

ROS 消息队列清空说明

李睿豪 2018.11.02

一、问题描述

MicROS 角色线程暂停，暂停期间，相应订阅者 (subscriber) 正常运行消息 (message) 订阅，在重启角色后，订阅者的回调函数 (callback) 首先调用处理过去的消息。根据 MicROS 角色要求，应对过去消息队列进行清空，实现在每次重启角色时，直接调用回调函数处理当前消息。

二、ROS Kinetic 源代码说明

在本问题中，消息队列有两个，一个订阅中的消息队列（详见 `subscription_queue.cpp`），一个是回调函数中的消息队列（详见 `callback_queue.cpp`）。

ROS Kinetic 在 `CallbackQueue` 类 (`callback_queue.cpp`) 中提供了 `CallbackQueue::clear()` 函数，对上层用户可见。但分析源代码发现，`CallbackQueue` 队列中，存放的是对订阅者消息队列操作的指针，`CallbackQueue` 队列为 `n`，可以通俗理解为对订阅者消息队列进行 `n` 次操作，通过调用 `CallbackQueue::clear()` 函数对 `CallbackQueue` 队列进行清空，并不能实现对订阅者消息队列的清空，重启角色时，回调函数依然会优先处理存放在订阅者消息队列中的旧消息。

在 `SubscriptionQueue` 类 (`subscription_queue.cpp`) 中，提供了 `SubscriptionQueue::clear()` 函数，此函数对上层用户不可见。同时，根据角色概念，每一个订阅者需要对应一个回调函数队列，因此，需要通过修改 `CallbackQueue` 类 (`callback_queue.cpp`)，通过

CallbackQueue 对象，同时对相应订阅者的消息队列和回调函数消息队列进行清空。

三、ROS Kinetic 版本源代码修改说明

修改源代码目录：/src/ros_comm/roscpp/

修改代码文件：/include/ros/callback_queue_interface.h

/include/ros/callback_queue.h

/include/ros/subscription_interface.h

/src/libros/callback_queue.cpp

/src/libros/subscription_queue.cpp

➤ /include/ros/callback_queue_interface.h 修改说明：

在 CallbackInterface 基类中，添加虚函数。

```
virtual void ClearSubQueue() {}
```

➤ /include/ros/subscription_queue.h 修改说明：

在 SubscriptionQueue 子类中，添加虚函数声明。

```
virtual void ClearSubQueue();
```

➤ /src/libros/subscription_queue.cpp 修改说明：

实现对 SubscriptionQueue::clear() 的访问，清空订阅消息队列。

```
void SubscriptionQueue::ClearSubQueue()
{
    clear();
}
```

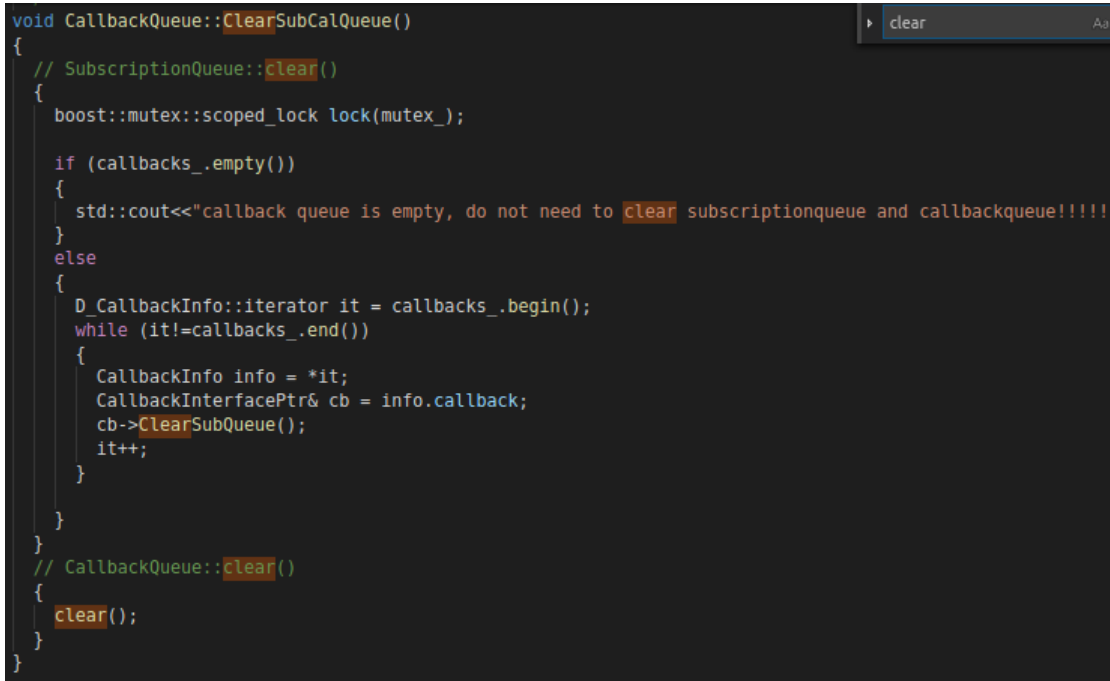
➤ /include/ros/callback_queue.h 修改说明：

在 CallbackQueue 类中，添加用户可见的 ClearSubCalQueue() 函数，实现对订阅者和回调函数消息队列的清空。

```
void ClearSubCalQueue();
```

➤ /src/libros/callback_queue.cpp 修改说明:

在 CallbackQueue 类中, 添加用户可见的 ClearSubCalQueue() 函数, 实现对订阅者和回调函数消息队列的清空。



```
void CallbackQueue::ClearSubCalQueue()
{
    // SubscriptionQueue::clear()
    {
        boost::mutex::scoped_lock lock(mutex_);

        if (callbacks_.empty())
        {
            std::cout<<"callback queue is empty, do not need to clear subscriptionqueue and callbackqueue!!!!"
        }
        else
        {
            D_CallbackInfo::iterator it = callbacks_.begin();
            while (it!=callbacks_.end())
            {
                CallbackInfo info = *it;
                CallbackInterfacePtr& cb = info.callback;
                cb->ClearSubQueue();
                it++;
            }
        }
    }
    // CallbackQueue::clear()
    {
        clear();
    }
}
```

● 源码编译说明:

ROS Kinetic 的源代码修改完毕后, 进入源代码所在目录 (home/XXX/ros_catkin_ws), 只对修改的 roscpp 包及其依赖进行重新编译:

```
./src/catkin/bin/catkin_make_isolate --install
--only-pkg-with-deps roscpp
```

四、测试说明

节点 1 (node 1, 命名 talker) 通过发布者 (publisher) 以 1Hz 的速度向主题 (topic) 发布消息, 节点 2 (node 2, 命名 listener) 通过订阅者 (subscriber) 以 10Hz 速度订阅主题上的消息。

节点 2 共有三个订阅者, 每个订阅者对应一个回调函数队列, 其中订阅者 1 的回调函数队列通过 ros::spinOnce() 触发, 订阅者 2 和

订阅者 3 对应同样的 callbackqueue(cq_2)，并都通过 ros::AsyncSpinner 方式触发。

开始时，三个订阅者（可以理解为角色）同时运行，回调函数在接收到消息后正常触发，当计数（timer）到 100 时，暂停订阅者 2 和订阅者 3，当计数到 200 时，重启订阅者 2 和订阅者 3。

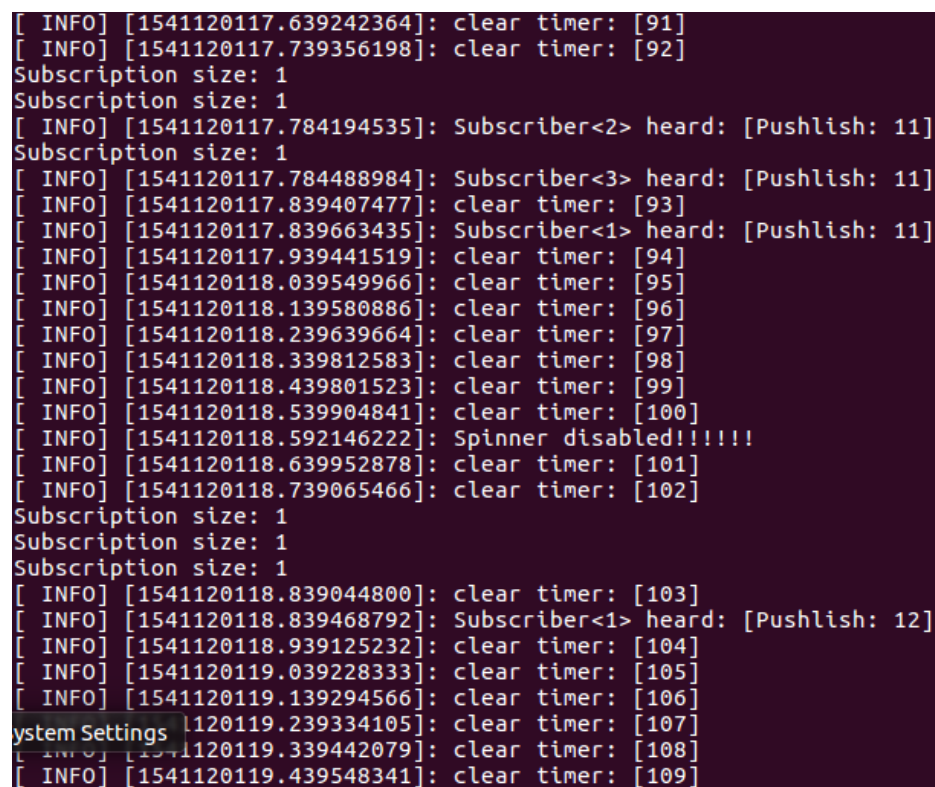
测试运行：catkin_make --pkg callback_msg_clear_tutorials

roscore & rosrun callback_msg_clear_tutorials talker

roslaunch callback_msg_clear_tutorials listener

五、实验结果

1、源代码修改前



```
[ INFO] [1541120117.639242364]: clear timer: [91]
[ INFO] [1541120117.739356198]: clear timer: [92]
Subscription size: 1
Subscription size: 1
[ INFO] [1541120117.784194535]: Subscriber<2> heard: [Pushlish: 11]
Subscription size: 1
[ INFO] [1541120117.784488984]: Subscriber<3> heard: [Pushlish: 11]
[ INFO] [1541120117.839407477]: clear timer: [93]
[ INFO] [1541120117.839663435]: Subscriber<1> heard: [Pushlish: 11]
[ INFO] [1541120117.939441519]: clear timer: [94]
[ INFO] [1541120118.039549966]: clear timer: [95]
[ INFO] [1541120118.139580886]: clear timer: [96]
[ INFO] [1541120118.239639664]: clear timer: [97]
[ INFO] [1541120118.339812583]: clear timer: [98]
[ INFO] [1541120118.439801523]: clear timer: [99]
[ INFO] [1541120118.539904841]: clear timer: [100]
[ INFO] [1541120118.592146222]: Spinner disabled!!!!
[ INFO] [1541120118.639952878]: clear timer: [101]
[ INFO] [1541120118.739065466]: clear timer: [102]
Subscription size: 1
Subscription size: 1
Subscription size: 1
[ INFO] [1541120118.839044800]: clear timer: [103]
[ INFO] [1541120118.839468792]: Subscriber<1> heard: [Pushlish: 12]
[ INFO] [1541120118.939125232]: clear timer: [104]
[ INFO] [1541120119.039228333]: clear timer: [105]
[ INFO] [1541120119.139294566]: clear timer: [106]
[ INFO] [1541120119.239334105]: clear timer: [107]
[ INFO] [1541120119.339442079]: clear timer: [108]
[ INFO] [1541120119.439548341]: clear timer: [109]
```

图 1 修改源代码前，计数到 100，暂停订阅者 2 和 3 的回调函数触发

如图 1 所示，当计数到 100 时，暂停订阅者 2 和订阅者 3 的回调函数触发（AsyncSpinner），订阅者 2 和订阅者 3 依然可以接收主题上的消息，消息队列长度增加，但回调函数不再触发调用。

```

[ INFO] [1541120471.837868104]: clear timer: [198]
[ INFO] [1541120471.958103597]: clear timer: [199]
[ INFO] [1541120472.058121970]: clear timer: [200]
Subscription size: 1
Subscription size: 10
Subscription size: 10
[ INFO] [1541120472.158227200]: clear timer: [201]
[ INFO] [1541120472.158740237]: Spinner Start again!!!!
[ INFO] [1541120472.158984476]: Subscriber<1> heard: [Pushlish: 22]
[ INFO] [1541120472.159280771]: Subscriber<2> heard: [Pushlish: 13]
[ INFO] [1541120472.159501944]: Subscriber<3> heard: [Pushlish: 13]
[ INFO] [1541120472.159718906]: Subscriber<2> heard: [Pushlish: 14]
[ INFO] [1541120472.159930553]: Subscriber<3> heard: [Pushlish: 14]
[ INFO] [1541120472.160290229]: Subscriber<2> heard: [Pushlish: 15]
[ INFO] [1541120472.160530995]: Subscriber<3> heard: [Pushlish: 15]
[ INFO] [1541120472.160745242]: Subscriber<2> heard: [Pushlish: 16]
[ INFO] [1541120472.160949567]: Subscriber<3> heard: [Pushlish: 16]
[ INFO] [1541120472.161249975]: Subscriber<2> heard: [Pushlish: 17]
[ INFO] [1541120472.161459449]: Subscriber<3> heard: [Pushlish: 17]
[ INFO] [1541120472.161659481]: Subscriber<2> heard: [Pushlish: 18]
[ INFO] [1541120472.161973302]: Subscriber<3> heard: [Pushlish: 18]
[ INFO] [1541120472.162257818]: Subscriber<2> heard: [Pushlish: 19]
[ INFO] [1541120472.162526603]: Subscriber<3> heard: [Pushlish: 19]
[ INFO] [1541120472.162779248]: Subscriber<2> heard: [Pushlish: 20]
[ INFO] [1541120472.163023123]: Subscriber<3> heard: [Pushlish: 20]
[ INFO] [1541120472.163298718]: Subscriber<2> heard: [Pushlish: 21]
[ INFO] [1541120472.163604496]: Subscriber<3> heard: [Pushlish: 21]
[ INFO] [1541120472.163864493]: Subscriber<2> heard: [Pushlish: 22]
[ INFO] [1541120472.164152722]: Subscriber<3> heard: [Pushlish: 22]
[ INFO] [1541120472.258283229]: clear timer: [202]
[ INFO] [1541120472.358370583]: clear timer: [203]
[ INFO] [1541120472.458463380]: clear timer: [204]
[ INFO] [1541120472.558595829]: clear timer: [205]
[ INFO] [1541120472.658662263]: clear timer: [206]
[ INFO] [1541120472.757778285]: clear timer: [207]
[ INFO] [1541120472.857825427]: clear timer: [208]
[ INFO] [1541120472.957957626]: clear timer: [209]
[ INFO] [1541120473.058081885]: clear timer: [210]
Subscription size: 1
Subscription size: 1
Subscription size: 1
[ INFO] [1541120473.122799825]: Subscriber<2> heard: [Pushlish: 23]
[ INFO] [1541120473.123143523]: Subscriber<3> heard: [Pushlish: 23]
[ INFO] [1541120473.158154690]: clear timer: [211]
[ INFO] [1541120473.158346278]: Subscriber<1> heard: [Pushlish: 23]
[ INFO] [1541120473.258224139]: clear timer: [212]
[ INFO] [1541120473.358357881]: clear timer: [213]

```

图 2 修改源代码前，计数到 200，重启订阅者 2 和 3 的回调触发

如图 2 所示，当计数到 200 时，重启订阅者 2 和订阅者 3 的回调函数触发，回调函数首先调用处理消息队列中旧的消息。回调函数消息队列（cq_2，长度为 20）中依次存放对订阅者 2 消息队列（长度为 10）和订阅者 3 消息队列（长度为 10）的操作指针。

2、源代码修改后

```

if (clear_flag==true)
{
    // Clear old callbacks queue and subscription queue
    cq_2.ClearSubCalQueue();
    /**
     * After we start the spinner again, the messages in the subscriber
     * will be push into the callbackqueue
     */
    async_spinner->start();
    ROS_INFO("Spinner Start again!!!!!!");
    clear_flag=false;
}

```

```

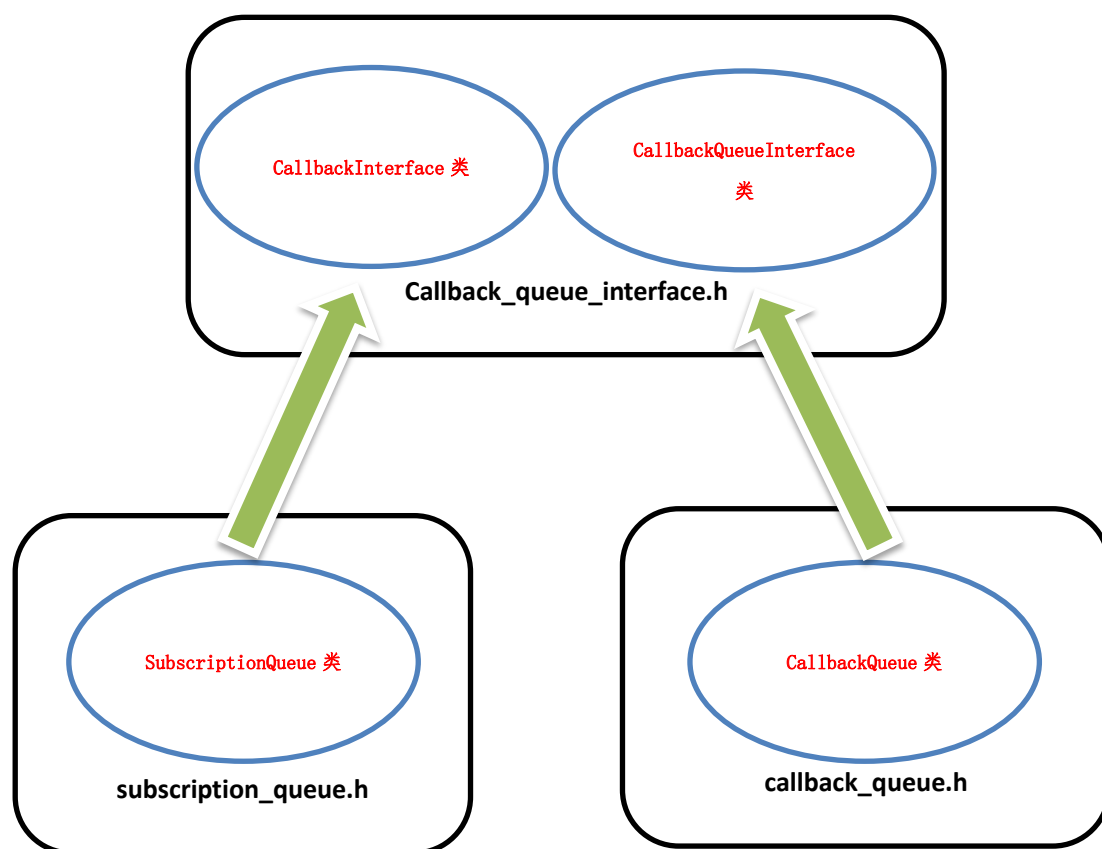
[ INFO] [1541121068.613749687]: clear timer: [196]
[ INFO] [1541121068.714344423]: clear timer: [197]
[ INFO] [1541121068.813883126]: clear timer: [198]
Subscription size: 1
Subscription size: 10
Subscription size: 10
[ INFO] [1541121068.913469421]: clear timer: [199]
[ INFO] [1541121068.913674651]: Subscriber<1> heard: [Pushlish: 22]
[ INFO] [1541121069.013909401]: clear timer: [200]
[ INFO] [1541121069.114037137]: clear timer: [201]
[ INFO] [1541121069.114382217]: Spinner Start again!!!!!!
[ INFO] [1541121069.213487098]: clear timer: [202]
[ INFO] [1541121069.313962572]: clear timer: [203]
[ INFO] [1541121069.414251090]: clear timer: [204]
[ INFO] [1541121069.513666069]: clear timer: [205]
[ INFO] [1541121069.614125121]: clear timer: [206]
[ INFO] [1541121069.713625819]: clear timer: [207]
[ INFO] [1541121069.814109993]: clear timer: [208]
Subscription size: 1
Subscription size: 1
Subscription size: 1
[ INFO] [1541121069.896154977]: Subscriber<2> heard: [Pushlish: 23]
[ INFO] [1541121069.896270852]: Subscriber<3> heard: [Pushlish: 23]
[ INFO] [1541121069.913446532]: clear timer: [209]
[ INFO] [1541121069.913623583]: Subscriber<1> heard: [Pushlish: 23]
[ INFO] [1541121070.014072935]: clear timer: [210]
[ INFO] [1541121070.113565283]: clear timer: [211]
[ INFO] [1541121070.213743543]: clear timer: [212]
[ INFO] [1541121070.314309875]: clear timer: [213]
[ INFO] [1541121070.413822956]: clear timer: [214]
[ INFO] [1541121070.514266815]: clear timer: [215]

```

图 3 修改源代码后，计数到 200，重启订阅者 2 的回调函数触发

如图 3 所示，当计数到 200 时，重启订阅者 2 的回调函数触发，重启前调用 CallbackQueue 类中的自定义函数 ClearSubCalQueue()，对回调函数中的消息队列（size 为 20）及其对应的订阅者 2（size 为 10）和订阅者 3（size 为 10）的消息队列进行清空，重启后，直接调用回调函数处理当前消息。

附录 1：函数继承示意图



附录 2：测试代码

```
int main(int argc, char **argv)
{
    /**
     * The ros::init() function
     */
    ros::init(argc, argv, "listener");
    ros::NodeHandle n;

    ros::Rate loop_rate(10);

    ros::CallbackQueue cq_1;
    //n.setCallbackQueue(&cq_1);

    ros::CallbackQueue cq_2;
    //n.setCallbackQueue(&cq_2);

    /**
     * For sub_1 with chatterCallback_1,
     * nothing special for internal ROS queue
     */
    ros::Subscriber sub_1 = n.subscribe("chatter", 100, chatterCallback_1);
```

```
ros::SubscribeOptions ops =
    ros::SubscribeOptions::create<std_msgs::String>(
        "chatter",          // topic name
        100,                // queue length
        chatterCallback_2,  // callback function
        ros::VoidPtr(),     // tracked object, we don't need one thus NULL
        &cq_2                // pointer to callback queue object
    );
ros::Subscriber sub_2 = n.subscribe(ops);

ros::SubscribeOptions ops1 =
    ros::SubscribeOptions::create<std_msgs::String>(
        "chatter",          // topic name
        100,                // queue length
        chatterCallback_3,  // callback function
        ros::VoidPtr(),     // tracked object, we don't need one thus NULL
        &cq_2                // pointer to callback queue object
    );
ros::Subscriber sub_3 = n.subscribe(ops1);

/**
 * spawn async spinner with 1 thread, running on our custom queue: cq_2
 * ros::AsyncSpinner async_spinner(1, &cq_2);
 */
async_spinner.reset(new ros::AsyncSpinner(1, &cq_2));
```



```

/**
 * start the async_spinner
 */
async_spinner->start();

int clear_timer=0;
bool clear_flag=false;
while (ros::ok())
{
    if (clear_timer>=100 && clear_timer<=200)
    {
        if (clear_flag==false)
        {
            async_spinner->stop();
            ROS_INFO("Spinner disabled!!!!!!");
            clear_flag=true;
        }
    }
}

```

```

    else
    {
        if (clear_flag==true)
        {
            /**
             * Clear old callbacks queue and subscription queue
             */
            cq_2.ClearSubCalQueue();
            /**
             * After we start the spinner again, the messages in the subscriber
             * will be push into the callbackqueue
             */
            async_spinner->start();
            ROS_INFO("Spinner Start again!!!!!!");
            clear_flag=false;
        }
    }

    //std::cout<<"cq2--callbackqueue size: "<<cq_2.size_queue()<<std::endl;
    ros::spinOnce();
    loop_rate.sleep();
}

```

```

        clear_timer++;
        ROS_INFO("clear timer: [%i]", clear_timer);
    }

    /**
     * release the spinner
     */
    async_spinner.reset();

    return 0;
}

```