QCM管理接口说明

日期	版本	作者	Email	备注
2018/08/28	0.1	刘伟	liuwei3-s@360.cn	创建文档
2018/08/31	0.1	刘伟	liuwei3-s@360.cn	第一版完成
2018/11/09	0.2	刘伟	liuwei3-s@360.cn	完善测试中发现的问题
2019/01/14	0.3	刘伟	liuwei3-s@360.cn	增加批量添加和删除方法

一. QCM与Hulk交互概要

1. 基本还是参考QBus的模式, QCM有名为 `Admin Server` 的组件,提供与QCM服务相关的基础接口;
2. `Admin Server` 不涉及任何的业务相关信息,即与具体业务无关;
3. Hulk依赖 `Admin Server` , 但 `Admin Server` 不依赖Hulk.

二. 用户申请配置相关

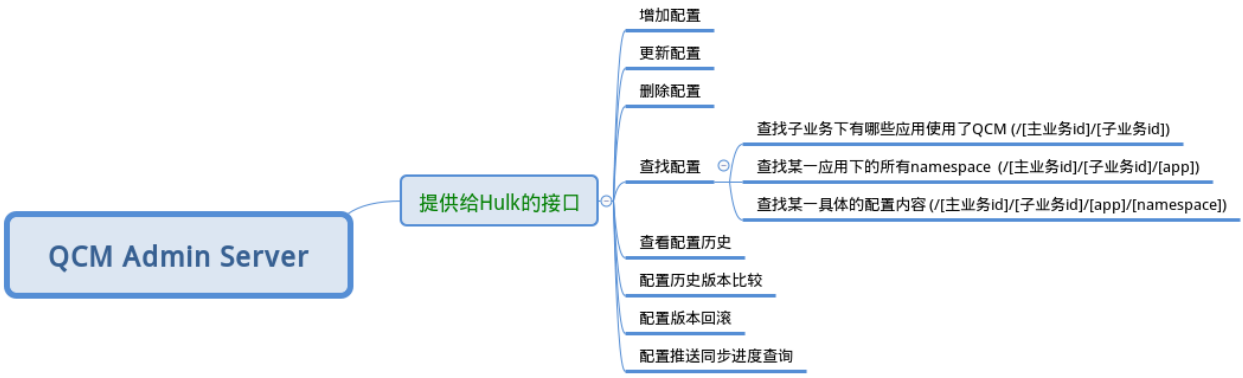
1. 用户申请配置必须首先选定 `子业务` ;
2. 用户需提供当前申请的配置对应的应用或服务, 即 `AppName` ;
3. 对同一应用的配置, 可能会存在于不同环境中,比如开发环境, 测试环境, 线上环境,我们称之为 `Namespace` , 用户在申请时需确定好;
4. 以上3点综合起来,即组成了用户申请的配置的ID, 这个ID也是这个配置在QCM系统中的存储Key

```
/[子业务ID]/[AppName]/[Namespace] 注：hulk上子业务id是全局唯一，且子业务迁移时其id不
```

三. QCM AdminServer相关接口

1. 所有接口均为Http接口,数据均使用 `JSON` 格式传输;

2. 先给出接口描述:



注：上图中的 `主业务ID` 已不再需要

四. 接口调用方式

1. 所有接口均以http接口形式给出；
2. 对于需要较少参数的接口，使用http get方法;
3. 对于需要较多参数的接口，使用http post方法，参数以json形式给出;
4. 返回值均为json形式，包含errcode和errmsg字段;
5. **对于配置项的内容, 从hulk传递到QCM时hulk需作urlencode, 从QCM返回到hulk时,hulk需作urldecode**;
6. 支持jsonp的跨域调用;
7. 支持basic验证.

五. 错误码说明

```
0:          "ok",
1:          "some parameters are empty",
2:          "config already exists",
3:          "internal error for etcd client",
4:          "failed to parse request",
5:          "config is not exists",
6:          "internal error for redis client",
7:          "failed to marshal etcd data",
8:          "failed to parse json parameter",
9:          "key is invalid",
```

六. Config相关接口

1. 添加新的配置项
   1.1 接口:

```
post /config/add
```

1.2 参数:

```
{
 "config_key":"/[子业务ID]/[AppName]/[Namespace]",
 "config_value":"配置的具体内容",
 "config_type":"text|json|xml|...",
 "config_operator":"作此操作的hulk用户账号"
}
```

1.3 返回值:

```
{
 "err_code":0为成功,其他为失败,
 "err_msg":"成功|失败的原因",
 "user_key":"返回给用户的key, 用户在使用sdk获取配置时,需要传入此key, 一般格式为:/[子
 "config_reversion":当前配置项的版本号,int64整型
}
```

1.4 说明: 如果调用此接口时,所新增的配置项已存在,则返回失败

2. 更新已有配置
   2.1 接口:

```
post /config/update
```

2.2 参数:

```
{
 "config_key":"/[子业务ID]/[AppName]/[Namespace]",
 "config_value":"配置的具体内容"
 "config_type":"text|json|xml|...",
 "config_operator":"作此操作的hulk用户账号"
}
```

2.3 返回值:

```
{
 "err_code":0为成功,其他为失败,
 "err_msg":"成功|失败的原因",
 "config_key":"",
 "config_reversion":更新后配置项的版本号,int64整型
}
```

2.4 说明: 如果调用此接口时,需要更新的配置还不存在,会返回错误,更新失败

3. 删除配置项
   3.1 接口:

```
post /config/delete
```

   3.2 参数:

```
{
 "config_key":"/[子业务ID]/[AppName]/[Namespace]"
}
```

   3.3 返回值:

```
{
 "err_code":0为成功,其他为失败,
 "err_msg":"成功|失败的原因"
}
```

   3.4 说明: 如果调用此接口时,需要更新的配置还不存在,会返回错误,更新失败

4. 查找子业务下的所有应用
   4.1 接口:

```
post /config/getapplist
```

   4.2 参数:

```
{
 "config_key":"/[子业务ID]",
}
```

   4.3 返回值:

```
{
 "err_code":0为成功,其他为失败,
 "err_msg":"成功|失败的原因",
 "config_key":"request中的config_key",
 "applist":[
 "app1",
 "app2"
 ]
}
```

5. 查找app下所有namespace的配置

5.1 接口:

```
post /config/getconfiglistbyapp
```

5.2 参数:

```
{
 "config_key":"/[子业务ID]/[appname]",
}
```

5.3 返回值:

```
{
 "err_code":0为成功,其他为失败,
 "err_msg":"成功|失败的原因",
 "config_key":"request中的config_key",
 "ns_config_list":[
     {
         "namespace":"xxx",
         "config_type":"xx",
         "config_value":"xxx",
         "config_operator":"xxx",
         "config_timestamp":"xxxxxx",
         "config_reversion:版本号,int64整型
     },
     {
         "namespace":"xxx",
         "config_type":"xx",
         "config_value":"xxx",
         "config_operator":"xxx",
         "config_timestamp":"xxxxxx",
         "config_reversion:版本号,int64整型
     }
 ]
}
```

6. 获取某一具体配置

6.1 接口:

```
post /config/getconfig
```

6.2 参数:

```
{
 "config_key":"/[子业务ID]/[appname]/[namespace]"
```

```
}
```

6.3 返回值:

```
{
 "err_code":0为成功,其他为失败,
 "err_msg":"成功|失败的原因",
 "config_key":"request中的config_key",
 "config":
     {
         "config_type":"xx",
         "config_value":"xxx",
         "config_operator":"xxx",
         "config_timestamp":"xxxxxx",
         "config_reversion:版本号,int64整型
     }
}
```

7. 获取某一配置的历史记录
   7.1 接口:

```
post /config/gethistorylist
```

7.2 参数:

```
{
 "config_key":"/[子业务ID]/[appname]/[namespace]"
}
```

7.3 返回值:

```
{
 "err_code":0为成功,其他为失败,
 "err_msg":"成功|失败的原因",
 "config_key":"request中的config_key",
 "history_list":[
     {
         "config_type":"xx",
         "config_value":"xxx",
         "config_operator":"xxx",
         "config_timestamp":"xxxxxx",
         "config_reversion:版本号,int64整型
     },
     {
         "config_type":"xx",
         "config_value":"xxx",
         "config_operator":"xxx",
```

```
            "config_timestamp":"xxxxxx",
            "config_reversion:版本号,int64整型
        }
    ]
}
```

8. 回滚配置
   8.1 接口:

```
post /config/rollbackconfig
```

   8.2 参数:

```
{
 "config_key":"/[子业务ID]/[AppName]/[Namespace]",
 "config_operator":"xxx",
 "config_reversion":版本号,int64整型
}
```

   8.3 返回值:

```
{
 "err_code":0为成功,其他为失败,
 "err_msg":"成功|失败的原因",
 "config_key":"request中的config_key",
}
```

9. 获取配置同步到客户机的情况
   9.1 接口:

```
post /config/getsyncstatus
```

   9.2 参数:

```
{
 "config_key":"/[子业务ID]/[AppName]/[Namespace]"
}
```

   9.3 返回值:

```
{
 "err_code":0为成功,其他为失败,
 "err_msg":"成功|失败的原因",
 "config_key":"request中的config_key",
 "sync_list":[
     "ip:port:timestamp:reversion",
```

```
    "ip:port:timestamp:reversion"
 ]
}
```

10. 获取申请过qcm服务的所有子业务列表
    10.1 接口:

```
get /config/getsubbusinesslist
```

    10.2 参数:

```
使用 GET 方法，无参数
```

    10.3 返回值:

```
{
"err_code":0为成功,其他为失败,
"err_msg":"成功|失败的原因",
"config_key":"request中的config_key",
"subbusiness_list":[
    "...",
    "..."
]
}
```

11. 批量添加配置
    11.1 接口:

```
get /config/batchadd
```

    11.2 参数:

```
{
"create_config_list":[
{
"config_key":"/[子业务ID]/[AppName]/[Namespace]",
"config_value":"配置的具体内容",
"config_type":"text|json|xml|...",
"config_operator":"作此操作的hulk用户账号"
},
...
{
"config_key":"/[子业务ID]/[AppName]/[Namespace]",
"config_value":"配置的具体内容",
"config_type":"text|json|xml|...",
"config_operator":"作此操作的hulk用户账号"
```

```
}
]
}
```

11.3 返回值:

```
{
"create_config_list": [
{
"err_code":0为成功,其他为失败,
"err_msg":"成功|失败的原因",
"user_key":"返回给用户的key, 用户在使用sdk获取配置时,需要传入此key, 一般格式为:/[子业
"config_reversion":当前配置项的版本号,int64整型
},
...
{
"err_code":0为成功,其他为失败,
"err_msg":"成功|失败的原因",
"user_key":"返回给用户的key, 用户在使用sdk获取配置时,需要传入此key, 一般格式为:/[子业
"config_reversion":当前配置项的版本号,int64整型
}
]
}
```

11.4 说明： 与add接口相似，就是将单条配置扩展成了配置的数组。每个配置的添加都是独立的，一个的失败不是影响其他的。

12. 批量添加配置
    12.1 接口:

```
get /config/batchdelete
```

12.2 参数:

```
{
"config_keys":[
{
"config_key":"/[子业务ID]/[AppName]/[Namespace]"
},
...
{
"config_key":"/[子业务ID]/[AppName]/[Namespace]"
}
]
}
```

12.3 返回值:

```
{
"config_keys": [
{
"err_code":0为成功,其他为失败,
"err_msg":"成功|失败的原因"
},
...
{
"err_code":0为成功,其他为失败,
"err_msg":"成功|失败的原因"
}
]
}
```

12.4 说明： 与delete接口相似，就是将单条配置扩展成了配置的数组。每个配置的删除都是独立的，一个的失败不是影响其他的。