|  |
| --- |
|  |

|  |
| --- |
|  |

**介绍**

**简介**

发布系统是2016年开发的淘米内部用于代码线上发布的一套系统。与原先手动上线发布代码的方式不同，新的发布系统不再需要程序员到线上服务器进行操作，而只需要提交代码至svn，然后在网页上操作就可以发布，减少了误操作的可能。并且所有程序的配置文件等相关信息都保存在数据库中，不再需要再担心服务器损毁后信息丢失的情况发生，也更容易进行服务器迁移等操作。无论在易用性和安全性上都相较于之前得到了提升。

**系统组成**

image1.png

发布系统组要由三个部分组成，网页前端（service-config.taomee.com），中心服务端，后台服务集。

网页前端主要负责查看、修改服务配置。所有数据的查看的修改都要与中心服务端进行交互（前端只会调用中心端接口，本身并不保存配置信息）。网页端还负责了服务的发布操作，从指定svn拉取代码编译并发布到后台服务器上，所有一系列的操作都由网页端负责。

中心服务端主要负责了数据存储，修改等，并与前端网页进行交互。中心服务端保存了后台所有服务的配置信息，以及所有配置信息的历史版本，并对这些信息做了相应的管理工作（版本控制）。

所有的后台服务运行时都会与中心服务端进行交互，服务启动时回去中心端拉取当前的配置文件，以及与之交互的服务的地址。并且每隔一段都会拉取一部分配置信息，保证服务在任何时间都可以自行调整配置以及交互服务地址。

**系统使用方法**

使用最新sync框架完成服务代码并上传svn。

在网页端配置服务相关配置。

在页面选择想要发布的服务版本，点击发布。

查看发布是否成功，如果失败，请回退版本后再次发布。

**网页端**

**页面显示**

**发布操作**

**中心服务端**

中心服务端主要分为两个服务，一个与前端页面交互，一个与后端服务交互，但都共享同一个数据库。如下图所示。

image2.png

**存储格式**

所有服务的信息都以组别为单位进行存储在中心服务端层。当客户端要访问某一个服务端的时候，并不是访问单一的服务，而是访问整个服务组。每个服务组可以包含1个以上的任意个服务。

客户端会根据从中心服务拉取到的service-group信息来决定具体访问到服务组内的哪一个服务。所有信息构成了一个树形结构（如下图所示），包括了：大组到小组的访问策略、小组到服务的访问策略、组内所有服务的地址信息。通过这些信息，客户端就能锁定索要访问的服务端服务。

大组到小组的目的是为了分流和数据隔离。访问策略只能设置为哈希，这样不同服务器内的数据都会根据哈希的规则保存数据，起到了数据隔离的作用。

小组到服务的目的是为了做备份，保证服务出现问题时有备份服务可以使用。

image3.png

**大组信息：**

服务组名称：每个服务组都有一个唯一的服务组名称。就是通过服务组名称来寻找索要查询的服务组。

外层访问策略：也就是从大组到小组的访问策略。

服务代码的svn路径：发布的时候就是从该地址拉取代码，编译后放到目标机器上的。

禁止或者启用标志位：表示该服务组是否被禁用了。就是该服务组没有被关闭，但是别的客户端不能访问到它。范围整个大组包含的服务。

大组和小组的关联：大组下面有哪些小组。

**小组信息：**

外层访问策略：小组到服务的访问策略。

禁止或者启用标志位：范围小组包含的服务。

小组到服务的关联：小组下有哪些服务。

**服务信息：**

服务器ID：每个ip地址对应一个服务器id，用来表示服务是发布在那个服务器上面的。

服务的绝对路径。

服务绑定的端口号。

禁止或者启用标志位：单个服务。

发布是启动服务的用户名密码。

该服务发布类型（PHP服务还是C++服务等）。

服务配置文件信息，以key-value形式保存。

**版本控制**

发布系统本控制与存储方式类似，分为三个层级的版本控制。分别为大组层，小组层，服务层。所有层级都包含了版本回退功能（删除的服务也可以通过版本回退找回）。每个大组，小组，服务都有自己的版本号，查询某一个版本的信息就是通过版本号进行查询的，版本号为自增id。下面说明了，不同的层级操作会有什么影响。

**修改信息（包括删除）：**

大组层：当大组层的信息发生改动时，会产生新的版本号以及一份大组层的信息重新保存在数据库中。小组层和服务层都不会发生变动，也不会产生新的版本重新保存。在这种情况下，大组的版本号一定是大于或者等于小组和服务的版本号的。这对后面查询操作来说很重要。

小组层：当小组层的信息发生改动时，同样会产生一个新的版本号并产生一份小组的信息保存在数据库中，同时小组所对应的大组也会加上一个版本，并且版本号新版本号就是新产生的版本号，随即保存入数据库中。但是与小组同级的其他小组不会发生改变，服务层的信息也不会发生改变。

服务层：当服务层的信息发生改动时，该服务会加一个版本保存，服务对应的小组也会加一个版本保存，对应的大组也会加一个版本保存。但不会影响到其他小组，以及其他服务。所有新增版本的版本号都是相同的。

PS.这个版本控制引入了svn版本控制的概念，类似于下层目录会影响上层目录的版本，但上层目录的改动不会影响下层目录。这里是大组级保存了所有的版本号，而小组级别和服务级别保存的版本号都要小于或者等于大组级别。

**查询信息（通过查询不同的版本实现版本回退）：**

根据版本号查询特定的服务组（服务组是由服务组名称区分的）时， 会先查询到该版本的大组，然后根据大组向下查询大组下的小组信息（大组保存了小组的id）。小组版本会选取小于等于大组版本中最大的那个版本，就是要查询的版本信息。然后再向下查询服务信息。与查询小组类似，会选取小于小组版本中最大的那个版本，最为当前版本的服务信息。

**center-web服务**

center-web服务是中心服务端内其中一个服务，与前端进行交互，为前端提供服务组信息的查看、修改等等功能。svn地址：http://svn.taomee.com/platform/refactor/service-center

该服务是用thrift框架编写的，不需要设立命令号，只需要调用接口就可以了，所有接口信息都可以在svn地址的thrift目录下中查看到。

**交互流程**

image4.png

**接口说明**

SetServiceGroup

功能：设置服务组信息，该接口会将接收到的服务组信息，作为一个新版本保存到数据库中。

其中每个大组，小组，服务都有自己id。在接收到的服务组信息中，当有id=-1时表示这个组或者服务是新添加的。当数据库中存在的id而收到的信息中不存在时，表示该组或者服务被删除了。

接口：SetServiceGroupAck SetServiceGroup(1: SetServiceGroupReq s)

接收数据：接收到的数据为整个服务组的信息，数据结构形式如下

SetServiceGroupReq

ServiceGroupDetail group：保存服务组信息的结构体

string user：保存该版本的用户

ServiceGroupDetail

i32 id：大组id（唯一）

string name：服务组名称（唯一）

string group\_strategy：外层策略

string svn：代码svn地址

bool forbidden：禁用标志位

list< ServiceReplicasDetail> vec：小组信息队列

ServiceReplicasDetail

i32 id：小组id（唯一）

string replicas\_strategy：内层策略（唯一）

bool forbidden：禁用标志位

list<ServiceDetail> vec：服务信息队列

ServiceDetail

i32 id：服务id（唯一）

string ip：服务发布ip地址

string path：服务发布绝对路径

i32 port：服务绑定端口号

i32 svn\_version：服务svn版本号

string user：启动服务用户名

string password：启动服务用户密码

i32 type：发布类型（php：1， c++：2， java:3 目前只支持c++）

string des：服务备注

bool forbidden：禁用标志位

list<ConfigDetail> vec：配置文件列表

ConfigDetail

id：配置id

key：配置key值

value：配置value值

deploy\_type：表示配置文件类型。1：普通key-value形式， 2：表示文件形式，服务取到这个类型的配置文件会在自己的conf目录下生成相应的配置文件 3：表示是服务依赖库信息，key是库名称，value是库svn地址。

annotation：注释

返回数据：

SetServiceGroupAck

StatusInfo status：返回状态码

功能：根据服务组名称查询服务组id

接口：QueryGroupIdAck QueryGroupId(1: QueryGroupIdReq s)

接收数据：

QueryGroupIdReq

string group\_name：服务组名称

返回数据：

QueryGroupIdAck

StatusInfo status：返回状态信息

i32 gid：服务组id

WebQueryGroup

功能：根据版本号查询对应服务信息

接口：WebQueryGroupAck WebQueryGroup(1: WebQueryGroupReq s)

接收数据：

WebQueryGroupReq

i32 gid：服务组号

i32 version\_id：版本号

返回数据：

StatusInfo status：返回状态信息

ServiceGroupDetail group：服务组信息（具体看SetServiceGroup接口）

SetPresentVer

功能：设置当前版本，就是在数据库中标记出那个版本是线上正在运行的。

接口：SetPresentVerAck SetPresentVersion(1: SetPresentVerReq s)

接收数据：

SetPresentVerReq

i32 gid：服务组id

i32 version\_id：要设置的版本号

string user：设置版本号的用户

返回数据：

SetPresentVerAck

StatusInfo status：返回状态信息

list<ReleaseInfo> vec：装有发布服务信息的队列

ReleaseInfo

i32 id：服务id

i32 action：服务操作，表示该服务是要停止，启动还是重启。

i32 type：服务类型，是c++服务还是php服务等。

string svn：服务的svn地址

i32 svn\_id：服务的svn版本号

string path：服务的绝对路径

string ip：服务发布的目标机器ip

i32 port：服务绑定的端口号

string user：启动服务的用户名

string password：启动服务的用户地址

map<string,string> lib\_list：服务链接库队列，map结果，key是库名称，vale是库的svn地址。

QuerySaveHistory

功能：查询一段时间内保存的版本信息

接口：QueryVersionListAck QuerySaveHistory(1: QueryVersionListReq s)

接收信息：

QueryVersionListReq

i32 gid：服务组号

i32 start\_time：起始时间戳

i32 end\_time：结束时间戳

返回信息：

QueryVersionListAck

StatusInfo status：返回状态信息

list<VersionInfo> version\_list：版本信息列表

VersionInfo：

i32 version\_id：版本号

i32 time：版本保存时间

string user：保存版本的用户

i32 flag：是否是线上的版本

QueryReleaseHistory

功能：查询一段时间内发布版本的信息

接口：QueryVersionListAck QueryReleaseHistory(1: QueryVersionListReq s)

接收信息：

QueryVersionListReq

i32 gid：服务组号

i32 start\_time：起始时间戳

i32 end\_time：结束时间戳

返回信息：

QueryVersionListAck

StatusInfo status：返回状态信息

list<VersionInfo> version\_list：版本信息列表

VersionInfo：

i32 version\_id：版本号

i32 time：版本发布时间

string user：发布版本的用户

i32 flag：是否是线上的版本

QueryPresentVer

功能：查询当前发布的版本号

接口：QueryPresentVerAck QueryPresentVer(1:QueryPresentVerReq s)

接收信息：

QueryPresentVerReq

i32 gid：服务组号

返回信息：

QueryPresentVerAck

StatusInfo status：返回状态信息

version\_id：当前线上版本的版本号

SetMonitorAddr

功能：设置报警用户邮件地址

接口：SetMonitorAddrAck SetMonitorAddr(1:SetMonitorAddrReq s)

接收信息：

SetMonitorAddrReq

list<string> addr\_list：邮件地址列表

返回信息：

SetMonitorAddrAck

StatusInfo status： 要保存的返回的状态信息，如果数据库中存有列表中没有的邮件地址，会将其注销

QueryMonitorAddr

功能：查询报警用户地址邮件

接口：QueryMonitorAddrAck QueryMonitorAddr()

接收信息：

无

发送信息：

QueryMonitorAddrAck

StatusInfo status：返回状态信息

list<string> addr\_list：查询的邮件地址列表。

**center-server服务**

center-server服务是中心服务端两个服务中的其中一个，负责与后台各个服务进行交互，为后台服务提供配置文件的信息，以及以及与其交互的服务的ip和port等信息。

svn地址：http://svn.taomee.com/platform/refactor/service-center-without-thrift

center-server服务使用的是async-server框架，本身启动的时候并不会从数据库拉配置，配置需要写在conf目录下。下面是center-server服务与后端交互的流程。

**交互流程**

后端服务与center-server服务交互主要分为两段时间，服务启动阶段和服务运行阶段。

服务启动阶段：

服务调用拉取配置文件的接口，向中心服务查询到服务的配置文件信息，其中包含了所有的配置信息，有些信息是启动服务所必须的。

服务调用拉取交互服务信息的接口，通过得到的交互服务信息初始化网络模块，与其他服务进行交互。

服务运行阶段：

定时获取交互服务信息，保证交互服务的ip或port等信息发生变化时，服务可以自己做出调整，不需要再修改任何文档。

定时获取服务可重载配置，使得有些可重载配置发生变化时，服务器可以在不重启的情况下获取这些可重载配置。

服务有脚本会定期对日志做扫描，从中心服务获取报警邮件地址，及时报警。

**接口说明**

**包头格式**：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **类型** | **字节数** | **说明** |
| pkg\_len | Uint32 | 4 | 协议长度 |
| seq\_num | uint32 | 4 | 包序列号，客户端填写，服务端原样返回 |
| cmd\_id | uint16 | 2 | 命令号 |
| status\_code | uint32 | 4 | 错误码 |
| user\_id | uint32 | 4 | 米米号 |

拉取配置文件

说明：每次服务启动时会调用这个这个接口拉取服务的配置文件，根据服务的ip和path确定唯一服务。

命令号：0x1000

**接收包包体**：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **类型** | **字节数** | **说明** |
| ip | char[64] | 64 | 服务所在ip地址 |
| path | char[1024] | 1024 | 服务所在绝对路径 |

**返回包包体**：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **类型** | **字节数** | **说明** |
| conf\_num | uint32\_t | 4 | 拉取的配置数量 |
| info | config\_info | 不定 | 配置信息 |
| info | config\_info | 不定 | 配置信息 |
| 。。。。 | 。。。 | 。。。 | 。。。 |

config\_info包体格式：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **类型** | **字节数** | **说明** |
| type | uint32\_t | 4 | 配置文件类型  1：key-value形式  2:文件形式 |
| value\_len | uint32\_t | 4 | value字段的长度 |
| key | char[64] | 64 | key值 |
| value | char[0] | value\_len | value值 |

拉取交互服务信息以及可重载配置

说明：后台服务可以通过这个接口拉取两种信息。

交互服务信息：

如果把后台服务看作客户端，那么客户端向别的服务端发送信息就必须要知道服务端的ip和port信息，而这些信息是保存在中心服务端的数据库中的。那么客户端就需要通过调用这个接口获取到相应的服务端的信息。

而且服务端的信息可能会发生改变，为了让客户端实时得获得这些信息，在不重启的情况下正确将这些信息发送到客户端。客户端会定时到中心服务端获取服务端的ip和port等信息，保证服务运行正常。

可重载配置：

服务在启动时会到中心服务拉取配置文件，但是如果配置发生改变，服务就必须重启才能重新加载配置。为了使得服务在不重启的情况下就可以重载一部分配置，服务会定期从中心服务端拉取可重载的配置文件。

命令号：0X2000

**接收包包体**：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **类型** | **字节数** | **说明** |
| ip | char[64] | 64 | 服务所在ip地址 |
| path | char[1024] | 1024 | 服务所在绝对路径 |
| name\_num | uint32\_t | 4 | 要拉取的服务组名称的数量 |
| name | char[64] | 64 | 服务组名称 |
| name | char[64] | 64 | 服务组名称 |
| 。。。。 | 。。。。 | 。。。。 | 。。。。 |

**返回包包体**：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **类型** | **字节数** | **说明** |
| group\_num | uint32\_t | 4 | 查询到的服务组数量 |
| group\_info | service\_group\_info | 不确定 | 服务组信息 |
| group\_info | service\_group\_info | 不确定 | 服务组信息 |
| 。。。。 | 。。。。 | 。。。 | 。。。 |
| conf\_num | uint32\_t | 4 | 查询到的可重载配置数量 |
| conf\_info | config\_info | 不确定 | 配置文件信息 |
| conf\_info | config\_info | 不确定 | 配置文件信息 |
| 。。。 | 。。。 | 。。。 | 。。。 |

config\_info包体格式：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **类型** | **字节数** | **说明** |
| type | uint32\_t | 4 | 配置文件类型  1：key-value形式  2:文件形式 |
| value\_len | uint32\_t | 4 | value字段的长度 |
| key | char[64] | 64 | key值 |
| value | char[0] | value\_len | value值 |

service\_group\_info包体格式：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **类型** | **字节数** | **说明** |
| name | char[64] | 64 | 服务组名称 |
| group\_strategy | uint8\_t | 1 | 服务组策略 |
| forbidden | uint8\_t | 1 | 禁用标志位 |
| num | uint32\_t | 4 | 小组数量 |
| info | service\_replicas\_info | 不确定 | 小组信息 |
| info | service\_replicas\_info | 不确定 | 小组信息 |
| 。。。。 | 。。。。 | 。。。。 | 。。。。 |

service\_replicas\_info包体格式：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **类型** | **字节数** | **说明** |
| replicas\_strategy | uint8\_t | 1 | 小组策略 |
| forbidden | uint8\_t | 1 | 禁用标志位 |
| num | uint32\_t | 4 | 服务数量 |
| info | host\_info | 41 | 服务信息 |
| info | host\_info | 41 | 服务信息 |
| 。。。 | 。。。 | 。。。 | 。。。 |

host\_info包体格式：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **类型** | **字节数** | **说明** |
| ip | char[32] | 32 | 交互服务ip地址 |
| port | uint32\_t | 4 | 交互服务端口号 |
| timeout | uint32\_t | 4 | 超时时间 |
| forbidden | uint8\_t | 1 | 禁用标志位 |

拉取报警邮件地址

说明：获取报警要通知的电子邮件地址

命令号：0X3000

**接受包包体**：空

**返回包包体：**

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **类型** | **字节数** | **说明** |
| num | uint32\_t | 4 | 电子邮件数量 |
| addr\_info | char[64] | 64 | 电子邮件地址 |
| addr\_info | char[64] | 64 | 电子邮件地址 |
| 。。。 | 。。。 | 。。。 | 。。。 |

**数据库设计**

中心服务数据库主要是提供了各个服务的版本信息的存储。由于有两个服务调用这个数据库，所以对数据库所有的增删改查等服务必须是原子操作。

**service\_group\_table大组信息表**

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| id | INT(10) | 服务组组号（索引）  每个服务组唯一，但是对应不同版本不唯一 |
| name | VCHAR(64) | 服务组名称（索引）  每个服务组唯一，但是对应不同版本不唯一 |
| saved\_user | VCHAR(64) | 保存版本用户 |
| group\_strategy | VCHAR(64) | 外层访问策略：  int\_region：整数哈希  string\_region：字符串哈希 |
| forbidden | SMALLINT(5) | 禁用标识位：  禁用1，启用0 |
| delete\_flat | SMALLINT(5) | 删除标志位，表示该服务组已经被删除 |
| version\_id | INT(10) | 版本号（索引） |
| version\_flag | SMALLINT(5) | 版本标志位（索引）：  当前版本1， 非当前版本0 |
| saved\_time | TIMESTAMP | 版本保存时间（索引）  数据插入数据库时间 |

**service\_replicas\_table小组信息表**

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| id | INT(10) | 小组组号（索引）  每个小组唯一，但是对应不同版本不唯一 |
| group\_id | INT(10) | 小组对应大组组号（索引） |
| replicas\_strategy | VCHAR(64) | 内层策略：  roll：轮询  fix：备份（一个挂了再启用另一个）  int\_region：整数哈希  string\_region：字符串哈希 |
| forbidden | SMALLINT(5) | 禁用标识位：  禁用1，启用0 |
| delete\_flat | SMALLINT(5) | 删除标志位，表示小组已经被删除 |
| forbidden | SMALLINT(5) | 禁用标识位：  禁用1，启用0 |
| delete\_flat | SMALLINT(5) | 删除标志位，表示该小组已经被删除 |
| version\_id | INT(10) | 版本号（索引） |
| version\_flag | SMALLINT(5) | 版本标志位（索引）：  当前版本1， 非当前版本0 |

**service\_table服务信息表**

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| id | INT(10) | 小组组号（索引）  每个服务唯一，但是对应不同版本不唯一 |
| replicas\_id | INT(10) | 服务对应小组号（索引） |
| server\_id | INT(10) | 服务器id：  每个server\_id对应一个服务器，根据这个id可以在对应itl数据库内获取到该机器的ip地址。  server\_id就是itl的t\_server\_info表的server\_id字段 |
| path | VCHAR(256) | 服务在目标机器上的部署目录，是绝对路径 |
| port | INT(10) | 服务绑定端口号 |
| svn\_version | INT(10) | svn版本号 |
| user | VCHAR(64) | 目标机器上启动服务的用户 |
| password | VCHAR(64) | 目标机器上启动服务的密码，明文 |
| type | SMALLINT(5) | 发布类型，0表示不熟即发布(php)，1表示编译发布(C++) |
| des | VCHAR(64) | 服务备注 |
| forbidden | SMALLINT(5) | 禁用标识位：  禁用1，启用0 |
| delete\_flat | SMALLINT(5) | 删除标志位，表示该服务已经被删除 |
| version\_id | INT(10) | 版本号（索引） |
| version\_flag | SMALLINT(5) | 版本标志位（索引）：  当前版本1， 非当前版本0 |

**config\_table配置文件表**

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| id | INT(10) | 配置项id（主键 自增）  每个小组唯一，但是对应不同版本不唯一 |
| service\_id | INT(10) | 配置项对应服务id（索引） |
| keyword | VCHAR(64) | key值 |
| value | TEXT | value值 |
| deploy\_type | SMALLINT(5) | 配置项类型：  1：普通字符串  2：文件，服务会生成对应key名称的文件在conf目录下  3：表示是库文件 |
| annotation | VCHAR(64) | 配置项说明 |
| version\_id | INT(10) | 版本号（索引） |

**id\_table获取大组小组服务的id表**

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| id | INT(10) | 为大组、小组、服务生成的id（主键 自增） |

**version\_table获取版本号的表**

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| id | INT(10) | 生成的版本号（主键 自增） |

**released\_version\_table发布版本信息表**

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| group\_id | INT(10) | 服务组组号（索引） |
| version\_id | INT(10) | 版本号 |
| release\_time | TIMESTAMP | 发布时间（索引） |
| release\_user | VCHAR(64) | 发布用户 |

**monitor\_table报警邮件地址表**

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| mail\_addr | VCHAR(64) | 邮件地址 |
| flag | INT(10) | 是否使用中 |

**后台服务框架**

**介绍**

后台服务与中心服务的交互流程在第三章（中心服务端）已经介绍过了。本章节主要从后台服务框架的角度介绍后台服务与中心端的交互是如何实现的。本章节还会介绍此框架的使用方法和规范。

后台服务框架是在arsync-server框架的基础上，做了一定的改动。本章节主要是对与发布系统相关的部分做一定程度的讲解，具体部分请看源代码，以及示例代码。

代码部分包含三个部分：

async-server源码

svn地址：http://svn.taomee.com/platform/refactor/public/async-server

async-server框架本身的代码，是框架核心代码，提供了一部分接口供服务调用。

common目录下代码

svn地址：http://svn.taomee.com/platform/refactor/public/common-without-thrift

common目录内包含了框架对发布系统支持的相关代码，主要包括了配置文件拉取和网络框架（通过服务组向服务端传输信息）相关代码

服务代码

需要自己编写的部分，主要包含了业务逻辑相关代码。但是中间包含了一部分可重用的代码，包括一部分服务组和可重载配置信息拉取的代码（这部分可能后面考虑移动到common目录下）。

**配置拉取**

框架支持两种形式的配置文件获取方式，一种是通过发布系统的中心端拉取配置文件，一种是原始的从conf目录下读取配置文件。在调试时可以通过不同的启动方式选择配置读取方式。这部分代码都写在common目录下。

配置文件的读取是在程序初始化阶段。在第三章提到过，通过发布系统拉取文件的方式，是通过调用center-server服务的拉取配置接口，从而得到配置信息。服务会得到两种配置信息，一种是简单的字符串形式，还有一种是文件形式。所有的配置都是以key-value形式得到的。

如果得到是字符串形式的数据，服务会将配置信息保存到名为m\_config\_map的map里，在业务代码中就可以通过调用接口获取到此map中的数据（接口就是server.cpp下的config\_get\_intval和config\_get\_strval函数）。

如果得到是文件形式的数据，那么服务就会在conf目录下生成以key为名称value为内容的文件。注意只有是string形式文件才是可重载的。

文件方式

启动方式：sh server.sh start file

代码：file\_config\_manager.cpp file\_config\_manager.h

中心拉取方式

启动方式：sh server.sh start center

代码：center\_config\_manager.cpp center\_config\_manager.h

代码都是继承至config\_manager.h类。可以在server.cpp的load\_config函数中看到调用流程。

**重载配置**

配置文件是分为可重载配置和不可重载配置的。重载配置的意思是，在程序不重启的情况下，服务可以重新加载配置信息。并且要其保证实时行，在中心端修改完可重载配置后，服务就可以立即重载该项配置。

为了实现这些功能，服务就需要定去去配置中心拉取可重载的配置项。那为什么要分为可重载配置和不可重载配置，不把所有配置都默认为可以重载的呢？

一个原因是有些服务的配置内容是很多的，而且所有服务都会定期去中心拉取可重载配置，如果每次拉取所有的配置信息的话，中心服务端的压力就会变得很大。还有一个原因，很多配置信息只有在服务重启的时候才会用到，就算在中心服务端对这些配置做出修改，服务也不会立即生效，所以这么做是没有意义的。

文件形式的配置信息不支持重载，只有普通的字符串形式才可以重载。服务在接收到从中心服务获取到的可重载配置后，会更新m\_config\_map中保存的配置信息，服务可以重新从map中读取该配置信息。

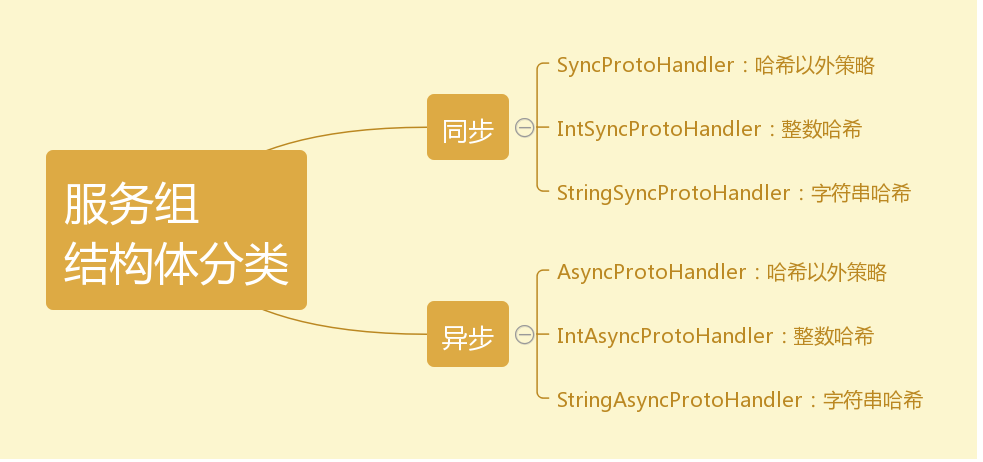
从中心服务加载可重载配置的接口，第三章已经讲解过了。是在拉取服务组信息的时候，一起拉取可重载配置。接口在文件startegy\_proto\_handler.cpp内，可以查看。

**服务组**

服务组是发布系统最核心的部分，其中最重要的网络模块代码也和服务组的概念融合在了一起，下面重点介绍该网络模块。

**使用方法**

使用该网络模块，需要使用者编写一个类，这个类要继承自网络模块中写好的一个类，也就是说不是简单得调用一个api就可以了。所有可继承的类都写在了common目录下的proto\_handler.h文件中。你可以看到该文件中有很多类，根据使用异步还是同步以及不同的外层访问策略来区分使用，具体分类如下图所示。



**使用规范**

**报警以及keepalive**