通过 init 的可选操作 实现 向 param 参数空间传入参数

```
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrun plumbing_apis demo01_apis_pub _length:=10
^Z
[2]+  已停止               rosrun plumbing_apis demo01_apis_pub _length:=10
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
```

```
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosparam list
/apis_pub/length
/rosdistro
/roslaunch/uris/host_qinghuan_system_product_name__45961
/rosversion
/run_id
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosparam get /apis_pub/length
10
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
```

通过添加 options 的方法

我们在创建 node 的时候 会给你随即复制一个参数 保证你在 一个文件启动两次 因为 node 名字相同 无法启动

```
/rosout_agg
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosnode list
/apis_pub
/apis_pub_1711634373489190043
/apis_pub_1711634455577411722
/rosout
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
```

这个是最后的效果

```
问题 6  输出  调试控制台  终端  端口  注释

[ INFO] [1711634496.795349953]: 我发布的信息是:hello world1204
[ INFO] [1711634496.895326192]: 我发布的信息是:hello world1205
[ INFO] [1711634496.995311392]: 我发布的信息是:hello world1206
[ INFO] [1711634497.095311337]: 我发布的信息是:hello world1207
[ INFO] [1711634497.195305587]: 我发布的信息是:hello world1208
[ INFO] [1711634497.295302177]: 我发布的信息是:hello world1209
[ INFO] [1711634497.395314095]: 我发布的信息是:hello world1210
[ INFO] [1711634497.495341702]: 我发布的信息是:hello world1211
[ INFO] [1711634497.595340261]: 我发布的信息是:hello world1212
[ INFO] [1711634497.695598312]: 我发布的信息是:hello world1213
[ INFO] [1711634497.795767756]: 我发布的信息是:hello world1214
[ INFO] [1711634497.895588639]: 我发布的信息是:hello world1215
[ INFO] [1711634497.995586277]: 我发布的信息是:hello world1216
[ INFO] [1711634498.095584806]: 我发布的信息是:hello world1217
^C[ INFO] [1711634498.195375783]: 我发布的信息是:hello world1218
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
```

```
[ INFO] [1711634492.283897372]: 我发布的信息是:hello world338
[ INFO] [1711634492.383875048]: 我发布的信息是:hello world339
[ INFO] [1711634492.483889219]: 我发布的信息是:hello world340
[ INFO] [1711634492.583872962]: 我发布的信息是:hello world341
[ INFO] [1711634492.683869845]: 我发布的信息是:hello world342
[ INFO] [1711634492.783872797]: 我发布的信息是:hello world343
[ INFO] [1711634492.883879000]: 我发布的信息是:hello world344
[ INFO] [1711634492.983886712]: 我发布的信息是:hello world345
[ INFO] [1711634493.083875481]: 我发布的信息是:hello world346
[ INFO] [1711634493.183885045]: 我发布的信息是:hello world347
[ INFO] [1711634493.283874159]: 我发布的信息是:hello world348
[ INFO] [1711634493.383890911]: 我发布的信息是:hello world349
[ INFO] [1711634493.483894756]: 我发布的信息是:hello world350
[ INFO] [1711634493.583874091]: 我发布的信息是:hello world351
^C[ INFO] [1711634493.683941360]: 我发布的信息是:hello world352
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
src  Debug  0
```

```
ros::init(argc,argv,"apis_pub",ros::init_options::AnonymousName);
```

ros::init_options::AnonymousName 为 node 节点的名字 在存在相同的时候 给个随机数

ros::Publisher pub=nh.advertise<std_msgs::String>("话题的名称",10)
10 是队列的长度

```
问题 3   输出   调试控制台   终端   端口   注释

qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrun plumbing_apis demo02_apis_sub
^Z
[1]+  已停止               rosrun plumbing_apis demo02_apis_sub
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
```

```
问题 3   输出   调试控制台   终端   端口   注释

qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrun plumbing_apis demo01_apis_pub
[ INFO] [1711715201.948573974]: 我发布的信息是:hello world0
[ INFO] [1711715201.949454676]: 我发布的信息是:hello world1
[ INFO] [1711715202.049564928]: 我发布的信息是:hello world2
[ INFO] [1711715202.149567417]: 我发布的信息是:hello world3
[ INFO] [1711715202.249549028]: 我发布的信息是:hello world4
[ INFO] [1711715202.349558355]: 我发布的信息是:hello world5
[ INFO] [1711715202.449816765]: 我发布的信息是:hello world6
[ INFO] [1711715202.549561325]: 我发布的信息是:hello world7
[ INFO] [1711715202.649809714]: 我发布的信息是:hello world8
[ INFO] [1711715202.749566042]: 我发布的信息是:hello world9
^Z
[1]+  已停止               rosrun plumbing_apis demo01_apis_pub
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
```

如果有 true 这个参数 就可以把最后一次发布的参数 给拿到

```
plumbing_apis > src > demo01_apis_pub.cpp > main(int, char * [])
#include"ros/ros.h"
#include"std_msgs/String.h"
int main(int argc, char* argv[])
{

    /*
    argc, argv name 可选参数 options

    name是节点的名字  ros 中不允许重名的操作

    argc是功能包 + rosrun plumbing_apis_pub demo01 中  dmeio01 +后面传入的实参  在这个demo01 我们是可以传入参数

    argc 是 int 等于 功能包的别名 + 后面传入参数的个数    实参个数 + 1

    argv 是一个char 的数组   这些实参被封入了 argv中 rosrun plumbing_apis_pub demo01_apis_pub  _length:=10  向param中传入了一个

    options 是一个节点启用选项  设置参数
    节点名称不能 相同  同时启动报错
    如果一个节点需要启动两次，回报错
    如何让一个节点启动两次正常运行

    */
    setlocale(LC_ALL, "");
    ros::init(argc,argv,"apis_pub",ros::init_options::AnonymousName);
    ros::NodeHandle nh;
    /*
    针对静态数据  只需要发布一次就ok了 将latch 这个可选参数设在为True
    */
    ros::Publisher pub=nh.advertise<std_msgs::String>("fang",10,true);

    ros::Rate rate(10);
    int num=0;
    //创建后待机
    ros::Duration(3.0).sleep();
    while(ros::ok())
    {
        std_msgs::String msg;
        std::stringstream ss;
        ss<<"hello world"<<num;

        msg.data=ss.str();
        if (num<10)
        {
            //test new 这里作的变更 主要三我们就发送这个数据几次 之后在发布万数据后 看看 能不能用
            pub.publish(msg);
            ROS_INFO("我发布的信息是:%s", msg.data.c_str());
        }


        num++;
        rate.sleep();
6   输出   调试控制台   终端   端口   注释
```

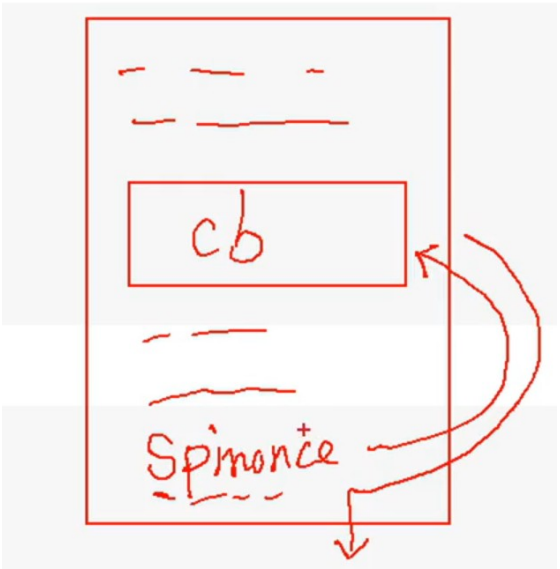
正好是最后一条参数

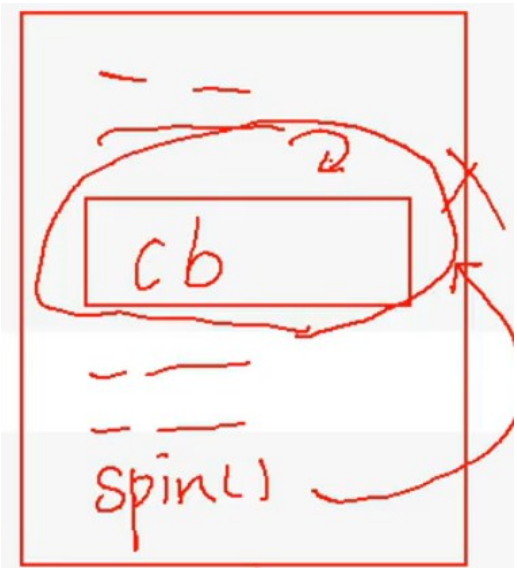
```
问题 6   输出   调试控制台   终端   端口   注释

qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrun plumbing_apis demo02_apis_sub
^Z
[1]+  已停止               rosrun plumbing_apis demo02_apis_sub
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrun plumbing_apis demo02_apis_sub
[ INFO] [1711715375.784094590]: 订阅到的信息hello world9
```

回旋函数
ros::spin()回头函数 进入循环 处理回调
ros::spinOnce() 循环的使用的时候 可以这个 在循环体 (for while)内处理所有可用的回调函数

函数的定义类型是 void 莫的返回值

spinOnce 回头 处理一轮回调函数 之后从回调函数继续运行 但是不会再一次运行 后续的代码会执行的
spin 是回到回调函数 后续的代码不会执行



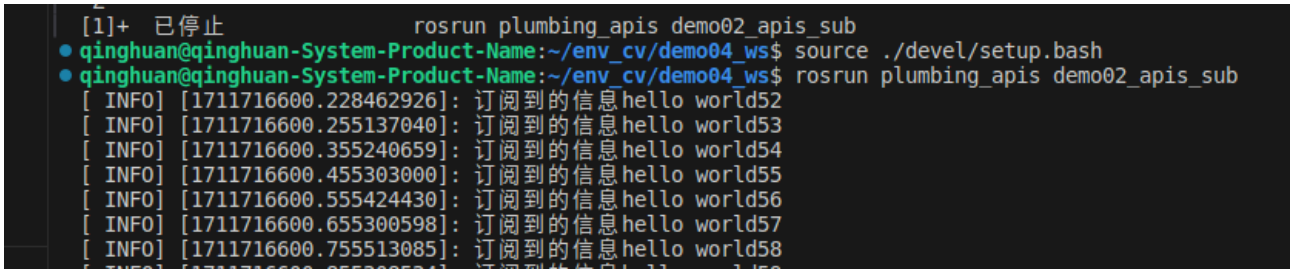


spinOnce 的结果

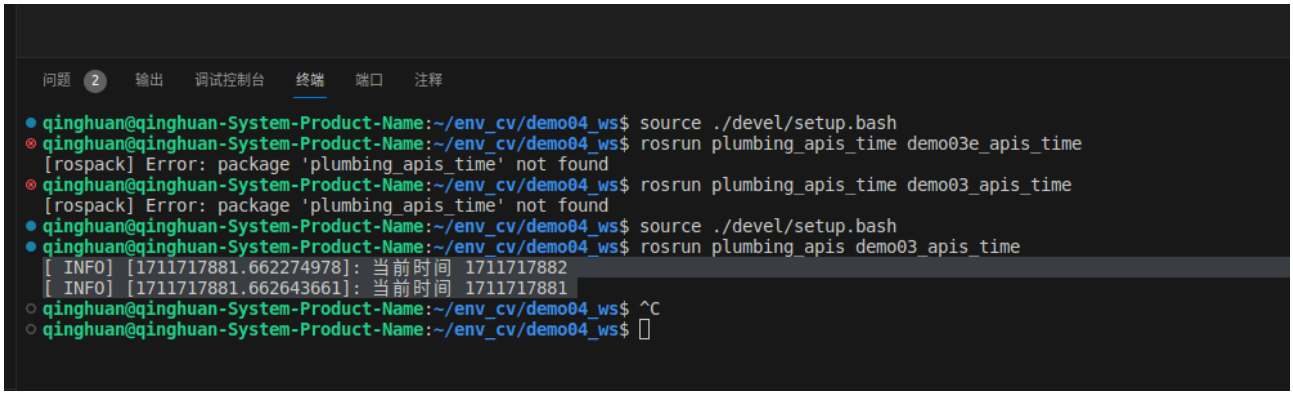
```
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrun plumbing_apis
[ INFO] [1711716401.356907366]: 我发布的信息是:hello world0
[ INFO] [1711716401.357759005]: 一轮回调执行完毕
[ INFO] [1711716401.357802684]: 我发布的信息是:hello world1
[ INFO] [1711716401.458107185]: 一轮回调执行完毕
[ INFO] [1711716401.458149034]: 我发布的信息是:hello world2
[ INFO] [1711716401.558112299]: 一轮回调执行完毕
[ INFO] [1711716401.558158208]: 我发布的信息是:hello world3
[ INFO] [1711716401.658106942]: 一轮回调执行完毕
[ INFO] [1711716401.658152300]: 我发布的信息是:hello world4
[ INFO] [1711716401.758105496]: 一轮回调执行完毕
[ INFO] [1711716401.758150444]: 我发布的信息是:hello world5
[ INFO] [1711716401.858111248]: 一轮回调执行完毕
[ INFO] [1711716401.858156966]: 我发布的信息是:hello world6
```

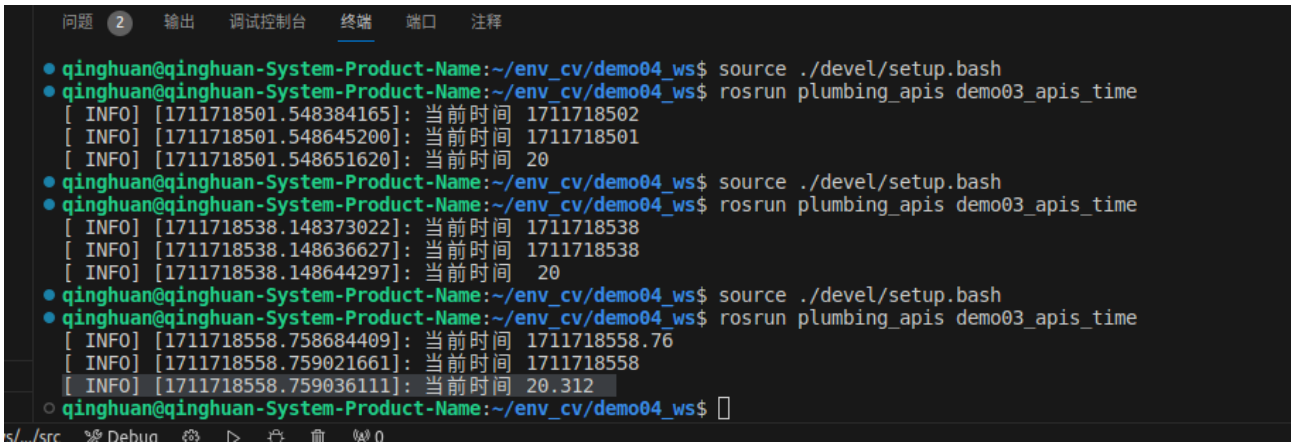
spin 自己进入循环不会自动推出
spin 后面的代码是不会执行的

订阅的信息 可以看到



ros::Time 获得时刻





ros::Time 是一个时刻 ros::Duration 是持续时间
在执行的时候 停止 3s
ros::Duaration du(4.5);
du.sleep();//休 4.5s

```
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrun plumbing_apis demo03_apis_time
[ INFO] [1711762692.675328157]: 当前时间 1711762692.68
[ INFO] [1711762692.675808830]: 当前时间 1711762692
[ INFO] [1711762692.675815540]: 当前时间 秒和毫秒23.123
[ INFO] [1711762692.675820210]: 当前时间 浮点数 20.312
[ INFO] [1711762692.675824210]: 开始休眠 浮点数 1711762692.676
[ INFO] [1711762697.176328511]: 休眠结束 浮点数 1711762697.176
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
```

时刻与持续时间 加减操作都可以
持续时间与持续时间 加减操作都可以
时刻之间只能减法

```
问题 3  输出  调试控制台  终端  端口  注释

qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrun plumbing_apis demo03_apis_time
[ INFO] [1711764559.497063164]: 当前时间 1711764559.50
[ INFO] [1711764559.497322078]: 当前时间 1711764559
[ INFO] [1711764559.497328368]: 当前时间 秒和毫秒23.123
[ INFO] [1711764559.497333248]: 当前时间 浮点数 20.312
[ INFO] [1711764559.497337498]: 开始休眠 浮点数  st  = 1711764559.497
[ INFO] [1711764563.997911347]: 休眠结束 浮点数  sted = 1711764563.998
[ INFO] [1711764563.997951816]: 开始 ros::Time st1 = ros::Time::now(); 1711764563.998
[ INFO] [1711764563.997965426]: 结束 ros::Time duuu = st1 - du2 1711764559.998
[ INFO] [1711764563.997989345]: 结束 ros::Time duuu = st1 + du2 1711764567.998
[ INFO] [1711764563.997996105]: 结束 ros::Duration sum_time = st- sted;; -4.501
[ INFO] [1711764563.998002785]: ros::Duration du3 = du2-sum_time;   8.501
[ INFO] [1711764563.998009155]: ros::Duration du3 = du2 + sum_time;   -0.501
[ INFO] [1711764563.998016795]: ros::Duration du44 =  du-  sum_time;  9.001
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
```

ros::Timer time1=nh.createTimer(ros::Duration(1), cv);//每隔开 1 秒输出一个 cv 中的信息
ros::spin();

```
问题 3  输出  调试控制台  终端  端口  注释

[ INFO] [1711765464.393888244]: 开始休眠 浮点数  st  = 1711765464.394
[ INFO] [1711765468.893985407]: 休眠结束 浮点数  sted = 1711765468.894
[ INFO] [1711765468.894011517]: 开始 ros::Time st1 = ros::Time::now(); 1711765468.894
[ INFO] [1711765468.894026547]: 结束 ros::Time duuu = st1 - du2 1711765464.894
[ INFO] [1711765468.894043987]: 结束 ros::Time duuu = st1 + du2 1711765472.894
[ INFO] [1711765468.894050757]: 结束 ros::Duration sum_time = st- sted;; -4.500
[ INFO] [1711765468.894057147]: ros::Duration du3 = du2-sum_time;   8.500
[ INFO] [1711765468.894064707]: ros::Duration du3 = du2 + sum_time;   -0.500
[ INFO] [1711765468.894070987]: ros::Duration du44 =  du-  sum_time;  9.000
[ INFO] [1711765468.894077077]: ----------------
[ INFO] [1711765469.894259337]: --------------
[ INFO] [1711765470.894505048]: --------------
[ INFO] [1711765471.894512820]: --------------
[ INFO] [1711765472.894219480]: --------------
[ INFO] [1711765473.894222867]: --------------
```

```
void cv(const ros::TimerEvent& event)
{
//传入一个时间变量 实参 const不改变值在callback内
    //时间的事件
    ROS_INFO_STREAM("--------------");
}
int main(int argc,char* argv[])
```

这个使用方案 就算一直使用

如何使用定时器 其他的两个参数
这个就算只是执行一次 就多一个 true 的结果

```
    ros::Timer time1=nh.createTimer(ros::Duration(1), cv,true);//每隔开1秒输出一个cv中的信息
    ros::spin();
    return 0;
}
```

```
问题 3  输出  调试控制台  终端  端口  注释

[ INFO] [1711767507.557469093]: 当前时间 1711767507.56
[ INFO] [1711767507.557992572]: 当前时间 1711767507
[ INFO] [1711767507.557999572]: 当前时间 秒和毫秒23.123
[ INFO] [1711767507.558004202]: 当前时间 浮点数 20.312
[ INFO] [1711767507.558017592]: 开始休眠 浮点数  st  = 1711767507.558
[ INFO] [1711767512.058147217]: 休眠结束 浮点数  sted = 1711767512.058
[ INFO] [1711767512.058189026]: 开始 ros::Time st1 = ros::Time::now(); 1711767512.058
[ INFO] [1711767512.058202946]: 结束 ros::Time duuu = st1 - du2 1711767508.058
[ INFO] [1711767512.058211546]: 结束 ros::Time duuu = st1 + du2 1711767516.058
[ INFO] [1711767512.058228896]: 结束 ros::Duration sum_time = st- sted;; -4.500
[ INFO] [1711767512.058234985]: ros::Duration du3 = du2-sum_time;   8.500
[ INFO] [1711767512.058241495]: ros::Duration du3 = du2 + sum_time;   -0.500
[ INFO] [1711767512.058247565]: ros::Duration du44 =  du-  sum_time;  9.000
[ INFO] [1711767512.058255925]: -------分割----------
[ INFO] [1711767513.058466819]: --------------
```

ros::Timer time1=nh.createTimer(ros::Duration(1), cv,true,false);//
false 这个参数 不自己启动，我们可以决定 这个 timer 什么时候启动 时候休息 yeah

```
78      ROS_INFO("-------分割----------");
79      /*
80      传入ros::Duration 时间间隔
81      const ros::TimeCallback &callback  回调函数 封装执行的任务
82      bool oneshot = false   是不是一次性的定时器 true 隔1s后，就会运行一次，但是运行一次就结束了
83      bool autostart = true
84      */
85
86      //os::Timer time1=nh.createTimer(ros::Duration(1), cv);//每隔开1秒输出一个cv中的信息
87
88      //ros::Timer time1=nh.createTimer(ros::Duration(1), cv,true);//每隔开1秒输出一个cv中的信息  ，家了true后就只运行一次了
89      ros::Timer time1=nh.createTimer(ros::Duration(1), cv,true,false);//
90      ros::spin();
91      return 0;
92  }
```

```
问题 3  输出  调试控制台  终端  端口  注释

qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrun plumbing_apis demo03_apis_time
[ INFO] [1711767672.340013357]: 当前时间 1711767672.34
[ INFO] [1711767672.340358881]: 当前时间 1711767672
[ INFO] [1711767672.340365291]: 当前时间 秒和毫秒23.123
[ INFO] [1711767672.340370191]: 当前时间 浮点数 20.312
[ INFO] [1711767672.340374461]: 开始休眠 浮点数  st  = 1711767672.349
[ INFO] [1711767676.840470653]: 休眠结束 浮点数  sted = 1711767676.840
[ INFO] [1711767676.840498732]: 开始 ros::Time st1 = ros::Time::now(); 1711767676.840
[ INFO] [1711767676.840517342]: 结束 ros::Time duuu = st1 - du2 1711767672.840
[ INFO] [1711767676.840525602]: 结束 ros::Time duuu = st1 + du2 1711767680.840
[ INFO] [1711767676.840531322]: 结束 ros::Duration sum_time = st- sted;; -4.500
[ INFO] [1711767676.840535921]: ros::Duration du3 = du2-sum_time;   8.500
[ INFO] [1711767676.840541791]: ros::Duration du3 = du2 + sum_time;   -0.500
[ INFO] [1711767676.840549711]: ros::Duration du44 =  du-  sum_time;  9.000
[ INFO] [1711767676.840556791]: -------分割----------
```

我们通过 time1.start() 启动 timer
能看到先输出了我们的 ros info 我们先用这个告诉你 我们要启动了， 之后启动 触发回调函数

```
77      // 使用定时器实现rate(10) 10hz的处理数据
78      ROS_INFO("-------分割----------");
79      /*
80      传入ros::Duration 时间间隔
81      const ros::TimeCallback &callback  回调函数 封装执行的任务
82      bool oneshot = false    是不是一次性的定时器 true 隔1s后，就会运行一次，但是运行一次就结束了
83      bool autostart = true
84      */
85
86      //os::Timer time1=nh.createTimer(ros::Duration(1), cv);//每隔开1秒输出一个cv中的信息
87
88      //ros::Timer time1=nh.createTimer(ros::Duration(1), cv,true);//每隔开1秒输出一个cv中的信息
89      ros::Timer time1=nh.createTimer(ros::Duration(1), cv,true,false);//
90      ROS_INFO("-------Timer 启动----------");
91      time1.start();
92      ros::spin();
93
```

问题 3　输出　调试控制台　终端　端口　注释

```
[ INFO] [1711767792.988289257]: 当前时间 1711767792
[ INFO] [1711767792.988299617]: 当前时间 秒和毫秒23.123
[ INFO] [1711767792.988304817]: 当前时间 浮点数 20.312
[ INFO] [1711767792.988308917]: 开始休眠 浮点数  st  = 1711767792.988
[ INFO] [1711767797.488755125]: 休眠结束 浮点数  sted = 1711767797.489
[ INFO] [1711767797.488791584]: 开始 ros::Time st1 = ros::Time::now(); 1711767797.489
[ INFO] [1711767797.488805964]: 结束 ros::Time duuu = st1 + du2 1711767793.489
[ INFO] [1711767797.488824543]: 结束 ros::Time duuu = st1 + du2 1711767801.489
[ INFO] [1711767797.488831003]: 结束 ros::Duration sum_time = st- sted;; -4.500
[ INFO] [1711767797.488836743]: ros::Duration du3 = du-sum_time;   8.500
[ INFO] [1711767797.488845953]: ros::Duration du3 = du2 + sum_time;   -0.500
[ INFO] [1711767797.488852993]: ros::Duration du44 = du-  sum_time;  9.000
[ INFO] [1711767797.488865503]: -------分割----------
[ INFO] [1711767797.488886952]: -------Timer 启动----------
[ INFO] [1711767798.489349380]: --------------
```

现在我们修改回调函数

```
void cv(const ros::TimerEvent& event)
{
//传人一个时间变量 实参 const不改变值在callback内
    //时间的事件
    ROS_INFO("--------------");
    ROS_INFO("调用当前时间%f",event.current_real.toSec());
}
int main(int argc, char* argv[])
```

我们使用了打印的时间

整体的一个总结 可以看赵虚左 也作了个总结

```
    ROS_INFO("-------分割----------");
    /*
    nh.createTimer的传入参数说明

    传入ros::Duration 时间间隔
    const ros::TimeCallback &callback  回调函数 封装执行的任务
    bool oneshot = false   是不是一次性的定时器 true 隔1s后，就会运行一次，但是运行一次就结束了
    bool autostart = true
    */


    //这个是 基础两个参数的使用 1
    //ros::Timer time1=nh.createTimer(ros::Duration(1), cv);//每隔开1秒输出一个cv中的信息

    // 只运行一次   oneshot的使用 用的时候 最好把其他的注释调 2
    //ros::Timer time1=nh.createTimer(ros::Duration(1), cv,true);//每隔开1秒输出一个cv中的信息  ，家了true后就只运行一次了


    // 这个是只运行一次 而且不自己启动 用我们的time1.start();让我们自己选择什么时候 启动
    //ros::Timer time1=nh.createTimer(ros::Duration(1), cv,true,false);//        3
    //ROS_INFO("-------Timer 启动----------");
    //time1.start();
    // 这个是修改了 回调函数的例子， 这个回调函数的例子，是和传入的参数 的中种类不相关的，就是需要我去查一下，这个类型的数据 里面都有什么成员函数
    ros::Timer time1=nh.createTimer(ros::Duration(1), cv,false,true);//                                                    4

    ros::spin();
    return 0;
}
```

节点的声明周期 与 发布消息

ros::ok() 判断节点是否运行正常

1、节点接受到了关闭信息 ctrl + c 在终端的输入

2、新启动了一个节点 与已有的节点名字相同

3、其余节点 ros::shutdown 调用了 shut_down 节点

shutdown 在 demo01_apis_pub.cpp 文件实现的看一下

```
62      return 0;
63  }

问题 3   输出   调试控制台   终端   端口   注释

[ INFO] [1711768700.215926048]: 我发布的信息是:hello world6
[ INFO] [1711768700.315735441]: 一轮回调执行完毕
[ INFO] [1711768700.315761219]: 我发布的信息是:hello world7
[ INFO] [1711768700.415737308]: 一轮回调执行完毕
[ INFO] [1711768700.415763515]: 我发布的信息是:hello world8
[ INFO] [1711768700.515742524]: 一轮回调执行完毕
[ INFO] [1711768700.515773932]: 我发布的信息是:hello world9
[ INFO] [1711768700.615739371]: 一轮回调执行完毕
[ INFO] [1711768700.715732369]: 一轮回调执行完毕
[ INFO] [1711768700.815746454]: 一轮回调执行完毕
[ INFO] [1711768700.915760110]: 一轮回调执行完毕
[ INFO] [1711768701.015733930]: 一轮回调执行完毕
[ INFO] [1711768701.115763332]: 一轮回调执行完毕
[ INFO] [1711768701.215744989]: 一轮回调执行完毕
[ INFO] [1711768701.215777257]: shutdown
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
```

log 四类 log 信息对应的内容

DEBUG(调试):只在调试时使用，此类消息不会输出到控制台；

- INFO(信息):标准消息，一般用于说明系统内正在执行的操作；
- WARN(警告):提醒一些异常情况，但程序仍然可以执行；
- ERROR(错误):提示错误信息，此类错误会影响程序运行；
- FATAL(严重错误):此类错误将阻止节点继续运行。