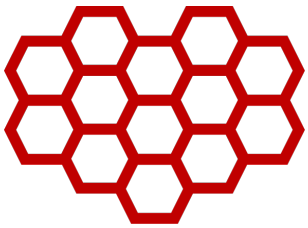
至简网格服务开发指导



作者 李国勇

日期 2024.3.1

修订记录

日期	内容	作者
2024.3.1	创建文档	李国勇

摘要

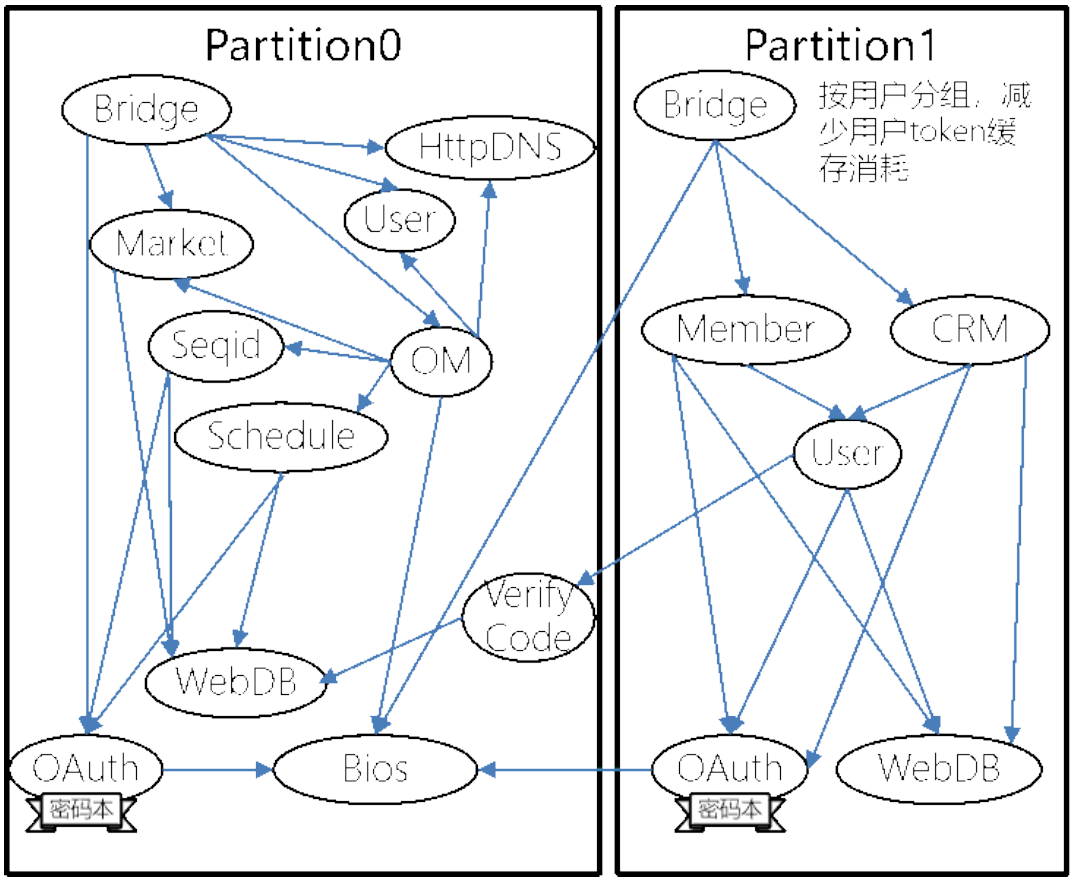
至简网格是一款 HTTP 服务器，用于开发基于数据库的端云结合的服务程序，可以运行在资源极其有限的设备上，比如安卓手机、树莓派等，使得服务器可以尽量前移到生产端，可以运用于边沿计算、企业生产信息化、办公自动化等，

它致力于简化开发、部署与运维工作；通过简单的配置即可实现数据库、接口开发；内置可靠性、安全性实现，业务开发无需过多关注；单例模式可以安装在一个安卓手机上，同时，集群模式可以跨实例、跨机房、跨城市部署。

本文主要用于指导至简网格服务端程序开发，包括数据库定义、接口定义等。

术语

术语	解释
服务	能完成一组功能的服务端程序
业务	具有完整的前后端功能的服务程序，与服务并没有明显的界限
实例	运行了一个或多个服务的服务器或虚拟机
平台	包括企业私网中的服务侧、端侧以及提供公共能力的云侧
AZ	Available Zone 可用区，通常可理解为一个机房，同城跨 AZ 部署，建议 AZ 距离 20km 左右（取自天津港事件）
Region	区域，通常可理解为一个城市，如果用于异地容灾，建议距离大于 500 公里（取自唐山、汶川地震最远破坏距离）
分区	Partition，分区是逻辑上的，一个分区一定在一个 AZ 中；同一个分区中的服务实例是共享的；除了公共分区（分区号 0-1023），不同分区之间不可互访



目录

1. 简介

至简网格为解决企业信息化、自动化而生，服务侧与端侧配合，简化信息记录、统计等繁琐的日常工作。至简网格服务端可以安装在多个节点上实现大规模集群工作，承载巨大的访问量， 也可以安装在单个节点上，满足一些小流量的使用场景。

服务器最小可以安装在一部老旧的安卓手机上，使得管理企业服务与使用普通手机应用一样简单。 只需要简单操作就可以实现企业服务的安装、启停、升级、卸载等维护工作，无需聘请专门的技术人员。

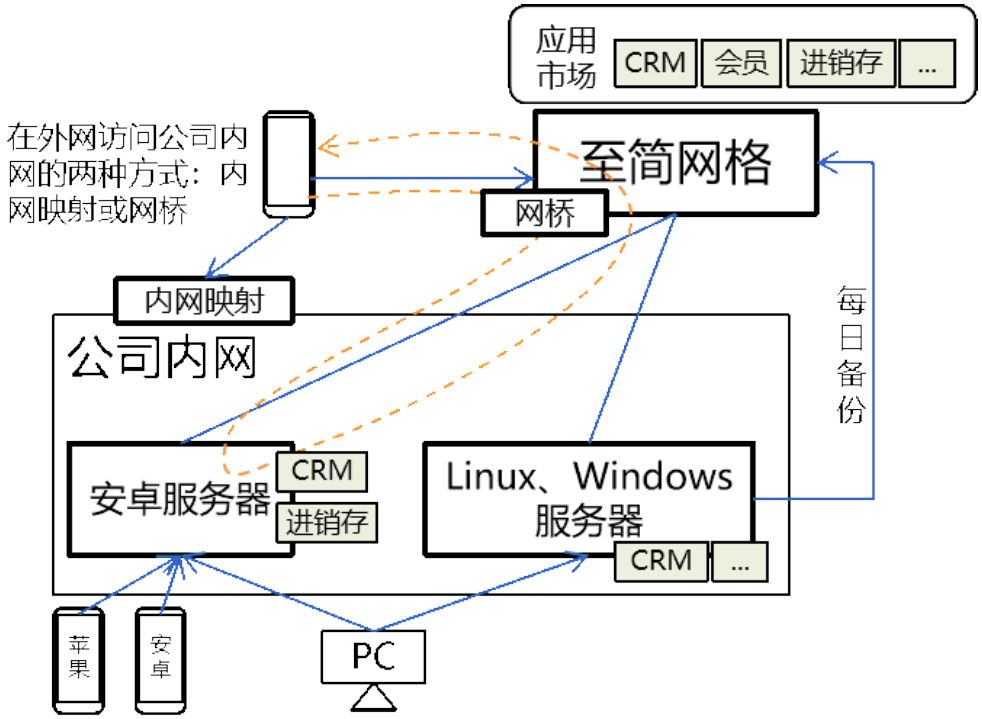
至简网格提供的业务软件都是开源的，永久免费使用，只有在使用每日备份等需要占用服务端资源的服务时，才会产生极少的费用，一般每年只需几十元。

如果您的企业有更高的要求，需要对服务进行定制，至简网格为此提供了极大的便利。服务源码都是 JSON 格式的文本，即使不是程序员，也容易理解；客户端程序就是普通的页面，通过简单的自学，非常容易掌握，可以根据需要自行修改。实在掌握不了的情况下，也可以聘请低级别的程序员进行修改，因为它的难度对于了解软件开发的人来说，是极其简单的。

安全性与可靠性实现难度高，日常使用时，体现不出价值，但是一旦出现问题，却是致命的。 所以，至简网格内置实现了安全、可靠的特性，应用开发时，不必关注底层的安全与可靠实现细节，这样，极大方便了服务的实现。

无论是服务端还是客户端，都竭尽全力地简化，降低开发难度与使用难度。总之，它是一套非常好用的端云结合的开发框架。

以下是至简网格端云结合的总体框架：



2. 为什么

已经有很多服务端开发框架与端侧开发框架，为什么还要重复造这两个轮子？

首先，没有找到合适的端云配合的框架。

其次，虽然云侧有 spring cloud 这样的框架，但是它们都太大，一旦使用就会扯进一堆一堆的组件，在极其受限的环境下使用，会带来无穷尽的兼容性问题， 让开发无法推进下去，而至简网格不会放弃安卓、树莓派等平台的兼容性，所以只能放弃。 移动端有 Capacitor，因为没有时间研究透彻，不敢轻易使用，electron 也是同样；当前的需求并不多，所以都自己动手，减少研究它们的时间消耗。

最后，除了上面那些需要重复造轮子的原因，至简网格还有哪些优势值得别人选择呢？

（一）小到极致

服务端、客户端，都不超过 15M，PC 客户端甚至不到 10M。

极小的端侧体现不出优势，但是极小的服务侧，就便于服务器边沿部署，一部手机、一个树莓派就绰绰有余。因为小，运用场景就可以扩大。 随着通讯能力提升，边缘计算、物联网会变得普遍，至简网格可以方便地部署到各类设备上，将计算尽量下沉。

（二）大到跨市

麻雀虽小五脏俱全，它可以部署在一部旧手机上，也可以跨城市多活部署。 底层实现是完全分布式的，只要采取合适的分片策略， 或者开启数据备份，完全可以跨机房、跨城市多活部署，不必担心单个设备、单个机房故障导致业务中断。 至简网格在底层实现时就为数据分片提供了便利。

（三）非常简单

开发简单

服务前、后端代码量都很小，代码很简单。服务侧业务开发，绝大部分情况使用简单的 JSON 配置就能完成；端侧交互开发，只要懂得 vue 就能胜任。

比如至简网格提供的 CRM、会员两个服务，其中 CRM 较大，接口定义部分约 3500 行 JSON 配置，端侧交互约 3500 行 js 代码，总共 7000 左右，安装包不到 100K。

代码即成本，代码量少，开发&维护的成本就少；开发难度低，即使是初级程序员也能开发维护。

维护简单

安卓版本的服务器，使用起来跟普通 App 一样安装、升级、卸载，服务一键启停，服务软件下载安装不过几秒钟。与服务对应的端侧应用使用更加简单，秒级安装，自动升级。

使用简单

已支持安卓与 Windows 客户端，端侧交互简单； 权限控制简单，支持多种业务维度的授权控制，并且用户可以在企业内网使用，也可以开放部分用户在公网访问。

（四）可靠安全

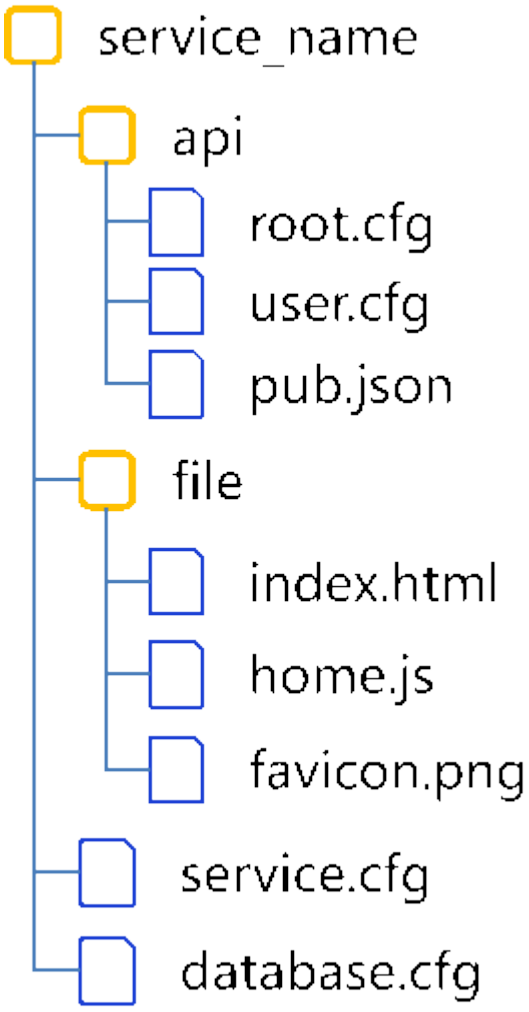
可靠与安全相关的设计实现，在至简网格随处可见，比如，分区隔离、公司隔离；传输都采用 https，使用 ECC256 证书，安全性达到 RSA3072 的强度；每个服务实例都自动分配了独一无二的证书； 服务间访问都需要 token，两个服务器即使在同一个局域网内，默认也不能窜访； 用户密钥经过 PBKDF2 加密存储，即使数据库泄露，也无法解开密码； 数据库两份拷贝，支持每日往云端备份……安全与可靠设计融入到至简网格的方方面面。

3. 服务开发概览

在至简网格中，每个服务对应一个独立的目录，目录中存放数据库、接口定义、端侧界面实现。

3.1. 目录结构

服务的根目录下有 api、file 两个目录，以及 service.cfg 与 database.cfg 两个文件。



1. api 目录中存放所有的接口定义文件，每个文件中可以定义多个接口；

A. 在调用接口时，url 需要携带文件名及接口名，比如调用 user.cfg 中的接口 add，url 为/user/add;

B. root.cfg 是特殊的，访问其中的接口不必携带/root，直接传/xxxx 即可；

C. json 扩展名的文件存放一个 Map 结构，Map 的每一项都是一个静态接口，其中的内容直接返回，比如 roles:{...}，访问时直接调用/roles 即可得到大括号中的内容；

D. def 扩展名的文件是宏定义文件，也是 Map 结构，每一项都是一个 process，在接口定义文件的 process 部分可以引用宏定义；

E. 如果无接口定义，可以没有此目录。

2. file 目录存放所有的交互页面，属于端侧开发，起始页固定为 index.html，在 index.html 中 import 所需的组件；

A. 建议一个组件对应一个 js 文件，比如 home.js、customer.js 等;

B. file 目录中内容就是端侧所需的交互文件，端侧在安装应用时，下载的就是这个目录的压缩包；

C. 如果无交互界面，可以没有此目录，但是如果希望服务有一个个性化 logo，建议增加 file 目录，并存放适合的 favicon.png 文件。

3. service.cfg 中定义了服务的名称、依赖服务等信息；

4. database.cfg 中定义了服务的数据库表结构，treedb、searchdb 无需建表，但是也需要在里面申明。

3.2. service.cfg

service.cfg 配置非常简单，格式如下：

```
{
    "author":"flyinmind@zhijian.net.cn", //作者
    "company" : "zhijian.net.cn", //公司或组织名称
    "version":"0.1.0", //版本号
    "dependencies":[
        //依赖的服务列表，如果不申明，就不能调用这个服务
        //webdb、bios、oauth 等服务无需申明
        //安卓等平台的单例版本，自动添加服务依赖
        //非单例版本，需要在 OM 平台上设置依赖关系及数据定义
        {"name":"user", "minVersion":"0.1.0", "maxVersion":"0.2.1"}
    ],
    "displayName":"系统接入控制" //对外显示的名称
}
```

3.3. database.cfg

database.cfg 定义了 rdb 的表结构，treedb、searchdb 没有建表操作，但是必须在此申明。

在单例模式（比如在安卓服务器中）运行时，至简网格会自动执行 database 中的建库、建表操作，也根据按服务器已有的数据版本，执行升级操作。

```
[
{
    "name":"crm", //库名称
    "version":"0.2.0", //版本号
    "type":"rdb", //类型，有 rdb（关系型数据库）、tdb（树形数据库）、sdb（搜索数据库）
    "versions":[
    {
    //minVer 与 maxVer 指定了最小、最大可执行版本
    //如果已有数据库版本不在此范围内，则里面的 sqls 不会执行
    "minVer":"0.0.0",
    "maxVer":"0.1.0",
    "toVer":"0.2.0", //升级后的数据库版本号
    "sqls":[
    //建表语句，或者升级 sql，可以有多个
    "create table if not exists orders ( -- 订单信息
    id int not null primary key, -- seq_id
    ...
    )"
```

```
]
},
{
    //另一个版本的初始或升级脚本
}
]
},
{
    "name":"crm", //searchdb 的名称，与 rdb 同名，表示与 rdb 在同一个库中
    "type":"sdb"
},
{
    //treedb 的名称，与 rdb 同名，表示与 rdb 在同一个库中
    //与 rdb 共库的情况，需要 rdb 中不能有 dir、item 这样的表名，否则会造成表名称冲
    突
    "name":"crm",
    "type":"tdb"
}
```

]

3.4. 接口文件

api 下每个".cfg"文件中都定义了一系列的接口。 比如存在接口文件 customer.cfg 定义了 create 接口，则可以通过 URL "/customer/create" 访问。

```
[
{
    "name": "create", //接口名称
    "method":"POST", //调用的 method，如果调用方使用的 method 错误，会返回
    API_NOT_FOUND 错误
    "property" : "private", //public 或 private，private 接口必须在请求头中携带服务 token
    才可以访问，
    "tokenChecker" : "USER", //鉴权类，USER|OAUTH|OM|APP
    "comment":"创建客户，需要在电子流中审批", //描述
    "request": [...],
    "process" : [...],
    "response":[...]
},
...
```

]

3.5. 接口宏定义

".def"中只可以定义的宏只能是 process，可以在接口的 process 里引用。比如定义一个 check_accounts 宏：

```
"check_accounts":{
    "name":"check_accounts",
    "comment":"检查帐号是否都存在",
    "type" : "call",
    "service": "user",
    "method":"POST",
    "url":"/user/userid",
    "tokenSign":"OAUTH",
    "parameters":"{\"accounts\":#ACCLIST#}"
}
```

这个宏可以在接口中引用，比如：

```
"process" : [
    {"macro": "check_accounts", "#ACCLIST#":"@{JSON|to,0}"},
    ...
```

]

宏可以传递参数，宏参数名称前后要加上“#”，比如上例中的"#ACCLIST#"

3.6. 静态接口

".json"文件用来定义静态内容的接口，与 type 为 static 的处理 不同之处在于“这些接口必须 public 的”。常用在 roles 接口中，roles 接口是服务定义用户角色用的，比如：

```
{
    "roles": {
        "admin":{"name":"企业主","rights":{"sku":"*","report":"*","proxy":"*"}},
        "sales":{"name":"销售","rights":{}},
        "finance":{"name":"财务","rights":{"report":"*"}},
        "support":{"name":"服务","rights":{}}
    }
}
```

这样在其他服务中就可以通过调用”/roles“获得服务支持的角色，以及各角色可以执行哪些类型接口的信息。

4. 接口定义

4.1. 总体格式

服务端开发主要是接口定义，每个接口定义分成四个部分， 基本信息（名称、可见性等）、变量定义、请求参数 request、处理逻辑 process、响应体 response。总体结构如下：

{

name:"api 名称",

method:"可接受的请求方法，不设置表示不限,POST|GET|PUT|DELETE",

property:"属性，private 或 public",

tokenChecker: "认证方式，property 有 public 时不必设置，USER|UNIUSER|OAUTH|OM|COMPANY|INIT|NODE|APP|APP-调用方服务名或*",

aclChecker: "接入检查，只支持 RBAC(Role Based Access Control)或者自定义实现",

sameAs:"如果接口的 request、vars、process、response 与某个其他的接口完全一致，则可以增加此配置，指定为那个接口的路径，比如与 stats.cfg 中的 report 接口相同，则可以写成/stats/report，此时 request、vars、process、response 不必配置",

feature: "特性，与 RBAC 配合，用于更加细致的控制角色授权",

comment:"描述，用于生成接口描述，可不提供",

vars:[

变量列表，可以有多个

],

request:[

请求参数列表，可以有多个，支持嵌套复杂结构

],

process:[

处理逻辑，可以有多个

],

response:[

响应结果，可以有多个，支持嵌套复杂结构

]

}

4.2. 请求 request

请求参数是一个 list，每个元素都是是一个请求参数定义，定义了名称、类型、值范围等信息，比如：

```
"request": [
    {"name":"name","type":"string","must":true,"regular":"^[a-z0-9]{1,30}$"},
    {"name":"val","type":"int","must":true,"max":0,"min":10}
]
```

参与与变量看作一类，在脚本中引用方式完全一样。

4.2.1. 参数

配置项	定义	类型	备注
name	参数名称	String	同一个接口的参数列表中，必须唯一
type	参数类型，不区分大小写	String	STRING、INT、LONG、FLOAT、BOOL、DATE、DOUBLE、OBJECT、BYTES、NOW、UUID、SEQUENCE、CONFIG、JSON。 几个特殊类型： Config：从 bios 的服务配置项中获取内容； Json：json 串，作为响应参数时会被转成 json 对象，作为输入参数时，被转为 json 字符串
must	是否为必须	Bool	如果为 true，当请求未携带此参数时，校验失败

配置项	定义	类型	备注
	参数		
max	Number 型： 最大值； String 型：最 大长度	Int	数值型的情况，默认为该类型能够表达的最大值，比如 type 为 int 时，默认为 Integer.MAX_VALUE。 long、double、float 以此类推。 String 类型默认为 255
min	Number 型： 最小值； String 型：最 小长度	Int	数值型的情况，默认为该类型能够表达的最大值，比如 type 为 int 时，默认为 Integer.MIN_VALUE。 long、double、float 以此类推。 String 类型默认为 0
list	是否为 list	Bool	list 中每个元素类型都由此参数的 type 指定； 在 Json 类型参数中，list 指定 json 是否为一个数组，true 时为数组，否则为 map
maxSize	List 中最多的 元素个数	Int	list 为 true 时，才有意义，默认为 10240
minSize	List 中最少的 元素个数	Int	list 为 true 时，才有意义，默认为 0
default	参数默认值	与 type 指定的 参数类 型一致	当不是必须参数时，可以设置默认值，当参数没有传递 时，则参数使用此值。 数值型、String、Bool：填写对应类型的值； Datetime：日期，格式由 format 指定； Bytes：base64 字符串，用 keytool 生成；

配置项	定义	类型	备注
			Json：一个 json 字符串；
const	是否为常量参数	Bool	常量参数，无需在请求时传递， 必须指定 default 值，接口定义中可以像普通参数一样使用
dataSeg	指定响应内容的字段名	String	默认就是 name。当响应内容是一个复杂的结构时，可以在 dataSeg 中指定分级，每级用“.”分隔
options	可选值列表	List	如果请求参数不在可选列表中，则参数校验失败
log	是否可以打印在日志中	Bool	默认为 true，表示可以打印到日志中
Object 特有的配置项			
props	嵌套定义复杂的结构	List	每一项是一个基本参数配置，用于指定 object 中的字段。 props 里面的字段还可以设为 object 类型，以此实现复杂的结构嵌套。 {"name":"infos", "type":"object", "must":true, "props":[{"name":"name", "type":"string"}, {"name":"val", "type":"string"}]}
checkAll	是否检查每个字段	Bool	默认为 true，检查 props 中定义的每个字段合法性。 如果是响应内容，checkAll 为 false 时，无论 object 有什么冗余内容，都会原样放过。
String 特有的配置项			

配置项	定义	类型	备注
len	字符串长度	Int	本质是将 min、max 设置成相同的值
regular	字符串合法性正则检查	String	只在 String 类型参数中有意义
tail	末尾添加的内容	String	在字符串的末尾额外添加的内容
trim	是否去除首尾空格	Bool	默认为 false
equalsTo	需要等于的字段	String	用在确认密码等场景，判断本参数必须要等于另外一个参数
maps	映射关系	Map	将一个值映射成其他值，只在 String 类型参数中有意义，通常用于多版本的兼容中
codeMode	编码模式	String	需要指定 keyName，keyName 在 OM 中配置 encode:加密数据 decode：解密数据
ipCheck	检查字符串是否为合法的 IP 地址	String	V4:是否为 IPv4 地址 V6:是否为 IPv6 地址 PORT:是否携带了端口号 LAN:是否为内网地址 WAN:是否为外网地址 LIST:以逗号分隔的多个地址
pwdChec	密码强度检	String	以逗号分隔成四个部分，从后向前，可以省略一部分，

配置项	定义	类型	备注
k	查		它们分别为： minLen:最短长度，默认为 4 charTypeNum:字符类型数量（大写字母、小写字母、数字、英文标点、其他），默认为 3 differentCharNum:不同字符数量，默认为 4 accountPara:账号字段名称，用于判断密码是否与账号类似，默认为 null，表示不判断
keyName	加密密钥名称	String	需要先在 OM 中配置，在服务配置项中配置密码在密码箱中的 UserKey 及名称，以":"分隔； 然后在密码箱中添加该密码，keyName 是密码箱中密码的名称 1)在密码箱中创建同名的密码； 2)在配置项中配置同名的配置项。 配置项内容由两部分组成： 经过根密钥加密的"密码箱的 userKey":经过密码箱加密后的"数据密钥" 其中的数据密钥就是用来加解密用户数据的。"根密钥、密码箱"请参照 4.5 节相关内容。
Sequence 特有配置			
len	序列值字节	Int	只可以为 4 或 8，为 4 时返回 Int 类型，为 8 时返回 Long

配置项	定义	类型	备注
	数		类型
Config 特有配置			
item	配置项名称	String	在服务设置中配置项的名称
Datetime、Now 特有配置			
format	日期格式	String	指定日期输出格式，作为输入参数时，只接受 UTC 时间戳
UUID 特有配置			
base64	是否为 base64 格式	Bool	如果为 true，则按 base64 输出，否则按 hex 输出

4.2.2. 变量

组合一个或多个参数，经过复杂计算厚，得到一个新的变量，得到的结果可以在脚本中像普通请求参数一样引用，多次引用并不会导致多次计算。

1）val:配置中可以使用内置占位符；

2）response：默认为 false，如果设为 true，生成的变量会插入到响应的 data 中。

```
"vars":[
    {"name":"flowid", "response":true, "val":"@{SEQUENCE|'flow',i}", "comment":"流程 id"}
]
```

4.3. 处理 process

一个接口 process 中，可以包括多个子处理，常用的处理类型有 js、java、rdb、treedb、search、localrdb、localtreedb、 localsearch、call、static、dataexists。

也可以自定义类型，实现 IProcessor 接口，或继承自已有的实现类，再通过 AbstractProcessor.register 注册到至简网格中； 配置接口时，将处理的 type 设置成注册的名称后，就可以使用。 或者不注册，直接将 handler 设置成对应的类即可。 相应的 class 文件要打包成 jar，放在服务根目录下的 libs 文件夹中即可，这属于 高阶开发，在此不赘述。

除 static 处理外，其他处理类型都有四个公共配置：

1) cache：表示是否缓存历史结果，只能指定一个占位符，比如@{HASH|cid,service}，此内容作为缓存的 key 值，读取缓存时，也用相同的 key，缓存有效期默认为 10 分钟；

2) ignores：可以忽略的错误码列表，如果发生的错误在这个列表中，就忽略它，返回 OK，否则结束当前处理以及后继的其他处理，返回错误码；[-1]表示忽略所有错误码；

3) when：一段 js 脚本，要求返回值是 boolean 型，确定当前 process 是否执行，配置中只能使用请求参数、变量、请求头或上一步的响应参数；

4) convert：将 start-end（包括 start、end）范围内的错误码全部转换成"to"错误码，info 中指定错误信息，如果 to 为 OK，可以设置 data，data 必须为一个 json 字符串，可以使用占位符；如果 start 与 end 相同，可以用 code 代替。

4.3.1. RDB

RDB 是使用最多的处理类型，调用 webdb/api/rdb/request 实现数据库读写。

```
{
    "name" : "sys",
    "type" : "rdb",
    "db":"companydb",
    "sharding":"@{cid}",
    "sqls" : ["replace into config(cid,service,k,v) values(@{cid}, '@{service}', '@{k}',
    '@{v}')"]
}
```

1) db：指定需要操作的数据库；

2) sharding：指定分片计算方法，可以引用请求参数、变量、请求头或者上一步的返回结果， 也可以使用 token 中的数据或请求头中的数据，但是最终结果要转换为一个无符号整型数，更多详情在 数据分片中；

3) sqls：可以只有一个 sql，也可以有多个；

   A. 多个 sql 是顺序执行的；

   B. 下一个 sql 可以使用上一个 sql 的查询结果，通过占位符@[!xxx]引用；

   C. 每个 sql 的配置可以是一个字符串，也可以是一个 map，通常增删改操作可以写成一个字符串，查询操作写成 map，因为需要对返回结果进行定义；

D. 多个写 sql 是放在一个事务中执行的，如果一个发生了错误，则所有操作都会回滚。

4) any：多个 sql 的情况，如果 any 为 true，则，任意一个执行成功就返回结果，否则将所有 sql 的执行结果都汇总后再返回。

4.3.1.1. 普通 SQL

SQL 操作是最常见的接口操作。增删改比较简单，只有成功失败的返回；而查询类 SQL，因为要设置结果集的返回格式，所以每个 sql 还有 name、multi、metas、merge 配置。

```
{
    "name":"vips",
    "multi":true,
    "metas":"each",
    "sql":"select id,name,mobile,update_time from vips
        order by update_time desc LIMIT @{num} OFFSET @{offset}",
    "comment":"返回字段与 search 保持一致"
}
```

1) name：执行结果的名称，在 merge 为 true 时，无意义，只用于日志中打印；

2) multi：返回结果是否为多行；

3) metas：返回结果中每一行是否携带字段名信息；

A. each：返回的每行记录中，每个字段都带有列名，如，{mobile:189...}；

B. none：每行记录都是一个数组，如，返回[1,"hello",4]，这样可以减少响应体大小；

C. oneCol：如果结果集有多行，且只有一列，可以指定 oneCol，返回一个数组，如，ids:[1,2,3,4...]，这样可以减少响应内容；

D. 列信息字段名：数据记录按数组返回，但是在最后添加一行各列的列名，如，cols:["name","age",...]，这里的 cols 就是用 metas 指定的， 解析时可以利用它，既可以减少返回内容的体积，又可以方便标识每一列。

4) merge：是否将结果直接存在 HandleResult.data 中，当 multi 为 false 时才有效；

A. false：响应形如 data.'name'.mobile:189...，其中的 name 就是 sql 配置中的列名称；

B. true：响应形如 data.mobile:189...，省去了中间一层。

【注意】

1) update_time 字段是系统在建表语句中插入的字段，用于辅助数据复制，查询时可以使用；

2) 简单增删改，系统自动添加 update_time 及对应的当前时间戳；

3) 复杂 sql，比如批量插入，系统需要将它们变成多行简单的 sql，逐行添加 update_time。

4.3.1.2. 带 JS 的 SQL

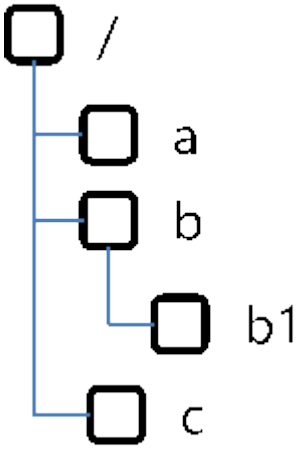
如果 SQL 比较复杂，需要一些简单的逻辑来拼装，则可以用"js:"开头，后面跟一段复杂的 js 脚本生成 SQL，比如实现一个批量插入数据的处理：

```
js:var sqls=['insert into tb(a,b,c,d) values']
var vv=@{signers};
for(var i in vv){
    if(i>0){sqls.push(',');}
    sqls.push("(@{a},'@{b}',");
    sqls.push(vv[i]);
    sqls.push(",@{ABSHASH|c,d})")
}
DB.sql(sqls.join(''));
```

使用 js 拼装 sql 时，所有占位符都可以用，占位符解析时会将字符串中的单引号"'"变为两个单引号"''"。也可以使用服务端内置的 js 函数（请参考 4.3.5）。

4.3.2. TreeRDB

TreeDB 是记录树状关系数据的数据库，比如：




```
{

   "name" : "createDb",

   "type" : "biosmeta",

   "actions" : [

      {"action":"crtDir", "key":"/service/@{service}/dbs"},

      {"action":"crtDir", "key":"/service/@{service}/dbs/@{db}"},

      {"action":"put", "key":"/service/@{service}/dbs/@{db}/tabledef", "value":""},

      {"action":"put", "key":"/service/@{service}/dbs/@{db}/type", "value":"@{type}"},

      {"action":"get", "key":"/service/@{service}/dbs/@{db}/type"}

   ]

}
```

1) action 是区分大小写的；

2) 所有的 action 都有 key 选项，key 唯一指定一个记录；

3) value 用于存储 key 对应的值，如果是一个 dir，不必指定 value；

4) 一个 dir 下可以有多条 K-V 键值对。

action	作用	备注
crtDir	创建 dir	创建 key 之前，必须创建对应的父目录，然后在其下面才可以创建多个 K-V 键值对
rmvDir	删除 dir	删除 dir 之前，必须保证 dir 没有 K-V
put	新增或更新键值对	必须保证父 dir 都存在
putIfAbsent	不存在则添加	如果键值对不存在则创建，否则放弃操作，返回 2000 错误码
putList	插入数组	把 value 当作一个数组，在其中添加元素； 如果 key 不存在，则创建它，如果存在，且 value 不在数组中，则添加
putMap	插入对象	把 value 当作一个 Map； 如果 key 不存在，则创建它，并将 value 转为 json 后存入； 如果存在，覆盖它; 传入的 value 是一个 Map
puts	在目录插入多对 K-V	在有 key 指定的目录下插入多对 K-V，value 参数是一个 Map 对象，指定多对 K-V
get	获得键值对	获得有 key 指定的 value，value 以字符串形式返回；返回值的名称可以有 as 指定，不指定则默认为 key 参数的最后一段
gets	获得一个目录下的所有键值对	键值对以 Map 数组返回，每个元素中有 key、val、ut；返回值的名称可以有 as 指定，不指定则默认为 key 参数的最后一段
getSubs	列举所有子目录	返回由 key 指定目录下的所有子目录，不包括 K-V；返回值的名称可以由 as 指定，不指定则默认为 key 参数的最后一段
getSubsAndItems	列举所有子目录及其拥有的键值对，	返回由 key 指定目录下的所有子目录，及它们下面的 K-V；每行包括 name,key,val 三个字段；返回值的名称

	比如 /service/config/dbs 下的所有子目录，及各子目录下的所有 K-V 项	可以有 as 指定，不指定则默认为 key 参数的最后一段
names	列举目录下所有 key	返回目录所有 key 的列表;返回值的名称可以有 as 指定，不指定则默认为 key 参数的最后一段
getMap	从 Map 中取一个字段	从 Map 形式返回，如果未用 value 指定字段名，则返回整个 Map，指定了则只返回字段名指定的值；返回值的名称可以有 as 指定，不指定则默认为 key 参数的最后一段
getsMap	返回目录下所有 K-V-UT	返回由 key 指定目录下的所有 K-V，以及 UT 更新时间；返回值的名称可以有 as 指定，不指定则默认为 key 参数的最后一段
getId	获得目录 id	返回目录 id
list	列举所有子目录	返回由 key 指定目录下的所有子目录，返回内容包括 id、name、ut；返回值的名称可以有 as 指定，不指定则默认为 key 参数的最后一段
rmv	删除 key	
rmvFromMap	删除 value 中的一个 key	把 value 当作 map，删除 Map 中由 value 参数指定的 key
rmvFromList	删除 value 中一个元素	把 value 当作 list，删除 List 中由 value 参数指定的元素
rmvs	删除目录下全部 K-V	删除由 key 指定的目录下的所有 K-V

4.3.3. Search

SearchDB 是逆向索引的数据库，用于分词查找。action 有 put、update、get、rmv。db 指定搜索的库名称，table 指定虚拟表名（并不存在实体的表），did 指定内容对应的数据唯一标识。

4.3.3.1. Put

添加搜索内容，title 指定标题，summary 指定摘要内容，content 指定具体内容；

```
{
    "name" : "createSearch",
    "type" : "search",
    "db": "crm",
    "action" : "put",
    "table":"customer",
    "did" : "@{custId}",
    "title" : "@{name}",
    "summary" : "@{address}",
    "content" : "@{CLEAN|comment} @{business} @{taxid}"
}
```

title、summary、content 并无本质区别，只是对照一篇文章的结构逻辑上分成标题、摘要、内容三部分。 每个部分在入库时都会经过分词处理，变成一个一个独立的词语存入库中，也可以在输入时就人为加空格，以提供分词的准去率。

4.3.3.2. Update

更新搜索内容，如果数据不存在，则，功能类似 put，如果数据已存在，则可以更新 title、summary、content，如果某项未传，则此项不更新

4.3.3.3. rmv

删除内容，只需传 did 参数；

4.3.3.4. get

搜索，get 后面可以增加传回的最大结果集行数，content 指定要搜索的内容，可以用空格分隔成多个词。

```
{
    "name" : "docs",
    "type" : "search",
    "db": "user",
    "table":"user",
    "action" : "get @{limit}",
    "content" : "@{s}"
}
```

content 即为要查找的内容，查找前会经过分词处理，也可以人为在词之间添加空格，以提升分词的准确率。

4.3.4. LocalxxxDB

每种 db 都对应有本地版本，localrdb、localtreedb、localsearch。

本地版的各类数据处理的数据都是只在服务实例本地可用，数据不会在不同实例间复制，没有两份拷贝，也不会往云端备份。比如地址库，包括了 localrdb、localsearch，它只能在一个服务实例中使用。

4.3.5. js

如果基本的数据库操作无法满足处理逻辑定义，可以使用 java、js 进行开发，因为安卓的字节码不同于 java，安卓上 java 逻辑开发很麻烦，所以本系统只支持 JS 脚本。 脚本中可以使用参数，通过@{xxx}引用，前面 processor 返回的结果可以通过@{!xxx}引用。

```
{
    "name" : "judgeExists",
    "type" : "js",
    "script" : "
        if(@{!vipNum}>0) {
        Mesh.error(RetCode.EXISTS,'vip already exists');
        } else {
        Mesh.success({});
        }
    "
}
```

JS 扩展能力

除了基本 js 基本功能外，系统提供了 Mesh、Logger、String、Secure 扩展能力。

函数	备注

函数	备注
Mesh 类中的函数	
success(jsonData)	返回成功的 HandleResult，jsonData 是返回数据，可以为"{}"，表示无数据
error(errCode, info)	返回失败的 HandleResult，errCode 在 RetCode 中定义
Logger 类中的函数	
debug(s)	输出 debug 级别的日志
info(s)	输出 info 级别的日志
warn(s)	输出 warn 级别的日志
error(s)	输出 error 级别的日志
String 类中的函数	
uuid()	产生 uuid 字符串，使用 base64 编码
replaceChars(str, ch, replaceWith)	在 str 中寻找 ch，并替换成 replaceWith
chkCreditCode(s)	判断是否为合法的统一信用码
base64CharCode(c)	返回一个字符的 base64 编码，c 必须是"a-z,A-Z,0-9,_,-"中的一个
isLanIP(v)	判断是否为局域网 IP，支持 IPv4 与 IPv6 判断
isIPv4(v)	是否为一个合法的 IPv4 地址

函数	备注
isIPv6(v)	是否为一个合法的 IPv6 地址
Secure 类中的函数	
pbkdf2(pwd, iterCount)	使用 pbkdf2 算法，将 pwd 迭代 iterCount 次
pbkdf2Check(pwd, savedPwd)	检查输入的 pwd 与 savedPwd 是否一致，savedPwd 由 pbkdf2 函数生成
hash(s)	计算多 int 型 hash 值
longHash(s)	计算字符串 long 型 hash code
absHash(s)	与 longHash 类似，但是返回的是大于 0 的 hash code
intHash(s)	将几个字符串连在一起，计算 int 型 hash code
cbcEncrypt(plain, key, kenLen)	使用 AES-CBC 算法加密，plain 为明文，key 为密钥，keyLen 可以选择 16/24/32。IV 为随机产生，并记录在密文的前面 16 字节中。
cbcDecrypt(cipher, key, kenLen)	使用 AES-CBC 算法解密，cipher 为密文，其中包括了随机 IV
gcmEncrypt(plain, key, kenLen)	使用 AES-GCM 算法加密，plain 为明文，key 为密钥，keyLen 可以选择 16/24/32。IV 为随机产生，并记录在密文的前面 16 字节中。
gcmDecrypt(cipher, key, kenLen)	使用 AES-GCM 算法解密，cipher 为密文，其中包括了随机 IV

函数	备注
md5(str)	使用 MD5 算法对 str 进行不可逆运算
sha1(str)	使用 SHA1 算法对 str 进行不可逆运算
sha256(s)	使用 SHA256 算法对 s1,s2,s3…进行不可逆运算，在它们之间会增加分隔符"-"
hmacSHA256(str)	使用 SHA256 算法对 str 进行不可逆运算。随机生成 16 字节 key，并记录在结果的前面
hmacSHA256Check(str, saved)	验证 str 与 saved 是否一致，saved 是 hmacSHA256 算法生成的
hmacSHA1(str, key)	使用 HMAC-SHA1 算法对 str 进行不可逆运算，key 可以是一个随机字符串
isPwdStrong(acc, pwd, min, max, charTypeNum, diffCharNum)	判断密码强度是否足够。 acc：帐号 pwd：密码 min：最小长度 charTypeNum：不同字符的数量 diffCharNum：不同类型字符数量，0-9|a-z|A-Z|其他，共四类

4.3.6. call

用于在一个接口中，调用其他接口，可以并发几个调用。call 只能调用同一个分区或公共分区中的服务。

1) service：指定需要调用的服务；

2) url：指定 url，其中不必包括“/api”；

3) method：指定调用的方法，只支持 POST 与 GET 两种；

4) parameters：指定请求参数，可以引用参数或上一步的返回结果，比如 GET 方法，写成 a=1&b=2...；POST 方法，只能用 json 格式，比如：{a:1,b:"x"...}，此 json 串可以直接写出，也可以放在字符串中，比如：：“{a:1,b:"x"...}”；

5) tokenSign：表示使用的 token 类型，可以选择 OMKEY、OAUTH、APPKEY 三种；

6) trans：表示请求是否将参数全部传递给被调服务。

```
{
    "name" : "addAcl",
    "type" : "call",
    "service":"bios",
    "method":"POST",
    "url":"/acl/set",
    "tokenSign":"OM",
    "trans":true
```

}

如果一个处理中需要发起多个请求，可以在 calls 中指定多个调用。此时，如果 any 设为 true（默认为 false），则只要一个请求成功，则最终结果为成功，其他请求的响应都丢弃；如果为 false，则只要有一个响应失败，则返回失败，所有响应都成功的情况下，会将多个响应合并在一起返回。

```
{
    "name" : "dbs&partInfo",
    "type" : "call",
    "any":false,
    "calls" : [
        {
            "service":"bios",
            "method":"GET",
            "url":"/db/serviceDbsDetail",
            "tokenSign":"OM",
            "parameters":"service=@{service}"
        },
        {
            "service":"bios",
            "method":"GET",
```

```
            "url":"/company/partInfo",
            "tokenSign":"OM",
            "parameters":"id=@{cid}"
        }
    ]
}
```

4.3.7. static

始终返回静态内容，只有一个 data 配置项，定义一个静态的 json 串。

```
{
    "name" : "segs",
    "type" : "static",
    "data": {"segs":["name","taxid","address","business","creator","createAt"]}
}
```

4.3.8. var

定义一个或多个参数，在下一步可以当作普通参数使用，比如@{varName}。

```
{
```

```
        "name":"get_user_id",
        "type" : "var",
        "toResp" : true,
        "vars":{
            "uid":"@{SEQUENCE|'userid',i}"
        }
    }
```

4.3.9. 组合处理

一个接口可能由多个 processor 组合而成，比如用户注册接口，首先校验验证码，然后判断用户是否已经存在，最后才是将用户名、密码录入数据库。每个 processor 可以是基本的数据库操作，也开始调用其他服务的接口。

多个 processor 是逐个执行的，后面的 processor 可以使用前面 processor 的返回结果，通过@{!xxx}引用，与请求参数唯一不同的地方是在名称前加一个感叹号。

当前存在一个限制，如果多个 processor 都是数据库写操作，比如有 A、B、C 三个数据库写操作，当 A 成功，B 失败，则 C 不会执行，但是 A 写入的内容不会回滚。这一点，在接口设计时必须给以特别关注。

下面是创建用户的例子，首先判断是否存在，如果存在，则在第一步就返回错误码了；然后生成用户 id；再然后记录用户数据；最后产生模糊搜索的内容。

四个步骤中，任何一个步骤出错，都会直接退出，并返回错误码。比如第三步创建用户数据成功后，但是创建搜索数据失败，则创建用户失败，但是用户数据并不会回退。

```
{
   "name": "add",
   "method":"POST",
   "property" : "private",
   "feature" : "user",
   "aclChecker" : "RBAC",
   "tokenChecker":"USER",
   "comment":"添加新用户",
   "request": [
        {"name":"account", "type":"string", "must":true, "regular": "^[a-zA-Z0-9_]{1,40}$"},
        {"name":"password", "type":"string", "must":true, "min":1, "max":40},
        {"name":"nickName", "type":"string", "must":true, "min":1, "max":40, "comment":"昵称"}
   ],
   "process" : [
        {
             "name" : "judge_whether_user_exists",
             "type":"dataexists",
             "db":"user",
             "expect" : false, //如果存在，则返回 EXISTS，否则返回 OK
             "numSeg":"rowNum",
             "sqls" : [{
                  "name":"countUser",
```

```
            "metas" : "each",
            "merge":true,
            "multi":false,
            "sql":"select count(*) rowNum from user where account='@{account}'"
        }]
    },
    {
        "name":"get_user_id","type" : "var","toResp" : true,
        "vars":{"uid":"@{SEQUENCE|'userid',i}"
    }
    },
    {
        "name" : "register",
        "type" : "rdb",
        "db":"user",
        "sqls" : [
            "insert into user(id,account,nickName,pwd)
            values(@{uid},'@{account}','@{nickName}','@{PBKDF|6,password}')"
        ]
    },
    {
        "name" : "create_search",
        "type" : "search",
        "db":"user",
        "action" : "put",
        "table":"user",
```

```
            "did" : "@{uid}",
            "title":"@{account}",
            "summary":"@{nickName}"
        }
    ],
    "response":[
        {"name":"uid", "type":"int", "comment":"用户 id"}
    ]
}
```

4.4. 响应 response

4.4.1. 响应格式定义

如果对 response 没有做特殊转换，可以不用定义。比如，当需要对响应的字段做格式检查、转换、解密等情况时，必须定义。responose 可以是一个 json 对象也可以是一个 json 数组，其中字段定义与 request 中的字段定义形式相同。

比如，下面这段是会员中的/vip/get 接口的响应格式定义，因为 mobile 字段需要解密，所以需要定义 response 的格式。

```
    "response": [
        {"name":"creator", "type":"string"},
        {"name":"createAt", "type":"long", "comment":"建档时间"},
        {"name":"name", "type":"string", "comment":"VIP 称呼"},
        {"name":"birth", "type":"int"},
```

```
        {"name":"sex", "type":"string"},
        {"name":"mobile", "type":"string", "codeMode":"decode", "keyName":"vipKey"},
        {"name":"ext", "type":"json", "comment":"扩展信息，解析为 json"}
    ]
```

响应内容的解析是需要占用 CPU 的，如果不是特别需要，可以不用定义。考虑到有些服务希望自动生成文档，那么就需要定义 response 的字段，但是，可以设置在运行时不解析。这时就需要将 response 定义成一个 json 对象，例如：

```
    "response":{
        "check":false, //默认为 true，即，只要定义了 response，就默认解析
        "segments":[
            {"name":"ver", "type":"int", "comment":"版本"},
            {"name":"serviceId", "type":"int", "comment":"服务 id"},
            {"name":"digest", "type":"string", "comment":"版本校验码"},
            {"name":"updateAt", "type":"string", "comment":"更新时间"},
            {"name":"features", "type":"string", "list":true, "comment":"更新的点"}
        ]
    }
```

如果中间处理过程产生了响应内容，但是又不希望它们返回，可以定义一个空的 response。

```
    ″response″:[]
```

4.4.2. 响应内容

所有响应的顶层结构都是一样的，包括返回码 code、信息 info，如果是查询类的请求，会包括数据 data 字段，每个查询类接口的 data 都不相同。data 中存放的内容就是在 response 中定义的。

```
{
    code:0,
    info:"Success",
    data:{
        a:1,
        b:"xxx",
        c:{...},
        d:[...]
    }
}
```

1) 如果定义了返回格式，在返回前，只返回定义了的字段内容，并且检查其合法性，其他内容都会丢弃；

2) 如果无 response 定义，则不会做任何过滤，处理中返回什么内容，全部返回；

3) 如果是一个长度为 0 的 response，则会丢弃所有内容，如："response":[]。

4.4.3. 返回码

响应体中的 code 为返回码，如果无错误则为 OK(0)，返回码在 js 脚本中用 RetCode.xx 可以直接引用，code 定义如下：

名称	值	含义
OK	0	成功
DEPRECATED	1	接口即将废弃
INTERNAL_ERROR	100	内部错误
INVALID_TOKEN	102	无效 token
EMPTY_BODY	103	请求体错误，用在 POST 请求中
DB_ERROR	104	数据库错误
INVALID_SESSION	105	无效的 session
SERVICE_NOT_FOUND	106	服务不存在
TOO_BUSY	107	系统太忙
SYSTEM_TIMEOUT	108	系统超时
NOT_SUPPORTED_FUNCTION	109	API 存在，但是所需的功能不支持
API_NOTFOUND	110	API 不存在
NO_RIGHT	111	无权调用
NO_NODE	112	找不到可用的节点提供服务
INVALID_NODE	113	无效的节点，比如数据库分片的情况下，请求发到错误的 webdb 实例上
THIRD_PARTY_ERR	114	调用第三方服务失败
UNKNOWN_ERROR	150	未知错误
EXISTS	2000	已经存在
NOT_EXISTS	2001	不存在
API_ERROR	3000	API 错误
WRONG_JSON_FORMAT	3001	JSON 体解析失败
INVALID_VERSION	3002	版本错误
DATA_WRONG	3003	数据错误
WRONG_PARAMETER	4000	参数错误，在参数定义中列表中，第几个参数错误，就加多少，比如第一个参数错误，返回 4001，以此类推
SERVICE_ERROR	5000	业务相关错误，可以自定义

5. 占位符

在 sql、js 脚本，以及一些配置项中（比如 searchdb、 treedb 的 action、 title、 when 中），可以引用请求参数、变量、响应参数、系统参数、请求头。

格式	类型	说明
@{xxx}	请求参数	引用参数列表中的字段，如果引用了不存在的请求参数，启动时会失败；可以包含'.'，表示多级引用
@{^xxx}	请求头参数	引用 http 请求头中的字段
@{!xxx}	响应参数	前面处理的响应内容；可以包含'.'，表示多级引用
@{#xxx}	系统参数	1)tokenCaller、tokenCallee、tokenPartId、tokenAcc、tokenCid、tokenExt：token 中的信息，在私有接口中才有； 2)reqAt 参数：接受到请求时的 utc 时间戳； 3)shard 分片号：从接口配置的 sharding 字段计算得出； 4)result：上一步的执行结果，与 convert 结合使用才有意义，因为任何一个处理只要返回值不是 OK，则整个处理就终止了，不会走到下一步。
@[!xxx]	前面步骤执行的响应内容	通常用在 RDB 处理中使用。当有多个 sql 时，上一个 sql 查询处理完毕，下一个 sql 可以使用上一个 sql 的结果集，比如 @[!UserNum]。 【注意】这种参数在请求端不会被编译替换，而是在 webdb 中执行时才会被替换，所以对性能有少许影响

单纯的变量不能够满足某些特定的功能，比如要对字段加解密，这时需要用到一些函数，使用时，将函数名放在参数前面，并用"|"分隔，参数可以是请求参数、响应参数，也可以是系统参数、请求头，比如:

@{HASH|#token...,para,!resp,1,'xxx'}

下表是系统可以支持的函数占位符:

名称	功能	举例	说明
HASH	计算 HASH 值	@{HASH|#token...,name, 1,'xxx'}	返回 HASH 值，HASH 算法与 Java 保持一致；如果有多个参数，它们之间使用"-"连接；默认为 long 型，如果第一个参数是"i"或"int"，则返回 int 型 hash 绝对值
ABSHASH	计算绝对 HASH 值	@{ABSHASH| #token...,name, 1,'xxx'}	返回 HASH 绝对值，HASH 算法与 Java 保持一致；如果有多个参数，它们之间使用"-"连接；默认为 long 型，如果第一个参数是"i"或"int"，则返回 int 型 hash 绝对值
HASHMOD	计算 HASH 绝对值，并求余	@{HASHMOD|mod,#token...,name, 1,'xxx'}	将参数进行 HASH 计算后得到一个整型绝对值，得数与 mod 求余；如果有多个参数，它们之间使用"-"连接；HASH 算法与 Java 保持一致

名称	功能	举例	说明
NOW	当前时间	@{NOW|yyyy-MM-dd HH:mm:ss}、@{NOW|unit86400000}转换成 UTC 天数	当前 UTC 时间戳，与@{#reqAt}是同一个值，在一次请求中，多次引用@{#reqAt}或@{NOW}，结果都相同； @{NOW|fmt}可以携带格式化信息，但是@{#reqAt}不可以； fmt：yyyy-MM-dd HH:mm:ss、hex（16 进制形式）、unitxxx（unit 后面指定毫秒数，比如按天为 86400000，按小时为 7200000）
MD5	计算 MD5	@{MD5|#tokenxxx, name, 1, 'xxx'}	格式类似 HASH，可有多个参数，它们之间用"-"连接，输出一个 base64 编码的字符串。
SHA256	计算 SHA256		类似 MD5，只是算法不同
HMACSHA256	计算 HMACSHA256	@{HMACSHA256| para1, name, 1, 'xxx'}	类似 MD5，只是算法不同；算法中的可以是随机生成的 16 字节内容，记录在结果的前 16 字节；在 js 脚本中可以使用 Secure.hmacSHA256Check(str, savedStr)进行校验，其中 savedStr 就是此处生成的字符串
PBKDF	计算	@{PBKDF|	iter 为迭代次数，para 为被混淆的字符串；在 js 脚本

名称	功能	举例	说明
	PBKDF2	iter,para}	中可以使用 Secure.pbkdf2Check(str, savedStr)进行校验，其中 savedStr 就是此处生成的字符串，也可以用进行校验，返回 true 或 false
PBKDFCHECK	PBKDF2 校验	@{PBKDFCHECK| str, savedStr}	str 为传入参数，savedStr 是用来检验的参数，比如从数据库取出
CONCAT	连接	@{CONCAT| para1, '-', para2, '-'...}	连接多个参数
COALESCE	返回第一个非空值	@{COALESCE| para1, para2, ''}	如果 para1 为空，则返回 para2，如果 para2 也为空，则返回空字符串
IFVALID	非空则返回	@{IFVALID| para1, 'xx-', para2}	如果 para1 为空返回''，否则返回"xx-para2"，用于解决 sql| 不能处理 java 的 null 问题
IFNULL	非空则返回	@{IFNULL| [!]para1, 'null'[, num]}	如果 para1 为空返回'null'字符串，否则 para1 的值；如果指定为 num，则不会加引号
UUID	产生 UUID 字符串	@{UUID| 16}、@{UUID| 64}	可以指定输出格式，16 表示 HEX 方式，64 表示 base64 方式

名称	功能	举例	说明
SRCIP	请求的源地址	@{SRCIP|remote}、@{SRCIP}	设置了 remote 表示返回地址考虑了 nat 转换，否则返回链路中上一跳的地址
UNIQUEID	先产生 UUID 字符串，然后输出该字符串的绝对 HASH 值	@{UNIQUEID|int}、@{UNIQUEID|l}	默认为 long 型，如果有参数"i"或"int"，则返回 int 型 hash 绝对值；经测试，int 型有千分之一的重复率，long 型约百万分之一的重复率
ENCODE	数据加密	@{ENCODE| keyName[#keyTime], paraName}	keyName 指定密钥的名称，可以加'#keyTime'，指定密钥最大有效天数，默认为 366 天，到期后会产生新的密钥，但是老密钥仍然可以解密；在一些安全性要求很高的场景中，可以设置较短的有效期。 在 js 或 sql 中可以通过@{DECODE| keyName, paraName}解密。也可以在参数配置中将 codeMode 设为 decode，并且设置 keyName
DECODE	数据解密	@{DECODE| keyName, paraName}	keyName 指定了密码的名称，无需事先创建密钥，运行时，如果无密钥则创建它
CONFIG	服务级配置	@{CONFIG|c	configItem 指定配置项名称，实际存储时会在前面增

名称	功能	举例	说明
	项	onfigItem}	加"para_"前缀
UPPER	转大写	@{UPPER| paraName}	将参数转为大写
LOWER	转小写	@{LOWER| paraName}	将参数转为小写
SUBSTR	取子字符串	@{SUBSTR| pname, 0, 2} 取字符串参数前面两个字符	@{SUBSTR| paraName, start[, len]} start 开始位置，len 指定子字符串的长度，可以未指定，则表示从 start 到末尾，如果 len 超过字符串末尾，则取到末尾为止
LIST	将 LIST 连接为字符串	@{LIST| paraName[, segName]}	将一个 list 内容链接成字符串，多个元素用逗号","分隔。用于解决 NORMAL 中数组自动加"[]"的问题，加了"[]"，在 sql 中就无法使用。如果 list 中元素是对象，可以指定字段名，处理时取出每个对象的指定字段，加","组成一个字符串
CLEAN	清除 JSON 中的字段名称	@{CLEAN|json}	只可用于 JSON 类型的参数，将 json 字段名全部清除，返回一个字符串。通常用在生成全文索引中
SIZE	返回数值参数的长度	@{SIZE| [!]para}	para 必须是一个数组

名称	功能	举例	说明
SEQUENCE	持续增长的id，不保证连续	@{SEQUENCE|i,'customer'}、@{SEQUENCE|customer,cidParaName}	第一个参数指定类型，有 i/int、l/long 两个选择，不输入则默认为 int；第二个参数是名称，可以加单引号，也可以不加，在一个服务内必须唯一；如果在公共接口中使用，需要提供 cid 参数的名称
ADD、SUB、MULTI、DIV	加减乘除	@{ADD| 类型, para1, para2}	类型有:int、long、float、double，para1 与 para2 必须是对应类型的数字
VER2INT	将字符串版本号转为一个整数	@{VER2INT|'11.22.33'}	版本号的没段存成十进制数的 3 位，比如例子中转为整数 11022033，所以版本号中每段不能超过三位数
RANDOM	产生随机数	@{RANDOM| l/d/f/c/s, min, max]}@{RANDOM|s,len]}>	l：长整型，d：双精度浮点数，f：单精度浮点数，c：字符（0-65535）。min、max 指定最小、最大值；s：字符串，len 指定字符串长度
JSON	将复杂对象转为 JSON	@{JSON| para, quote,	para 为任意类型的参数，可以指定引号

名称	功能	举例	说明
	串	safequote}	
SPLIT	字符串切割	@{SPLIT|para,len_or_splitterChar,spliter}	将 para 参数按固定长度 len 切割成多段 (不足 len 的不会填充尾部）；或者通过分隔符分成多段；分隔后再使用分隔符 spliter 连接起来，连接时会加上合适的引号

6. 认证&鉴权

6.1. 服务间认证&鉴权

服务间调用如果不加限制，会导致滥用却难以定位的问题。认证后，可以清晰地知道请求方是谁，并能做相应的限制，比如流控、鉴权等。

当接口定义的 property 中有 private 属性，则在服务接口定义中可以指定 tokenChecker 来对请求鉴权，tokenChecker 有以下几种：

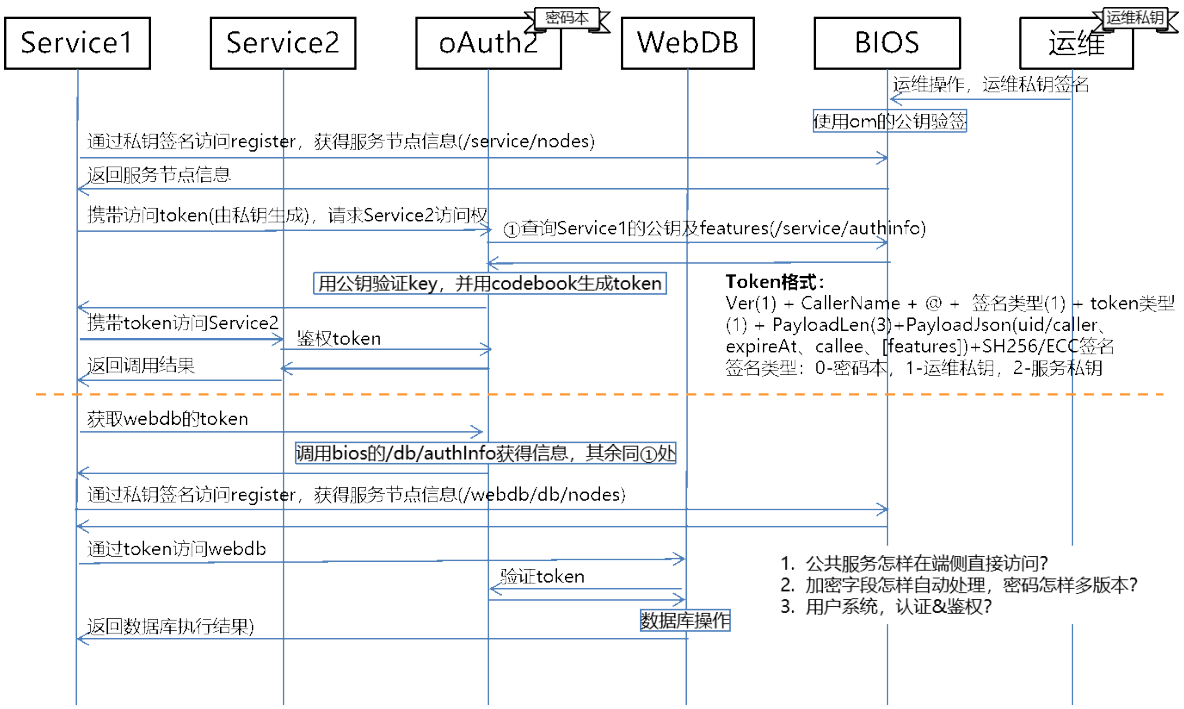
名称	描述
OAUTH	服务间认证，必须在管理台设置调用关系；在安卓服务器中，启动时已根据 service.cfg 中申明的依赖关系，自动设置
OM	只容许 OM 管理台调用
APP	同一个服务内部不同接口之间的互相调用，比如 webdb 的数据同步接口； 如果是"**APP-允许的调用方服务名"，**则表示只允许某个服务调用此接口，这点极大地方便了服务回调，调用方服务调用时，必须使用自身的私钥签名； 也可以指定为 APP-*，表示任何服务都可以调用

NODE	容许同一内网环境中其他节点调用本接口，比如每个节点都要给统计服务上报运行情况时，可以使用此认证

服务间 OAUTH 认证是最重要的一类认证，APP(调用方服务名)、OM 也是服务间认证，除了实现不同，其他是一样的。 以下主要介绍服务间 OAUTH 认证。

6.1.1. 认证

OAUTH 依赖 oAuth2 服务，如下图，Service1 访问 Service2：



1. Service1 先用自己的私钥签名生成 AppToken，访问 oAuth2；

2. oAuth2 从 BIOS 中获得 Service1 的公钥与可访问的 features 列表，并用公钥验证 AppToken；

3. 如果通过，则用自己的 codebook 生成 Service2 的 token 返回给 Service1；

4. Service1 用这个 token 调用 Service2 的接口，Token 中包括分区、调用方、被调方、超时时间、可调用 features 列表等信息；

5. Service2 接到请求后，向 oAuth2 验证 token 是否有效，通过后才会执行后继操作。

以上的 Service1 获取 token、oAuth2 获取公钥、Service2 验证 token，在服务中都会做缓存，不会每次请求都完整走一遍 oAuth 过程。

oAuth2 服务使用的密码本，在安卓服务器中，第一次启动时生成，每个实例都不相同，保证不同私有云服务器之间不能互访，以此来解决多家公司在同一个局域网的问题。

6.1.2. 鉴权

在 token 中携带了 Service1 可以访问的 features，Service2 中通过判断接口的 feature 来判断 Service1 是否可以调用该接口。

6.1.3. 数据库访问

数据库是一种特殊的服务，但是认证操作与普通服务类似，只是 token 中的 callee 字段填写的是 db 的名称，features 填写“*”。

因为业务只能访问自己的数据库，所以不做 C、R、U、D 的权限限制，也就是说，业务对数据库具备所有权限，但是不建议数据库执行 DDL 类 SQL，DML 类 SQL 不建议单次做大批量操作。

数据库有 OM 接口，可以指定数据库只读、可读写。只读状态下，写入都会失败。此特性可用于数据库升级等 OM 操作时。

/webdb/api/om/setWritable?service=xxx&db=yyy&writable=trueORfalse

6.2. 用户认证&鉴权

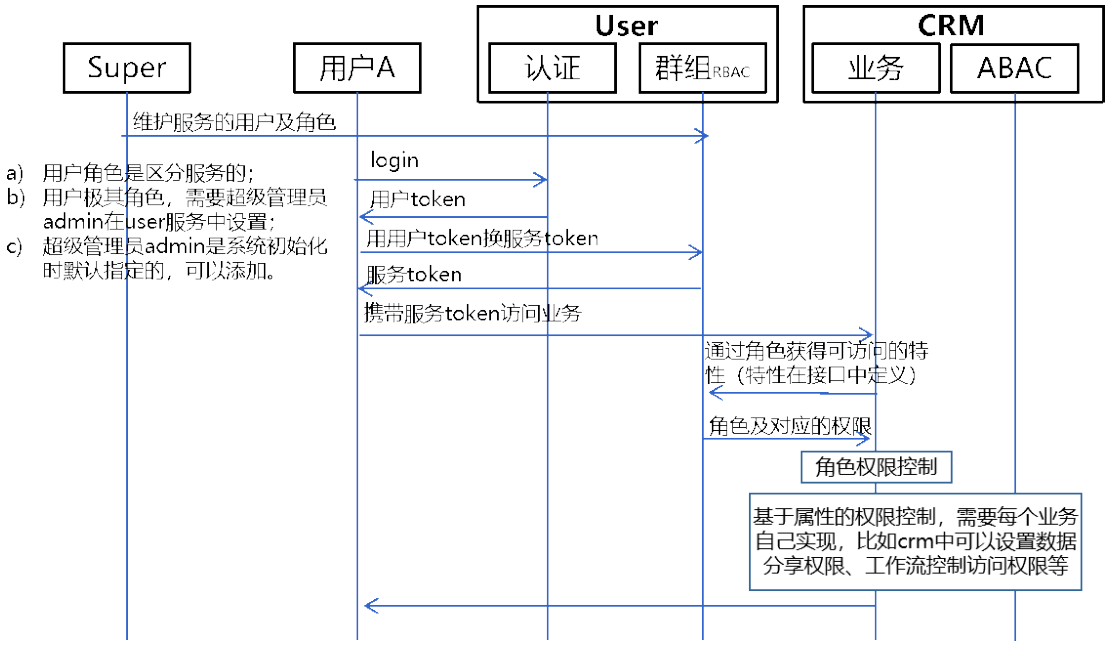
名称	描述
USER	端侧公司用户调用服务侧接口时使用，端侧必须登录了一个公司帐号才可以通过
UNIUSER	端侧个人用户调用服务侧接口时使用，端侧必须登录了个人帐号才可以通过分详细描述

接口定义的 property 中有 private 属性，公司服务的 tokenChecker 为 USER，个人服务 tokenChecker 为 UNIUSER。 这样的接口，需要用户输入账号、密码登录后才可以访问。

【注意】UNIUSER 只能在至简网格服务端提供的个人服务中使用，企业服务中需要进行用户认证 USER。

6.2.1. 认证

公司服务的用户认证通过 user 服务实现，个人服务的用户认证通过 uniuser 服务实现，包括登录、验证等基本操作，uniuser 还包括注册功能。



上图示例展示 CRM 服务的用户认证鉴权过程：

1) 用户从端侧向 CRM 服务发起请求前，需要输入用户名、密码，获得用户 token；

2) 如果是访问 user 服务本身的接口，携带用户 token 即可访问；

3) 如果访问 CRM 服务，则需要拿用户 token 向 user 服务换取服务 token；

4) 携带服务 token 请求 CRM 服务接口，CRM 服务根据 token 中的用户信息，向 user 服务获得该用户的角色确定用户可以调用哪些 feature 的接口；

5) 如果权限确认通过，才会执行后继的业务逻辑。

6.2.2. 鉴权

鉴权分为 RBAC(Role Based Access Control 基于角色的接入控制)与 ABAC(Attribute Based Access Control 基于属性的接入控制)。

6.2.2.1. RBAC

在至简网格中，RBAC 在 user 服务中实现。在 user 服务中为服务添加用户时，需要指定角色。角色是服务实现时定义的，在其中指定角色可以访问的接口范围，通过服务的/roles 接口提供给 user 服务。

```
"roles": {
    "admin":{
      "name":"企业主",
      "rights":{
```

```
            //sku 是接口定义文件的名称(sku.cfg)，* 表示其中的所有特性的接口都可以调
            用
            "sku":"*",
            //如果接口定义中指定了 feature，就可以更加细致的授权
            "report":"featureA,featureB...",
            "proxy":"*"
        }
    },
    "sales":{
        "name":"销售",
        "rights":{
  //这里没有  指定任何接口文件，则，只能访问没有设置 feature 的接口
        }
    }
    ...
}
```

为了实现对角色功能更加细致的限制，在每个接口中都可以定义 feature，在角色定义时， 限制角色在某个接口定义文件中，只能执行特定的几类接口。详情请参照 接口定义

6.2.2.2. ABAC

ABAC 的权限控制更加精细化，与业务特征紧密相关，难以提供统一的实现，每个服务需要自己实现。比如，CRM 中有独立的 powers 表，控制每个用户可以访问哪些数据，权限控制达到行级别。比如，数据分享以及工作流赋权，都可以控制到单个客户、联系人、订单级别。

7. 数据库开发

7.1. 数据库定义

至简网格支持三种数据库，分别是 RDB、SDB、TDB，其中 RDB 关系型数据库最为常用，分为本地与通用两种。

1) 通用（在服务根目录的 database.cfg 文件中定义）：由 webdb 服务统一管理， 所以可以跨实例访问，支持数据跨实例同步与定期异地备份。

2) 本地（在 database.loc.cfg 文件中定义）：这类数据只存在于本服务目录， 数据库实例存在数据库根目录 dbs 下，比如 address 服务的数据库就是这类。 因为是存在本地的，服务接口实现时是直接操作的，所以它性能更好，但是它不会同步与定期备份， 适合存放需要经常访问，但是极少变更的数据。

7.1.1. SDB 搜索数据库

实现对内容进行分词以及模糊搜索的功能，使用方法请参照 “处理”部分的描述。不涉及表结构定义，只需要在其中申明即可，type 设为 sdb，如下所示：

```
{
    "name":"crm",
    "type":"sdb"
}
```

7.1.2. TDB 树状数据库

实现树状关系数据的增删改查，使用方法请参照“处理”部分的描述。不涉及表结构定义，只需要在其中申明即可，type 设为 tdb，如下所示：

```
{
    "name":"crm",
    "type":"tdb"
}
```

7.1.3. RDB 关系型数据库

实现关系型数据的增删改查，使用方法请参照“处理”部分的描述。涉及多个版本表结构升级或定义：

```
{
    "name":"crm",
    "version":"0.2.0", //升级后的目标版本
    "type":"rdb",//固定为 rdb
    "versions":[] //每个版本对应 map 对象
```

}

versions 中可以有多个 map 对象，在执行时会判断本地版本是否在 minVer（包括）、maxVe（包括）所指定的范围之中。如果包括，则执行其中 sqls 中每个数据库脚本。

每行脚本最好只指定一条 DDL 语句，多条 DDL 语句指定多个 SQL 执行。

DDL 语句执行完毕，会将本地数据库版本号改为 toVer，然后再继续后面 version 执行。

```
{
    "minVer":"0.0.0", //最新
    "maxVer":"0.1.0",
    "toVer":"0.2.0",
    "sqls":[]
}
```

7.2. 数据分片

数据量较小时（Sqlite：<1 百万行记录，MySQL：<1 千万行记录）不必分库，大数据量时建议分库。 单个数据库太大会引起性能下降、维护困难。

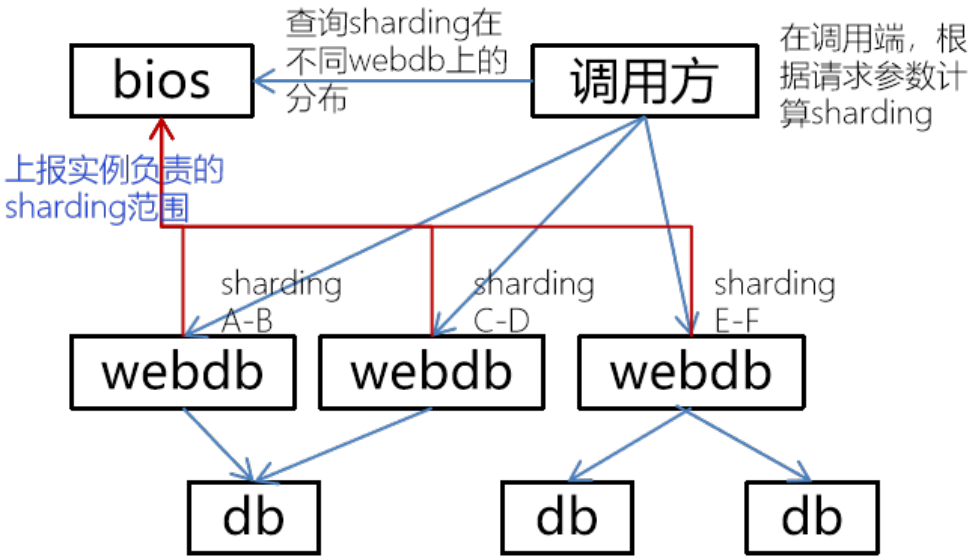
分库是针对某个服务的某个数据库的。分库需要计算每行数据的分片号，并将其分配到不同的数据库中。 至简网格的分片号范围为大于或等于 0，小于或等于 32767，也就是最大支持 32768 个分片。

多个分片可以放在一个库中，也可以一个分片单独放在一个库中，即，最多可支持 32768 个分库，如果一个分片存放最大 1 百万行记录，最大可容纳约 327 亿行记录。

分片号可以用多个字段共同计算得到，分片号计算结果只能是一个整型数，所以通常要用到 ABSHASH 占位符，计算字段可以是请求参数、系统参数或前面处理的响应结果，比如：

"sharding" : "@{ABSHASH | account, #tokenCaller, !custId, ^agent...}"

其中的 account 是请求参数，或者接口中定义的变量；#tokenCaller 是 token 中的字段；!custId 是前面的响应结果；^agent 是请求头中的字段。 参数定义请参照占位符的介绍。数据库分片的实现原理，见下图所示：



每个 webdb 实例负责一个分片范围，在它启动后会定期向 bios 服务上报自己的分片范围，调用方发起请求时需要先从 bios 中获得分片分布情况， 然后再根据接口定义中 sharding 计算结果，找到合适的 webdb 实例。

【注意】

1) webdb 只管记录数据，如果调用端选择了错误的分片号，webdb 只会返回 INVALID_NODE(113)错误，此错误需要调用方自己处理；

2) 如果分片信息发生了调整，需要重启相关的 webdb 实例，调用方需要等待几分钟才会更新分片分布信息，如果需要及时调整，调用方也需重启；

3) treedb 不支持分片。

8. 高阶开发

如果 sql 脚本、js 脚本已不能满足业务要求时，则需要做 java 开发。 比如系统内置的 user、oauth、webdb 等服务，都内置了 java 实现的逻辑。 这时需要用 java 实现 IProcessor 接口，或者继承 RDBProcessor、TreeDBProcessor 等类进行扩展。 在 process 中，指定 handler 为自定义的实现类，比如：

```
{
    "name" : "get_token",
    "type" : "java",
    //因为 type 为 java，所以 SampleDBProcessor 必须继承自 AbstractProcessor，
    或者 IProcessor
    "handler" : "cn.net.zhijian.mesh.builtin.xsv.SampleDBProcessor"
}
{
      "name" : "get_token",
      "type" : "rdb",
      //因为 type 为 rdb，所以 SampleDBProcessor 必须继承自 RDBProcessor
      //类似的情况，比如 treedb、search 必须分别继承自 TreeDBProcessor、
   SearchProcessor
```

"handler" : "cn.net.zhijian.mesh.builtin.xsv.SampleDBProcessor"

}

至简网格中内置了加载第三方 jar 的能力，但是，因为安卓中需要对 jar 做转换后才能加载，对研发人员有一定的要求， 所以暂时没有放开此能力。如果对此有强烈的需求，请联系至简网格，有两种方法可以解决此问题：

1) 至简网格将功能内置到系统中；

2) 放弃兼容性，只提供 java 版本，这是我们不愿意看到的。

9. 基础服务

9.1. 公司帐号服务

维护一个企业内部的用户数据，包括对用户数据、群组数据的增删改查，以及用户授权。

9.1.1. 用户数据维护

系统初始化时，已经创建了超级用户 admin，密码默认为"123456"， 建议第一次使用时就更改这个密码，并且记住它。

超级用户可以对用户数据进行增删改查，admin 可以初始化任何一个用户的密码（解决忘记密码的情况）， admin 也可以添加其他的超级用户，但是 admin 不可以删除自己。

9.1.2. 群组数据维护

admin 可以对群组数据增删改查，调整结构；一个群组的管理员可以对下一层群组进行增删改查。群组功能在当前系统中使用较少，业务可以自行决定怎样使用群组数据。

9.1.3. 用户授权

只实现按角色的授权(RBAC)，管理员在为每个服务添加用户时，可以指定它在其中的角色， 根据角色决定用户可以执行哪些操作。

为了实现这点，需要服务在实现时，对接口做功能划分， 并在 角色定义中指定可执行的功能范围。

给用户授予某个业务系统中的角色时，还可以指定是否可以外网接入的权限。 此权限在开通外网访问的情况下有效，只有开通此权限的用户才可以在公网环境访问公司内网的服务。

用户在调用服务接口时，必须携带服务 token，此 token 会在用户系统中进行验证，通过后才可以访问。

9.2. 个人帐号服务

面向个人的用户系统，它包括"公司帐号服务"的绝大部分功能。在端侧，可以同时登录个人帐号与多个企业帐号。个人帐号服务与公司帐号服务的区别有：

(1) 不实现外网接入控制，因为本身就在外网；

(2) 一个端侧能有零个或一个个人帐号登录，但是可以有多个企业帐号登录；

(3) 帐号通过注册获得，而不是管理员添加；

(4) 只有面向个人用户的服务才可以使用个人帐号，比如密码箱、计算器、专注力等，就是面向个人的服务。

9.3. 序列 ID 服务

实现一个持续增长（不保证连续）的 ID 服务，通过 SEQUENCE 占位符获得。此占位符可以用在 sql、js 脚本中，也可以用在 vars 的 val 中， 或者 var 处理中。

9.4. 定时任务

如果需要定期执行一个任务，比如定期备份等，可以在 service.cfg 中申明依赖 schedule 服务完成。

定时任务是通过定期调用服务提供的接口实现业务需要的功能。此接口不能占用太长时间，否则会堵塞定时任务执行。 如果此任务需要占用很长时间，建议在接口中先返回 RetCode.EXECUTING(2)，然后启动一个线程执行耗时的任务。 等任务完成后，再调用 schedule 的/callback 接口，传回 taskId、code、info 三个参数，异步告知执行结果。 在任务执行期间，schedule 每分钟会检查服务是否已经返回了结果，如果返回 code 为 RetCode.OK(0)， 则本轮定时任务结束，否则会在下一次重试时间到达时，再次发起调用。

在服务的初始化接口中调用 schedule 的/task/create 接口创建定时任务，此接口可以多次调用，以最后一次为准。 创建定时任务的请求的 tokenSign 为 APP，请求参数如下：

名称	说明	举例
name	任务名称，必须为数字字母下划线	test123
type	周期类型，有 D(天)、W(星期)、M(月)、C(周期性)	D
val	从起点推后的分钟数，D/W/M:离周期起点间隔，C:周期的分钟间隔	比如 type 为 D，val 为 540，表示每天 9 点执行；type 为 W，val 为 1440，表示每周一 0 点执行
minTime	最小重试时间间隔，单位为分钟，每重试一次翻一倍；	1

名称	说明	举例
	此参数的大小取决于定时任务耗时长短，比如预计最长 5 分钟完成， 则设置为 5 分钟，通常设置时要保有一定的余量，比如预计耗时 5 分钟，设置为 10 分钟	
maxRetry	最大失败重试次数，每重试一次减 1，到 0 时，本轮周期内停止尝试	3
url	给定时服务调用的服务接口 url，特别注意：此接口的 tokenChecker 为"APP-schedule"，且必须尽快返回，否则会堵塞定时任务服务	/om/backup

9.5. 验证码服务

当前只提供图片验证码。

1) /image?w=xx&h=yy：返回一个 base64 形式的验证码图片 img，与一个 session；

2) /verify?session=xxx&code=yyy：在服务端验证输入是否正确，session 是/image 接口返回的。

9.6. 配置服务

存放 K-V 形式存储的配置，每个公司是独立的，所以请求必须是公司级的。 端侧公司用户发起请求，会在请求头中携带 cid，服务端接到请求后，将 cid 通过头部再转给 config 服务。

提供了 put、putIfAbsent、remove、get、list 接口， 详细定义请直接在码云、Github、CSDN 中查看接口定义文件 config/api/root.cfg。

9.7. 工作流服务

工作流中可以定义一个工作流程中步骤，在业务中控制工作的推进，工作可以向下一步推进， 也可以回退到上一步。每一步可以写入当前责任人的意见，并指定下一步的执行人（可多人）。 每一步操作，包括回退，都有详细的记录，在需要回溯工作时，可以清晰地查看每一步的记录。

详细接口定义请直接在码云、Github、CSDN 中的 workflow 里查看。

/workflow/api/flow.cfg：定义工作流的相关接口，这类接口在业务的管理台中调用，工作流服务提供了默认的管理页面，业务中可以直接引用，样例请参照/crm/file/index.html。router 定义时，引入"/workflow/file/pub/settings.js"即可

```
{path: '/flowdef', component:() => import('/workflow/pub/settings.js')}
```

/workflow/api/root.cfg：启动或删除工作流、确认工作、查询任务的接口，这类接口在业务中调用，具体样例可以参照码云、Github、CSDN 中的/crm/api 下的一系列接口定义。

客户审批工作流

名称	南京建工
统一信用码	299898999999999999
地址	秦淮区
主营业务	建筑
创建人	admin
创建时间	2022/11/17 14:36:47

2022/11/17 14:36:47

创建客户(创建客户信息)

admin

2022/11/17 14:36:47

2022/11/25 15:03:20

区域销售主管确认(区域销售主管确认)

admin

2022/11/25 15:03:20

ok

2022/11/25 15:03:20

参与人决策(参与人确认是否可以参与)

admin

意见

权签人

下一步 返回

lgy

未处理