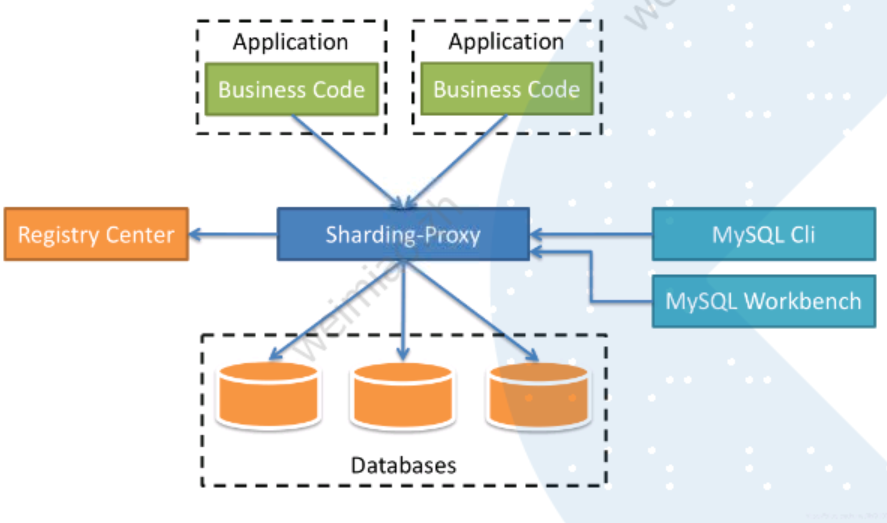
**2）驱动层ShardingJDBC**

=====================



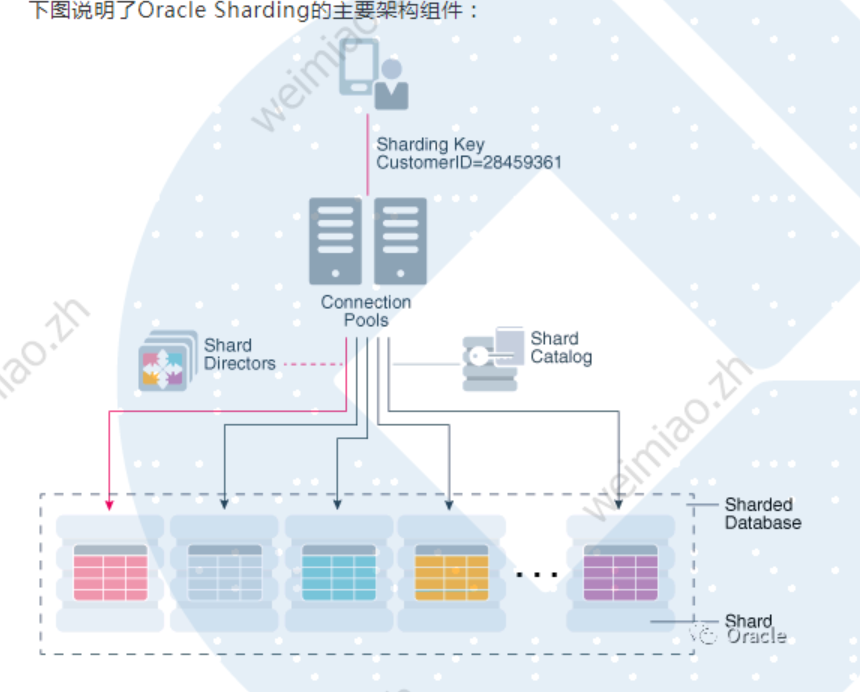
**SharingProxy**

**================**



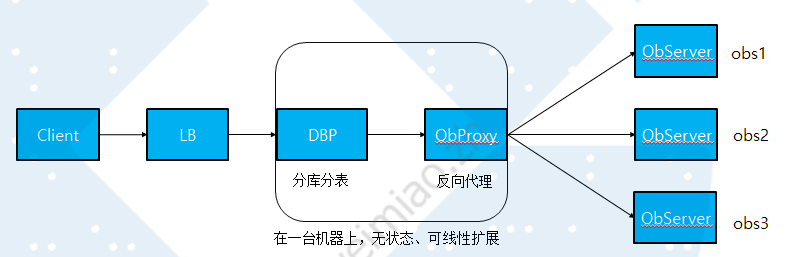
**4)Oracle Sharding**

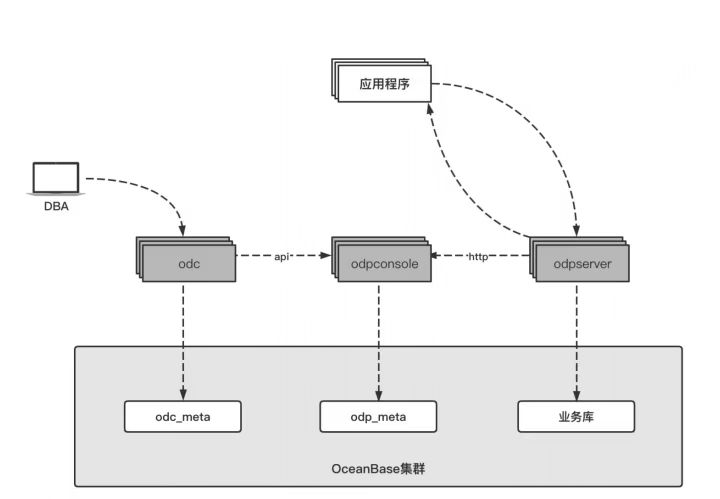
========================

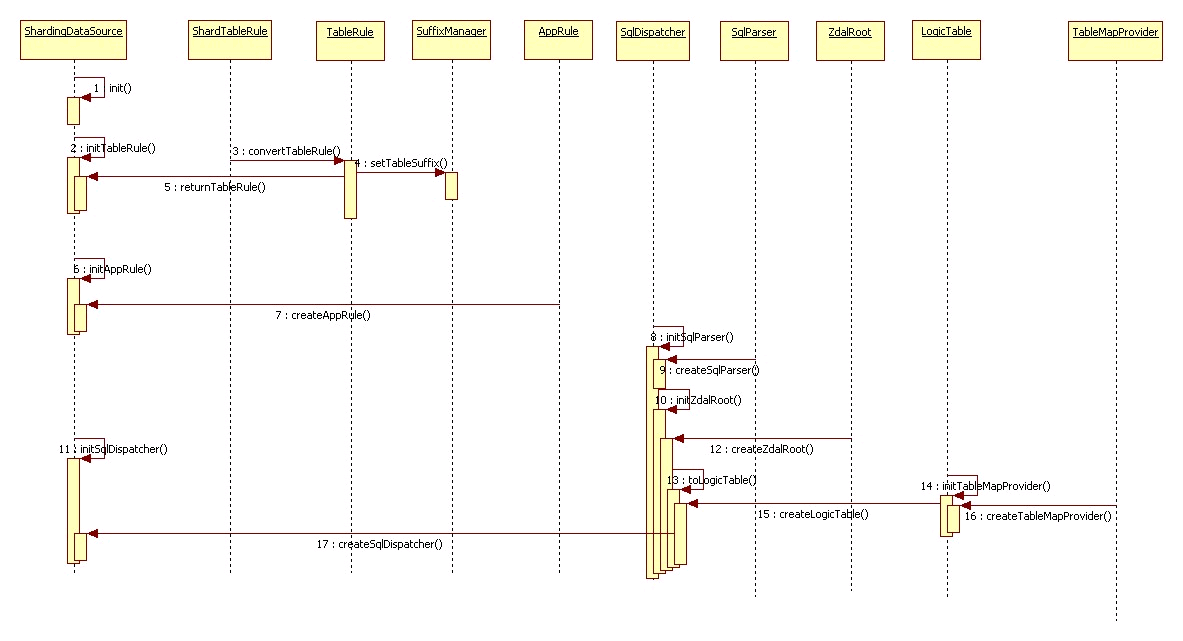


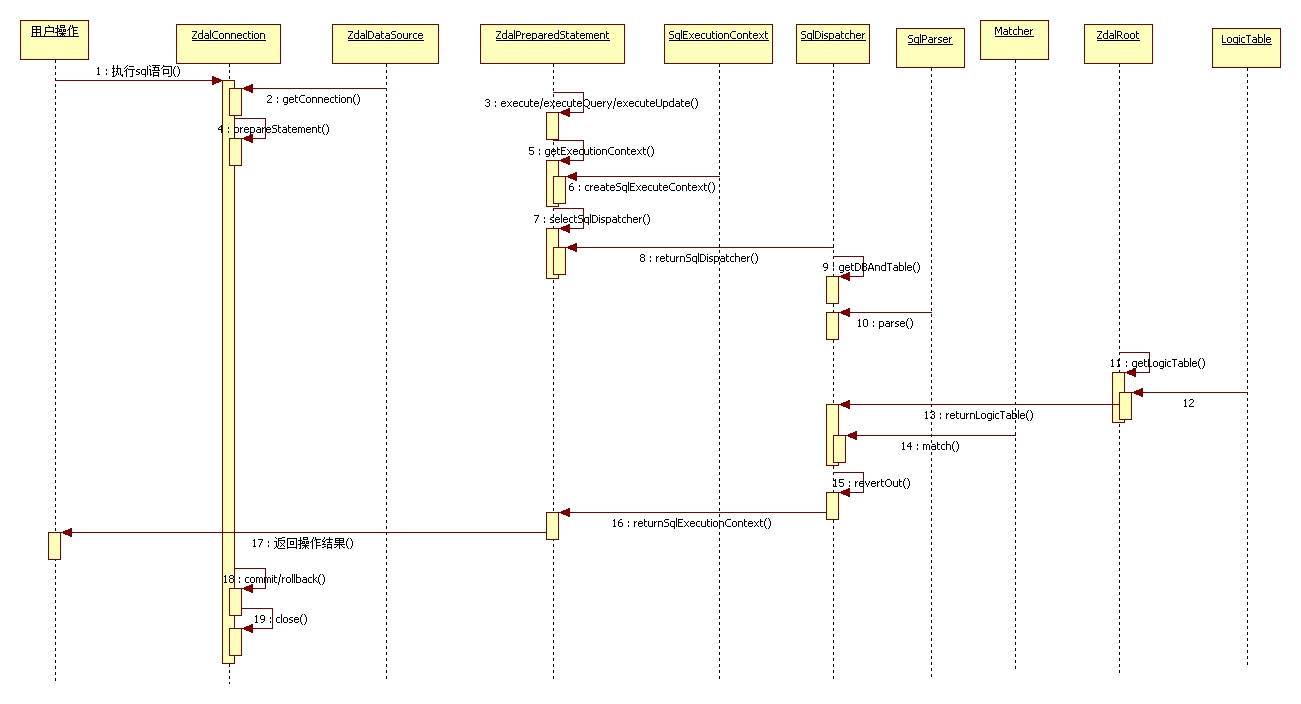
**5)DBP+Oceanbase**

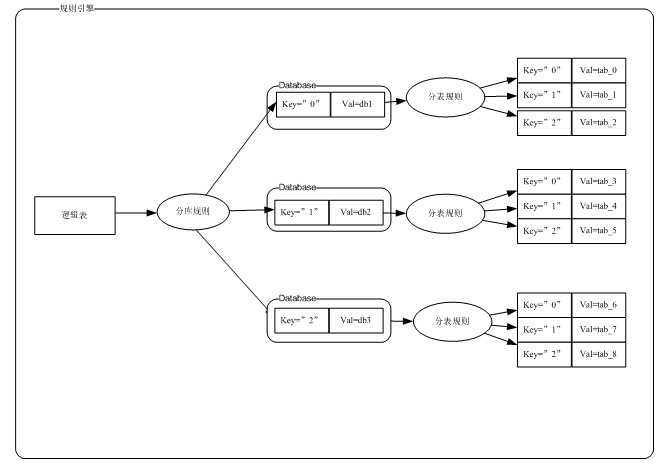
**=========================**

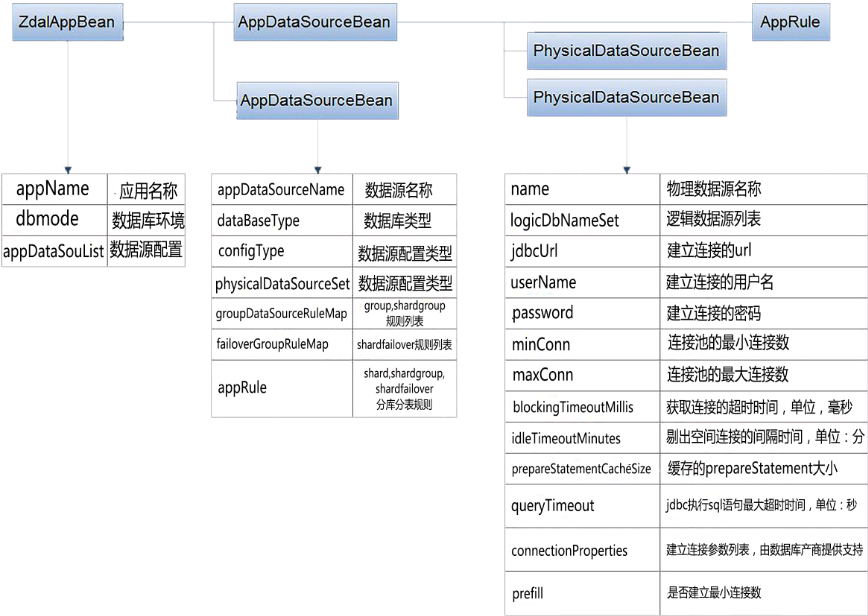


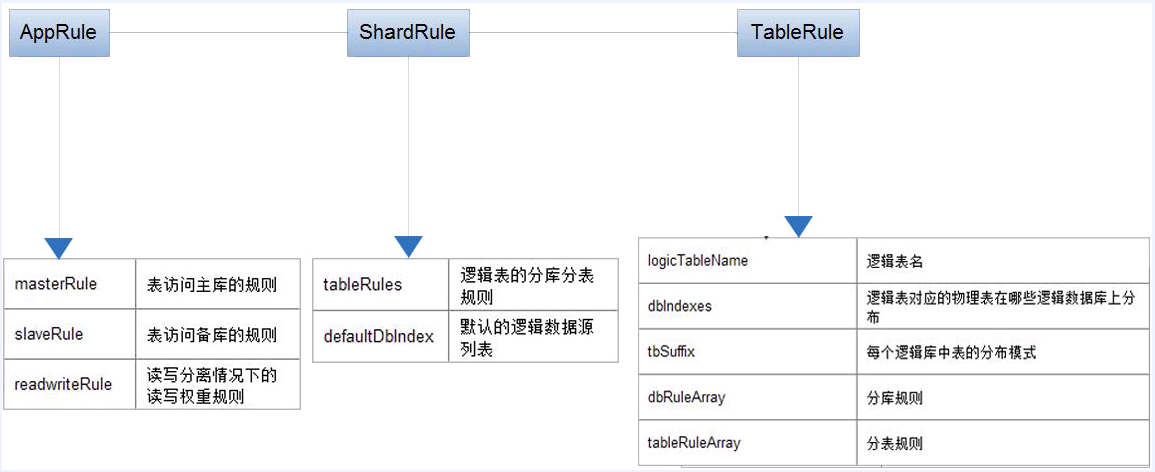












▲Zdal提供两类日志：

1.zdal-client-config.log：初始化的日志信息，主要打印appName,appDsName,dbmode,configPath,分库分表规则，groupRule,failoverRule,动态failover，markdown，markup等动态推送的日志信息。

2.zdal-datasource-pool.log：各个物理数据源连接池的连接状态日志信息；超时连接剔除的日志，异常连接的日志信息等。

【数据库灾备】

数据访问层组件是以jar包的方式嵌入app中，相应的DB灾备由app和DBA决定,在数据库出现故障需要failover，markdown，markup时，通过分布式环境的zookeeper等工具进行动态切换，zdal目前支持各个数据源权重的动态切换（这部分功能需要由使用方进行改造以便支持）

建立：逻辑实例，物理实例，写入分库分表规则

逻辑连接：分库分表规则，物理服务（observer）连接信息

**更进一步的架构需求和容灾需求**

**=======================**

OBProxy的主要架构可以简单描述如上图所示，其中异步框架实现高效的代理转发，通信协议实现OBProxy与Client和ObServer之间通信方式，连接管理实现OBProxy的连接池功能，**路由选择实现对用户请求的的最优路由选择,而容灾模块则负责监控OceanBase集群状态。**监控运维提供对OBProxy的丰富运维手段，集群管理实现OBProxy对OceanBase多集群的支持。以上组件相互依赖配合，共同实现OBProxy的整体功能。

OBProxy的容灾策略主要影响路由选择和连接管理，主要包含黑名单和灰名单两种检查，用于处理ObServer错峰合并、升级、宕机、启动/停止，网络分区等状态。黑名单采取定期刷新维护，由ObServer反馈哪些server节点死不可用。灰名单采取主动触发维护，当OBProxy转发请求给ObServer，如果发现ObServer返回特定的系统错误，或者ObSerer在N秒内有M次连续不可用，则将该ObServer加入灰名单。黑名单中的ObServer将被过滤不访问，但是灰名单中的ObServer每隔一段时间会重试一次，检查是否需要洗白，以避免长时间将ObServer拉黑。

对于用户的连接，通常情况下是Client <--> OBProxy <--> 多个Observer。而在Client的一个事务中，只会使用到一个OBProxy <--> ObServer的连接。对于其他非事务中的连接，如果ObServer发生异常导致网络连接断掉，OBProxy会关闭该无效的连接，Client端将对此无感知不受影响。