Actionlib 的使用

AI 航团队

Actionlib 是 ROS 非常重要的库，像执行各种运动的动作，例如控制手臂去抓取一个杯子，这个过程可能复杂而漫长，执行过程中还可能强制中断或反馈信息，这时 Actionlib 就能大展伸手了。

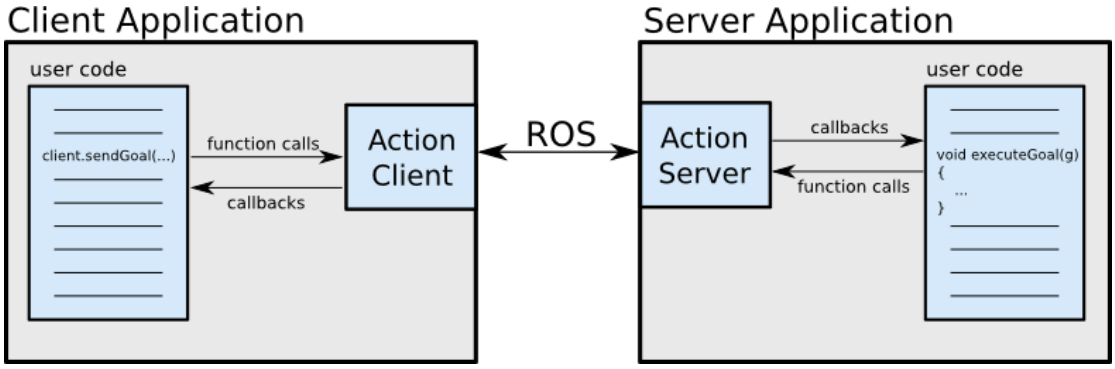
1.原理

1.1 功能

在任何一个比较大的基于 ROS 的系统，都会有这样的情况，向某个节点发送请求执行某一个任务，并返回相应的执行结果，这种通常用 ROS 的服务（services）完成。然而，有一些情况服务执行的时间很长，在执行中想要获得任务处理的进度，或可能取消执行任务，Actionlib 就能实现这样的功能，它是 ROS 的一个非常重要的库。轻舟机器人的目标定就可以借助此功能完成。 http://wiki.ros.org/cn/actionlib_tutorials/Tutorials

1.2 框架

如下图所示， Actionlib 的框架实际是一种特殊的客户-服务的模式。除了服务请求的功能外，还可以实时获取服务器执行任务的进度状态，以及强制中断服务的功能。



下面以洗碟子为例子，实现客户端调用服务器执行洗盘子的动作。这个例子是官网的一个改进版本，涵盖 actionli 的基本功能，例如获取服务器执行任务的进度状态，强制终端服务的功能，服务活动状态提示。

2.1 服务服务端实现

服务端实现了服务器执行任务的反馈信息，中断抢占功能。具体实现较为简单，反馈信息通过发布反馈的消息实现，中断抢占通过注册中断毁掉函数实现，代码如下：

```cpp
//这是actionlib的服务端

#include <first_actionlib_sample/DoDishesAction.h>
#include <actionlib/server/simple_action_server.h>

//这样定义下会用起来简洁许多
typedef actionlib::SimpleActionServer<first_actionlib_sample::DoDishesAction> Server;

class DoDishesActionServer
{
public:
    DoDishesActionServer(ros::NodeHandle n):
            server(n, "do_dishes",
                   boost::bind(&DoDishesActionServer::ExecuteCb, this, _1), false)
    {
        //注册抢占回调函数
        server.registerPreemptCallback(boost::bind(&DoDishesActionServer::preemptCb, this));
    }

    //启动服务
    void Start()
    {
        server.start();
    }

    //回调函数，在此添加代码实现你要的功能
    void ExecuteCb(const first_actionlib_sample::DoDishesGoalConstPtr& goal) {
        // 在次添加你所要实现的功能
        ROS_INFO("Received goal,the dish id is :%d", goal->dishwasher_id);
        //反馈
        first_actionlib_sample::DoDishesFeedback feedback;
        ros::Rate rate(1);
        int cur_finished_i = 1;
        int toal_dish_num = 10;
        for(cur_finished_i = 1; cur_finished_i <= toal_dish_num; cur_finished_i++)
        {
            if(!server.isActive())break;

            ROS_INFO("Cleanning the dish::%d", cur_finished_i);
            feedback.percent_complete = cur_finished_i/10.0;
            server.publishFeedback(feedback);
            rate.sleep();
        }
        first_actionlib_sample::DoDishesResult result;
        result.toal_dishes_cleaned = cur_finished_i;

        if(server.isActive())server.setSucceeded();

    }

    //中断回调函数
    void preemptCb()
    {
        if(server.isActive()){
            server.setPreempted();//强制中断
        }
    }

    Server server;
};

int main(int argc, char** argv) {
    ros::init(argc, argv, "do_dishes_server");
    ros::NodeHandle n;
    //初始化，绑定回调函数

    DoDishesActionServer actionServer(n);
    //启动服务器，等待客户端信息到来
    actionServer.Start();
    ros::spin();
    return 0;
}
```

2.2 客户端实现

客户端注册了三个回调函数，DoneCb,ActivCb,FeedbackCb，分别地，DoneCb：用于监听

服务器任务执行完后的相应消息以及客户端的相应处理，ActivCb：服务器任务被激活时的消息提示以及客户端的相应处理，FeedbackCb：接收服务器的反馈消息以及客户端的相应处理。代码如下：

```
//这是actionlib的客户端

#include <first_actionlib_sample/DoDishesAction.h>
//#include <actionlib_msgs/GoalStatusArray.h>
#include <actionlib/client/simple_action_client.h>

//这样定义下会用起来简洁许多
typedef actionlib::SimpleActionClient<first_actionlib_sample::DoDishesAction> Client;

class DoDishesActionClient {
private:
    // Called once when the goal completes
    void DoneCb(const actionlib::SimpleClientGoalState& state,
            const first_actionlib_sample::DoDishesResultConstPtr& result) {
        ROS_INFO("Finished in state [%s]", state.toString().c_str());
        ROS_INFO("Toal dish cleaned: %i", result->toal_dishes_cleaned);
        ros::shutdown();
    }

    // 当目标激活的时候，会调用一次
    void ActiveCb() {
        ROS_INFO("Goal just went active");
    }

    // 接收服务器的反馈信息
    void FeedbackCb(
            const first_actionlib_sample::DoDishesFeedbackConstPtr& feedback) {
        ROS_INFO("Got Feedback Complete Rate: %f", feedback->percent_complete);
    }
public:
    DoDishesActionClient(const std::string client_name, bool flag = true) :
            client(client_name, flag) {
    }

    //客户端开始
    void Start() {
        //等待服务器初始化完成
        client.waitForServer();
        //定义要做的目标
        first_actionlib_sample::DoDishesGoal goal;
        goal.dishwasher_id = 1;
        //发送目标至服务器
        client.sendGoal(goal,
                boost::bind(&DoDishesActionClient::DoneCb, this, _1, _2),
                boost::bind(&DoDishesActionClient::ActiveCb, this),
                boost::bind(&DoDishesActionClient::FeedbackCb, this, _1));
        //等待结果，时间间隔5秒
        client.waitForResult(ros::Duration(10.0));

        //根据返回结果，做相应的处理
        if (client.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
            printf("Yay! The dishes are now clean");
        else {
            ROS_INFO("Cancel Goal!");
            client.cancelAllGoals();
        }

        printf("Current State: %s\n", client.getState().toString().c_str());
    }
private:
    Client client;
};

int main(int argc, char** argv) {
    ros::init(argc, argv, "do_dishes_client");
    DoDishesActionClient actionclient("do_dishes", true);
    //启动客户端
    actionclient.Start();
    ros::spin();
    return 0;
}
```

2.3CMakeLists 编写

```
cmake_minimum_required(VERSION 2.8.3)
project(first_actionlib_sample)

find_package(catkin REQUIRED COMPONENTS
  actionlib
  actionlib_msgs
  roscpp
  rospy
  std_msgs
)

## Generate actions in the 'action' folder
 add_action_files(
   DIRECTORY action
   FILES
   DoDishes.action
 )

## Generate added messages and services with any dependencies listed here
 generate_messages(
   DEPENDENCIES
   actionlib_msgs#   std_msgs
 )
```

```
catkin_package()

###########
## Build ##
###########
include_directories(
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ executable
add_executable(do_dishes_action_client_node src/DoDishesActionClient.cpp)
add_executable(do_dishes_action_client_node1 src/DoDishesActionClient1.cpp)
add_executable(do_dishes_action_server_node src/DoDishesActionServer.cpp)

## Add cmake target dependencies of the executable
## same as for the library above
 add_dependencies(do_dishes_action_client_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
 add_dependencies(do_dishes_action_client_node1 ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
add_dependencies(do_dishes_action_server_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
## Specify libraries to link a library or executable target against
 target_link_libraries(do_dishes_action_client_node
   ${catkin_LIBRARIES}
 )
  target_link_libraries(do_dishes_action_client_node1
   ${catkin_LIBRARIES}
 )
 target_link_libraries(do_dishes_action_server_node
   ${catkin_LIBRARIES}
 )
```

2.4package.xml 编写

添加 actionlib 的编译和执行依赖，如下

```
<build_depend>actionlib</build_depend>
<build_depend>actionlib_msgs</build_depend>
<run_depend>actionlib</run_depend>
<run_depend>actionlib_msgs</run_depend>
```

1.同学们在使用过程中，如果发现内容有疏漏或者不严谨的地方，请与我们联系，将会有轻舟积分送上！QQ：270220858

2.内容如有雷同，侵删！

2021 年 5 月