

The logo for Gdevops features a stylized orange 'G' followed by the word 'devops' in white lowercase letters. The background is a solid blue color with faint, light-colored geometric shapes and patterns.

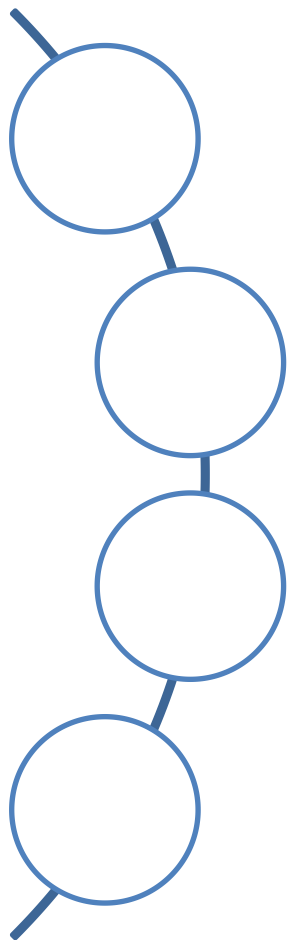
Gdevops

# 全球敏捷运维峰会

分布式数据库中间件  
的设计与实现

演讲人：张亮

# 互联网领域数据库面临的问题



# 各种数据库方案对比

	RDBMS	NoSQL	NewSQL
SQL支持	原生	不支持	不完善
事务	ACID+XA	BASE	F1
存储引擎	成熟	较成熟	待验证
数据分片	有限支持	支持	支持
动态扩容	不支持	有限支持	支持较好

# RDBMS解决方案的优缺点

开发友好，面向SQL  
存储引擎稳定  
单节点事务引擎成熟  
未达阈值的单机性能高



单节点并发访问频率受限  
单节点数据承载量受限  
分布式事务性能难以接受  
分布式扩展困难



# 当当数据库中间层的关注重点

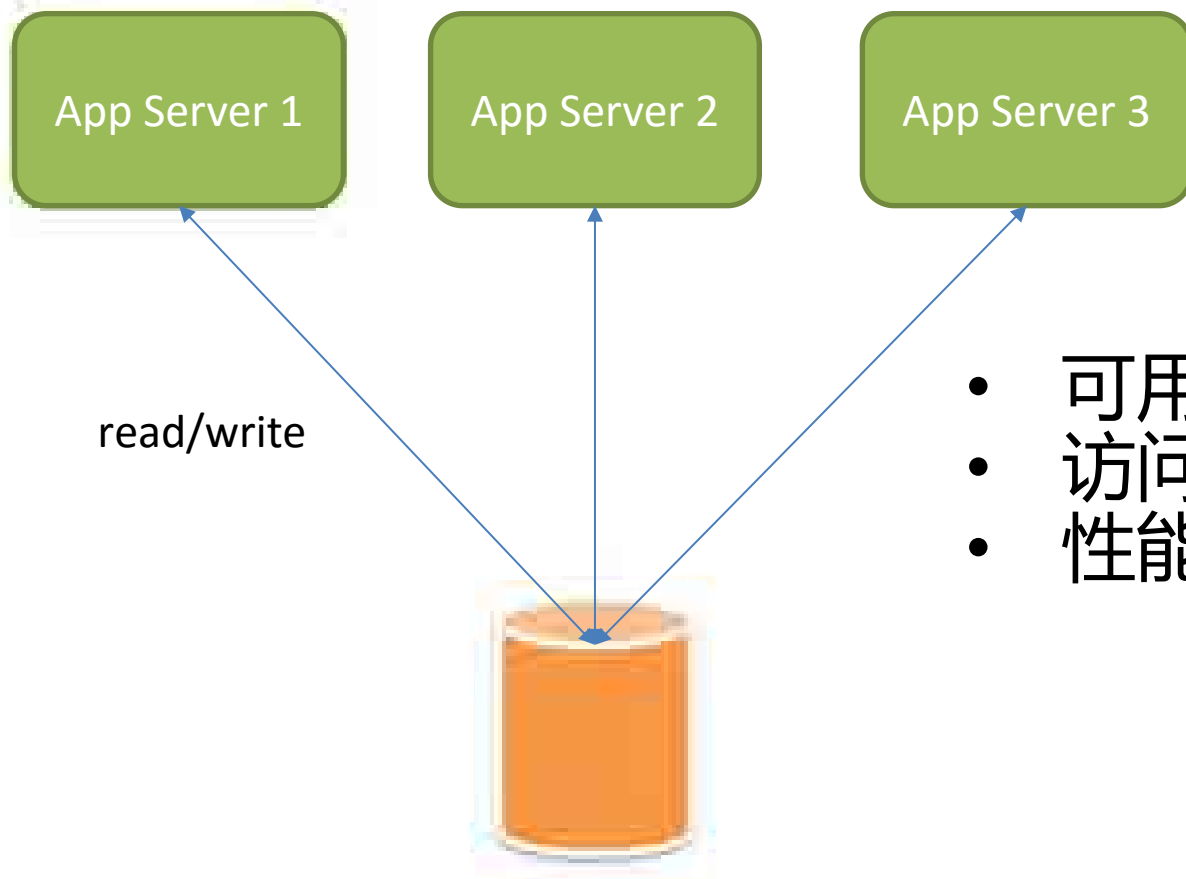
- 配置动态化
- 数据源自动切换

- 弱XA
- 柔性事务

分片

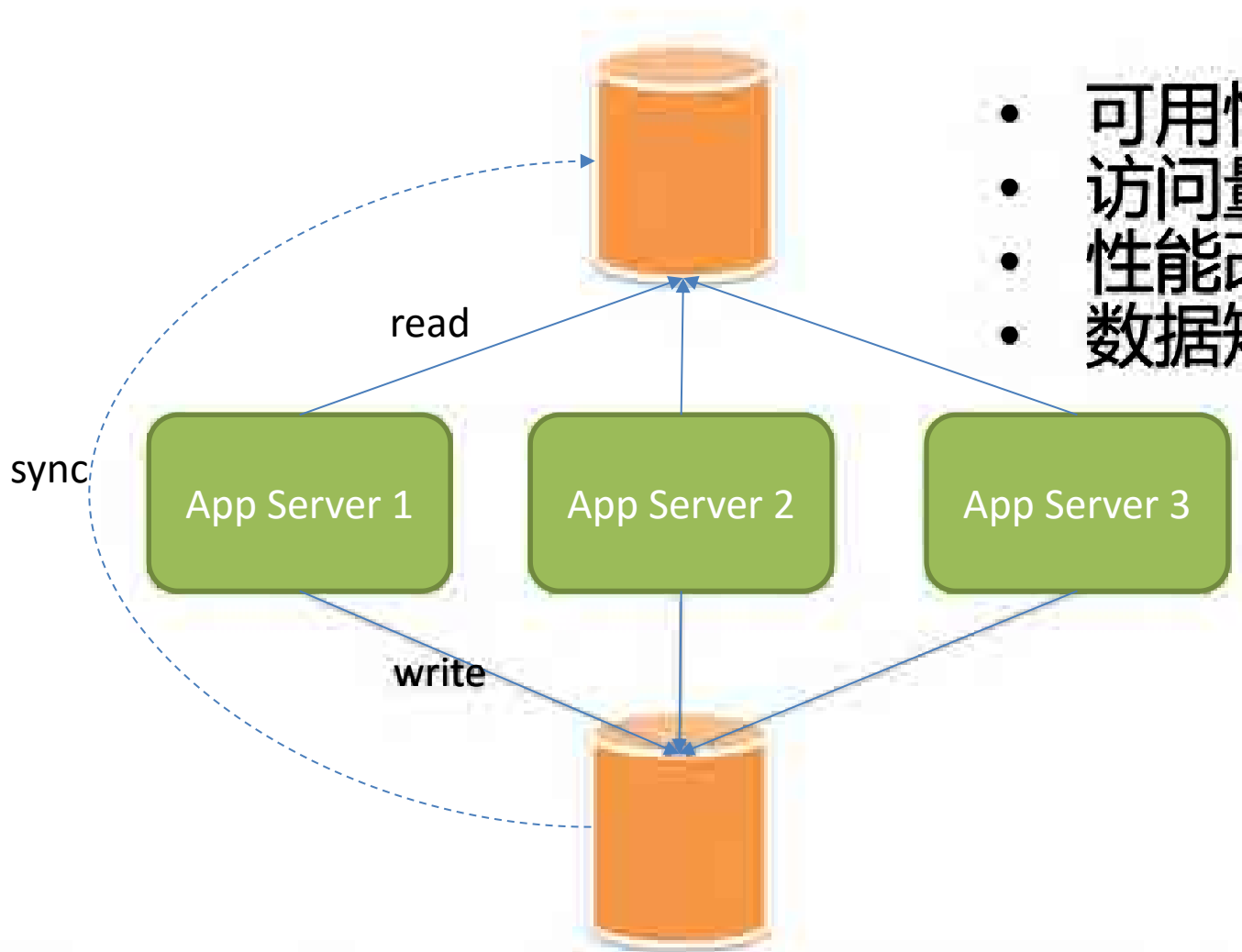
- 分库分表
- 读写分离
- 分布式主键

# 单库



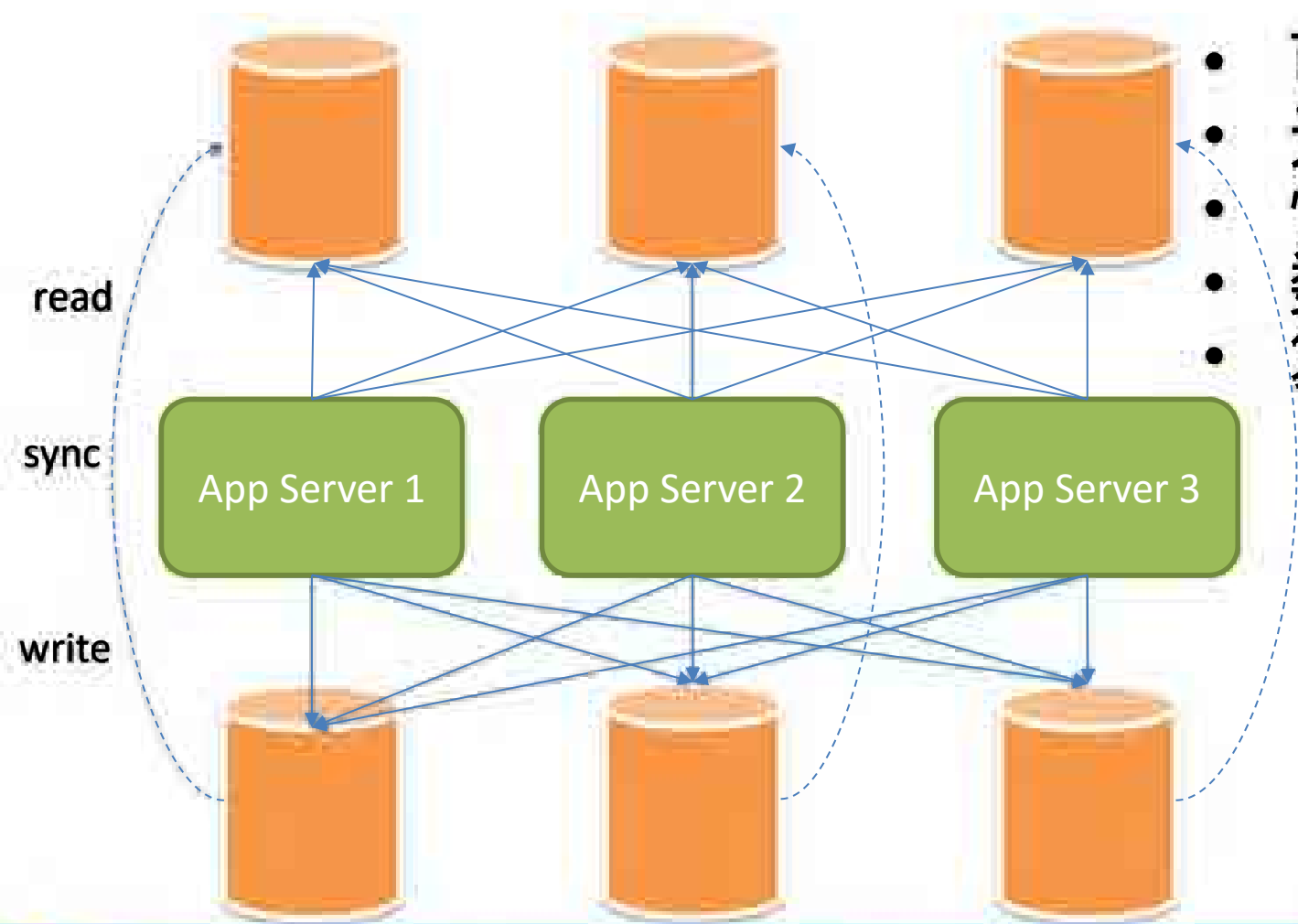
- 可用性差
- 访问量瓶颈受限
- 性能低下

# 读写分离



- 可用性略提升
- 访问量瓶颈提升
- 性能改善
- 数据短期不一致

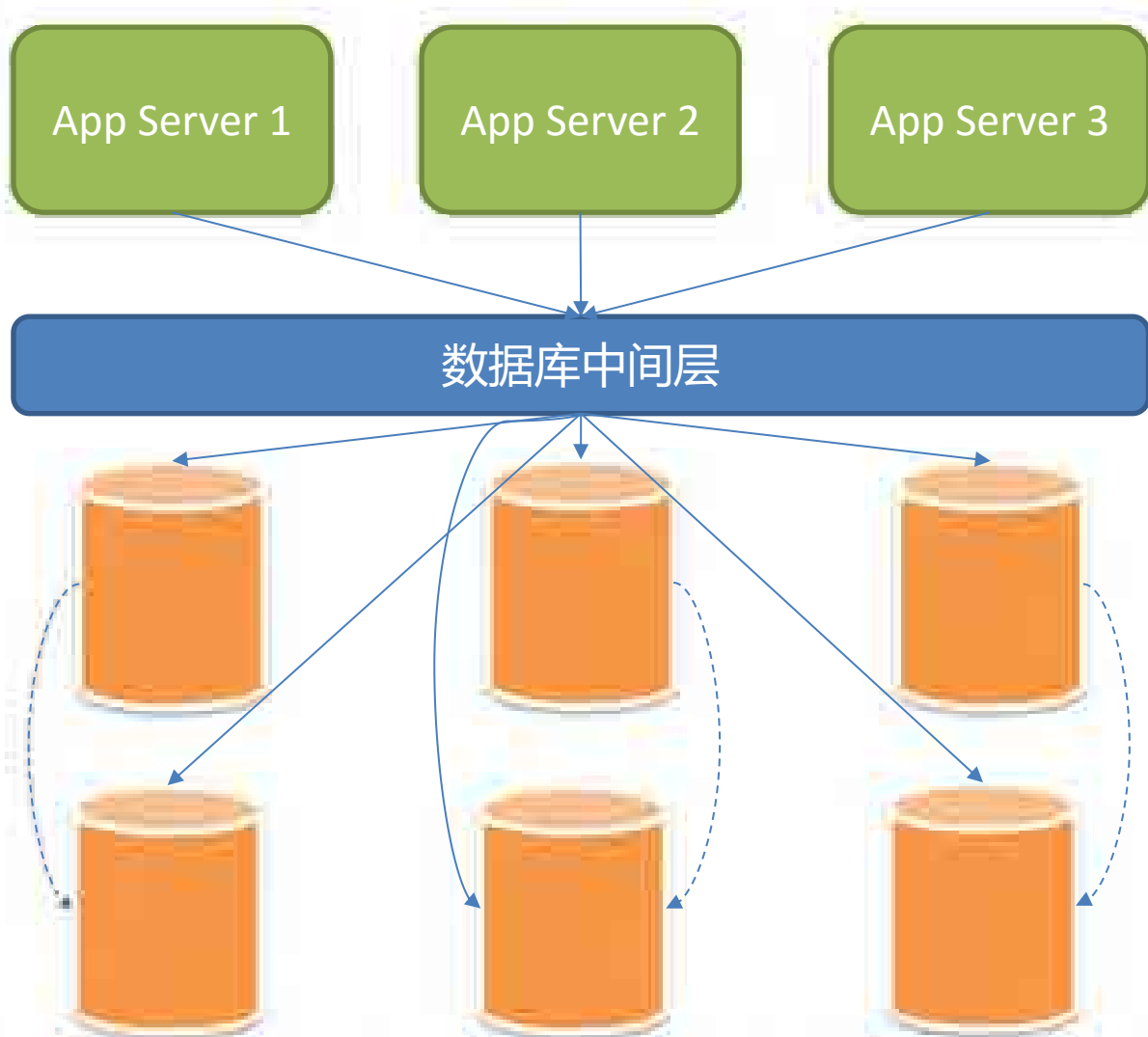
# 分库分表+读写分离



- 可用性提升
- 无访问量瓶颈
- 性能大幅改善
- 数据短期不一致
- 复杂



# 引入数据库中间层



- 可维护性提升
- 简明清晰



# 分片类型

# 垂直分片

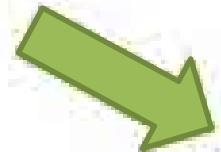
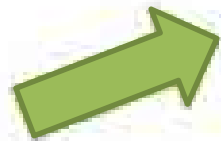
```
SELECT * FROM t_user WHERE id=1
```

```
SELECT * FROM t_order WHERE id=1
```



```
SELECT * FROM t_user WHERE id=1
```

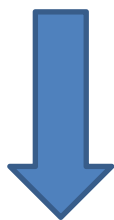
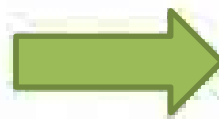
```
SELECT * FROM t_order WHERE id=1
```



# 水平分片

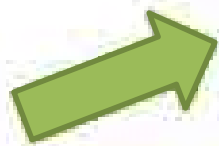
```
SELECT * FROM t_user WHERE id=1
```

```
SELECT * FROM t_user WHERE id=2
```



```
SELECT * FROM t_user WHERE id=1
```

```
SELECT * FROM t_user WHERE id=2
```



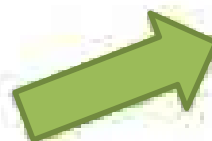


# 水平分片策略

# 哈希取模分片策略

```
SELECT * FROM t_user WHERE id=1
```

```
SELECT * FROM t_user WHERE id=2
```

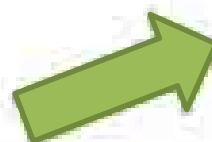


# 范围分片策略

```
SELECT * FROM t_user WHERE id=1
```

```
SELECT * FROM t_user WHERE id=1001
```

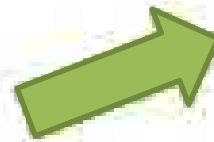
```
SELECT * FROM t_user WHERE id=2001
```



# 标签分片策略

```
SELECT * FROM t_user WHERE location= 'bj'
```

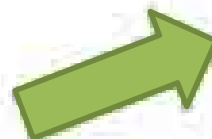
```
SELECT * FROM t_user WHERE location  
= 'sh'
```



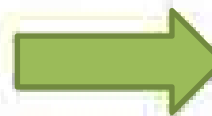


# 时间分片策略

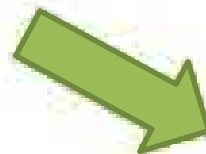
```
SELECT * FROM t_order WHERE year=2015
```



```
SELECT * FROM t_order WHERE year=2016
```



```
SELECT * FROM t_order WHERE year=2017
```

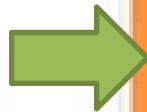


# 复合分片策略

```
SELECT * FROM t_order  
WHERE user_id=1 AND order_id=1001
```



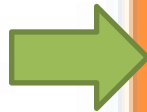
```
SELECT * FROM t_order  
WHERE user_id=1 AND order_id=1002
```



```
SELECT * FROM t_order  
WHERE user_id=2 AND order_id=2001
```

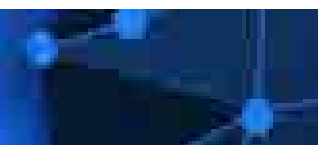


```
SELECT * FROM t_order  
WHERE user_id=2 AND order_id=2002
```





# 实现方案



# 透明化实现方案选型

	Proxy	ORM	JDBC
数据库	单一	任意	任意
ORM	任意	单一	任意
异构语言	任意	仅Java	仅Java
性能	损耗略高	损耗低	损耗低

# Sharding-JDBC是什么

- **开源**的分布式数据库中间件，它无需额外部署和依赖，旧代码**迁移成本**几乎为**零**。
- 面向开发的**微服务**与**云原生**的基础类库。
- 完整的实现了**分库分表**、**读写分离**和**分布式主键**功能，并初步实现了**柔性事务**，**治理**正在功能开发中。

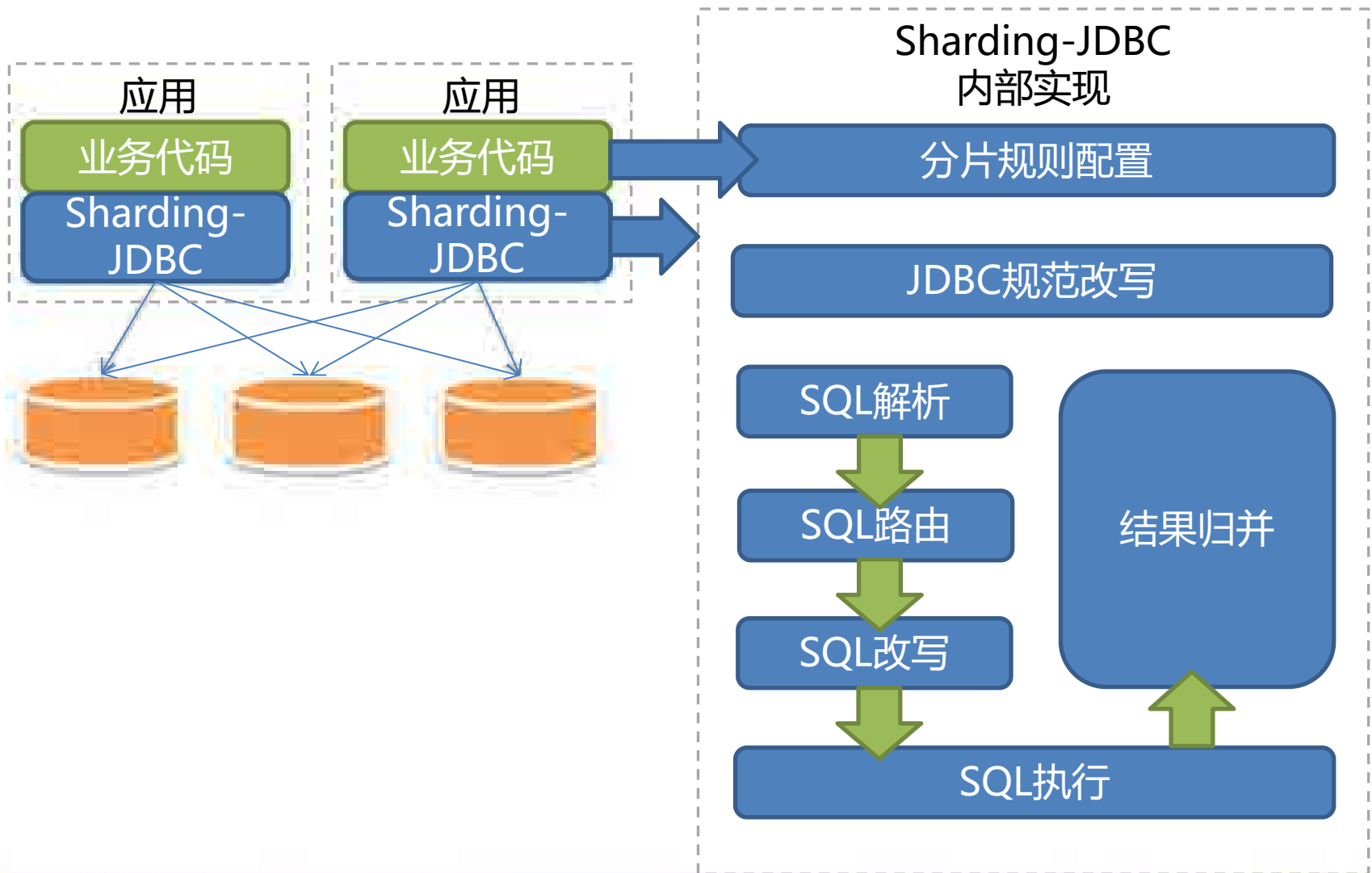
# Sharding-JDBC兼容性

- Mybatis
- JPA
- Hibernate
- JDBC

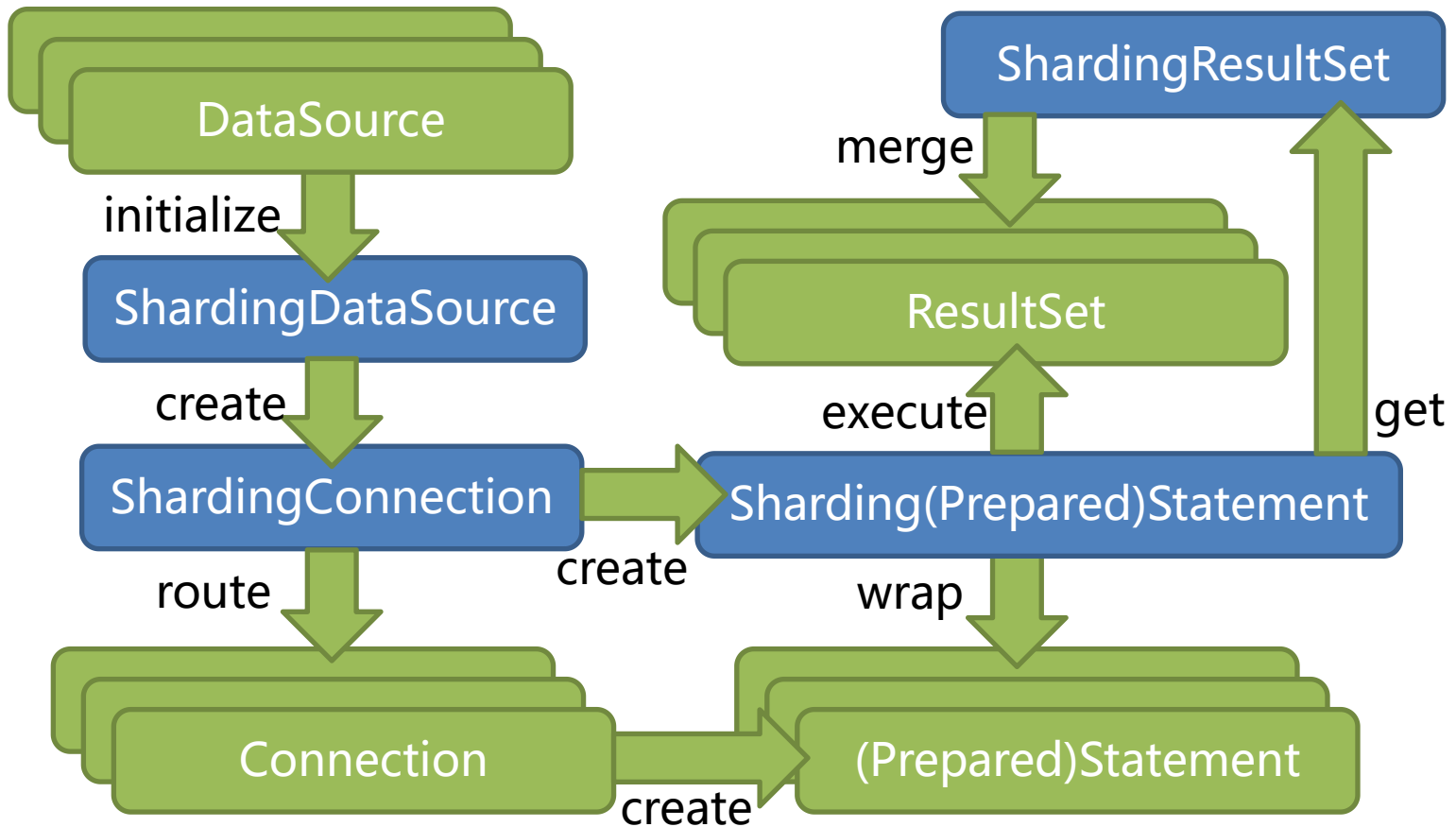
- MySQL
- Oracle
- SQLServer
- PostgreSQL

- DQL
- DML
- DDL

# 分片架构图



# JDBC规范改写





# 为什么需要解析SQL

- 仅分库的单分片查询
- 仅分库的跨分片的无聚合、排序、分组查询
  
- 包含分表的查询
- 跨分片的聚合、排序、分组查询
- 复杂查询，如：OR、UNION、子查询等

# SQL解析示例

```
SELECT
HIGH_PRIORITY STRAIGHT_JOIN SQL_BUFFER_RESULT SQL_NO_CACHE
id, name FROM table_x WHERE id=1
INTO OUTFILE 'file_a' LOCK IN SHARE MODE
```

```
SELECT id, name FROM table_a WHERE id=1
```

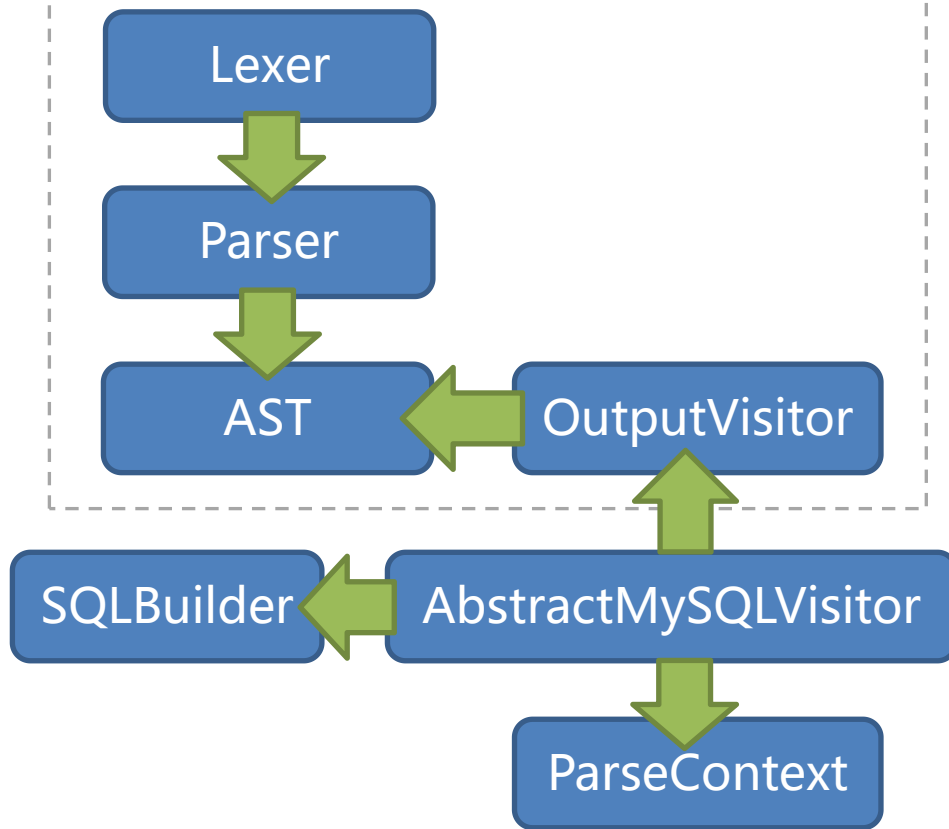
```
{
table : table_x
sharding-column : id
sharding-value : 1
}
```

```
SELECT
HIGH_PRIORITY STRAIGHT_JOIN SQL_BUFFER_RESULT
SQL_NO_CACHE
id, name FROM token WHERE id=1
INTO OUTFILE 'file_a' LOCK IN SHARE MODE
```

# SQL解析 初版

使用Druid

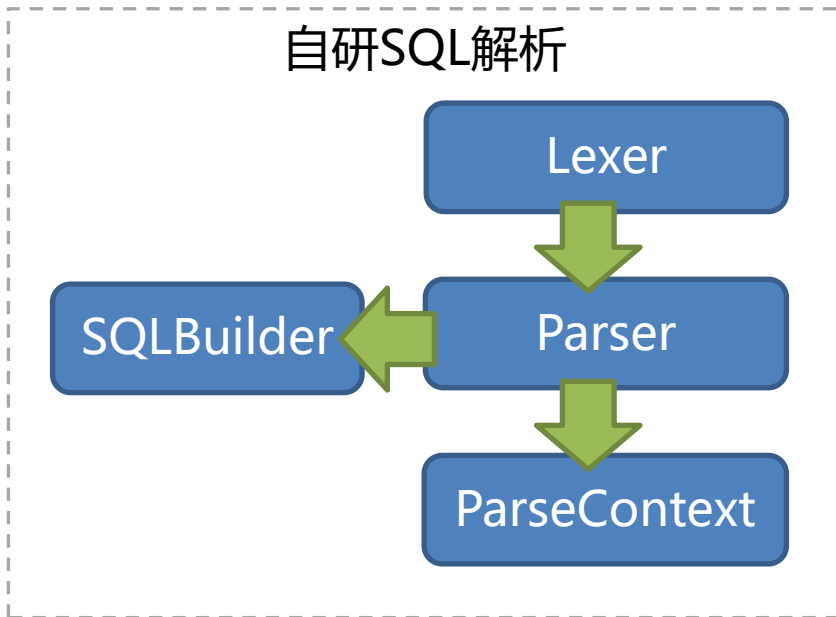
Druid



## 问题（仅针对专注于 Sharding 的解析）

- 性能
  - 需解析全SQL，语法树复杂
  - 需二次访问语法树，再次生成解析结果
- 准确性
  - 重新生成导致原SQL变化
- 易读性
  - 针对OutputVisitor重写，编码零散
- 兼容性
  - Druid升级后不兼容之前版本

# SQL解析 再版



## 提升

- 性能
  - 仅解析与分片相关部分
  - 无需二次访问语法树
- 准确性
  - 使用原SQL，无需再生成
- 易读性
  - 代码聚合，无零散编码
- 稳定性
  - 减少依赖，剥离第三方lib升级兼容问题

# 路由算法

- 单分片键
- 精确路由 ( = , IN ) + 范围路由 ( Between )

- 单分片键
- Inline表达式, eg : t\_user\_ \${userid % 8}

- 多分片键
- 分片算法的复杂度由用户自行控制

- 无分片键
- 通过Hint直接指定DataNode

- 不分片

# 路由类型

- SQL中仅存在单一表
- SQL中仅存在多表，  
但分表策略完全一致
- SQL中仅存在多表，  
且分表策略不一致

# SQL改写

- 标记Token

- `LIMIT m, n => LIMIT 0, n`

- `AVG(expr) => SUM(expr), COUNT(expr)`

- 排序列生成补列

- 分组列生成补列
- ORDER BY补充

- 主键生成补列

# 为何改写Limit

表t\_score\_0

score
100
90
80

表t\_score\_1

score
95
85
75

**SELECT score FROM t\_score ORDER BY score DESC LIMIT 1 , 2**

SQL不改写的查询结果

表t\_score\_0

score
90
80

表t\_score\_1

score
85
75

最终归并结果

score
85
80



# 为何改写Limit

SQL改写

```
SELECT score FROM t_score ORDER BY score DESC LIMIT 0, 3
```

SQL改写后的查询结果

表t\_score\_0

score
100
90
80

表t\_score\_1

score
95
85
75

最终归并结果

score
95
90

# 为何补列

- 无需补列的场景

```
SELECT id, name FROM t_user WHERE ORDER BY name
```

- 需要补列的场景

```
SELECT id, age FROM t_user WHERE ORDER BY name
```

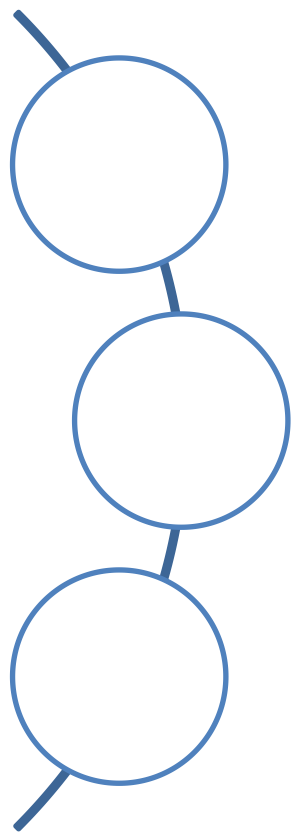
```
SELECT id, name AS n FROM t_user WHERE ORDER BY name
```

```
SELECT u.* FROM t_user AS u JOIN t_order AS o ON  
u.user_id=o.user_id WHERE ORDER BY name
```

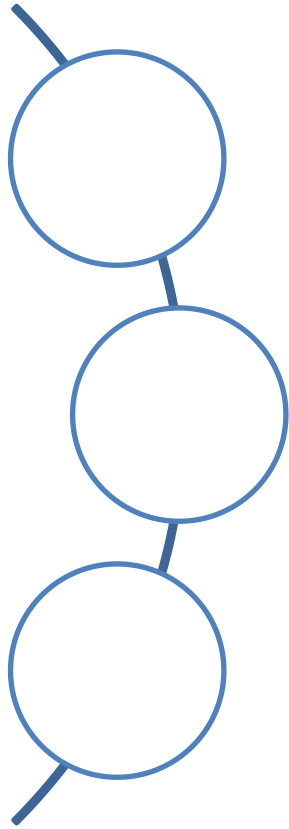
# 结果归并

- LIMIT m, n => LIMIT 0, n
- 跳过前n条数据
- 比较并返回最小(大)值
- 多结果集累加, DISTINCT暂未实现
- AVG(expr) => SUM(expr), COUNT(expr)
- return SUM / COUNT
- 排序列生成补列
- 多结果集归并排序
- 分组列生成补列
- 所有结果集加载至内存进行分组、聚合、排序

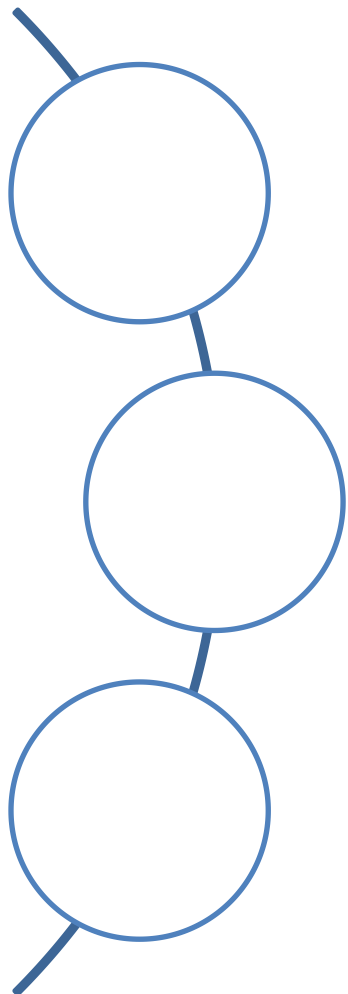
# 读写分离



# 其他功能



# Sharding-JDBC注意事项





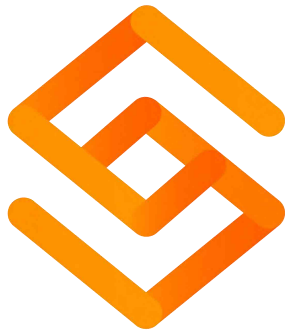
# Roadmap

- 数据源动态切换
- 数据库访问层治理

- 子查询深度支持
- OR
- 批量INSERT

- TCC

# Sharding-JDBC成绩单



## Sharding Jdbc

<https://shardingjdbc.io>

Github Star **2759** fork **1198**

获得开源中国2016年最受欢迎的开源软件  
第**17**名

入选码云年度**GVP**项目

明确采用公司**24**家





Gdevops

# 全球敏捷运维峰会



THANK YOU !

