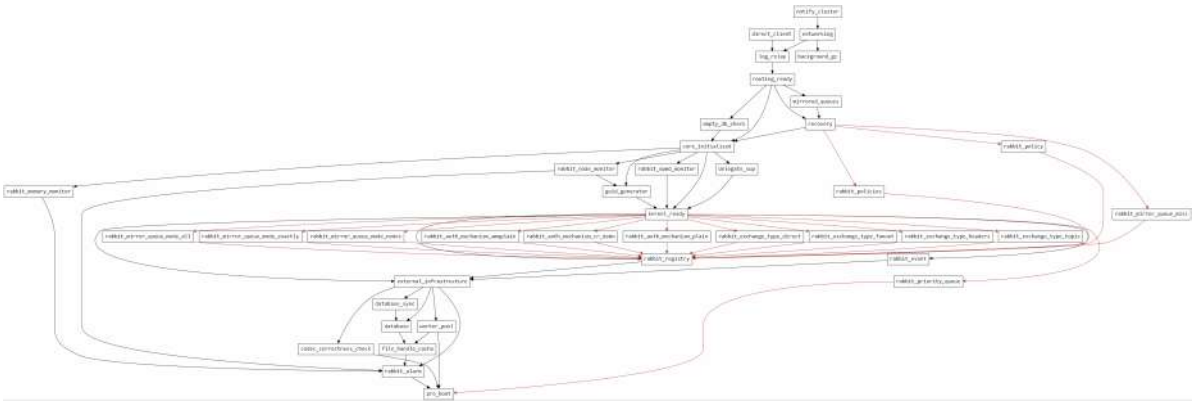
RabbitMQ进程启动有向图

笔记本： RabbitMq源代码阅读系统原理分析

创建时间： 2015/6/4 12:30 **更新时间：** 2015/11/17 16:12



graphviz Dot脚本

```
digraph RabbitMQ{
     node[shape="box" fontsize=18 size="1,1" fontname="Yahei Mono" height=.5
width=.8];
     edge[fontsize=18];

     pre_boot;
     codec_correctness_check;
     rabbit_alarm;
     database;
     database_sync;
     file_handle_cache;
     worker_pool;
     external_infrastructure;
     rabbit_registry;
     rabbit_event;
     kernel_ready;
     rabbit_memory_monitor;
     guid_generator;
     delegate_sup;
     rabbit_node_monitor;
     rabbit_epmd_monitor;
     core_initialized;
     empty_db_check;
     recovery;
     mirrored_queues;
     routing_ready;
     log_relay;
     direct_client;
     notify_cluster;
     background_gc;
     rabbit_auth_mechanism_amqplain;
     rabbit_auth_mechanism_cr_demo;
     rabbit_auth_mechanism_plain;
     rabbit_exchange_type_direct;
     rabbit_exchange_type_fanout;
```

```
    rabbit_exchange_type_headers;
    rabbit_exchange_type_topic;
    rabbit_mirror_queue_misc;
    rabbit_mirror_queue_mode_all;
    rabbit_mirror_queue_mode_exactly;
    rabbit_mirror_queue_mode_nodes;
    rabbit_policies;
    rabbit_policy;
    rabbit_priority_queue;


    external_infrastructure -> codec_correctness_check;
    codec_correctness_check -> pre_boot;
    external_infrastructure -> rabbit_alarm;
    rabbit_alarm -> pre_boot;
    external_infrastructure -> database;
    database -> file_handle_cache;
    external_infrastructure -> database_sync;
    database_sync -> database;
    worker_pool -> file_handle_cache;
    file_handle_cache -> rabbit_alarm;
    external_infrastructure -> worker_pool;
    worker_pool -> pre_boot;
    kernel_ready -> rabbit_registry;
    rabbit_registry -> external_infrastructure;
    kernel_ready -> rabbit_event;
    rabbit_event -> external_infrastructure;
    kernel_ready -> external_infrastructure;
    core_initialized -> rabbit_memory_monitor;
    rabbit_memory_monitor -> rabbit_alarm;
    core_initialized -> guid_generator;
    guid_generator -> kernel_ready;
    core_initialized -> delegate_sup;
    delegate_sup -> kernel_ready;
    core_initialized -> rabbit_node_monitor;
    rabbit_node_monitor -> rabbit_alarm;
    rabbit_node_monitor -> guid_generator;
    core_initialized -> rabbit_epmd_monitor;
    rabbit_epmd_monitor -> kernel_ready;
    core_initialized -> kernel_ready;
    routing_ready -> empty_db_check;
    empty_db_check -> core_initialized;
    routing_ready -> recovery;
    recovery -> core_initialized;
    routing_ready -> mirrored_queues;
    routing_ready -> core_initialized;
    mirrored_queues -> recovery;
    networking -> log_relay;
    log_relay -> routing_ready;
    direct_client -> log_relay;
    notify_cluster -> networking;
    networking -> background_gc;
    kernel_ready -> rabbit_auth_mechanism_amqplain[arrowhead="vee" color ="brown"];
    rabbit_auth_mechanism_amqplain -> rabbit_registry[arrowhead="vee" color
="brown"];
```

```
    kernel_ready -> rabbit_auth_mechanism_cr_demo[arrowhead="vee" color ="brown"];
    rabbit_auth_mechanism_cr_demo -> rabbit_registry[arrowhead="vee" color
="brown"];
    kernel_ready -> rabbit_auth_mechanism_plain[arrowhead="vee" color ="brown"];
    rabbit_auth_mechanism_plain -> rabbit_registry[arrowhead="vee" color ="brown"];
    kernel_ready -> rabbit_exchange_type_direct[arrowhead="vee" color ="brown"];
    rabbit_exchange_type_direct -> rabbit_registry[arrowhead="vee" color ="brown"];
    kernel_ready -> rabbit_exchange_type_fanout[arrowhead="vee" color ="brown"];
    rabbit_exchange_type_fanout -> rabbit_registry[arrowhead="vee" color ="brown"];
    kernel_ready -> rabbit_exchange_type_headers[arrowhead="vee" color ="brown"];
    rabbit_exchange_type_headers -> rabbit_registry[arrowhead="vee" color ="brown"];
    kernel_ready -> rabbit_exchange_type_topic[arrowhead="vee" color ="brown"];
    rabbit_exchange_type_topic -> rabbit_registry[arrowhead="vee" color ="brown"];
    recovery -> rabbit_mirror_queue_misc[arrowhead="vee" color ="brown"];
    rabbit_mirror_queue_misc -> rabbit_registry[arrowhead="vee" color ="brown"];
    kernel_ready -> rabbit_mirror_queue_mode_all[arrowhead="vee" color ="brown"];
    rabbit_mirror_queue_mode_all -> rabbit_registry[arrowhead="vee" color ="brown"];
    kernel_ready -> rabbit_mirror_queue_mode_exactly[arrowhead="vee" color
="brown"];
    rabbit_mirror_queue_mode_exactly -> rabbit_registry[arrowhead="vee" color
="brown"];
    kernel_ready -> rabbit_mirror_queue_mode_nodes[arrowhead="vee" color ="brown"];
    rabbit_mirror_queue_mode_nodes -> rabbit_registry[arrowhead="vee" color
="brown"];
    recovery -> rabbit_policies[arrowhead="vee" color ="brown"];
    rabbit_policies -> rabbit_registry[arrowhead="vee" color ="brown"];
    recovery -> rabbit_policy[arrowhead="vee" color ="brown"];
    rabbit_policy -> rabbit_registry[arrowhead="vee" color ="brown"];
    kernel_ready -> rabbit_priority_queue[arrowhead="vee" color ="brown"];
    rabbit_priority_queue -> pre_boot[arrowhead="vee" color ="brown"];
}
```

以下是实际的启动过程研究：

1.rabbit_alarm启动步骤(先执行rabbit_alarm:start()函数)

(1).启动一个rabbit_alarm_sup的supervisor2监督进程同时在该监督进程下启动一个rabbit_alarm的gen_event进程

rabbit_alarm进程作为整个RabbitMQ系统的报警进程，例如内存，磁盘大小的报警，报警后，如果有人向rabbit_alarm进程注册，

则会进程回调，同时会通知集群中的额其他节点

(2).启动一个vm_memory_monitor_sup的supervisor2监督进程同时在该进程下启动一个vm_memory_monitor进程

RabbitMQ系统虚拟机内存监督报警进程，如果虚拟机中的内存少于配置文件配置的大小，则会立刻通知rabbit_alarm进程

该进程就是RabbitMQ系统对内存使用情况监视的进程，如果当前内存使用量超过了配置文件中配置的大小，则会立刻向rabbit_alarm

进程发布内存使用量过大的报警信息，则集群中的所有节点的rabbit_alarm进程则会将内存报警信息回调注册到rabbit_alarm进程的函数

(3).启动一个rabbit_disk_monitor_sup的supervisor2监督进程同时在该监督进程下启动一个rabbit_disk_monitor进程

RabbitMQ系统磁盘使用报警进程，如果磁盘剩余大小少于配置文件中配置的大小，则会立刻通知rabbit_alarm进程

该进程就是RabbitMQ系统对磁盘使用情况监视的进程，如果磁盘剩余量少于配置文件中配置的大小，则会立刻向rabbit_alarm进程

发布报警信息，则集群中的所有节点的rabbit_alarm进程都会将报警信息回调注册到rabbit_alarm进程的函数

2.file_handle_cache启动步骤
(1).执行rabbit:start_fhc()函数启动file_handle_cache进程(该进程是RabbitMQ系统文件打开关闭操作关键进程)
该进程是RabbitMQ系统所有操作磁盘文件相关操作的进程

3.worker_pool启动步骤(RabbitMQ系统异步执行任务的小系统)
(1).首先启动一个worker_pool_sup的supervisor的监督进程
(2).worker_pool_sup监督进程下再启动一个worker_pool的进程池管理进程
(3).worker_pool_sup监督进程下再启动调度线程个数的worker_pool_worker的工作进程
该监督树下的进程是用来异步提交函数让工作进程执行完成，worker_pool_worker为工作进程，worker_pool为所有worker_pool_worker进程的管理者，哪个
worker_pool_worker进程空闲，哪个worker_pool_worker正在工作，worker_pool进程都有记录

4.database启动步骤(初始化RabbitMQ中的mnesia数据库，如果配置有集群数据库，自动连接到集群数据库)
(1).执行rabbit_mnesia:init()函数
该操作步骤是启动当前RabbitMQ节点的Mnesia数据库，同时将本节点同集群中的其他节点连接起来，并将集群中的数据同自己本地的数据进行同步，然后将
RabbitMQ系统所有的数据库表启动起来

5.database_sync启动步骤
(1).执行rabbit_sup:start_child(mnesia_sync)函数
mnesia:sync_transaction操作没有保证Mnesia数据库的日志同步到磁盘上，则调用mnesia_sync:sync()函数的进程进行同步阻塞等待mnesia成功的将数据库
操作日志写入磁盘上

6.codec_correctness_check启动步骤
(1).执行rabbit_binary_generator:check_empty_frame_size()函数
确保空的Frame数据格式的正确性

7.rabbit_registry启动步骤
(1).执行rabbit_sup:start_child(rabbit_registry)函数
RabbitMQ系统内部各种定义类型注册处理模块的进程，该进程启动了rabbit_registry名字的一个ETS表，用来存储分类，类型名字，以及处理模块的数据

8.rabbit_auth_mechanism_cr_demo启动步骤
(1).执行rabbit_registry:register(auth_mechanism, <<"RABBIT-CR-DEMO">>, rabbit_auth_mechanism_cr_demo)函数
RabbitMQ系统用户验证处理模块之一

9.rabbit_auth_mechanism_amqplain启动步骤
(1).执行rabbit_registry:register(auth_mechanism, <<"AMQPLAIN">>, rabbit_auth_mechanism_amqplain)函数
RabbitMQ系统用户验证处理模块之一

10.rabbit_auth_mechanism_plain启动步骤
(1).执行rabbit_registry:register(auth_mechanism, <<"PLAIN">>, rabbit_auth_mechanism_plain)函数
RabbitMQ系统用户验证处理模块之一

11.rabbit_mirror_queue_mode_all启动步骤(高可用队列相关)

(1).执行rabbit_registry:register(ha_mode, <<"all">>, rabbit_mirror_queue_mode_all)函数

12.rabbit_event启动步骤(RabbitMQ系统事件管理器进程)
(1).执行rabbit_sup:start_restartable_child(rabbit_event)函数,启动rabbit_event进程
RabbitMQ系统中所有的事件都是发布到rabbit_event事件管理器中，只要有rabbit_event事件管理器的事件处理进程，则该进程就能接收到所有的事件

13.rabbit_exchange_tye_direct启动步骤
(1).执行rabbit_registry:register(exchange, <<"direct">>, rabbit_exchange_type_direct)函数
RabbitMQ系统exchange交换机direct类型向rabbit_registry进行注册

14.rabbit_exchange_type_fanout启动步骤
(1).执行rabbit_registry:register(exchange, <<"fanout">>, rabbit_exchange_type_fanout)函数
RabbitMQ系统exchange交换机fanout类型向rabbit_registry进行注册

15.rabbit_exchange_type_headers启动步骤
(1).执行rabbit_registry:register(exchange, <<"headers">>, rabbit_exchange_type_headers)函数
RabbitMQ系统exchange交换机headers类型向rabbit_registry进行注册

16.rabbit_exchange_type_topic启动步骤
(1).执行rabbit_registry:register(exchange, <<"topic">>, rabbit_exchange_type_topic)函数
RabbitMQ系统exchange交换机topic类型向rabbit_registry进行注册
%% rabbit_topic_trie_node表里存储的是节点数据(里面存储的是topic_trie_node数据结构，所有的路由信息都是从root节点出发)
%% rabbit_topic_trie_edge表里存储的是边数据(里面存储的是topic_trie_node数据结构，边的数据结构里面存储的有路由信息的单个单词)
%% rabbit_topic_trie_binding表里存储的是某个节点上的绑定信息(里面存储的是topic_trie_binding数据结构)
%% 比如路由信息hello.#.nb：
%% 1.有四个节点，第一个节点始终是root节点，然后是其他三个节点，
%% 2.有三条边信息，每个边数据结构里面带有对应的单词hello，#，nb
%% 3.在第四个节点上存在绑定的队列名字

17.rabbit_mirror_queue_mode_exactly启动步骤(高可用队列相关)
(1).执行rabbit_registry:register(ha_mode, <<"exactly">>, rabbit_mirror_queue_mode_exactly)函数

18.rabbit_mirror_queue_mode_nodes启动步骤(高可用队列相关)
(1).执行rabbit_registry:register(ha_mode, <<"nodes">>, rabbit_mirror_queue_mode_nodes)函数

19.rabbit_priority_queue启动步骤(启动RabbitMQ系统优先级队列)
(1).执行rabbit_priority_queue:enable()函数
该步骤是允许RabbitMQ系统的队列支持简单的优先级队列

20.rabbit_epmd_monitor启动步骤
(1).执行rabbit_sup:start_restartable_child(rabbit_epmd_monitor)函数
该进程主要用来监视epmd进程的存在，有可能epmd进程被无端的删除掉，则该进程发现epmd进程被kill掉后，会立刻进行对epmd进程进行重启

21.guid_generator启动步骤(生成独一无二的各种ID对应的模块)
(1).执行rabbit_sup:start_restartable_child(rabbit_guild)函数
RabbitMQ系统中生成唯一16为字符串ID的进程

22.rabbit_node_monitor启动步骤
(1).执行rabbit_sup:start_restartable_child(rabbit_node_monitor)函数
RabbitMQ系统中节点管理的进程，该进程会保留集群中的其他节点以及它们对应的GUID，同时节点的删除，增加都会根据该进程通知集群中的其他节点

23.delegate_sup启动步骤(多次的向远程节点发送消息，则此代理会将发送到同一个远程节点的多个消息操作统一成一个发送消息操作)
(1).执行rabbit:boot_delegate()函数
RabbitMQ系统中的代理进程监督树，这些代理进程主要用来多次对远程的某个节点进行多次访问，用了代理进程后，可以将多次访问变成一次访问操作

24.rabbit_memory_monitor启动步骤
(1).执行rabbit_sup:start_restartable_child(rabbit_memory_monitor)函数
rabbit_memory_monitor进程统计RabbitMQ系统中内存使用情况，它会收到当前系统中所有的消息队列统计的数字，同时将内存使用速率返回给各个消息
队列

25.empty_db_check启动步骤(如果RabbitMQ系统是第一次启动则需要插入默认的账号，账号密码，vhost等默认信息)
(1).执行rabbit:maybe_insert_default_data()函数
RabbitMQ系统是第一次启动则需要插入默认的账号，账号密码，vhost等默认信息

26.rabbit_mirror_queue_misc启动步骤(高可用队列相关)
(1).执行rabbit_registry:register(policy_validator, <<"ha-mode">>, rabbit_mirror_queue_misc)函数
(2).执行rabbit_registry:register(policy_validator, <<"ha-params">>, rabbit_mirror_queue_misc)函数
(3).执行rabbit_registry:register(policy_validator, <<"ha-sync-mode">>, rabbit_mirror_queue_misc)函数
(4).执行rabbit_registry:register(policy_validator, <<"ha-promote-on-shutdown">>, rabbit_mirror_queue_misc)函数

27.rabbit_policies启动步骤
(1).执行rabbit_policies:register()函数
该启动步骤主要是向rabbit_registry进程注册policy_validator类型的相关数据(主要包括是消息队列的参数类型)

28.rabbit_policy启动步骤
(1).执行rabbit_policy:register()函数
该启动步骤主要是向rabbit_registry进程注册runtime_parameter类型的相关数据
该模块是队列和交换机的公用配置参数模块的处理模块

29.recovery启动步骤
(1).执行rabbit:recover()函数
该启动步骤主要是恢复所有的持久化队列，将所有的持久化队列进程启动起来，将持久化的交换机信息恢复到非持久化的mnesia数据库表中，
将持久化的绑定信息恢复到非持久化的绑定数据库表中

30.mirrored_queues启动步骤(高可用队列相关)
(1).执行rabbit_mirror_queue_misc:on_node_up()函数

31.log_relay启动步骤(主要是将error_logger事件中心的事件发布到<<"amq.rabbit.log">>交换机对应的队列里)

(1).执行rabbit_sup:start_child(rabbit_error_logger_lifecycle, supervised_lifecycle, [rabbit_error_logger_lifecycle, {rabbit_error_logger, start, []},
{rabbit_error_logger, stop, []}]]}}函数

该启动步骤会在rabbit_sup监督进程下启动名字为rabbit_error_logger_lifecylce监督进程，该rabbit_error_logger_lifecylce监督进程执行

supervised_lifecycle:start_link函数，该监督进程的回调初始化函数会执行rabbit_error_logger:start()函数，该函数启动rabbit_error_logger进程，

该进程会处理error_logger事件中心发布的事件，然后将事件依次发布到<<"amq.rabbit.log">>名字对应的交换机上(<<"amq.rabbit.log">>交换机会在

rabbit_error_logger事件处理进程启动的时候创建)

32.background_gc启动步骤

(1).执行rabbit_sup:start_restartable_child, [background_gc])函数

(该进程启动一个定时器，定时的对RabbitMQ系统进行垃圾回收，定时器的间隔时间是不断变化的，如果垃圾回收的时间过长，则增加时间间隔，反之则减少垃圾回收的时 间间隔)

33.networking启动步骤

(1).执行rabbit_networking:boot()函数

根据配置文件中的端口，启动网络连接进程树，等待客户端通过Socket连接过来

34.direct_client启动步骤

(1).执行rabbit direct:boot()函数

启动rabbit direct client sup监督进程，该监督进程的动态启动子进程是执行{rabbit channel sup, start link, []}

所有客户端的连接进程都是启动在rabbit_direct_client_sup监督进程下

35.notify_cluster启动步骤

(1).执行rabbit_node_monitor:notify_node_up()函数

通知RabbitMQ集群系统当前节点启动