# 跨服模块介绍

## 综述

跨服模块总体通过skynet的cluster实现，采用的从服主动上报，主服定时检查的逻辑来维护从服到主服的点对点的连接，不支持从服之间的直接交互，避免加大开发时的逻辑复杂度

## 具体实现

主从都是通过cross\_monitor服务来实现整个跨服连接，通过setting

配置中的server\_type来区分服务器类型，现阶段只做普通服和跨服主服的区分

此外每个服务器都会配置主服cluster具体的ip端口，名字统一使用”master”来完成，同时会配置一个自己本地的cluster服务使用的ip端口，对于所有服务器都会主动挂载cluster\_node对应的具体端口，主服配置是配置给所有从服，让他们主动去连接主服用的。主服从配置知道其他从服的信息，都靠运行起来后的从服自行上报完成

如果某个服务器暂时不参与跨服活动可通过配置中的 is\_corss字段来控制，

字段修改后可以自动连接，不需要重启，但是已经连接的不会去主动断开

配置完成后的例子

从服

server\_type = "normal",

master\_node = "0.0.0.0:2528",  
cluster\_node = "0.0.0.0:2529",  
is\_cross = true,

主服

server\_type = "ndinfo\_master",

master\_node = "0.0.0.0:2528",  
cluster\_node = "0.0.0.0:2528",  
is\_cross = true,

也就是对于从服只需要关注 cluster\_node字段，根据每个服务器不同进行配置，确保不要冲突即可

对于从服的corss\_monitor，需要在启动的时候挂载自己的cluster端口，同时对master进行一次上报操作，这个上报操作也扮演着ping操作，上报的时候，会上报自己囊括的服务器ID，本地的cluster\_node配置，还有对应的node名，如果在启动那次ping失败了或者后续有ping失败的操作，会立刻启动一个一次性的快速重连服务用来快速恢复连接

对于主服，启动的时候首先同样挂载自己的cluster端口，同时启动一个定时刷新本地node列表的任务，node列表缓存在share\_data里用来给其他功能服务，刷新服务会定时检查这些node是不是可用，如果不是则进行剔除，对于有差异的上报，永远是新的上报覆盖老的数据，不过一般不会出现冲突，后续如果出现问题，可以考虑在上报节点信息和本地缓存不一致的时候做日志处理，定位冲突

## 使用

整个跨服模块包装了一个cross\_lib模块供外部使用，最常用的接口为

* 从服对主服进行异步或者同步调用，传参为主服node所在的本地服务名和具体消息

function cross\_lib.send\_master(master\_service, msg)

function cross\_lib.call\_master(master\_service, msg)

* 主服回复从服时的调用

function cross\_lib.send\_slave(server\_id, service, msg)

function cross\_lib.call\_slave(service, server\_id, msg)

也就是对外其他功能不需要感知node系统，也不需要专门去维护主从之间的连接，只需要在发送和回复的时候指定服务器ID即可(将来如果有功能特别蛋疼要求一部分跨服开启，一部分不开启，那最好把限制做到功能里，不要改动跨服基础架构)

同时也提供了一些配置相关的简单调用，

cross\_lib.server\_type()

cross\_lib.is\_master\_node()

cross\_lib.server\_id()

这些都只是简单的配置操作，并不复杂这里就不再做解释

需要注意的是所有的call和send在连接断开或者目标节点不存在的时候都会直接返回异常，所以不需要依赖call的返回来检测目标节点ok，开发中，劲量避免使用call，最好能将任务拆分成清晰的多个无上下文的单独操作

send 和call的具体返回格式参照requester模块，使用方法可以参照cross\_monitor