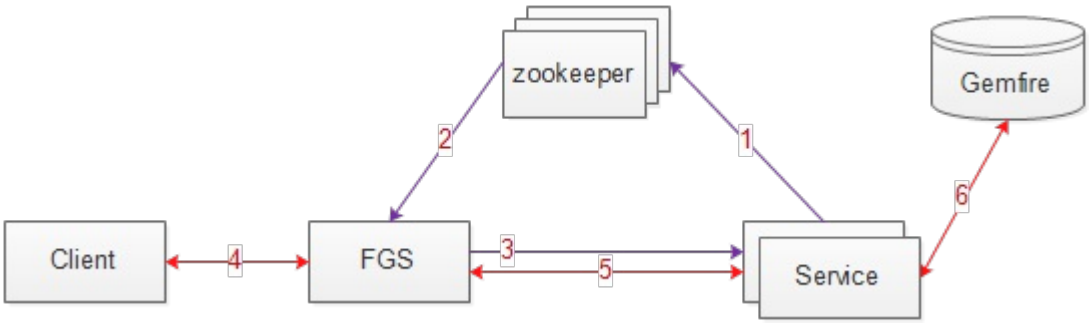
FGS User's Guide(v1.0.3)

FGS简介

FGS(Front Gateway System 前置网关系统) 系统架构工作流程图如下所示：



- <1> Service向zookeeper注册服务
- <2> FGS监听到zookeeper中服务变化，获取新增的Service服务信息
- <3> 根据获取的新增服务，连接到新发现的Service服务
- <4> Client端向FGS发送登录请求
- <5> FGS验证转发Client的登录请求至指定的Service服务(CAS)
- <6> CAS(Chronos Account System)接收登录请求，读取gemfire数据验证登录信息，同时将Client登录结果写入gemfire(fgs_sessions)中
- <5.> CAS将登录处理结果返回FGS
- <4.> FGS收到CAS的登录回报，解析登录结果后，将回报转发至指定的Client
- <4> Client接收登录回报，登录成功后方可发送其他服务请求以及主题订阅

FGS功能介绍

FGS系统使用QtpMessage协议作为唯一的通信协议，协议主要由Header、Option以及Data三部分组成

- Header

  Header块中包含消息类型(msg_type), 服务ID(service_id)以及订阅主题(topic)组成

  - 消息类型 Service端的消息类型，由Service端自己定义,须确保在该Service端唯一,取值范围(1000,60000]
  - 服务ID 后台服务的唯一id，全局唯一，取值范围(0,255)
  - 订阅主题 全局唯一(0,65535)

  FGS系统提供的部分消息类型如下：

消息类型	ID	备注
kMtFgsStdAns	100	FGS标准错误回报
kMtLogin	101	登录请求
kMtLoginAns	102	登录回报

kMtLogout	103	登出请求
kMtSubscribe	105	订阅(单key)
kMtUnsubscribe	106	取消订阅(单key)
kMtPublish	107	发布(单key)
kMtComboSubscribe	108	订阅(组合key)
kMtComboUnsubscribe	109	取消订阅(组合key)
kMtComboPublish	110	发布(组合key)

tip: FGS系统提供的功能服务中，消息类型为订阅发布时，service_id=0, topic=[!0]；消息类型非订阅发布时，service_id=[!0], topic=0; [另外当service_id=[!0]且topic=[!0]时，此消息将被当作普通消息转发至后端服务同时会被当作订阅消息在FGS内部处理(主要用于行情订阅)]

- Option

  Option块由选项id和选项值组成，Option的id由Client端和Service端协商自定义，取值范围（0,59900）

  FGS系统提供了部分FGS系统所需Option定义,如下:

选项名	ID	类型	备注
kMoItemSize	59901	uint32_t	Data块中单个数据包的长度
kMoItemCnt	59902	uint32_t	Data块中数据包的个数
kMoInstanceID	59903	int64_t	Service实例ID(Client端根据Service端注册时的负载均衡模式按需要提供)
kMoSessionID	59904	int64_t	Client端在FGS中的session_id, Service端需要将收到的该字段在消息中返回
kMoSubscribeKey	59905	int64_t	发布消息的key(用于topic过滤消息)

- Data

  Data数据块为Service端和Client端进行交互的数据块，由各服务自定。

FGS对前后端服务提供登录/登出、订阅、发布以及消息路由等功能

- 登录/登出

  Client端要使用FGS的任何其他功能都需要先登录FGS,只有成功登录的用户才享有FGS的其他权限。跟FGS的同一连接，重复发送登录请求时，新的登录请求会将前一次登录的用户踢出。

  登录请求的报文(data)格式如下所示：

```
{
```

```
    "data":{
        "user_id": 100101,
        "password": "*",
        "dsn": "*",
        "cpuid": "*",
        "mac": "*",
        "ip": "*"
    }
}
```

登出请求的报文格式如下所示：

```
{
    "data":{
        "user_id": 100101
    }
}
```

- user_id 操作员id
- password 密码(密文)
- dsn 硬盘序列号
- cpuid cpu序列号
- mac mac地址
- ip IP地址

登录回报的报文(data)格式如下所示(ret_code代表的值参见文件include/fgs_inc.h)：

```
{
    "data":{
        "user_id": 100101,
        "term_id": 0,
        "ret_code": 0,
        "ret_msg": "login successed."
    }
}
```

- user_id 操作员id(int64_t)
- term_id 终端ID(uint32_t)
- ret_code 返回代码(0：成功; else 失败)

- ret_msg 错误消息

FGS返回的标准错误消息格式同登录回报，如下所示(ret_code代表的值参见文件include/fgs_inc.h)：

```
{
    "data":{
        "ret_code": -1,
        "ret_msg": "error message."
    }
}
```

- 订阅

Client端已登录的用户可以进行订阅Service端提供的主题

当前系统订阅主要支持两种订阅模式，单key和组合key，具体订阅模式选择需要依赖发布端选择的发布模式。单key模式的key为int64_t类型；组合key模式的key是以多个key-value对的json对象。

- 单key订阅 (kMtSubscribe/kMtUnsubscribe)

  订阅的key为int64_t类型，当订阅key=0时，表示订阅单主题下的所有key。订阅消息打包实例如下：

```
{
    // 同时订阅topic=2212，key=600446和341000
    auto subscribe = std::make_shared<QtpMessage>();
    subscribe->BeginEncode(kMtSubscribe, 0, 2212);
    uint32_t item_size = sizeof(int64_t);
    uint32_t item_cnt = 2;
    subscribe->AddOption(fgs::kMoItemSize, &item_size,
sizeof(uint32_t));
    subscribe->AddOption(fgs::kMoItemCnt, &item_cnt,
sizeof(uint32_t));
    int64_t key = 0;
    std::string data;
    data.append((char*)&key, sizeof(int64_t));
    key = 341000;
    data.append((char*)&key, sizeof(int64_t));
    subscribe->SetData((void*)data.data(), data.length(), false);
    subscribe->Encode();
}
```

- 组合key订阅 (kMtComboSubscribe/kMtComboUnsubscribe)

  组合订阅的可以为key-value的json对象，其中value的值可以使用字符[*]来表示通配该项key的的所有值，如果订阅的组合key中所有key能在发布消息的组合key中匹配成功，则认为订阅了该主题的消息。订阅时的key对象位于整个Data块的/data/keys数组对象下。 组合订阅消息打包实例如

下：

```
{
    // 同时订阅两组组合key
    auto subscribe = std::make_shared<QtpMessage>();
    subscribe->BeginEncode(kMtComboSubscribe, 0, 2212);
    std::string key = "{\"data\":{\"keys\":
[{\"k1\":\"v1\",\"k2\":\"600446\"},
{\"k1\":\"v1\",\"k2\":\"341000\"}]}}";
    subscribe->SetData((void*)key.data(), key.length(), false);
    subscribe->Encode();
}
```

组合订阅的data数据块的格式：

```
{
    "data":{
        "keys"[
            {//组合key对象}
        ]
    }
}
```

取消订阅的消息包和订阅消息的数据打包内容格式一样，仅msg_type不一样

- 发布

Service端在成功注册FGS服务后，可进行主题消息的发布，FGS会将发布的消息转发给已订阅该主题消息的Client端。当前系统发布主要支持两种发布模式，单key和组合key，具体介绍同上一篇订阅。发布消息时，将发布消息的key以Option选项的方式存储于消息的Option中，Data块存放消息的具体内容。

  - 单key发布(kMtPublish)

```
{
    auto publish = std::make_shared<QtpMessage>();
    publish->BeginEncode(kMtPublish, 0, 2212);
    int64_t key = 600446;
    publish->AddOption(kMoSubscribeKey, &key, sizeof(int64_t));
    std::string str = "600446-value:";
    publish->SetData((void*)str.data(), str.length(), false);
    publish->Encode();
}
```

  - 组合key发布(kMtComboPublish)

```
{
    auto publish = std::make_shared<QtpMessage>();
    publish->BeginEncode(kMtComboPublish, 0, 2212);
    std::string key = "{\"k1\":\"v1\", \"k2\":\"600446\"}";
    publish->AddOption(kMoSubscribeKey, key.data(), key.length());
    std::string str = "600446-value:";
    publish->SetData((void*)str.data(), str.length(), false);
    publish->Encode();
}
```

- 消息路由

  Client端和Service端的相互通信，FGS会按指定要求路由到指定的目的地

  - Client端

    为了FGS能正确路由到指定的Service端，客户端的消息头必须指定service_id，如果Service端注册的lb类型是按Instance_id路由的，需要在Option中指定kMoInstanceID选项的值，其值由Service服务端事先提供

  - Service端

    为了FGS能正确将Service端的消息路由至指定的Client端，Service端在返回消息时候，复制请求消息Option选项中的kMoSessionID选项后再发送给FGS。[普通消息需要在消息头中带上Service端的service_id，但发布消息service_id=0]

Client对接FGS开发

Client端开发只需按上述要求打包消息即可，无其他注意事项。上述消息类型和Option定义等参见发布的附件include/fgs_inc.h

Service对接FGS开发

Service端开发要对接FGS需先注册服务，注册成功后，FGS会根据Service端注册服务时提供的信息来连接及管理Service。

注册服务使用提供的zoo_election.so库, 程序启动时创建zoo_election实例用于注册服务，程序退出时需要调用zoo_election实例的stop()方法或析构该实例来停止已注册的服务，以便重启程序后服务能迅速注册成功（不停止该实例，重启程序时可能需要等待20~30s左右服务才能注册成功，以下示例采用捕获SIGTERM和SIGINT信号来处理该服务的停止）。开发示例如下：

```
#include "zoo_election.h"
#include "fgs_inc.h"
#include <signal.h>

ZooElectionPtr zoo_watcher = nullptr;

void sigcatcher(int sig){
    switch (sig) {
```

```
        case SIGTERM:
        case SIGINT: {
            if (zoo_watcher) zoo_watcher->stop();
            exit(0);
            break;
        }
        default: break;
    }
}

int main(int argc, char** argv){
    // signal catcher
    struct sigaction act;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    act.sa_handler = &sigcatcher;
    sigaction(SIGTERM, &act, NULL);
    sigaction(SIGINT, &act, NULL);

    // register service.
    ZooKeeperPtr zookeeper = CreateZooKeeper("172.24.13.23:4180", 10);
    if (!zookeeper){
        LOG(error, "zookeeper create failed.");
        return -1;
    }

    ZooData zoo_data;
    zoo_data.service_id = 254;
    zoo_data.instance_id = 1;
    zoo_data.port = 50001;
    zoo_data.lb = kZtMasterSlave;
    strcpy(zoo_data.address, "172.24.13.23");

    zoo_watcher = std::make_shared<ZooElection>(zookeeper, zoo_data);
    zoo_watcher->detach();

    // todo: do other things
}
```

tip: 因示例程序通过捕获SIGTERM和SIGINT信号来处理服务的停止，若要正常停止程序，则不能使用kill -s 9 ${pid}来杀死程序，而是通过kill ${pid}来停止程序即可。

开发库中ZooData数据结构各字段介绍如下：

- service_id Service服务id，全局唯一
- instance_id Service服务实例id，同一Service下须唯一
- lb 负载均衡算法类型，当前支持类型有[0: 指定instance_id路由模式，1: master_slave模式]
- address Service端提供服务的地址(该地址需要FGS能访问)
- port Service端提供服务的端口

Service端其他注意事项参考上述各篇章。上述内容中消息类型和Option定义等参见发布的附件include/fgs_inc.h

FGS对接约束

为了方便各前后端服务的对接，对service_id和topic的定义做如下一些约束

- service_id 定义

  对当前已开发的系统，每组系统预留10个service_id, 供其相关系统使用。service_id定义如下：

0	fgs	前置网关系统
10	cas	账户应用系统
20	ss	策略系统
21	ssgw	策略网关
30	coms	订单管理系统
40	cms	配置管理系统
41	tmis	交易监测指标服务模块
50	sds	行情数据系统
60	bs	回测(模拟撮合)
61	fs	仿真(模拟撮合)
71	cs	清算服务
80	ogs	订单网关服务
...	...	...

- topic 定义

  为了使所有系统的topic定义在fgs中唯一,将topic的值定义在service_id之下，并为每一个service预分配100个topic主题(sub_topic_id)供其使用:

  topic = service_id * 100 + sub_topic_id

  如:

  cas的topic定义范围: (1001, 1099)

  ss的topic定义范围: (2001, 2099)

  等等

FGS运行依赖配置

FGS运行默认依赖当前目录下的config.json文件，也可通过gflags方式配置config_file参数项来修改启动配置文件

及路径。配置文件以json格式进行配置，其中各配置对象分别定义如下：

- log

  log对象配置程序运行日志输出配置

  - dir 日志输出目录，为空时直接输出至控制台
  - level 日志输出基本级别
  - verbose info日志详细级别(值越大日志信息显示越详细，level>2时，此值不起作用)

```
"log":{
    "dir": "",
    "level": 0,
    "verbose": 150
}
```

- zookeeper

  zookeeper对象用于配置Zookeeper服务的连接信息等

  - host zookeeper服务器的地址和端口（address:port）
  - timeout 超时时间(秒)
  - base_dir zookeeper发现Service的基础路径(此项默认无须另行配置)

```
"zookeeper":{
    "host": "172.24.13.23:4180",
    "timeout": 10
}
```

- server

  server对象配置FGS提供的服务信息

  - address FGS绑定的地址
  - port FGS绑定的端口

```
"server":{
    "address": "0.0.0.0",
    "port": 8001
}
```

- services

  services对象配置service信息（当前系统默认只配置account_system相关项），各子对象如下：

  - account_system

account_system 对象配置账户系统的一些信息

- service_id AccountSystem系统的服务id，用于FGS转发Client端的登录信息。
- params 对象下配置用户登录验证的一些个性化信息(如白名单/黑名单)
  - account_verify 账户认证启用功能，默认启用(*正式发布系统不启用此项配置)
  - white_list 白名单用户配置，配置白名单用户的用户名和密码
  - back_list 黑名单用户配置，配置黑名单用户的用户名，[**黑名单优先于白名单]

```
"services":{
    "account_system": {
        "service_id": 10,
        "params": {
            "white_list": [{
                "user_id": 100101,
                "password": "*"
            }],
            "back_list": [100102]
        }
    }
}
```

FGS启动高级配置(可选)

FGS默认直接运行可执行程序即可，若需要特别定制高级配置项可通过启动参数进行配置。FGS启动参数采用gflags进行配置。通过gflags文件配置启动方式如下：

```
fgs --flagfile fgs.config
```

高级配置参数可自定义FGS系统内部各部分线程数量，各参数定义如下：

- config_file 配置文件(默认当前目录下的config.json)
- access_actor_multi 接入端消息处理线程数
- service_actor_multi 服务端消息处理线程数
- subscribe_actor_multi 单key订阅发布消息处理线程数
- combo_subscribe_actor_multi 组合key订阅发布消息处理线程数

各线程数量范围限制在[1~16]，请根据实际情况进行配置部署。参数配置格式参考如下：

```
--config_file=config.json
```

FGS运行依赖项

- zookeeper-3.4.10

zookeeper启动依赖 jdk-1.8.0_*以上版本

zookeeper单机模式部署启动：

- 进入zookeeper-3.4.10目录下的conf子目录，创建配置文件zoo.cfg，内容如下:

```
tickTime=1000
dataDir=/tmp/zookeeper
clientPort=4180
```

  - tickTime: zookeeper中使用的基本时间单位, 毫秒值.
  - dataDir: 数据目录. 可以是任意目录.
  - clientPort: 监听client连接的端口号.

- 在zookeeper-3.4.10目录下执行如下命令启动zookeeper server

```
$ ./bin/zkServer.sh start
```