RUST CHINA CONF 2020

首届中国 Rust 开发者大会

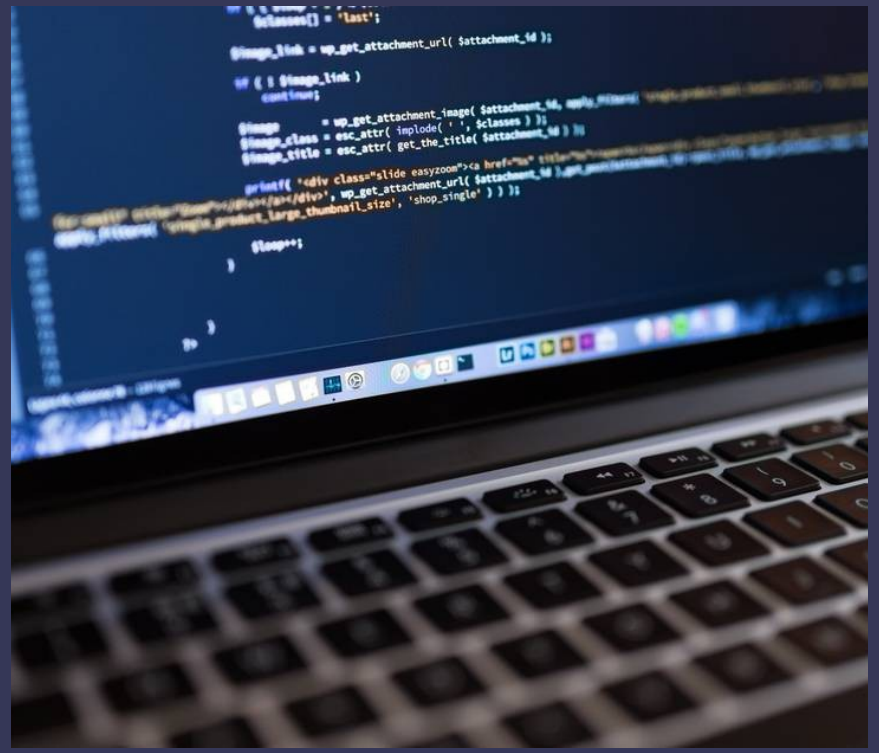
2020.12.26–27 深圳

Rust China Conf 2020

Shenzhen, China

2020conf.rustcc.cn

历代 ORM 之痛



无缘高并发 - rust 大部分 ORM 使用同步模型，要么使用 unsafe FFI+ 同步 调用原生驱动程序，要么只支持单一数据库，而且某些 ORM 需要环境变量增加运维难度

难以自定义 SQL - 传统 ORM 难以编写便捷的自定义 SQL

自定义 SQL 时写 xml 累人 - 使用 XML 组织 SQL 复杂且冗余代码量大

Rbatis 如何解决痛点?

① 驱动实现选用 sqlx-core （是 sqlx 数据库通讯的实现），使用条件编译 + 判断抽象包装出通用的 DBPool, 初始化各个数据库连接池，仅需一行代码

② Rbatis 上层同时支持 SQL 包装器和 Py-Sql 解释器，以超低的代价写出正确 + 灵活 + 缩短的 SQL

③ Py-SQL 解释器在增强灵活性的同时，缩短对等 XML 形式 30%-40% 代码量

④ 借鉴于 Mybatis， Mybatis Plus,GoMybatis

Rbatis-ORM 介绍

- 使用 serde_json 处理序列化，简化对象 -sql 转换
- 灵活易用性，内置 SQL 包装器 + 内置 py_sql + 宏 + 四合一驱动 (可选条件编译)

- 丰富插件支持，分页插件，逻辑删除插件，SQL 拦截器插件
- 百分百安全安全代码实现，完全基于 Future，支持高并发运行时 Tokio / AcyncStd

成为下一个 SSM 中的 Rust 版本 M

Mybatis 与 Rbatis 灵活性 PK - 1

这个例子是实现最常见的插入数据库操作例子
要求插入跳过空字段
Rbaits 保持语义和 Mybatis 描述一样情况下
不仅代码量缩减 50% 以上
而且 Rbatis 为 py_sql 宏提供分页插件支持
提升可维护性（增删字段无需改 SQL ）

第 1 步假设我们定义表结构如右图

```rust
#[crud_enable]
#[derive(Serialize, Deserialize, Clone, Debug)]
pub struct BizActivity {
    pub id: Option<String>,
    pub name: Option<String>,
    pub pc_link: Option<String>,
    pub h5_link: Option<String>,
    pub remark: Option<String>,
    pub create_time: Option<NaiveDateTime>,
    pub delete_flag: Option<i32>,
}
```

Mybatis-xml （ 21 行，未计入 java， 需要定义 BaseRuseltMap 配置）

```xml
<insert id="insert">
    insert into biz_activity
    <trim prefix="(" suffix=")" suffixOverrides=",">
        <if test="id != null">id,</if>
        <if test="name != null">name,</if>
        <if test="pc_link != null">pc_link,</if>
        <if test="h5_link != null">h5_link,</if>
        <if test="remark != null">remark,</if>
        <if test="create_time != null">create_time,</if>
        <if test="delete_flag != null">delete_flag,</if>
    </trim>

    <trim prefix="values (" suffix=")" suffixOverrides=",">
        <if test="id != null">#{id},</if>
        <if test="name != null">#{name},</if>
        <if test="pc_link != null">#{pc_link},</if>
        <if test="h5_link != null">#{h5_link},</if>
        <if test="remark != null">#{remark},</if>
        <if test="create_time != null">#{create_time},</if>
        <if test="delete_flag != null">#{delete_flag},</if>
    </trim>
</insert>
```

Rbatis-PySQL 实现 (9 行 - 包含 Rust 定义 - 维护性高 - 异步！)

```rust
#[py_sql(RB, "insert into biz_activity
                (
                trim ',': for key,item in arg: if item != null:
                    ${key},
                ) VALUES (
                trim ',': for item in arg: if item != null:
                    #{item},
                )   ")]
pub async fn py_insert(arg: &BizActivity) -> DBExecResult {}
```

mybatis 与 Rbatis 灵活性 PK - 2

也许你会说，那我可以在 Mybatis 中使用 foreach, 那么代码改成以下内容（13 行，要求参数 arg 为集合）

```
<insert id="insert">
    insert into biz_activity
    <trim prefix="(" suffix=")" suffixOverrides=",">
        <foreach collection="arg" item="item" index="index" >
            #{arg}
        </foreach>
    </trim>
    <trim prefix="values (" suffix=")" suffixOverrides=",">
        <foreach collection="arg" item="item" index="index" >
            #{item}
        </foreach>
    </trim>
</insert>
```

Rbatis-PySQL 实现 (仅 9 行 - 包含 Rust 定义 - 维护性高)

arg 可以是对象，结构体，Map，Array，Slice....

```
#[py_sql(RB, "insert into biz_activity
                (
                trim ',': for key,item in arg: if item != null:
                    ${key},
                ) VALUES (
                trim ',': for item in arg: if item != null:
                    #{item},
                )   ")]
pub async fn py_insert(arg: &BizActivity) -> DBExecResult {}
```

mybatis 与 Rbatis 灵活性 PK - 3

Mybatis 的参数会因为 JVM 编译丢失，因此需要 @Param 指定名称。包括 golang 参数名称也是丢失的

```java
@Repository
public interface ActivityMapper {

    List<Activity> selectByCondition(@Param("name") String name,
                                     @Param("startTime") Date startTime,
                                     @Param("endTime") Date endTime,
                                     @Param("page") Integer page,
                                     @Param("size") Integer size);

}
```

Rbatis 因为使用过程宏，参数名称不丢失，参数名只需定义一次。我喜欢干净代码 +1

```rust
#[py_sql(RB, "select * from biz_activity where delete_flag = 0 where
                and name=#{name} and start_time = #{start_time} and end_time = #{end_time}")]
fn select_page(page_req: &PageRequest,
               name: &str,
               start_time:&NaiveDateTime,
               end_time:&NaiveDateTime) -> Page<BizActivity> {}
```

GoMybatis 与 Rbatis 压测 PK

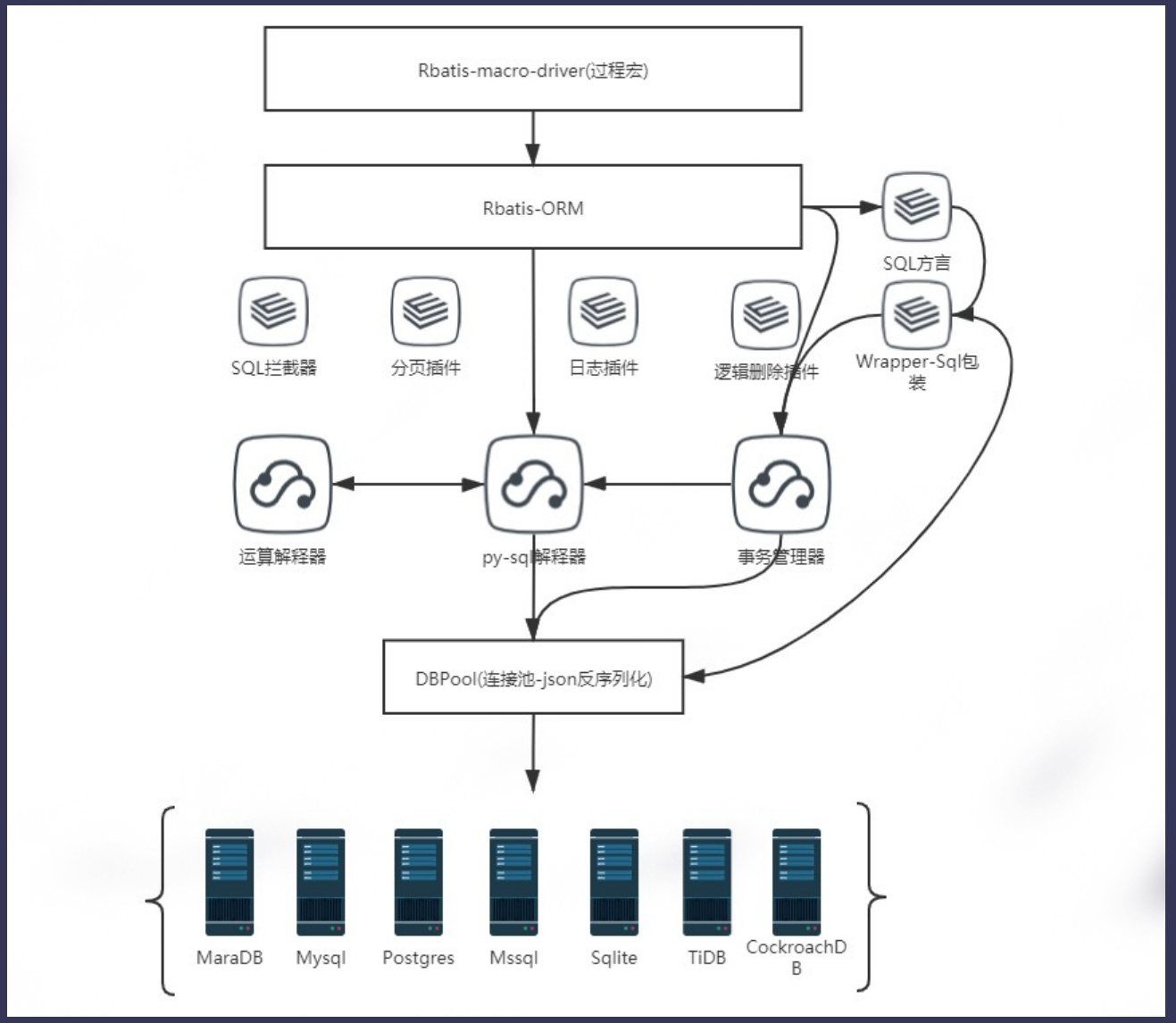
GoMybatis 是本人在 golang 语言的实现版本（仅包含 json 序列化加 XML 实现），连接池是使用 sql/DB （标准库） +go-mysql

Rbatis 借助 Rust 解决更多不足，提高生产力的情况下，且 QPS 提升大约 20%，内存消耗减少 13 倍！性能 - 安全 - 生产力兼顾

Performance comparison with Golang (in a docker environment)

Framework	Mysql (docker)	SQL statement (10k)	ns/operation (lower is better)	Qps(higher is better)	Memory usage(lower is better)
Rust-rbatis/tokio	1 CPU, 1G memory	select count(1) from table;	965649 ns/op	1035 Qps/s	2.1MB
Go-GoMybatis/http	1 CPU, 1G memory	select count(1) from table;	1184503 ns/op	844 Qps/s	28.4MB

Rbatis- 架构总览

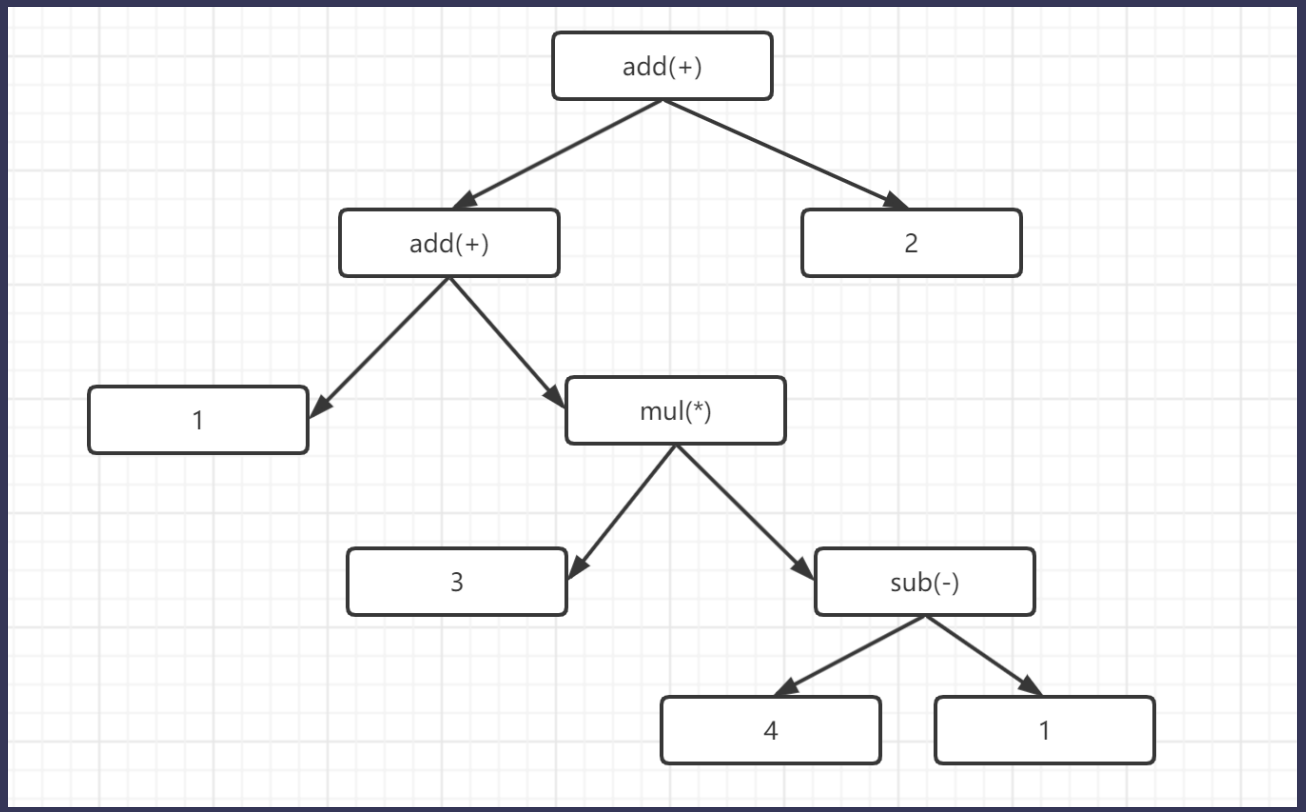

Rbatis-macro-driver(过程宏)
Rbatis-ORM
SQL方言
SQL拦截器
分页插件
日志插件
逻辑删除插件
Wrapper-Sql包装
运算解释器
py-sql解释器
事务管理器
DBPool(连接池-json反序列化)
MaraDB
Mysql
Postgres
Mssql
Sqlite
TiDB
CockroachDB

Wrapper-SQL 包装 - 实现细节披露

```rust
let mut m : Map<String, Value> = Map::new();
m.insert( k: "a".to_string(), v: json!("1"));
let w : Wrapper = Wrapper::new( driver_type: &DriverType::Mysql).eq( column: "id", obj: 1) : &mut Wrapper
    .ne( column: "id", obj: 1) : &mut Wrapper
    .in_array( column: "id", obj: &[1, 2, 3]) : &mut Wrapper
    .not_in( column: "id", obj: &[1, 2, 3]) : &mut Wrapper
    .all_eq( arg: &m) : &mut Wrapper
    .like( column: "name", obj: 1) : &mut Wrapper
    .or() : &mut Wrapper
    .not_like( column: "name", obj: "asdf") : &mut Wrapper
    .between( column: "create_time", min: "2020-01-01 00:00:00", max: "2020-12-12 00:00:00") : &mut Wrapper
    .group_by( columns: &["id"]) : &mut Wrapper
    .order_by( is_asc: true, columns: &["id", "name"]) : &mut Wrapper
    .check().unwrap();
```

Wrapper 包装器利用了 rust 泛型和 serde_json 提供的序列化实现，用于不想编写纯 SQL 的开发者们。并且 Wrapper 不使用宏系统，所以是完整支持 IDE 智能提示的！

运算解释器 - 实现细节披露



运算解释器是针对 json 对象运算的解释器，除了加减乘除外，还支持字符串拼接，括号，数组访问， json 对象属性访问，对比操作 ... 等等，实现细节如下

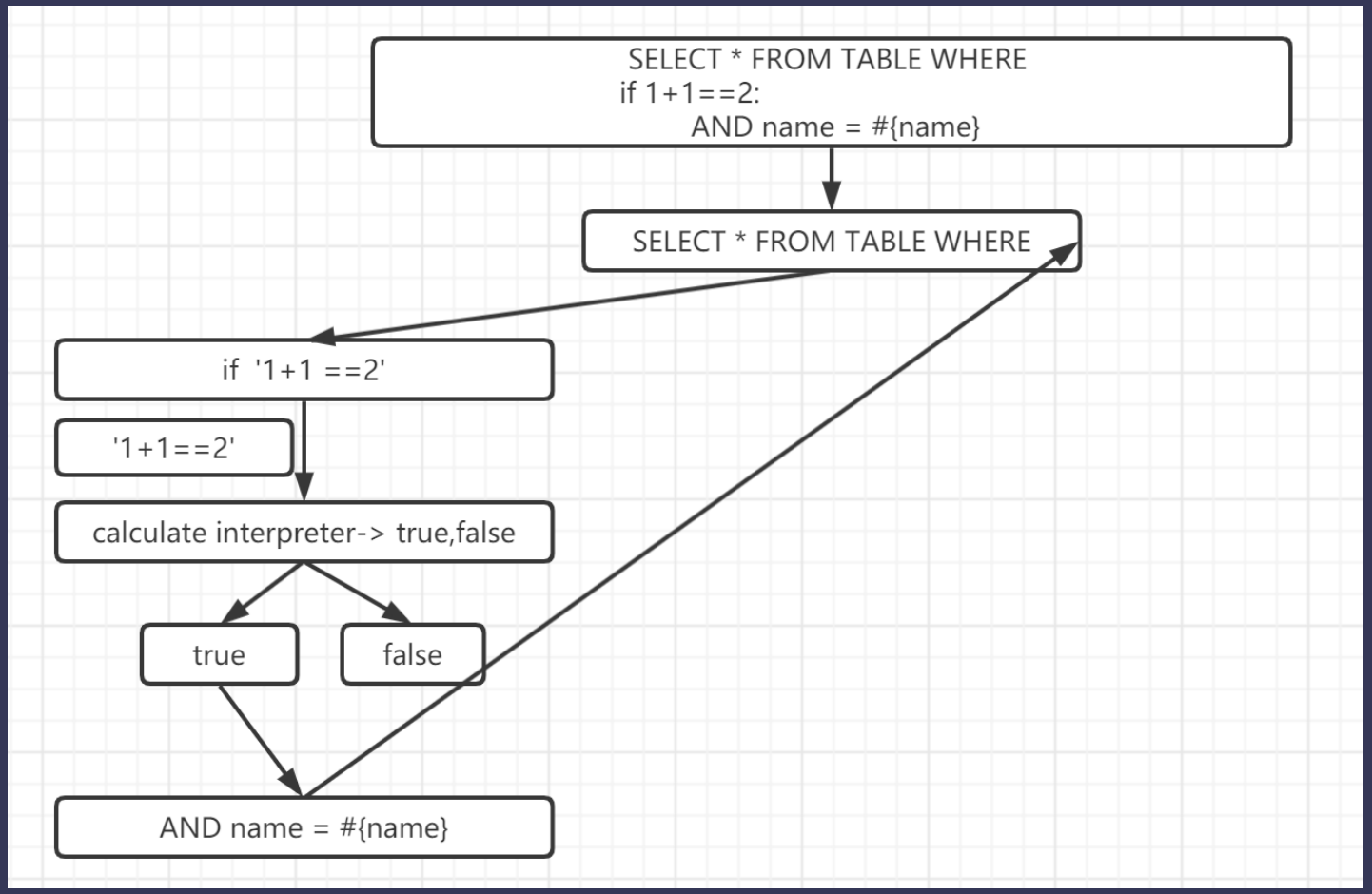
Lexer 阶段，例如 文本 ‘1+1’ 转变为字符串数组 ['1','+','1']

Parse 阶段， ['1','+','1'] 字符串数组，转变为抽象语法树（实际就是二叉树）

Eval 执行阶段，从根结点（root）开始，根据自身的类型运算左右 2 边叶子节点结果，返回数据。

- 当然框架会在首次执行的时候缓存编译结果，第二次执行只需读取编译缓存执行

Py-SQL 解释器 - 实现细节披露



- Py-SQL 语法是建立在数组结构之上的，实现内容如图
  1. Lexer 阶段，例如 文本转变为字符串数组
  2. Parse 阶段，字符串数组识别为 py 语法节点，提取预编译字段名称表达式，转变为抽象语法数组
  3. Eval 执行阶段，从数组第 0 位依次开始运算结果，计入 SQL 和参数进入 BtreeMap， 返回 SQL 和顺序参数。
- 当然框架会在首次执行的时候缓存编译结果，第二次执行只需读取编译缓存执行

分页插件配合过程宏 - 实现细节披露

```rust
#[py_sql(RB, "select * from biz_activity where delete_flag = 0
              if name != '':
                and name=#{name}")]
fn py_select_page(page_req: &PageRequest, name: &str) -> Page<BizActivity> {}

#[async_std::test]
pub async fn test_macro_py_select_page() {
    fast_log::init_log( log_file_path: "requests.log", channel_cup: 1000, level: log::Level::Info, filter: None, debug_mode: true);
    RB.link( url: "mysql://root:123456@localhost:3306/test").await.unwrap();
    let a = py_select_page( page_req: &PageRequest::new( current: 1, size: 10), name: "test").await.unwrap();
    println!("{:?}", a);
}
```

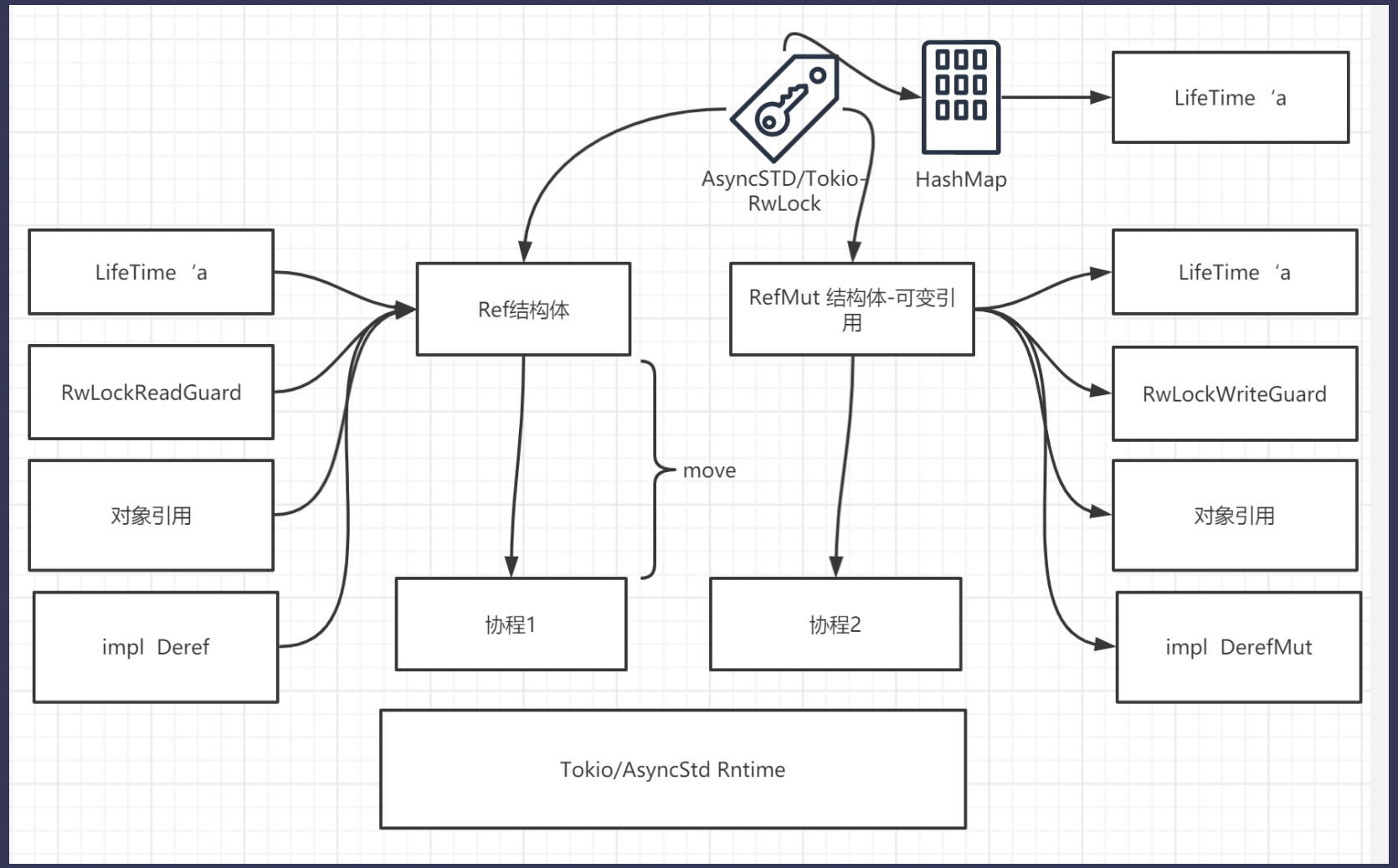
```rust
.....gen macro py_sql :
 pub async fn py_select_page(page_req : & PageRequest, name : & str) -> rbatis
:: core :: Result < Page < BizActivity > >
{
    let mut rb_args = serde_json :: Map :: new() ; rb_args .
    insert("page_req" . to_string(), serde_json :: to_value(page_req) .
           unwrap_or(serde_json :: Value :: Null)) ; rb_args .
    insert("name" . to_string(), serde_json :: to_value(name) .
           unwrap_or(serde_json :: Value :: Null)) ; let mut rb_args =
    serde_json :: Value :: from(rb_args) ; return RB .
    py_fetch_page("",
                  "select * from biz_activity where delete_flag = 0
                  if name != '':
                    and name=#{name}",
                  & rb_args, page_req) . await ;
}
.....gen macro py_sql end.....
```

过程宏会翻译标记实现为具体函数（如左图）
过程宏可识别 tx_id 事务号（未指定则空字符串）
过程宏可识别分页插件参数

分页插件的原理则是替换语句 中 * 为 count(1)，
同时剔除 order by,limit 等语句。

同时插件如果不满足你的需要，可以自定义自己 的
实现

跨多协程生命 + 协程共享安全 SyncMap- 实现细节披露



社区评价和反馈

RBatis is the most promising ORM for rust, more than sqlx, diesel or arysn

喜欢 py 语法和过程宏

ORM 这块，除了 Rbatis 比较好用，其他的缺点明显

- 社区评价绝大部分来自 JAVA 转 Rust 的开发者

谢谢

欢迎试用

RUST CHINA CONF 2020

首届中国 Rust 开发者大会

2020.12.26–27 深圳