请求接口设计

Register

请求参数示例：

```
{
    "PhoneNumber":1139084122,
    "Password":"zyb",
    "Username":"zyb",
    "Environment":{
        "IP":"123.56.40.182",
        "DeviceID":"2019CP1964"
    }
}
```

返回参数示例：

```
{
    "code": 200/400,
    "msg": "注册成功"/"注册失败"
}
```

Login

请求参数示例：

不需要验证码

```
{
    "PhoneNumber":11390841222,
    "Password":"zyb",
    "Environment":{
        "IP":"123.56.40.182",
        "DeviceID":"2019CP1964"
    }
}
```

需要验证码

```
{
    "PhoneNumber":113908412,
    "Password":"zyb",
    "VerifyCode":"xjp6",
    "Environment":{
        "IP":"123.56.40.182",
        "DeviceID":"2019CP1964"
    }
}
```

返回参数示例:

```
{
    "code": 200,
    "msg": "登陆成功"
}
```

```
{
    "code": 400,
    "msg": 登录失败/缺少验证码/验证码缺失或错误.."
}
```

Applycode

请求参数示例 :

```
{
    "PhoneNumber":1139084122,
    "Url":"login",
    "Environment":{
        "IP":"123.56.40.182",
        "DeviceID":"2019CP1964"
    }
}
```

返回参数示例:

```
{
    "code": 200,
    "msg": "获取验证码成功",
    "path": "localhost:4000/svg/login1139084122.svg",
    "timeout": "有效期60s"
}
```

组件

ApplyCode 组件

timeout = 60

- 描述：验证码有效期 (s)

generate(url,phone)

- 描述：根据给定的 电话号，创建该电话号在对应接口的验证码，并返回存储地址
- 参数：接口url，电话号（唯一标识）
- 返回：静态验证码存储地址（可访问）

check(url, phone, crayCode,callback)

- 描述：判断给定的验证码是否正确（转换为小写后匹配）
- 参数：接口url，电话号，需要检验的验证码，回调函数
- 返回：是否正确（bool）

Method 组件

封装了常用的方法

getUID()

- 描述：获取一个新的UID
- 参数：无
- 返回：UID（String）

getTime()

- 描述：获取当前系统时间
- 参数：无
- 返回：Time（String）

crypto_md5(pwd)

- 描述：加密给定的明文
- 参数：明文
- 返回：加密好的密文（String）

Interface 类

类成员变量：

- id：接口的id (UID)
- url：接口的url
- name：接口的名称

Rule 类

Rule的形式为：

在xxx时间内 访问不超过xxx次，触犯了此规则后，采用xxx方法，并判断是否拦截该ip未来一段时间的操作.

类成员变量：

- id：规则的id（UID）
- timeout：规则的限定时间
- num：规则的限定次数
- crime：规则的描述（验证码/滑块 等）
- level：规则的等级，等级越大风控程度越严重
- lockTime：（额外）触犯规则后拦截该ip的时限（可为0）
- remark：规则的备注（详细信息）

Server类

askStatus(num)

- 描述：查看一个接口是否开启拦截器
- 参数：接口号 num
- 返回：是/否 （bool）

set(num,app)

- 描述：将给定的express对象，分配给接口num （默认不开启拦截，需手动设置）
- 参数：接口号 num，express对象 app
- 返回：无

openServer(num)

- 描述：开始监听接口num的访问请求
- 参数：接口号 num
- 返回：无

closeServer(num)

- 描述：停止监听接口num的访问请求
- 参数：接口号 num
- 返回：无

delServer(num)

- 描述：删除接口的服务
- 参数：接口号 num
- 返回：无

openFilter(num)

- 描述：打开接口num的拦截器
- 参数：接口号 num
- 返回：无

closeFilter(num)

- 描述：关闭接口num的拦截器
- 参数：接口号 num
- 返回：无

风控系统

接口样式：url样式

接口控制块： ICB (Interface Control Block) 用于存放指定URL样式接口的控制信息

风控中心：RMC (Risk Managing Center) 存放所有接口的icb，并进行维护

接口控制信息：用于存放接口的历史访问情况，IP评级，规则向量表等

接口控制工具

ipRecordQ

- 描述：请求事件队列 [List]：存放IP的访问记录历史
- 访问记录：ipRecord 类型
    - ip
    - time：时间戳

ruleList

- 描述：规则向量表 [map]，存放所有规则向量**（全局）**
    - key：规则id
    - value：规则向量（具体参照规则（Rule）类信息）

ruleRecorder

- 描述：规则计数器 [map]，记录 ip 的累计访问次数
    - key：规则 id
    - value：IP计数 [map]
        - key：IP
        - value：num ，该IP累计访问次数

executeList

- 描述：规则执行列表 [list]
    - 规则执行对象
        - Rule：规则向量
        - pointer：该规则向量指针指向请求事件队列的位置

ipLevel

- 描述：IP评级表[map]，记录每个IP评级
    - key：IP
    - value：对应IP的评级

ipifLock

- 描述：IP锁定表[map]，记录每个IP锁定状况
- key : ip
- value :
    - islock：标记是否被锁定
    - endtimeStamp(系统时间 结束)
    - timeoutID：setTimeout 的 ID, 可以手动结束锁定

ICB (Interface Control Block)

接口:

register(environment)

- 描述：将该条访问记录在ICB里进行登记，同时维护IP风控等级
- 参数：environment
- 返回：无

update()

- 描述：ICB自动更新，刷新掉过期记录，同时更新IP等级
- 参数：无
- 返回：无

getLevel(environment)

- 描述：获取一个IP在某接口的风控等级
- 参数：environment
- 返回：风控等级 level

addRule(id)

- 描述：添加一条规则给某一接口
- 参数：规则对应的id
- 返回：无

delRule(id)

- 描述：删除一条接口内的规则
- 参数：规则对应的id
- 返回：无

getAllRules()

- 描述：导出当前接口有哪些规则
- 参数：无
- 返回：[Rule,Rule...]

exportRecorder()

- 描述：导出当前icb的计数器快照
- 参数：无
- 返回：ruleRecorder

exportIpLevel()

- 描述：导出当前icb的IP评级表快照
- 参数：无
- 返回：ipLevel

clear()

- 描述：清空icb数据
- 参数：无
- 返回：无

importRule(new_ruleList)

- 描述：对当前接口导入（多个）规则
- 参数：规则编号列表 new_ruleList
- 返回：无

importRecordQ(ipRecordQ)

- 描述：对当前接口导入事件队列
- 参数：ipRecordQ
- 返回：无

deepClone(target)

- 描述：对指定 tartget 进行深拷贝，并返回拷贝结果
- 参数：深拷贝目标 target
- 返回：拷贝结果 source

recover(target)

- 描述：以指定target进行还原，还原类方法，并返回还原
- 参数：还原目标 target
- 返回：还原结果 source

RMC（Risk Managing Center）

getInterID(url)

- 描述：获取某个url的id
- 参数：接口url
- 返回：对应id

isLocking(ip)

- 描述：查询指定ip是否被锁定
- 参数：ip
- 返回：bool

unlock(ip)

- 描述：对指定ip进行解锁
- 参数：ip
- 返回：无

lock(ip,timeout)

- 描述：锁定指定ip,时长为 timeout (ms)
- 参数：ip, 锁定时长timeout
- 返回：无

checkLock(lockList)

- 描述：检测ip是否应该被上锁
- 参数：lockList [map]
  - key：ip
  - value：[list]
    - timeout
- 返回：无

getLevel(url,environment)

- 描述：查询指定ip在指定接口的风控等级
- 参数：接口url，environment
- 返回：level

register(url,environment)

- 描述：将指定ip在指定接口进行注册
- 参数：接口url，environment
- 返回：无

update(url)

- 描述：指定某一接口进行信息更新
- 参数：接口url
- 返回：无

addInterface(inter)

- 描述：添加一个接口
- 参数：inter（详细见 interface类）
- 返回：无

delInterface(inter)

- 描述：删除一个接口
- 参数：inter（详细见 interface类）
- 返回：无

importInterface(inter_list)

- 描述：导入多个接口
- 参数：inter_list [list]
  - inter（详细见 interface类）
- 返回：无

exportInterface()

- 描述：导出目前的接口信息
- 参数：无
- 返回：[list]
  - inter（详细见 interface类）

addRule(rule)

- 描述：向规则向量表添加一条规则
- 参数：rule
- 返回：无

delRule(id)

- 描述：从规则向量表删除一条规则
- 参数：规则id
- 返回：无

importRule(rule_list)

- 描述：导入多条规则到规则向量表
- 参数：rule_list
- 返回：无

exportRule()

- 描述：导出当前规则向量表
- 参数：无
- 返回：[list]
    - rule

inter_addRule(url,id)

- 描述：对指定接口添加一条指定的规则
- 参数：接口url，规则id
- 返回：无

inter_delRule(url,id)

- 描述：对指定接口删除一条指定的规则
- 参数：接口url，规则id
- 返回：无

inter_importRule(url,id_list)

- 描述：导入多条规则进指定接口
- 参数：接口url，规则id队列
- 返回：无

inter_exportRule(url)

- 描述：导出指定接口的规则
- 参数：接口url
- 返回：[list]
    - 规则id

interImportICB(url,new_icb)

- 描述：导入指定接口的icb信息
- 参数：接口url，icb信息
- 返回：无

interExportICB(url)

- 描述：导出指定接口的icb
- 参数：接口url
- 返回：JSON.stringify(icb)

clear()

- 描述：清空RMC数据
- 参数：无
- 返回：无

业务层

1、 Svg.js

function work(req)

描述：注册静态资源/svg访问记录，动态更新评级

参数：request请求体

返回：无

2、 Applycode.js

function work(req,callback)

描述：用于获取验证码以及判别验证码获取条件是否满足

参数：request请求体，回调函数

返回：无

3、 Login.js

async function work(req,callback)

描述：判别能否登录，并且返回正确信息或者错误信息。注意异步操作套用async

参数：request请求体，回调函数

返回：无

4、 Logout.js

function work(req,callback)

描述：登出操作，注销用户Session

参数：request请求体，回调函数

返回：无

5、 register.js

function work(req,callback)

描述：注册操作，并根据是否注册成功，以及数据正确与否，返回信息

参数：request请求体，回调函数

返回：无

6、 SystemIO

async function importICB_DB(url)

描述：根据给定的端口，从数据库导入对应的ICB控制块信息

参数：url

返回：返回一个信息体

async function exportICB_DB(url,icb)

描述：导出指定端口的ICB控制块信息到数据库

参数：url，icb

返回：返回一个信息体

async function addInter_DB(data)

描述：添加一条接口信息至数据库

参数：无

返回：返回全部的查询信息

async function importInter_DB()

描述：向系统导入数据库中接口信息

参数：无

返回：返回全部的查询信息

async function exportInter_DB(data)

描述：导出系统接口信息

参数：所需数据

返回：无

async function addRule_DB(data)

描述：添加一条规则信息至数据库

参数：无

返回：返回全部的查询信息

async function importRule_DB()

描述：向系统导入数据库中所有规则信息

参数：无

返回：返回全部的查询信息

async function exportRule_DB(data)

描述：导出系统接口信息

参数：所需数据

返回：无

async function importInter_Rule_DB(uid)

描述：导入指定接口的接口-规则关系

参数：uid,即接口ID

返回：返回全部的查询信息

async function exportInter_Rule_DB(uid,rid)

描述：导出一条接口-规则关系

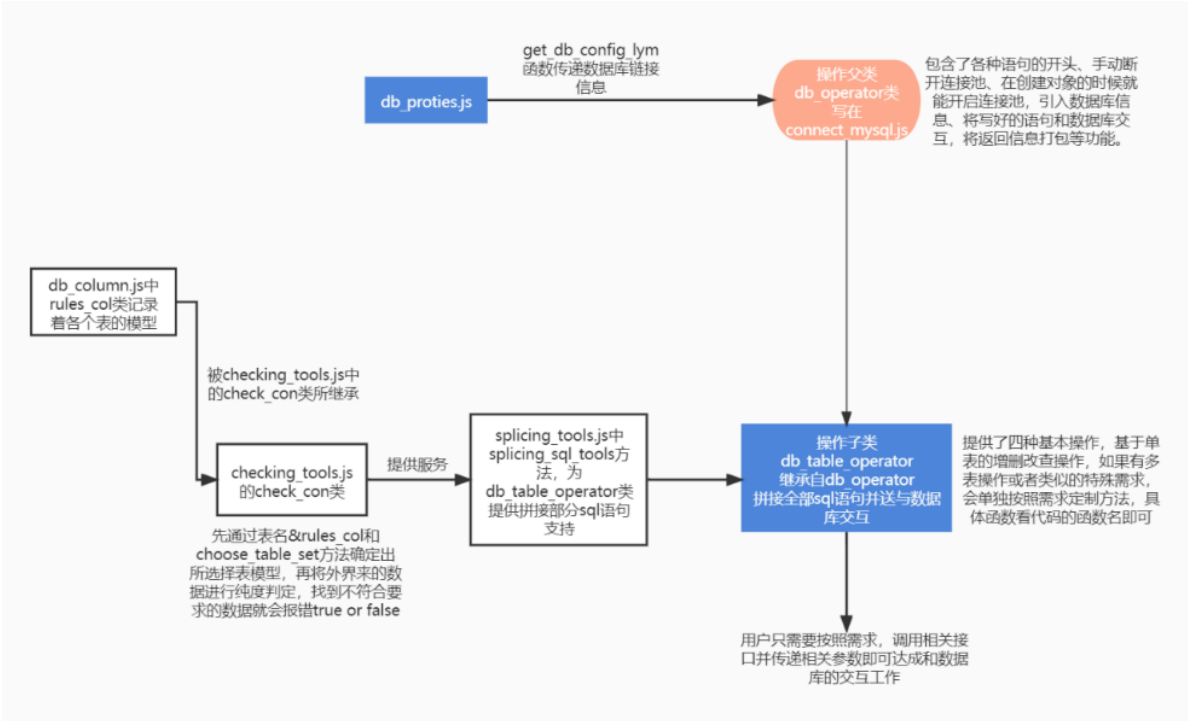
参数：接口ID+规则ID

返回：无

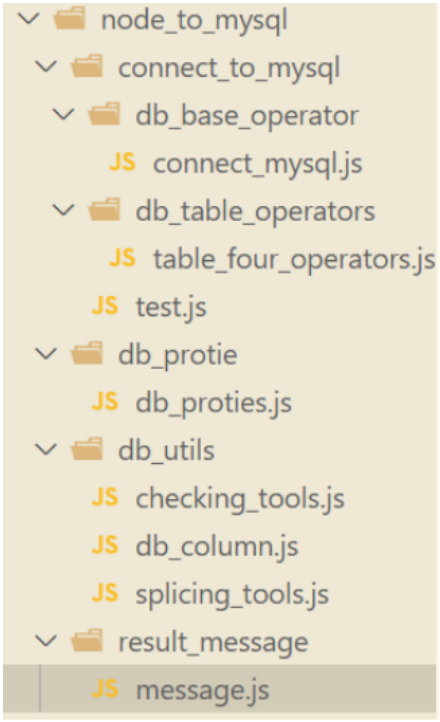
数据库

MySQL操作设计

1、整体设计：



文件结构



2、返回信息类

```
function return_message(code, messages,data){//返回信息
    var message = {};
    message["CODE"] = code;
    message["MESSAGE"] = messages;
    message["DATA"] = data;
    return JSON.parse(JSON.stringify(message));
}

module.exports = {return_message};//向外展示哪些函数可用
```

3. table_four_operators接口说明

dynamic_select_one_table_sql(select_con,where_con,table_name)

为向用户提供的查询接口

参数:

<1>拼接select部分语句所用的信息，格式为:

```
var l = [ 'id', 'password'];
```

其内部为想构造的结果集样式，名称一定要和表的属性名一致。

<2>拼接where部分语句所用的信息，格式为:

```
var item = {
    user_name : {
        operator:'=',
        value:'lwc'
    },
    id : {
        operator:'=',
        value:'16'
    }
};
```

不可以有任何一处为undefined、operator和value咬拼对，外层key要求和上面一个参数一致。

<3> 表名，用于选择模型来校验数据。

dynamic_insert_one_table_sql(insert_con,table_name,primary_key,attribute_num)

<1>构建insert部分sql时候所用数据，属性名(key)不可空，value不能是undefined

格式：

```
var items = {
    interface_id : '10',
    rules_id : '8'
}
```

<2>表名，用于选择模型来校验数据

<3>主键列表，用于进行重复插入判断，加入的是主键列表！！！！！！！！！不要加其他属性进去！！！！！！！！！！！！！！！！！！！！！！！！！！！！！

<4>表中的属性个数，用来判断给的数据是否包含了全部数据(根据需求约定，插入时数据必须全都得有)

dynamic_update_one_table_sql(update_con,where_con,table_name)

<1>拼接set部分语句用的数据，属性值(key)不能写错，value不可空，操作符为=(不变的)

格式：

```
var items = {
    id : '16',
}
```

<2>拼接where时候语句，格式见上。

<3>表名，用于选择模型来校验数据

dynamic_delete_one_table_sql(where_con,table_name)

<1>拼接where时候语句，格式见上。

<2>表名，用于选择模型来校验数据

值得注意的是，这是逻辑删除，内部使用update语句.

7、调用举例

查询

```
target.dynamic_select_one_table_sql(1,item,"user").then(res => {
    console.log(res);
    target.close_pool();
})
```

插入

```
target.dynamic_insert_one_table_sql(items,"interface_rules",
['interface_id','rules_id'],2).then(res => {
    console.log(res);
    target.close_pool();
})
```

更新

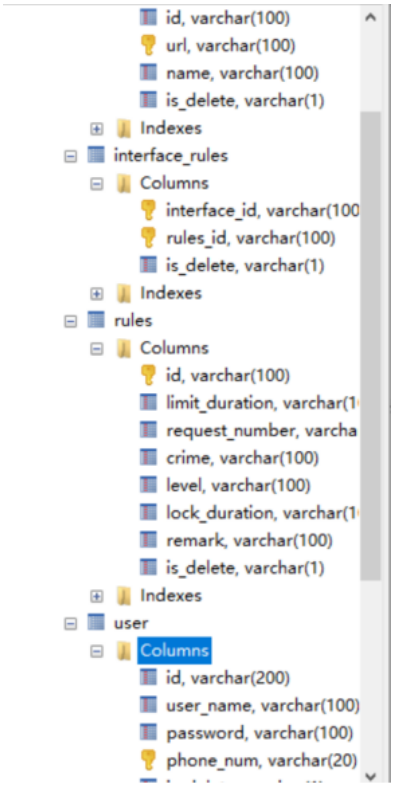
```
target.dynamic_update_one_table_sql(items,item,"user").then(res => {
    console.log(res);
    target.close_pool();
})
```

删除

```
target.dynamic_delete_one_table_sql(item,"user").then(res => {
    console.log(res);
    target.close_pool();
})
```

res 即为结果；一个对象有一个连接池，该对象不用了之后，要手动关闭连接池。

8、数据库设计说明：



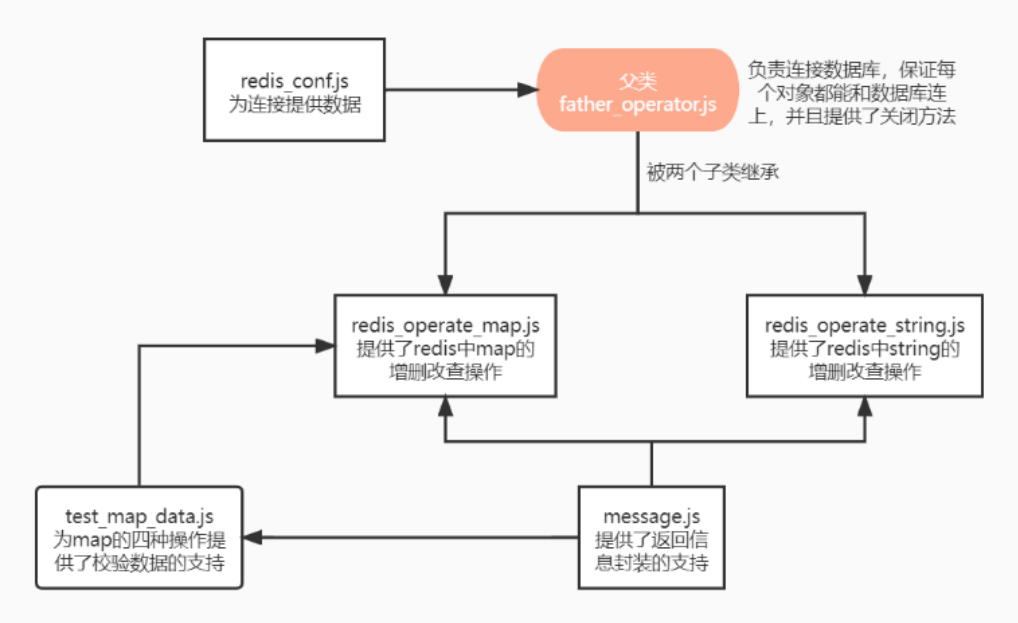
interface表主键 url

interface_rules 主键 interface_id+rules_id

rules表主键 id

user 表主键 phone_num

Redis库操作设计



1、 文件结构：

2、 redis_operate_map.js接口说明：

insert_into_redis(table_name,data,limit_time)

向redis插入数据，参数：

第一个：表名(String)，作为key

第二个：数据(map<phone,Applycode>)

第三个：有效时间(秒)

功能：插入数据，更新数据(只要键值一样，重复插入就是更新)

校验：所有部分数据不能是undefined

select_from_redis(table_name)

向redis查询数据，参数：

第一个：表名(作为key值在数据库里查询)

功能：查询数据

校验：所有部分数据不能是undefined

正确返回：一串JSON数据

delete_from_redis(table_name,field_list)

在redis里面删除键值对，参数：

第一个，表名

第二个：key对应的值域列表，要写全，针对本项目是两个：phone,Applycode

功能：删除数据

校验：所有部分数据不能是undefined+对field_list是否正确以及是否写全的校验。

3、 redis_operate_string.js接口说明：

insert_into_redis(table_name,data,limit_time)

向redis插入数据，参数：

第一个：表名(String)，作为key

第二个：数据(String)，作为value

第三个：有效时间(秒)

功能：插入数据，更新数据(只要键值一样，重复插入就是更新)

校验：所有部分数据不能是undefined

select_from_redis(table_name)

向redis查询数据，参数：

第一个：表名(作为key值在数据库里查询)

功能：查询数据

校验：所有部分数据不能是undefined

正确返回：一串JSON数据

delete_from_redis(table_name)

在redis里面删除键值对，参数：

第一个，表名

功能：删除数据

校验：所有部分数据不能是undefined