初始化

ros : i init (argc. argv, "xxx"); 参数下数 参数数组 节点名(催一)

可以在命令行中直接向数组(argv) 佳爹,

131: rosrun api_vscoole time _length = 3

苦节点名相同,则同名的节点中占启动的节点会导致较先启动物节点死亡。 可使用options结相同的节点名店增加险机数,本质上使节点名字互采放 同等点可以同时启动

ros::init (argo, argv, ros::init::options:: AnonymousName);

可以用 rosnode list 看出节点之后有一串随机数。

治经与服务相关对象(进阶)

ros:: Publisher pub = nh. advertise < stol_msgs:: String > ("fany", lu, t/f) 自栖 图数 消息类型 指型为 Latch (bal) latch (bool)

里大认为false, 苦设置为true, 则会保留发布的—— 最后的一条消息,直至有订阅者订阅话题再发送出去。

7克1/结表示开始传号

spin() \$ spin Once()

可将 Spin L) 视作一个死循环, 因此 Spin L) 后的代码均平执行

spinOncel) 视作一个f九行一次的循环。

双方均用于处理回调函数

ros;:Time:;now();返回值也为ros:Time类型 获取当前时刻: 这回当前时刻,即 now()函数被调用的时刻个 岩考练: 1970年 01月01日 00:00:00, 类型

函数: 如Sec()可以将助约转换为秒数, 返回 double 美型数据

字段: .sec 取将的刻转换为秒数, 效图 int 美型数据

F/31! ros::Time right-now = ros:: Time::now(); ROS_INFO(" %. 2f " right_now. to Sec());

ROS_INFO['%d', right_now.sec); 设置指定时刻: ros:: Time t1(20,124321); 20秒+124321 納秒 ros: Time +21 20, 124); 20, 124 \$%

时间类型 (持续时间): rosi: Duration.

设置持续时间(停顿时间):

停顿了粉

设置好停顿时间台调用steepl); ros:: Duration du2(s.20); 停顿5.2秒 du2.slcep();

的自运第(时刻与时刻之间相:成,不可相加)结果为 Duration类型

ros: Time begin = ros: ! Time :: now (); ros: Duration dul (5); dul. sleep();
ros:: Time stop = ros:: Time::nowc);
ros:: Duration du = begin - stop;

时刻与时间相加或相流,结果为时刻 (Time 类型) 时间与时间相加城,结果为Duration类型。

定时罢:

ros::Timer 类型

差数: callback

one shot

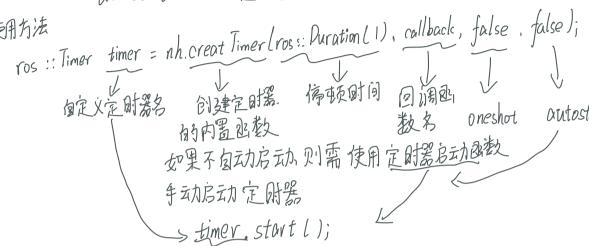
autostart

回调避

定时器是否是一次性执行的、黑大小为加金。

定时器是否自动启动,黑大认为true.

使用方法



包定 x 头 文件 的 创建 与调用 在功能包的/include 下创建。h头文件

```
#ifndef _HELLO_H
#define _HELLO_H
             void run();
```

特别地在CMakellet的配置中应当把 [18~12]放开海解

```
${catkin INCLUDE DIRS}
```

使用头文件示例

```
#include "ros/ros.h"
#include "head vscode/hello.h"
      void Myhello::run()
     ros::init(argc,argv,"hello_head");
hello_ns::Myhello myhello;
      myhello.run();
```

自定义源文件的调用 先按以上操作创建头文件之后,用一个Cpp文件提供用(在同一命名空间内)

```
## Declare a C++ library
add_library(head_src
include/${PROJECT_NAME}/hello.h
src/hello.cpp

## Add cmake target_dependencies of the

add_dependencies(head_src ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

add_dependencies(use_hello ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target against

target_link_libraries(head_src
${catkin_LIBRARIES}

} target_link_libraries(use_hello
head_src ##前面的库名148 和124 都是用的库名
${catkin_LIBRARIES}
}
```

其他函数

ros::shut down() --->美闭(杀死)节点. 抗行完毕后2位图 false.

bool ok(); return true——> 节点存活 return false——> 节点死亡

Void shutdown() 关闭函数

自志函数

```
ROS_DEBUG("调试信息");//不会显示在控制台上面4
ROS_INFO("一般消息");
ROS_WARN("警告信息");
ROS_ERROR("错误信息");
ROS_FATAL("严重错误");
```



```
ros@ros-VirtualBox:~/demo_ws$ rosrun apis_vscode log
[ INFO] [1683463143.826006827]: 一般消息
[ WARN] [1683463143.827035820]: 警告信息
[ERROR] [1683463143.827091336]: 错误信息
[FATAL] [1683463143.827098126]: 严重错误
```