初始化

```
ros::init(argc, argv, "xxx");
```

argc：参数个数（n+1）个
argv：参数数组
"xxx"：节点名（唯一）

可以在命令行中直接向数组（argv）传参，下划线表示开始传参

例：`rosrun api_vscode time _length:=3`

若节点名相同，则同名的节点中后启动的节点会导致较先启动的节点死亡。

可使用options给相同的节点名后增加随机数，本质上使节点名字互异，故同名节点可以同时启动

使用方法：`ros::init(argc, argv, ros::init::options::AnonymousName);`

可以用 rosnode list 看出节点之后有一串随机数。

话题与服务相关对象（进阶）

```
ros::Publisher pub = nh.advertise<std_msgs::String>("fancy", 10, true);
```

ros::Publisher：内置函数
nh：句柄
advertise：函数
std_msgs::String：消息类型
"fancy"：话题名
10：队列长度
true：latch(bool)

latch(bool)

默认为false，若设置为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。

设置latch为true就不用再循环发布同一条消息了

回旋函数 spin()与spinOnce()

可将spin()视作一个死循环，因此spin()后的代码均不执行

spinOnce()视作一个执行一次的循环。

双方均用于处理回调函数

时间与时刻

获取当前时刻：ros::Time::now()；返回值也为ros::Time类型

返回当前时刻，即now()函数被调用的时刻（时刻类型）

参考系：1970年01月01日 00:00:00.

函数：.toSec()可以将时刻转换为秒数，返回double类型数据.

字段：.sec可以将时刻转换为秒数，返回int类型数据.

示例：
```
ros::Time right_now = ros::Time::now();
ROS_INFO("%.2f", right_now.toSec());
ROS_INFO("%d", right_now.sec);
```

设置指定时刻：
```
ros::Time t1(20, 124321);   20秒+124321纳秒
ros::Time t2(20.124);       20.124秒
```

时间类型（持续时间）：ros::Duration.

设置持续时间（停顿时间）：

设置好停顿时间后调用sleep();
```
ros::Duration du2(5.20);   停顿5.2秒
du2.sleep();
```

时间运算（时刻与时刻之间相减，不可相加）结果为Duration类型

如
```
ros::Time begin = ros::Time::now();
ros::Duration du1(5);
du1.sleep();
ros::Time stop = ros::Time::now();
ros::Duration du = begin - stop;
```

时刻与时间相加或相减，结果为时刻（Time类型）

时间与时间相加减，结果为Duration类型。

定时器：

ros::Timer 类型

参数：

callback 回调函数

oneshot 定时器是否是一次性执行的，默认为 false.

autostart 定时器是否自动启动，默认为 true.

使用方法

```
ros::Timer timer = nh.creatTimer(ros::Duration(1), callback, false, false);
```

- timer：自定义定时器名
- creatTimer：创建定时器的内置函数
- ros::Duration(1)：停顿时间
- callback：回调函数名
- 第一个 false：oneshot
- 第二个 false：autostart

如果不自动启动，则需使用定时器启动函数手动启动定时器

```
timer.start();
```

自定义头文件的创建与调用

在功能包的 /include 下创建 .h 头文件

src > head_vscode > include > head_vscode > C hello.h > { } hello_ns > Myhello

```
#ifndef _HELLO_H
#define _HELLO_H
/*
    声明namespace
            |--class
                |--run
*/
namespace hello_ns
{
    class Myhello
    {
        private:

        public:
        void run();
    };
}
#endif
```

使用头文件示例

```
#include "ros/ros.h"
#include "head_vscode/hello.h"

namespace hello_ns
{
    void Myhello::run()
    {
        ROS_INFO("函数执行");

    }
}
int main(int argc, char  *argv[])
{
    setlocale(LC_ALL,"");
    ros::init(argc,argv,"hello_head");
    hello_ns::Myhello myhello;
    myhello.run();
    return 0;
}

```

特别地，在 CMakeList 的配置中应当把 118～121 放开注释

```
118  include_directories(
119  include
120    ${catkin_INCLUDE_DIRS}
121  )
```

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文件去使用(在同一命名空间内)

src > head_src_vscode > src > hello.cpp > ...

```cpp
#include "head_src_vscode/hello.h"
#include "ros/ros.h"
namespace hello_ns
{
    void Myhello::run()
    {
        ROS_INFO("源文件中的run函数");
    }
}
```

src > head_src_vscode > src > use_hello.cpp > main(int, char * [])

```cpp
#include "ros/ros.h"
#include "head_src_vscode/hello.h"
int main(int argc, char  *argv[])
{
    setlocale(LC_ALL,"");
    ros::init(argc,argv,"use_hello");

    //空间
    hello_ns::Myhello myhello;
    myhello.run();

    return 0;
}
```

调用自定义源文件

CMakelist.txt 配置

```
## 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::shutdown() ---> 关闭(杀死)节点.

执行完毕后返回 false.

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

void shutdown() 关闭函数

日志函数

```
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
```