

APIJSON 项目中期报告

邱俊霖

目录

1. 项目进度	2
2. APIJSON 学习	3
2.1 解析大体流程图	3
2.2 部分学习知识	3
3. ClickHouse 学习	5
3.1 Jdbc 测试	5
3.2 安装	6
3.3 数据类型	6
3.4 sql 语法	7
3.5 数据迁移	7
4. Clickhosue 接入	10
4.1 功能符测试	10
4.1.1 查询数组&分页	10
4.1.2 匹配选项范围	10
4.1.3 匹配条件范围	11
4.1.4 包含选项范围	11
4.1.5 判断是否存在	12
4.1.6 远程调用函数	12
4.1.7 存储过程(待测试)	12
4.1.8 引用赋值	12
4.1.9 子查询(待测试)	12
4.1.10 模糊搜索	12
4.1.11 正则匹配(需进一步测试)	13
4.1.12 连续范围(待测试)	13
4.1.13 新建别名	13
4.1.14 增加 或 扩展	13
4.1.15 减少 或 去除	13
4.1.16 比较运算	13
4.1.17 逻辑运算	13
4.1.18 join 关联查询(待测试)	14
4.1.19 对象关键词, 可自定义	14
4.1.20 order 排序查询(待测试)	14
4.1.21 group 分组查询(待测试)	14
4.1.21 sql 查看&性能分析	14
4.2 函数支持测试	14
4.2.1 日期函数	14
4.2.2 string 函数	17

4.2.3 json 函数（待加入）	21
4.2.4 数学函数.....	21
4.2.5 类型转换函数（待加入）	24
4.2.6 数组函数（待加入）	24
4.2.7 高阶函数（待加入）	24
4.2.8 ip/hash/url/uuid/GEO 函数（待加入）	24
4.2.9 其他函数（待加入）	24
5. 发现的问题.....	24
5.1 执行时 rsmd.getTableNames 返回“ ‘Comment’ ”	24
5.2 包含选项范围<> 方式解决	25
5.3 不支持关键字 exist （待解决）	25
5.5 Clickhouse 很多函数不支持	26
5.6 clickhouse 中文官方文档有误.....	26
5.7 遇见的一些 bugs.....	26
6 后期规划	26
6.1 完成报告中其余没有的测试功能，使接入没有问题	26
6.2 进行复杂查询的测试	26
6.3 完成所有函数的测试	26
7. 中期感悟	27

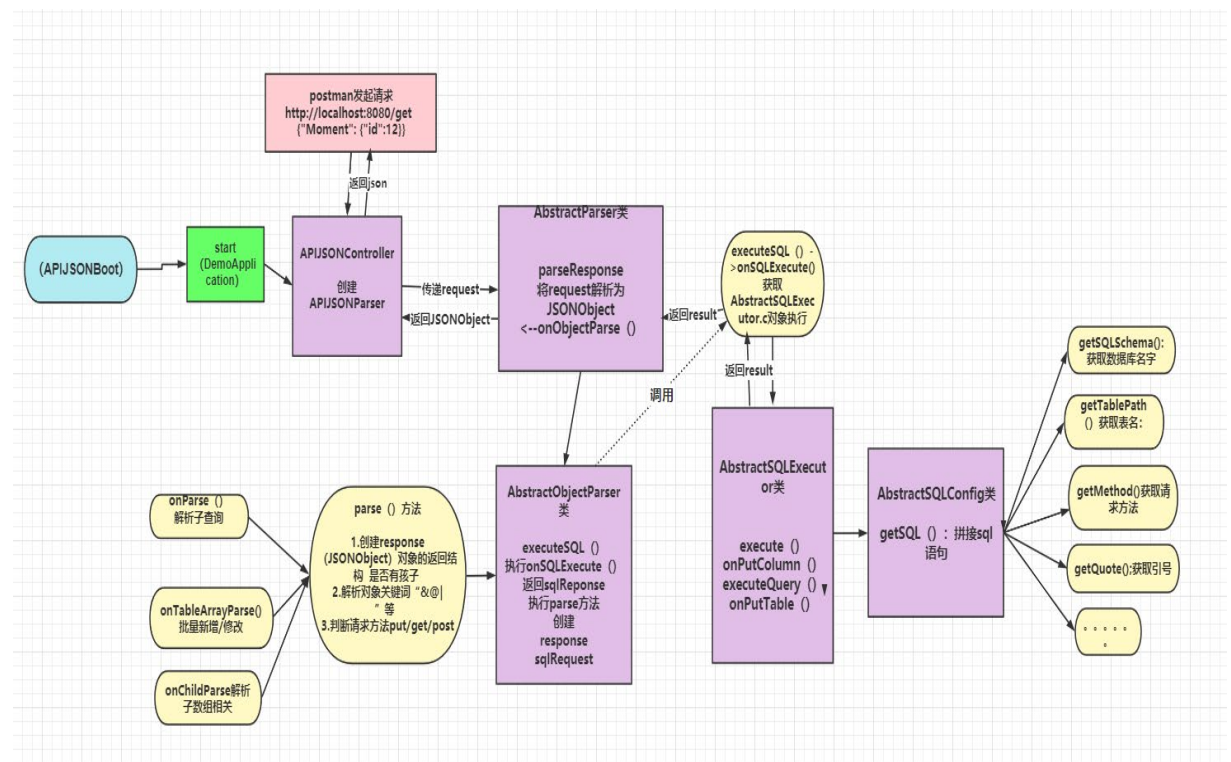
1. 项目进度

APIJSON 接入 clickhouse，目的，项目使用 clickhouse 数据库时候。保证使用 APIJSON 查询不会出错。

1. 学习项目，包括项目结构，代码
2. 学习 clickhouse
3. 配置 clickhouse 环境
4. 测试 apijson 中的所有功能是否在 clickhouse 中适用。已完成部分测试，具体已完成内容在后面详细描述

2. APIJSON 学习

2.1 解析大体流程图



2.2 部分学习知识

1. 学习 APIJSON 的请求语法格式

包括（字段过滤，字段别名，逻辑运算，模糊查询，正则匹配，列表查询，分页，排序，分组查询）响应格式。编写一些自定义请求测试，对请求的格式有了一个把握。

2. 学习源码部分：

为了更贴近生产环境，选择 APIJSONBOOT 这个 demo，在启动类里，这个 Demo 用了 apijson-framework，在项目启动了中调用了 APIJSONApplication.init。APIJSONApplication 这个类属于 apijson-framework 包，apijson-framework

是 APIJSON 的服务端框架。那么查看 APIJSONApplication 中的 init () 方法这个方法 是属于 APIJSON 的启动起点。

1. 构造器赋值。构造器在程序初始化的时候就已经创建，static 变量，APIJSONSQLConfig;APIJSONParser, APIJSONController;共用一个构造器，
2. 权限校验 APIJSONVerifier 重要 类，调用 initAccess () 进行权限校验
3. 远程函数配置 APIJSONFunctionParser.init()
4. 请求结构校验配置 APIJSONVerifier.initRequest()
5. Request 和 Response 的数据结构校验 APIJSONVerifier.testStructure();

3. get 请求进行关联查询， "/" 的用法

/ 开头的是缺省路径（相对路径），缺省了 / 之前的一段，直接从所处容器的父容器路径开始。

4. 关于远程函数：

"key() ":"函数表达式"，函数表达式为 function(key0, key1...), 会调用后端对应的函数 function(JSONObject request, String key0, String key1...), 实现 参数校验、数值计算、数据同步、消息推送、字段拼接、结构变换 等特定的业务逻辑处理，可使用 - 和 + 表示优先级，解析 key-() > 解析当前对象 > 解析 key() > 解析子对象 > 解析 key+()

5. 关于 get 和 gets

GETS 能使用的功能比较受限，例如不能查数组，查单条记录。而且必须传 tag, APIJSONORM 或根据 method, tag, version 去查 Request 表里的 structure, 强制校验请求参数。GET 就没有这些限制，也不需要传 tag, 不会经过 Request 表的配置来校验。但是 GET, GETS, HEAD, HEADS, POST, PUT, DELETE 所有方法都会用 Access 表的配置来校验权限，有权限的才允许使用。如果对安全性要求比较高，就不要开放 GET, HEAD, 只允许 GETS, HEADS, 参考 apijson_privacy 这张表的 Access 配置

6. 主要的类 AbstractPraserparse 解析请求 json 并获取对应结果

onVerifyRole 校验角色及对应操作的权限 extendSuccessResult 添加请求成功的状态内容 onObjectParse 解析 onArrayParse 获取对象数组，该对象数组处于 parentObject 内 onJoinParse 解析 joinexecuteSQL 行 SQL 并返回 JSONObject

7. AbstractSQLConfig

经过初步的尝试修改。发现 apijson 接入 clickhosue 主要的是修改 AbstractSQLConfig 中的代码。这个文件里面的方法很多，涵盖了生成 sql 的各个方面。无法一一去看。只有测试出了问题的时候再去排查。

3. ClickHouse 学习

3.1 Jdbc 测试

clickhouse 的 jdbc 驱动一共有三个：

分别是：

ClickHouse JDBC driver

```
1. <dependency>
2.     <groupId>ru.yandex.clickhouse</groupId>
3.     <artifactId>clickhouse-jdbc</artifactId>
4.     <version>0.3.1</version>
5. </dependency>
```

Clickhouse4j

lighter and faster alternative for the official ClickHouse JDBC driver

```
1. <!-- normal version -->
2. <dependency>
3.     <groupId>com.github.housepower</groupId>
4.     <artifactId>clickhouse-native-jdbc</artifactId>
5.     <version>${clickhouse-native-jdbc.version}</version>
6. </dependency>
```

3.2 安装

本次测试实验使用的是 Deb 安装包，在 ubuntu 环境下安装。

DEB 安装包_运行以下命令来安装包：

1. `sudo apt-get install apt-transport-https ca-certificates dirmngr`
2. `sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv E0C56BD4`
3. `echo "deb https://repo.clickhouse.tech/deb/stable/ main/" | sudo tee /etc/apt/sources.list.d/clickhouse.list`
4. `sudo apt-get update`
5. `sudo apt-get install -y clickhouse-server clickhouse-client`
6. `sudo service clickhouse-server start` 启动服务
7. `clickhouse-client` 启动客户端

3.3 数据类型

基础数据类型

MySQL	Hive	ClickHouse
byte	TINYINT	Int8
short	SMALLINT	Int16
int	INT	Int32
long	BIGINT	Int64
varchar	STRING	String
timestamp	TIMESTAMP	DateTime
float	FLOAT	Float32
double	DOUBLE	Float64
boolean	BOOLEAN	-

注意点：ClickHouse 没有 boolean 类型

高级数据类型

数据类型	含义	特点
inf	正无穷大	正数/0时获得，分母为0就认为是无穷大
nan	正无穷小	0/0时获得，分子分母都为0得到的就是无穷小
Enum8	枚举TINYINT	比如:Enum8('true'=1,'false'=0)
Enum16	枚举SMALLINT	比如:Enum16('true'=1,'false'=0)
Array(T)	数组	由T类型元素组成的数组
tuple()	元组	存进去的数据每个都有自己的数据类型

3.4 sql 语法

和标注的 mysql 差距不大

1. [WITH expr_list | (subquery)]
2. SELECT [DISTINCT] expr_list
3. [FROM [db.]table | (subquery) | table_function] [FINAL]
4. [SAMPLE sample_coeff]
5. [ARRAY JOIN ...]
6. [GLOBAL] [ANY|ALL|ASOF] [INNER|LEFT|RIGHT|FULL|CROSS] [OUTER|SEMI|ANTI] JOIN (subquery) | table (ON <expr_list>) | (USING <column_list>)
7. [PREWHERE expr]
8. [WHERE expr]
9. [GROUP BY expr_list] [WITH TOTALS]
10. [HAVING expr]
11. [ORDER BY expr_list] [WITH FILL] [FROM expr] [TO expr] [STEP expr]
12. [LIMIT [offset_value,]n BY columns]
13. [LIMIT [n,]m] [WITH TIES]
14. [UNION ALL ...]
15. [INTO OUTFILE filename]
16. [FORMAT format]

3.5 数据迁移

目前没有找到一个好的办法。所以直接在 clickhouse 中使用 sql 语句来讲本地 mysql 中的数据移动到 clickhouse 中。


建库：

```
create database test;  
use test;
```

导入数据:

```
insert into ck_db_name.ck_tb_name (字段1, 字段2, ...)  
SELECT 字段1, 字段2, ...  
FROM mysql('host:port', 'database_name', 'table_name',  
           'user_name', 'passport')  
  
eg:  
CREATE TABLE dg_sku_inventory_day ENGINE = MergeTree ORDER BY id AS SELECT * FROM  
mysql('192.168.1.20:3306', 'dgdata', 'dg_sku_inventory_day', 'root', '123456');前提条件:  
MySQL的主键必须为not null  
192.168.1.20:3306 为MySQL的IP地址和端口号  
dgdata 为MySQL的库, 作为数据源的  
dg_sku_inventory_day 为MySQL的表, 源表  
root 为MySQL的账号  
123456 为MySQL的账号对应的密码 账号需要具备远程连接访问的权限。
```

使用语句:



```
CREATE TABLE apijson_privacy ENGINE = MergeTree ORDER BY id AS SELECT * FROM
mysql('10.20.253.73:3306', 'sys', 'apijson_privacy', 'root', '123456');

CREATE TABLE apijson_user ENGINE = MergeTree ORDER BY id AS SELECT * FROM
mysql('10.20.253.73:3306', 'sys', 'apijson_user', 'root', '123456');

CREATE TABLE Comment ENGINE = MergeTree ORDER BY id AS SELECT * FROM
mysql('10.20.253.73:3306', 'sys', 'comment', 'root', '123456');

CREATE TABLE Document ENGINE = MergeTree ORDER BY id AS SELECT * FROM
mysql('10.20.253.73:3306', 'sys', 'document', 'root', '123456');

CREATE TABLE Function ENGINE = MergeTree ORDER BY id AS SELECT * FROM
mysql('10.20.253.73:3306', 'sys', 'function', 'root', '123456');

CREATE TABLE Request ENGINE = MergeTree ORDER BY id AS SELECT * FROM
mysql('10.20.253.73:3306', 'sys', 'request', 'root', '123456');

CREATE TABLE Response ENGINE = MergeTree ORDER BY id AS SELECT * FROM
mysql('10.20.253.73:3306', 'sys', 'response', 'root', '123456');

CREATE TABLE Moment ENGINE = MergeTree ORDER BY id AS SELECT * FROM
mysql('10.20.253.73:3306', 'sys', 'moment', 'root', '123456');

CREATE TABLE Method ENGINE = MergeTree ORDER BY id AS SELECT * FROM
mysql('10.20.253.73:3306', 'sys', 'method', 'root', '123456');

CREATE TABLE Login ENGINE = MergeTree ORDER BY id AS SELECT * FROM
mysql('10.20.253.73:3306', 'sys', 'login', 'root', '123456');

CREATE TABLE Verify ENGINE = MergeTree ORDER BY id AS SELECT * FROM
mysql('10.20.253.73:3306', 'sys', 'verify', 'root', '123456');

CREATE TABLE Random ENGINE = MergeTree ORDER BY id AS SELECT * FROM
mysql('10.20.253.73:3306', 'sys', 'random', 'root', '123456');
```

4. Clickhosue 接入

4.1 功能符测试

4.1.1 查询数组&分页

```
{
  "User[]": {
    "page": 0,
    "count": 1,
    "User": {
      "sex": 0
    }
  }
}
```

生成 Sql:

```
SELECT * FROM `test`.`apijson_user` WHERE ( (`sex` = 0) ) LIMIT 1
```

4.1.2. 匹配选项范围

```
{
  "User[]": {
    "count": 3,
    "User": {
      "id{}": [
        38710,
        82001,
        70793
      ]
    }
  }
}
```

Sql : SELECT * FROM `test`.`apijson_user` WHERE ((`id` IN (38710,82001,70793))) LIMIT 3

4.1.3. 匹配条件范围

```
{
  "User[]": {
    "count": 3,
    "User": {
      "id{}": "<=80000,>90000"
    }
  }
}
```

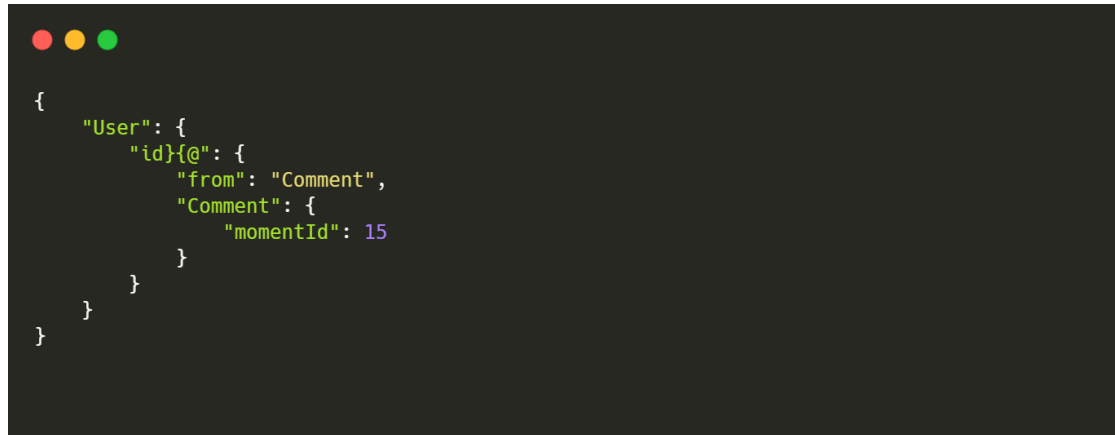
```
SELECT * FROM `test`.`apijson_user` WHERE ( (`id` <=80000 OR `id` >90000) )
LIMIT 3
```

4.1.4 包含选项范围

```
{
  "User[]": {
    "count": 3,
    "User": {
      "contactIdList<=": 38710
    }
  }
}
```

```
SELECT * FROM `test`.`apijson_user` WHERE ( (`contactIdList` is NOT null
AND (has(JSONExtractArrayRaw(assumeNotNull(`contactIdList`)), '38710')) ) )
LIMIT 3
```

4.1.5 判断是否存在



Sql:

```
SELECT * FROM `test`.`apijson_user` WHERE ( ( EXISTS (SELECT * FROM  
`test`.`Comment` WHERE ( (`momentId` = 15) ) ) ) ) LIMIT 1
```

查询结果出错，加入需解决问题。

4.1.6 远程调用函数

4.1.7 存储过程(待测试)

4.1.8 引用赋值

经过测试，没有问题，都可以为引用赋值

4.1.9 子查询(待测试)

4.1.10 模糊搜索

4. 1. 11 正则匹配(需进一步测试)

4. 1. 12 连续范围(待测试)

4. 1. 13 新建别名

支持

4. 1. 14 增加 或 扩展

支持

4. 1. 15 减少 或 去除

支持

4. 1. 16 比较运算

支持

4. 1. 17 逻辑运算

支持

4.1.18 join 关联查询(待测试)

4.1.19 对象关键词，可自定义

4.1.20 order 排序查询(待测试)

4.1.21 group 分组查询(待测试)

4.1.21 sql 查看&性能分析

clickhosue 不支持 explain 关键字

4.2 函数支持测试

4.2.1 日期函数

在如何测试函数的问题上的问题上，我先选择对日期函数进行测试。
clickhosue 对日期时间处理函数的支持，相对于 mysql

mysql 函数	clickho use 测 试	修改方案	修 改 后 测 试
adddate	✗	addDays,addYears, addMonths, addWeeks, addDays, addHours, addMinutes, addSeconds,addQuarters	✓
adtime	✗		
curdate	✗	today	no w
current_date	✗	today	
current_time	✗		
current_time stamp	✗		

mysql 函数	clickhouse 测试	修改方案	修改后测试
curtime	✗		
date	✓		
datediff	✗	dateDiff	
date_add	✗		
date_format	✗	formatDateTime	
date_sub	✗	subtractYears, subtractMonths, subtractWeeks, subtractDays, subtractHours, subtractMinutes, subtractSeconds, subtractQuarters	
day	✓		
dayname	✗		
dayofmonth	✗	toDayOfMonth	
dayofweek	✗	toDayOfWeek	
dayofyear	✗	toDayOfYear	
extract	✓	支持，内部会转换为 clickhouse 对应的 toXX	
from_days	✗		
hour	✓		
last_day	✗		
localtime	✗		
localtimestamp	✗		
makedate	✗		
maketime	✗		
microsecond	✗		
minute	✓		

mysql 函数	clickhouse 测试	修改方案	修改后测试
monthname	✗		
month	✓		
now	✓		
period_add	✗		
period_diff	✗		
quarter	✓		
second	✓		
sec_to_time	✗		
str_to_date	✗		
subdate	✗		
subtime	✗		
sysdate	✗		
time	✗		
time_format	✗	formatDateTime	✓
time_to_sec	✗		
timediff	✗	dateDiff	✓
timestamp	✗		
to_days	✗		
week	✓		
weekday	✗	toDayOfWeek	✓
weekofyear	✗		

mysql 函数	clickhouse 测试	修改方案	修改后测试
year	✓		
yearweek	✗		
unix_timestamp	✗		
from_unixtime	✗		

经过第一次测试，我发现 mysql 中支持的函数 clickhouse 很多都不支持，而 clickhouse 中支持的函数在 mysql 中都不支持。于是测试了其他类型的函数，发现也是这样。于是我决定放弃这样的测试方式。

想了两种解决方案：

1. 把所有 clickhouse 中的函数加入 SQL_FUNCTION_MAP 中
2. 新添加一个 map，在 map 中加入 clickhouse 的函数，在获取 sql 语句的时候进行这样判断 (isClickHouse() && map.contains(fun))

和老师讨论了一下。决定把所有 clickhouse 中的函数加入 SQL_FUNCTION_MAP 中，因为这样的方式省去了代码的复杂性，同时 clickhouse 的函数也就几百个，加入对 apijson 的性能几乎没有什么影响。

4.2.2 string 函数

翻译的英文文档，并将函数加入到 apijson 中。

经过上一轮的测试，整理了所有 clickhouse 中相关的函数。并在查询中通过

1. SQL_FUNCTION_MAP.put("empty", ""); // empty(s) 对于空字符串 s 返回 1, 对于非空字符串返回 0
2. SQL_FUNCTION_MAP.put("notEmpty", ""); //notEmpty(s) 对于空字符串返回 0, 对于非空字符串返回 1。
3. SQL_FUNCTION_MAP.put("lengthUTF8", ""); //假定字符串以 UTF-8 编码组成的文本, 返回此字符串的 Unicode 字符长度。如果传入的字符串不是 UTF-8 编码, 则函数可能返回一个预期外的值
4. SQL_FUNCTION_MAP.put("lcase", ""); //将字符串中的 ASCII 转换为小写
5. SQL_FUNCTION_MAP.put("ucase", ""); //将字符串中的 ASCII 转换为大写。
6. SQL_FUNCTION_MAP.put("lowerUTF8", ""); //将字符串转换为小写, 函数假设字符串是以 UTF-8 编码文本的字符集。
7. SQL_FUNCTION_MAP.put("upperUTF8", ""); //将字符串转换为大写, 函数假设字符串是以 UTF-8 编码文本的字符集。
8. SQL_FUNCTION_MAP.put("isValidUTF8", ""); // 检查字符串是否为有效的 UTF-8 编码, 是则返回 1, 否则返回 0。
9. SQL_FUNCTION_MAP.put("toValidUTF8", ""); //用 (U+FFFD) 字符替换无效的 UTF-8 字符。所有连续的无效字符都会被替换为一个替换字符。
10. SQL_FUNCTION_MAP.put("concatAssumeInjective", ""); // concatAssumeInjective(s1, s2, ...) 与 concat 相同, 区别在于, 你需要保证 concat(s1, s2, s3) → s4 是单射的, 它将用于 GROUP BY 的优化。
11. SQL_FUNCTION_MAP.put("substringUTF8", ""); // substringUTF8(s, offset, length) 与 'substring' 相同, 但其操作单位为 Unicode 字符, 函数假设字符串是以 UTF-8 进行编码的文本。如果不是则可能返回一个预期外的结果 (不会抛出异常)。
12. SQL_FUNCTION_MAP.put("appendTrailingCharIfAbsent", ""); // appendTrailingCharIfAbsent(s, c) 如果 's' 字符串非空并且末尾不包含 'c' 字符, 则将 'c' 字符附加到末尾
13. SQL_FUNCTION_MAP.put("convertCharset", ""); // convertCharset(s, from, to) 返回从 'from' 中的编码转换为 'to' 中的编码的字符串 's'。
14. SQL_FUNCTION_MAP.put("base64Encode", ""); // base64Encode(s) 将字符串 's' 编码成 base64
15. SQL_FUNCTION_MAP.put("base64Decode", ""); //base64Decode(s) 使用 base64 将字符串解码成原始字符串。如果失败则抛出异常。
16. SQL_FUNCTION_MAP.put("tryBase64Decode", ""); //tryBase64Decode(s) 使用 base64 将字符串解码成原始字符串。但如果出现错误, 将返回空字符串。
17. SQL_FUNCTION_MAP.put("endsWith", ""); //endsWith(s, 后缀) 返回是否以指定的后缀结尾。如果字符串以指定的后缀结束, 则返回 1, 否则返回 0。
18. SQL_FUNCTION_MAP.put("startsWith", ""); //startsWith(s, 前缀) 返回是否以指定的前缀开头。如果字符串以指定的前缀开头, 则返回 1, 否则返回 0。
19. SQL_FUNCTION_MAP.put("trimLeft", ""); //trimLeft(s) 返回一个字符串, 用于删除左侧的空白字符。
20. SQL_FUNCTION_MAP.put("trimRight", ""); //trimRight(s) 返回一个字符串, 用于删除右侧的空白字符。
21. SQL_FUNCTION_MAP.put("trimBoth", ""); //trimBoth(s), 用于删除任一侧的空白字符
22. SQL_FUNCTION_MAP.put("splitByChar", ""); //splitByChar (分隔符, s) 将字符串以 'separator' 拆分成多个子串。'separator' 必须为仅包含一个字符的字符串常量。

23. SQL_FUNCTION_MAP.put("splitByString", ""); // splitByString(分隔符, s) 与上面相同, 但它使用多个字符的字符串作为分隔符。该字符串必须为非空。
24. SQL_FUNCTION_MAP.put("arrayStringConcat", ""); //arrayStringConcat(arr[, 分隔符])使用 separator 将数组中列出的字符串拼接起来。' separator' 是一个可选参数: 一个常量字符串, 默认情况下设置为空字符串。
25. SQL_FUNCTION_MAP.put("alphaTokens", ""); //alphaTokens(s) 从范围 a-z 和 A-Z 中选择连续字节的子字符串。返回子字符串数组。
26. SQL_FUNCTION_MAP.put("splitByRegexp", ""); //splitByRegexp(regexp, s) 将字符串分割为由正则表达式分隔的子字符串。它使用正则表达式字符串 regexp 作为分隔符。
27. SQL_FUNCTION_MAP.put("splitByWhitespace", ""); //splitByWhitespace(s) 将字符串分割为由空格字符分隔的子字符串。返回选定子字符串的数组
28. SQL_FUNCTION_MAP.put("splitByNonAlpha", ""); //splitByNonAlpha(s) 将字符串分割为由空格和标点字符分隔的子字符串
29. SQL_FUNCTION_MAP.put("extractAllGroups", ""); //extractAllGroups(text, regexp) 从正则表达式匹配的非重叠子字符串中提取所有组
30. SQL_FUNCTION_MAP.put("leftPad", ""); //leftPad('string', 'length'[, 'pad_string']) 用空格或指定的字符串从左边填充当前字符串(如果需要, 可以多次), 直到得到的字符串达到给定的长度
31. SQL_FUNCTION_MAP.put("leftPadUTF8", ""); //leftPadUTF8('string', 'length'[, 'pad_string']) 用空格或指定的字符串从左边填充当前字符串(如果需要, 可以多次), 直到得到的字符串达到给定的长度
32. SQL_FUNCTION_MAP.put("rightPad", ""); // rightPad('string', 'length'[, 'pad_string']) 用空格或指定的字符串(如果需要, 可以多次)从右边填充当前字符串, 直到得到的字符串达到给定的长度
33. SQL_FUNCTION_MAP.put("rightPadUTF8", ""); // rightPadUTF8('string', 'length'[, 'pad_string']) 用空格或指定的字符串(如果需要, 可以多次)从右边填充当前字符串, 直到得到的字符串达到给定的长度。
34. SQL_FUNCTION_MAP.put("CRC32", ""); // 使用 CRC-32-IEEE 802.3 多项式和初始值 0xffffffff (zlib 实现)返回字符串的 CRC32 校验和
35. SQL_FUNCTION_MAP.put("CRC32IEEE", ""); // 使用 CRC-32-IEEE 802.3 多项式返回字符串的 CRC32 校验和
36. SQL_FUNCTION_MAP.put("CRC64", ""); // 使用 CRC-64-ECMA 多项式返回字符串的 CRC64 校验和
37. SQL_FUNCTION_MAP.put("normalizeQuery", ""); //normalizeQuery(x) 用占位符替换文字、文字序列和复杂的别名。
38. SQL_FUNCTION_MAP.put("normalizedQueryHash", ""); //normalizedQueryHash(x) 为类似查询返回相同的 64 位散列值, 但不包含文字值。有助于对查询日志进行分析
39. SQL_FUNCTION_MAP.put("encodeXMLComponent", ""); //encodeXMLComponent(x) 转义字符以将字符串放入 XML 文本节点或属性中
40. SQL_FUNCTION_MAP.put("decodeXMLComponent", ""); // 用字符替换 XML 预定义的实体
41. SQL_FUNCTION_MAP.put("extractTextFromHTML", ""); //extractTextFromHTML(X) 从 HTML 或 XHTML 中提取文本的函数

```

42. SQL_FUNCTION_MAP.put("positionCaseInsensitive", ""); //positionCaseInsensitive(s,
    needle[, start_pos])与 position 相同, 返回在字符串中找到的子字符串的位置(以字节为
    单位), 从 1 开始。使用该函数进行不区分大小写的搜索。
43. SQL_FUNCTION_MAP.put("positionUTF8", ""); // positionUTF8(s, needle[, start_pos
    ]) 返回在字符串中找到的子字符串的位置(以 Unicode 点表示), 从 1 开始。
44. SQL_FUNCTION_MAP.put("positionCaseInsensitiveUTF8", "");positionCaseInsensitiveUTF8
    (s, needle[, start_pos]) 与 positionUTF8 相同, 但不区分大小写。返回在字符串中找到
    的子字符串的位置(以 Unicode 点表示), 从 1 开始
45. SQL_FUNCTION_MAP.put("multiSearchAllPositions", ""); // 与 position 相同, 但返回字
    符串中找到的相应子字符串的位置数组(以字节为单位)。位置从 1 开始索引。
46. SQL_FUNCTION_MAP.put("multiSearchAllPositionsUTF8", ""); //See multiSearchAllPo
    sitions.
47. SQL_FUNCTION_MAP.put("multiSearchFirstPosition", ""); // multiSearchFirstPosition
    (s, [needle1, needle2, ..., needlen])
48. SQL_FUNCTION_MAP.put("multiSearchFirstIndex", ""); //multiSearchFirstIndex(s, [ne
    edle1, needle2, ..., needlen]) 返回字符串 s 中最左边的 needlei 的索引 i(从 1 开始),
    否则返回 0
49. SQL_FUNCTION_MAP.put("multiSearchAny", ""); // multiSearchAny(s, [needle1, need
    le2, ..., needlen])如果至少有一个字符串 needlei 匹配字符串 s, 则返回 1, 否则返回
    0。
50. SQL_FUNCTION_MAP.put("match", ""); //match(s, pattern) 检查字符串是否与模式正则
    表达式匹配。re2 正则表达式。re2 正则表达式的语法比 Perl 正则表达式的语法更有局限
    性。
51. SQL_FUNCTION_MAP.put("multiMatchAny", ""); //multiMatchAny(s, [pattern1, patte
    rn2, ..., patternn]) 与 match 相同, 但是如果没有匹配的正则表达式返回 0, 如果有匹配
    的模式返回 1
52. SQL_FUNCTION_MAP.put("multiMatchAnyIndex", ""); //multiMatchAnyIndex(s, [pattern1
    , pattern2, ..., patternn]) 与 multiMatchAny 相同, 但返回与干堆匹配的任何索引
53. SQL_FUNCTION_MAP.put("multiMatchAllIndices", ""); //multiMatchAllIndices(s, [patt
    ern1, pattern2, ..., patternn]) 与 multiMatchAny 相同, 但返回以任意顺序匹配干堆
    的所有索引的数组。
54. SQL_FUNCTION_MAP.put("multiFuzzyMatchAny", ""); //multiFuzzyMatchAny(s, distance,
    [pattern1, pattern2, ..., patternn])与 multimchany 相同, 但如果在固定的编辑距离内
    有匹配的模式则返回 1
55. SQL_FUNCTION_MAP.put("multiFuzzyMatchAnyIndex", ""); //multiFuzzyMatchAnyIndex(s,
    distance, [pattern1, pattern2, ..., patternn])与 multiFuzzyMatchAny 相同, 但返回
    在固定编辑距离内匹配干草堆的任何索引。
56. SQL_FUNCTION_MAP.put("multiFuzzyMatchAllIndices", ""); // multiFuzzyMatchAllInd
    ices(s, distance, [pattern1, pattern2, ..., patternn])
57. SQL_FUNCTION_MAP.put("extract", ""); // extract(s, pattern) 使用正则表达式
    提取字符串的片段
58. SQL_FUNCTION_MAP.put("extractAll", ""); //extractAll(s, pattern) 使用正则表达式
    提取字符串的所有片段

```

```

59. SQL_FUNCTION_MAP.put("extractAllGroupsHorizontal", ""); //extractAllGroupsHorizontal(s, pattern)
60. 使用模式正则表达式匹配 s 字符串的所有组
61. SQL_FUNCTION_MAP.put("extractAllGroupsVertical", ""); //extractAllGroupsVertical(s, pattern) 使用模式正则表达式匹配 s 字符串的所有组
62. SQL_FUNCTION_MAP.put("like", ""); //like(s, pattern) 检查字符串是否与简单正则表达式匹配
63. SQL_FUNCTION_MAP.put("notLike", ""); //和 'like' 是一样的, 但是是否定的
64. SQL_FUNCTION_MAP.put("countSubstrings", ""); //countSubstrings(s, needle[, start_pos]) 返回子字符串出现的次数
65. SQL_FUNCTION_MAP.put("countSubstringsCaseInsensitive", ""); //countSubstringsCaseInsensitive(s, needle[, start_pos])
66. 返回不区分大小写的子字符串出现次数。
67. SQL_FUNCTION_MAP.put("countMatches", ""); //返回干 s 中的正则表达式匹配数。countMatches(s, pattern)
68. SQL_FUNCTION_MAP.put("replaceOne", ""); //replaceOne(s, pattern, replacement) 将 's' 中的 'pattern' 子串的第一个出现替换为 'replacement' 子串。
69.
70. SQL_FUNCTION_MAP.put("replaceAll", ""); //replaceAll(s, pattern, replacement) 用 'replacement' 子串替换 's' 中所有出现的 'pattern' 子串
71. SQL_FUNCTION_MAP.put("replaceRegexpOne", ""); //replaceRegexpOne(s, pattern, replacement) 使用 'pattern' 正则表达式进行替换
72. SQL_FUNCTION_MAP.put("replaceRegexpAll", ""); //replaceRegexpAll(s, pattern, replacement)
73. SQL_FUNCTION_MAP.put("regexpQuoteMeta", ""); //regexpQuoteMeta(s) 该函数在字符串中某些预定义字符之前添加一个反斜杠

```

4.2.3 json 函数（待加入）

4.2.4 数学函数

Clickhosue 数学函数

```

1. SQL_FUNCTION_MAP.put("plus", ""); //plus(a, b), a + b operator 计算数值的总和。
2. SQL_FUNCTION_MAP.put("minus", ""); //minus(a, b), a - b operator 计算数值之间的差, 结果总是有符号的。

```

3. SQL_FUNCTION_MAP.put("multiply", ""); //multiply(a, b), a * b operator 计算数值的乘积
4. SQL_FUNCTION_MAP.put("divide", ""); //divide(a, b), a / b operator 计算数值的商。结果类型始终是浮点类型
5. SQL_FUNCTION_MAP.put("intDiv", ""); //intDiv(a,b)计算数值的商，向下舍入取整（按绝对值）。
6. SQL_FUNCTION_MAP.put("intDivOrZero", ""); // intDivOrZero(a,b)与' intDiv' 的不同之处在于它在除以零或将最小负数除以-1时返回零。
7. SQL_FUNCTION_MAP.put("modulo", ""); //modulo(a, b), a % b operator 计算除法后的余数。
8. SQL_FUNCTION_MAP.put("moduloOrZero", ""); //和 modulo 不同之处在于，除以 0 时结果返回 0
9. SQL_FUNCTION_MAP.put("negate", ""); //通过改变数值的符号位对数值取反，结果总是有符号
10. SQL_FUNCTION_MAP.put("gcd", ""); //gcd(a,b) 返回数值的最大公约数。
11. SQL_FUNCTION_MAP.put("lcm", ""); //lcm(a,b) 返回数值的最小公倍数
12. SQL_FUNCTION_MAP.put("e", ""); //e() 返回一个接近数学常量 e 的 Float64 数字。
13. SQL_FUNCTION_MAP.put("pi", ""); //pi() 返回一个接近数学常量 π 的 Float64 数字。
14. SQL_FUNCTION_MAP.put("exp2", ""); //exp2(x)接受一个数值类型的参数并返回它的 2 的 x 次幂。
15. SQL_FUNCTION_MAP.put("exp10", ""); //exp10(x)接受一个数值类型的参数并返回它的 10 的 x 次幂。
16. SQL_FUNCTION_MAP.put("cbrt", ""); //cbrt(x) 接受一个数值类型的参数并返回它的立方根。
17. SQL_FUNCTION_MAP.put("erf", ""); //erf(x) 如果 'x' 是非负数，那么 $\text{erf}(x / \sigma \sqrt{2})$ 是具有正态分布且标准偏差为 σ 的随机变量的值与预期值之间的距离大于 $\langle x \rangle$
18. SQL_FUNCTION_MAP.put("erfc", ""); //erfc(x) 接受一个数值参数并返回一个接近 $1 - \text{erf}(x)$ 的 Float64 数字，但不会丢失大 $\langle x \rangle$ 值的精度。
19. SQL_FUNCTION_MAP.put("lgamma", ""); //lgamma(x) 返回 x 的绝对值的自然对数的伽玛函数。
20. SQL_FUNCTION_MAP.put("tgamma", ""); //tgamma(x)返回 x 的伽玛函数。
21. SQL_FUNCTION_MAP.put("intExp2", ""); //intExp2 接受一个数值类型的参数并返回它的 2 的 x 次幂 (UInt64)
22. SQL_FUNCTION_MAP.put("intExp10", ""); //intExp10 接受一个数值类型的参数并返回它的 10 的 x 次幂 (UInt64)。
23. SQL_FUNCTION_MAP.put("cosh", ""); // cosh(x)
24. SQL_FUNCTION_MAP.put("sinh", ""); //sinh(x)
25. SQL_FUNCTION_MAP.put("asinh", ""); //asinh(x)
26. SQL_FUNCTION_MAP.put("atanh", ""); //atanh(x)
27. SQL_FUNCTION_MAP.put("atan2", ""); //atan2(y, x)
28. SQL_FUNCTION_MAP.put("hypot", ""); //hypot(x, y)

```
30. SQL_FUNCTION_MAP.put("log1p", ""); //log1p(x)
31. SQL_FUNCTION_MAP.put("trunc", ""); //和 truncate 一样
32. SQL_FUNCTION_MAP.put("roundToExp2", ""); //接受一个数字。如果数字小于 1，它返回
    0。
33. SQL_FUNCTION_MAP.put("roundDuration", ""); //接受一个数字。如果数字小于 1，它返回
    0。
34. SQL_FUNCTION_MAP.put("roundAge", ""); // 接受一个数字。如果数字小于 18，它返回
    0。
35. SQL_FUNCTION_MAP.put("roundDown", ""); //接受一个数字并将其舍入到指定数组中的一个
    元素
36. SQL_FUNCTION_MAP.put("bitAnd", ""); //bitAnd(a,b)
37. SQL_FUNCTION_MAP.put("bitOr", ""); //bitOr(a,b)
38. SQL_FUNCTION_MAP.put("abs", ""); //bitXor(a,b)
39. SQL_FUNCTION_MAP.put("abs", ""); //bitNot(a)
40. SQL_FUNCTION_MAP.put("abs", ""); //bitShiftLeft(a,b)
41. SQL_FUNCTION_MAP.put("abs", ""); //bitShiftRight(a,b)
42. SQL_FUNCTION_MAP.put("abs", ""); //bitRotateLeft(a,b)
43. SQL_FUNCTION_MAP.put("abs", ""); //bitRotateRight(a,b)
44. SQL_FUNCTION_MAP.put("abs", ""); //bitTest(a,b)
45. SQL_FUNCTION_MAP.put("abs", ""); //bitTestAll(a,b)
46. SQL_FUNCTION_MAP.put("abs", ""); //bitTestAny(a,b)
47. SQL_FUNCTION_MAP.put("abs", ""); //bitCount(a)
48. SQL_FUNCTION_MAP.put("abs", ""); //bitHammingDistance(a,b)
```

4.2.5 类型转换函数（待加入）

4.2.6 数组函数（待加入）

4.2.7 高阶函数（待加入）

4.2.8 ip/hash/url/uuid/GEO 函数（待加入）

4.2.9 其他函数（待加入）

5. 发现的问题

5.1 执行时 rsmd.getTableName 返回” ‘Comment’ ”

经过验证和测试。使用 clickhouse4j 执行 rsmd.getTableName 返回” ‘Comment’ ”，有反引号

而使用官方驱动 clickhouse-jdbc 返回的是 "Comment"，没有反引号，所以这个问题，如果在 getQuote() 中去掉`"，那么 clickhouse 生成的 sql 语句都没有反引号。生成的 sql 保留反引号，在进行判断的时候如果使用的 jdbc 驱动是 clickhouse4j，获取的表名是有反引号的。将它截取下来再比较。修改方法：

```
1. String sqlTable = rsmd.getTableName(i);
2. if (config.isClickHouse() && sqlTable.startsWith("`")) {
3.     sqlTable = sqlTable.substring(1, sqlTable.length() - 1);
4. }
5. ...
6. } else if ( ! config.getSQLTable().equalsIgnoreCase(sqlTable)) {
```

官方文档有一句话：

关键字不是保留的；它们仅在相应的上下文中才会被认为是关键字。如果

你使用和关键字同名的 标识符 ，需要使用双引号或反引号将它们包含起来。例如：如果表 table_name 包含列 "FROM"，那么 SELECT "FROM" FROM table_name 是合法的

说明 clickhouse 是支持表同名的。并且必须要使用反引号或者双引号。

5.2 包含选项范围<> 方式解决

包含选项范围**请求：

```
1.  {
2.     "User[]": {
3.         "count": 3,
4.         "User": {
5.             "contactIdList<>": 38710
6.         }
7.     }
8. }
```

解析后的 sql 语句为：SELECT * FROM `sys`.`apijson_user` WHERE ((`contactIdList` is NOT null AND (json_contains(`contactIdList`, '38710'))))) LIMIT 3

执行报错 没有对应函数，修改为 clickhouse 中相应函数，查询不出来

SELECT * FROM `test`.`apijson_user` WHERE ((`contactIdList` is NOT null AND (JSONHas(`contactIdList`, '38710'))))) LIMIT 3

SELECT * FROM `test`.`apijson_user` WHERE ((`contactIdList` is NOT null AND (visitParamHas(`contactIdList`, '38710'))))) LIMIT 3

解决办法 参考另外一个同学的解决办法

https://github.com/chenyannlann/APIJSONDemo_ClickHouse/issues/4

5.3 不支持关键字 exist （待解决）

5.5 Clickhouse 很多函数不支持

解决办法是将 clickhosue 函数加入进去

5.6 clickhosue 中文官方文档有误

Clickhosue 中文官方文档很多内容都没有写。并且是直接翻译。错了很多。

于是舍弃了中文官方文档，直接看的英文。只不过过程有些缓慢

5.7 遇见的一些 bugs

解决了 click-jdbc jar 包版本直接引入与 springboot-web 包冲突的问题

解决了远程数据库访问权限的问题

解决了 linux 中 clickhouse 数据库表名大小写识别的问题

6 后期规划

6.1 完成报告中其余没有的测试功能，使接入没有问题

继续测试所有的关键字和语法功能，看有没有问题。

6.2 进行复杂查询的测试

前期进行的测试都是一些简单的测试。没有试过很复杂的查询。

6.3 完成所有函数的测试

把 clickhosue 中所有的函数（项目中不存在的）加入到项目中。

虽然 clickhouse 中有些函数的名字和功能 and mysql 中的一样。但是这个不影响结果。虽然 clickhosue 中的函数存在于项目中。但是有些函数可能会存在问题，比如使用的范围等等。只有测试出现问题的时候，再解决。

7. 中期感悟

首先自己感觉还是有很多不足的地方。对项目中一些功能的理解有欠缺。前期一大部分时间都花在了看项目结构和源码上，只对 clickhosue 进行了少量测试，前期阅读代码的时候，可以说是非常的吃力。花了大量的时间，不断地 debug。最后还是没有怎么搞很明白，但是是说对每个方法在做什么事情能大体明白。能够进行修改 sql 组装过程的代码。在不断的看代码，调试的过程中。学到了很多知识点与解决办法。

后期需要做的工作还比较多。目前完成的功能对于后面的工作有很大的帮助，因为越来越熟悉，后面做得会快一些。希望能进一步了解，为项目作出贡献。能成为 APIJSON 的贡献者。

这次能认识大佬老师和诸多优秀同学，也很幸运。