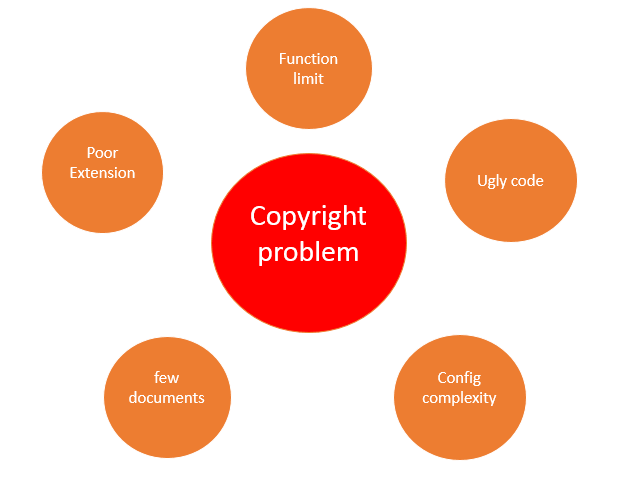
Dragon项目设计文档

# 1 为什么要使用Dragon替代TDDL

## 1.1 使用TDDL的六大罪状



## 1.2主流的数据库中间件设计方式

主要分为3种Database Proxy、ORM框架集成、实现JDBC规范

### 1.2.1 Database Proxy

目前主流的以Database Proxy方式设计的中间件有：Mysql-proxy、Mycat、Atlas。

Database Proxy的方式中，应用不直接连接数据库，而是连接proxy，由proxy来完成sql的解析、路由、合并查询结果、返回结果给客户端。

优点：

对应用来说是完全透明的，配置也完全和配置单个数据源一样。

缺点：

1、资源竞争：如果多个应用需要进行分库分表，访问同一个proxy，那么必然对proxy的资源产生竞争，最恶劣情况，如果由于某个应用由于请求非常频繁，导致proxy宕机，那么，其他的应用也受到影响，不能继续访问这个proxy。

2、proxy 需要做集群：由于所有的应用都通过proxy访问数据库，如果只有一个proxy实例，一般其挂了，那么所有的应用都无法访问数据库了。

### 1.2.2 ORM框架集成

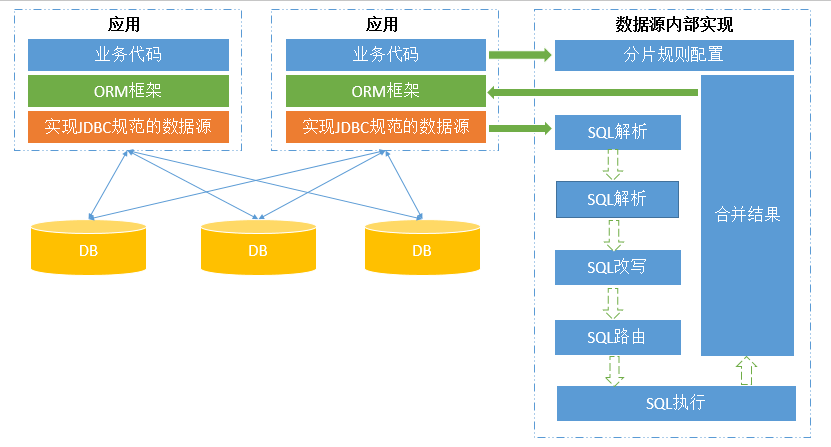
目前以ORM框架集成形式提供的主流的数据库中间件有：hibernate shards、ibatis-sharding。

优点：应用直连数据库，省去了中间的代理，应用之间相互无影响

缺点：必须要使用特定的ORM框架

### 1.2.3 实现JDBC规范的数据源

目前实现jdbc规范形式的数据库中间件主要由： TDDL、Zebra、sharding-jdbc。

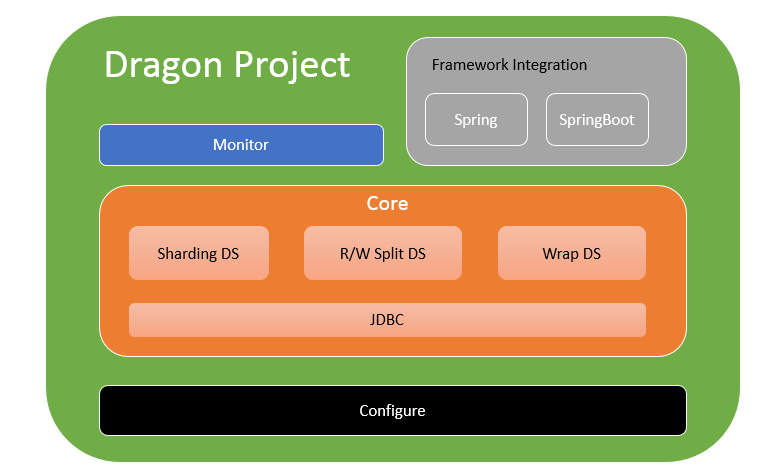


# 2、Drangon项目设计

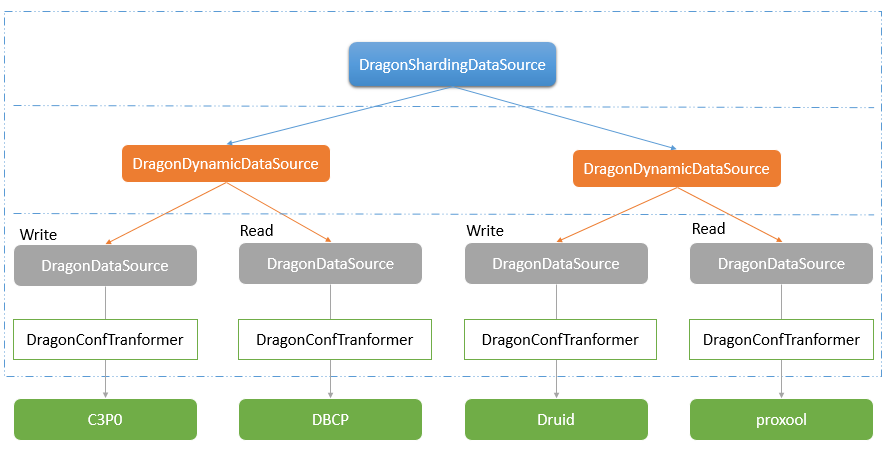
DragonDataSource是一个支持读写分离、分库分表的数据源，实现了JDBC规范。

Drangon项目名称的由来，通常支持分库分表，比支持读写分离复杂的多，而在实现分库分表的过程中，最复杂的部分莫过于sql-parser，这需要掌握一些编译原理相关的知识，编译原理三大圣经中的龙(Dragon)书，是必备的参考书籍，因此项目起名为Dragon。

## 2.1 整体架构



## 2.2 数据源设计



其中：

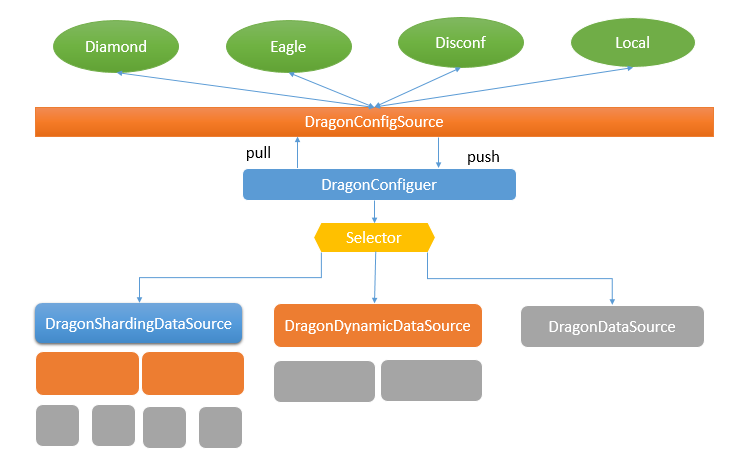
**DragonShardingDataSource**是用于支持分库分表的数据源

**DragonHADataSource**是用于支持读写分离的数据源

**DragonDataSource**是一个真实物理数据源的wrapper(包装设计模式)。

**DragonConfTranformer**是一个数据源配置的转换器。由于每个真实的物理数据源的配置项可能各不相同，DragonDataSource会定一些标准的配置项，通过DragonConfTranformer将标准的配置项转换为某个具体实际数据源的配置项。要支持某个真实的物理数据源，只要增加对应的DragonConfTranformer实现即可。(桥接设计模式)

## 2.3 配置模块



Dragon针对于配置，抽象出一个配置源(DragonConfigSource)的概念，不论配置是放在本地文件中，还是放在远程，Dragon并不关心，因为Dragon是从DragonConfigSource中来获取配置的信息。

DragonConfigSource需要支持通过pull和push的方式获取配置：

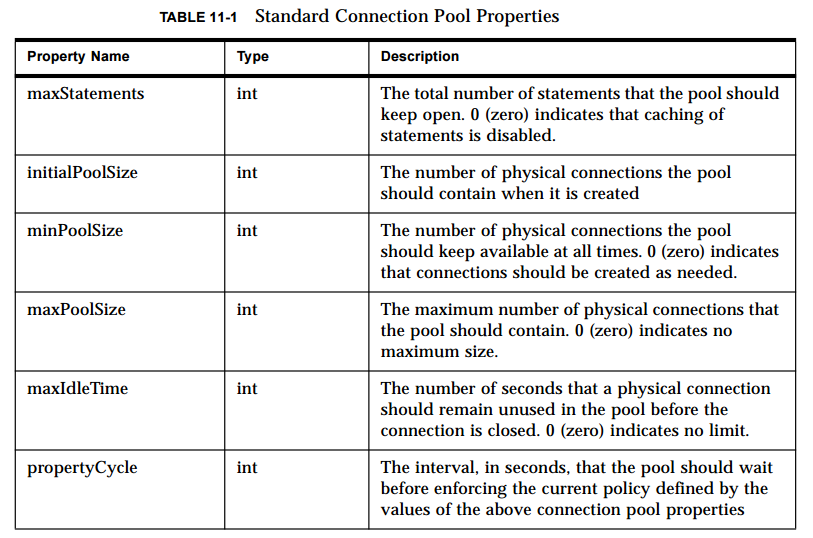
* pull方式指的是主动去外部数据源拉取配置，在Dragon启动的时候，需要主动pull。
* push方式指的是外部数据源主动向Dragon推送配置，例如想再运行时修改配置。

在启动的时候，Dragon会根据主动pull的配置信息，通过DragonConfiguer进行解析，Selector根据解析的结果选择到底初始化哪一个数据源。例如：如果配置信息是分库分表的配置，那么就初始化DragonShardingDataSource；如果是读写分离的配置信息，就初始化DragonHADataSource。

当Dragon在运行时接受到push的配置信息时，会首先解析出变化的部分，例如一个DragonHADataSource下只是某一个DragonConfigSource的配置发生了变化，只将变化的部分传递过来，然后通过Selector选择对应的DragonConfigSource，将变化的部分配置交给其处理。

# 3 DragonDataSource设计

## 3.1 JDBC 3.0规范中的标准连接池属性



除了这些标准属性，一般数据库连接池还会提供以下属性：

url、

driverClassName、

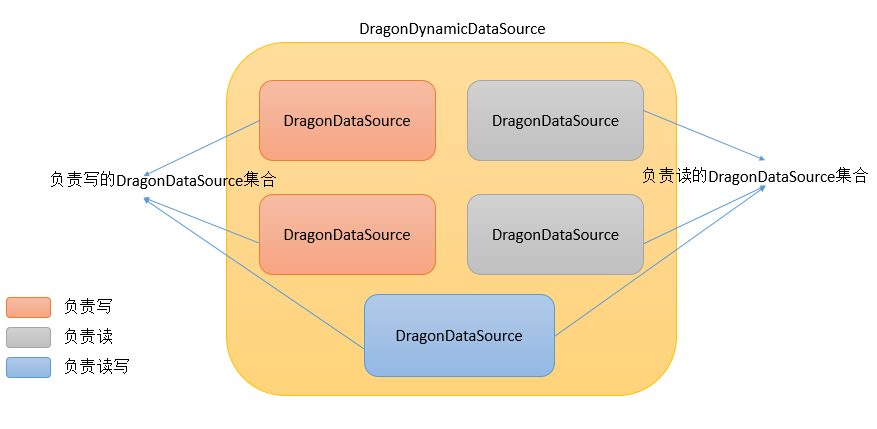
username、

password、

Extra config，由具体的DragonConfTransformer实现完成附加属性的配置。

# 4 DragonHADataSource设计

## 4.1 整体设计



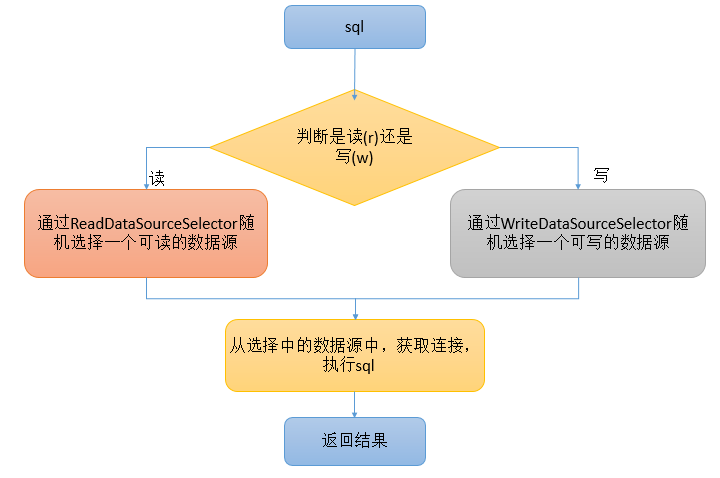
设计一个数据源DragonHADataSource，这个数据源下面可以管理多个真实物理数据源(DragonDataSource)。不同的DragonDataSource职责不同，有的负责读(R)，有的负责写(W)，也可以同时负责读写。

总的来说，DruidGroupDataSource下面管理的DruidDataSourcde，大致就是分为2类：

用于读的DragonDataSource集合：当有一个读的sql请求时，会从这个集合中选择一个数据源执行这个sql，这是通过一个类ReadDataSourceSelector实现的。

用于写的DragonDataSource集合：当有一个写的sql请求时，会从这个集合中选择一个数据源执行这个sql，这是通过一个类WriteDataSourceSelector实现的。

## 4.2 ****路由****



需要注意的是：在选择一个物理数据源时，每个物理数据源选中的概率是完全相同的。例如读的DragonDataSource集合大小为N，那么对于一个读sql，每个被选中的概率都是1/N。这里并没有使用权重的概念。

此外，为了便于区分，每个物理数据源都需要指定的一个唯一的dbIndex。

## ****4.3 Hint****

支持使用Hint的方式直接指定使用某个数据源。一些场景下，可能需要直接指定使用某个数据源，例如写完数据，立即读的情况。主库的数据同步到从库可能会存在延迟，导致读不到数据。因此需要可以手工影响路由方式的办法，这就是Hint的作用。

如

/\*DRUID\_GROUP\_HINT(dbIndex:**yourDbIndex**)\*/ SELECT id,name FROM user where id=xxxx;

其中/\*...\*/部分就是所谓的Hint。yourDbIndex是变量，根据用户配置的真实的DruidDataSource的dbIndex时指定。

## ****4.4 事务支持****

一旦开启了事务，事务中所有的sql语句，不论是读还是写，都是在写库(主库)上执行。

## 4.5 连接管理

## 4.6 ExeptionSorter

读失败重试的基础是对SqlException进行排序，如果是读取超时异常，可以考虑重试，但是如果是致命异常，例如某个mysql实例挂了，那么重试也没有意义，不光不应该重试，而且以后的请求都不应该路由到对应的数据源上。这就涉及到对异常进行排序。

### 4.6.1 SqlState

根据 X/Open 和 SQL Access Group SQL CAE 规范 (1992) 所进行的定义，SQLERROR 返回 SQLSTATE 值。SQLSTATE 值是包含五个字符的字符串 。五个字符包含数值或者大写字母， 代表各种错误或者警告条件的代码。SQLSTATE 有个层次化的模式：头两个字符标识条件的通常表示错误条件的类别， 后三个字符表示在该通用类中的子类。成功的状态是由 00000 标识的。SQLSTATE 代码在大多数地方都是定义在 SQL 标准里的。

表 7. 类代码 08：连接异常

| **SQLSTATE 值** | **含义** |
| --- | --- |
| 08001 | 应用程序请求器不能建立连接。 |
| 08002 | 连接已存在。 |
| 08003 | 连接不存在。 |
| 08004 | 应用程序服务器拒绝建立连接。 |
| 08007 | 事务解析未知。 |
| 08502 | 用 TWOPHASE 的 SYNCPOINT 运行的应用程序进程发出的 CONNECT 语句无效，因为无事务管理器可用。 |
| 08504 | 当处理指定的路径重命名配置文件时遇到错误。 |

### 4.6.2 vendor code

Java中的SqlException除了包含sqlstate，还包含一个vendor code。这是特定数据库供应商定义的错误。针对这类问题也要处理。

## ****4.7 读失败重试****

对于一个sql请求失败了，支持进行自动重试。例如，有2个DruidDataSource用于读，当第一个读取失败的时候，那么会尝试使用第二个进行读。默认最多重试3次(包含第1次)，且只会通过未尝试过的数据源进行重试。只对读操作进行重试，写操作不会进行重试，如果开启了事务，则也不会进行重试。

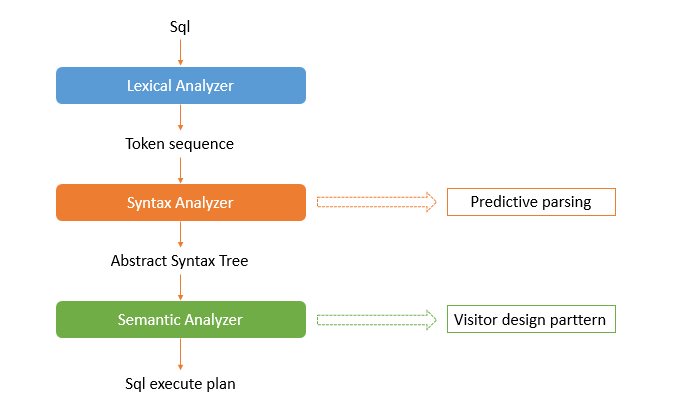
对于一些致命的异常，例如某个数据库server实例宕机，导致某个物理数据源不可用的情况，会将这个物理数据源标记为不可用，之后有sql的时候，不会再转发到这个数据源上。会有一个线程，定时测试被标记为不可用的数据源，一旦测试成功，重新标记为可用。

# 5 DragonShardingDataSource设计

## 5.1 规则引擎

## 5.2 Sql Parser

### 5.2.1 Sql 解析流程



### 5.2.2 主流的开源sql-parser

目前开源的sql-parser有很多，例如：

**FoundationDB SQL Parser：**

项目地址https://github.com/FoundationDB/sql-parser

**Jsqlparser**

项目地址：https://github.com/JSQLParser/JSqlParser

**Druid SQL Parser**

项目地址：<https://github.com/alibaba/druid/wiki/SQL-Parser>

其中Fdbparser和jsqlparser都是基于javacc实现的。

#### 5.2.2.1 sql解析性能对比

Mycat团队曾经做一个性能测试，对同一个sql语句，使用3种解析器解析出ast语法树，执行10万次、100万次的时间对比。测试结果表明：

1. 10万次：druid比fdbparser快10倍，比JSQLParser快6倍
2. 100万次：druid比fdbparser快15倍，比JSQLParser快近10倍
3. 如果sql越长，其快的倍数会越多，之前在公司有一条长的select语句，druid解析能比fdbparser快40倍。

Mycat团队同时还给出了druid解析更加快的原因。

1. Druid的解析过程使用“统筹方法”，就是几件事情同时一起做。Druid解析包含词法解析和语法解析，这两件事情它只需要从字符的第一个到最后一个遍历一遍，就同时完成了词法解析和语法解析，语法树也已经构造完成。
2. Fdbparser由于依赖于javacc解析，做了很多无用功，它是解析完所有的词法，中间的对象得到后最后再构造出ast语法树对象。

#### 5.2.2.2 数据库方言支持对比

SQL-92、SQL-99等都是标准SQL，mysql/oracle/pg/sqlserver/odps等都是方言，sql-parser需要针对不同的方言进行特别处理。

Druid的sql parser是目前支持各种数据语法最完备的SQL Parser。目前对各种数据库的支持如下：

|  |  |  |
| --- | --- | --- |
| 数据库 | DML | DDL |
| mysql | 完全支持 | 支持大部分 |
| oracle | 大部分 | 支持大部分 |
| postgresql | 完全支持 | 支持大部分 |
| sql server | 支持常用的 | 支持常用的ddl |
| db2 | 支持常用的 | 支持常用的ddl |

druid还缺省支持sql-92标准的语法，所以也部分支持其他数据库的sql语法。

Fdbparser虽然有很多弊端，但是也有它存在的理由，它更偏向于标准sql。

更能引导大家写标准sql，而不是mysql方言，这样应用程序才有可能支持多种数据库，领导说要换数据库时，不至于因为都是非标准sql要大改才能迁移到其他数据库。

结论：Druid支持的方言比fdbparser多。

需要注意的是，并不是偏向于sql标准，就一定好。例如limit是mysql数据库方言，如果不支持这个关键字，估计开发者要抓狂了。

## 5.3 跨库操作与事务

# 6 配置模块

# 7 框架整合模块

## 7.1 Spring

## 7.2 Spring Boot