

## 进阶课程③⑥ | Apollo ROS深入介绍

ROS是一个强大而灵活的机器人编程框架，从软件构架的角度说，它是一种基于**消息传递通信的分布式多进程框架**。ROS本身是基于消息机制的，可以根据功能把软件拆分成为各个模块，每个模块只是负责读取和分发消息，模块间通过消息关联。

本周阿波君将继续与大家分享**Apollo ROS**的相关课程。下面，我们一起进入进阶课程第36期。

**目前ROS仅适用于Apollo 3.0之前的版本，最新代码及功能还请参照Apollo 3.5及5.0版本。**

以下，ENJOY

本节内容主要介绍ROS中一些不是特别常见的属性。



# ROS Packages

## ROS Packages

- ROS software is organized into *packages*, which can contain source code, launch files, configuration files, message definitions, data, and documentation
- A package that builds up on/requires other packages (e.g. message definitions), declares these as *dependencies*

To create a new package, use

```
> catkin_create_pkg package_name  
{dependencies}
```



创建一个ROS开发环境和写一个C++工程有点类似，通过**catkin create**可以创建一个简单的工程。其中的文件组织方式如上图所示，包括：

SRC存放源文件；

MSG存放节点之间进行通信的消息定义；

SRV存放节点之间进行服务通信的时候的服务定义；

CONFIG存放配置文件相关的信息；

INCLUDE存放头文件相关的信息；

Launch存放节点启动和它相关的节点之间的启动文件。

上面介绍的package组织方式只是官方推荐的一种组织方式，使用catkin build编译，当source完环境变量之后，通过Ros提供的命令比如ROS run或者ROS launch启动时，package name可以自动补全，package里面包含的节点或者launch文件也是可以自动寻找，所以官方推荐使用这种组织方式。

DEVEL和BUILD这两个目录是build时自动产生的两个临时目录。

此外，开发环境还有描述ROS Package相关工作区的两个文件：**Package.xml**和**Cmakelists.txt**。

## ROS Packages

### package.xml

- The package.xml file defines the properties of the package
  - Package name
  - Version number
  - Authors
  - Dependencies on other packages
  - ...

#### package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>ros_package_template</name>
  <version>0.1.0</version>
  <description>A template for ROS packages.</description>
  <maintainer email="pfankhauser@ethz.ch">Peter Fankhauser</maintainer>
  <license>BSD</license>
  <url type="website">https://github.com/ethz-asl/ros_best_pr</url>
  <author email="pfankhauser@ethz.ch">Peter Fankhauser</author>

  <buildtool_depend>catkin</buildtool_depend>

  <depend>roscpp</depend>
  <depend>sensor_msgs</depend>
</package>
```

**Package.xml**定义了可执行文件依赖的一些库，包括编译和运行时依赖库，同时定义了软件版本信息等常见的描述文件。

## ROS Packages

### CMakeLists.xml

The CMakeLists.txt is the input to the CMakebuild system

1. Required CMake Version (cmake\_minimum\_required)
2. Package Name (project())
3. Find other CMake/Catkin packages needed for build (find\_package())
4. Message/Service/Action Generators (add\_message\_files(), add\_service\_files(), add\_action\_files())
5. Invoke message/service/action generation (generate\_messages())
6. Specify package build info export (catkin\_package())
7. Libraries/Executables to build (add\_library()/add\_executable()/target\_link\_libraries())
8. Tests to build (catkin\_add\_gtest())
9. Install rules (install())

#### CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_package_template)

## Use C++11
add_definitions(--std=c++11)

## Find catkin macros and libraries
find_package(catkin REQUIRED COMPONENTS
  roscpp
  sensor_msgs
)

--
```

[More info](#)

**Cmakelists.txt**定义了怎么编译ROS工程的规则，主要定义了以下几个部分：

- 1、指定Cmake的版本；
- 2、工程Package name；
- 3、工程的头文件信息；
- 4、指定所依赖的库，与在Package.xml里面所指定的依赖库是一一对应的；
- 5、install命令，将编译出来的临时文件放在指定的目录里。

下面是Cmakelists文件的一个例子。

## ROS Packages

### CMakeLists.xml Example

<pre>cmake_minimum_required(VERSION 2.8.3) project(husky_highlevel_controller) add_definitions(--std=c++11)  find_package(catkin REQUIRED COMPONENTS roscpp sensor_msgs)  catkin_package(   INCLUDE_DIRS include   # LIBRARIES   CATKIN_DEPENDS roscpp sensor_msgs   # DEPENDS )  include_directories(include \${catkin_INCLUDE_DIRS})  add_executable(\${PROJECT_NAME} src/\${PROJECT_NAME}_node.cpp src/HuskyHighlevelController.cpp)  target_link_libraries(\${PROJECT_NAME} \${catkin_LIBRARIES})</pre>	<p>Use the same name as in the package.xml</p> <p>We use C++11 by default</p> <p>List the packages that your package requires to build (have to be listed in package.xml)</p> <p>Specify build export information</p> <ul style="list-style-type: none"><li>• INCLUDE_DIRS: Directories with header files</li><li>• LIBRARIES: Libraries created in this project</li><li>• CATKIN_DEPENDS: Packages dependent projects also need</li><li>• DEPENDS: System dependencies dependent projects also need (have to be listed in package.xml)</li></ul> <p>Specify locations of of header files</p> <p>Declare a C++ executable</p> <p>Specify libraries to link the executable against</p>
---	---

从上到下依次指定了Cmake的版本、project的名字、ROS工程所依赖的c++的版本、另外是依赖的库文件，最后生成可执行文件以及这个文件所链接的[依赖库](#)。



# Eclipse下 编译ROS基本工程

## Setup a Project in Eclipse

- Build the Eclipse project files with additional build flags

```
> catkin build package_name -G"Eclipse CDT4 - Unix Makefiles"  
-D__cplusplus=201103L -D__GXX_EXPERIMENTAL_CXX0X__=1
```

- To use flags by default in your catkin environment, use the catkin config command.
- The Eclipse project files will be generated in  
~/catkin\_ws/build

The build flags are already setup in the provided installation.

工程建立好之后，catkin build可以直接对工程进行编译。直接用 catkin build去编译，会把整个工程目录里面的所有的package进行统一编译，如果是构建一个比较复杂的系统，可能一个文件夹包含了很多节点或者package包，编译时间会比较久，可以通过指定package名，编译某一个固定的package包，提升编译效率。

下面介绍在Eclipse下如何编译ROS基本工程

## Setup a Project in Eclipse

- Start Eclipse and set the workspace folder

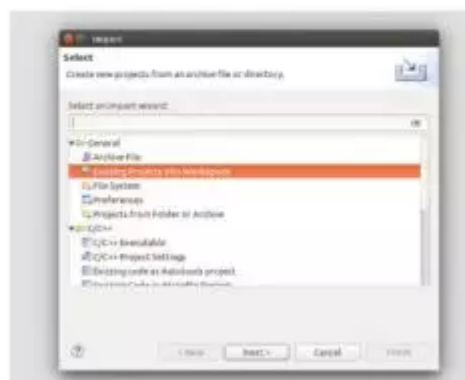


The Eclipse workspace is already set in the provided installation.

## Setup a Project in Eclipse

- Import your project to Eclipse

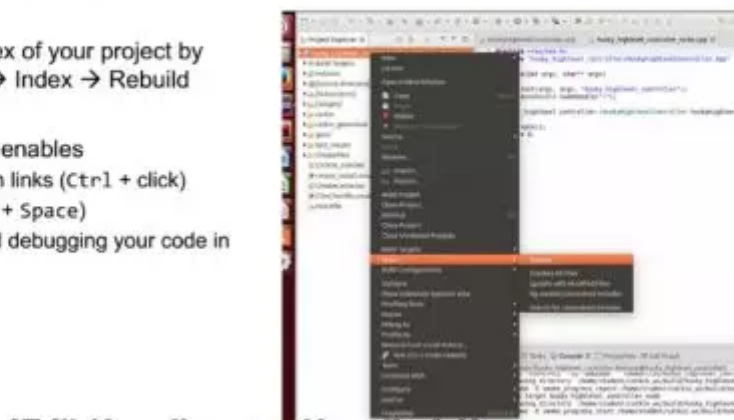
File → Import → General  
→ Existing Projects into Workspace



首先是设置工作ROS工作区，然后将ROS package导入到Eclipse设置的工作区。

## Setup a Project in Eclipse

- Rebuild the C/C++ index of your project by  
Right click on Project → Index → Rebuild
- Resolving the includes enables
  - Fast navigation through links (Ctrl + click)
  - Auto-completion (Ctrl + Space)
  - Building (Ctrl + B) and debugging your code in Eclipse



通过Eclipse提供的build或者run等功能去调试ROS工程。同时Eclipse里面提供的快捷键在编译程序里面同样适用。



# 通过hello world 了解ROS基本的运行逻辑

## ROS C++ Client Library (*roscpp*)

```
hello_world.cpp
#include <ros/ros.h>

int main(int argc, char** argv)
{
    ros::init(argc, argv, "hello_world");
    ros::NodeHandle nodeHandle;
    ros::Rate loopRate(10);

    unsigned int count = 0;
    while (ros::ok()) {
        ROS_INFO_STREAM("Hello World " << count);
        ros::spinOnce();
        loopRate.sleep();
        count++;
    }

    return 0;
}
```

- ROS main header file include
- ros::init(...) has to be called before calling other ROS function
- The node handle is the access point for communications with the ROS system (topics, services, parameters)
- ros::Rate is a helper class to run loops at a desired frequency
- ros::ok() checks if a node should continue running  
Returns false if SIGINT is received (Ctrl + C) or ros::shutdown() has been called
- ROS\_INFO() logs messages to the filesystem
- ros::spinOnce() processes incoming messages via callbacks

[More info](#)



上图的hello world程序展现了ROS框架写Node所使用的核心要素：

Include 就是include ROS的一个基本环境；

Main函数里面有三行需要重点关注：

**init**：引入ROS的一个基本环境，指定节点使用的node名字和一些参数信息；

**NodeHandle**：node和整个ROS框架进行通信所使用的一个句柄指针；

**数据发送的频率**：looprate ( 10 ) 以10赫兹发送消息。

最底下的while循环以10赫兹的消息频率进行发送，同时进行计数。

**Spinonce**：有一帧消息就把这消息立马发送出去，同时进行下一轮的消息等待。



## ROS提供的日志系统

## ROS C++ Client Library (roscpp)

### Logging

- Mechanism for logging human readable text from nodes in the console and to log files
- Instead of `std::cout`, use e.g. `ROS_INFO`
- Automatic logging to console, log file, and `/rosout` topic
- Different severity levels (Info, Warn, Error etc.)
- Supports both `printf`- and stream-style formatting

```
ROS_INFO("Result: %d", result);
ROS_INFO_STREAM("Result: " << result);
```
- Further features such as conditional, throttled, delayed logging etc.

	Debug	Info	Warn	Error	Fatal
<code>stdout</code>	x	x			
<code>stderr</code>			x	x	x
<code>Log file</code>	x	x	x	x	x
<code>/rosout</code>	x	x	x	x	x

! To see the output in the console, set the output configuration to screen in the launch file

```
<launch>
  <node name="listener" output="screen"/>
</launch>
```

[More info](#)

在示例程序里面有一个`ROS_INFO`，它就是ROS提供的日志系统；

ROS的日志系统是分级的，即在编写节点程序的时候对打印的信息进行分级，对不同的分级，ROS会提供不同的颜色和格式进行展示。分级的作用是为了帮助开发者快速地定位到关键信息，不会对整个节点的逻辑产生实质性的影响。

日志系统提供了两种格式`ROS_INFO`与`ROS_INFO_STREAM`：

`ROS_INFO`：默认把信息打印到当时运行的屏幕上。

`ROS_INFO_STREAM`：它是流式数据，默认输出到后台这个节点所对应的日志文件。



Subscriber与Publisher有三点明显的区别:

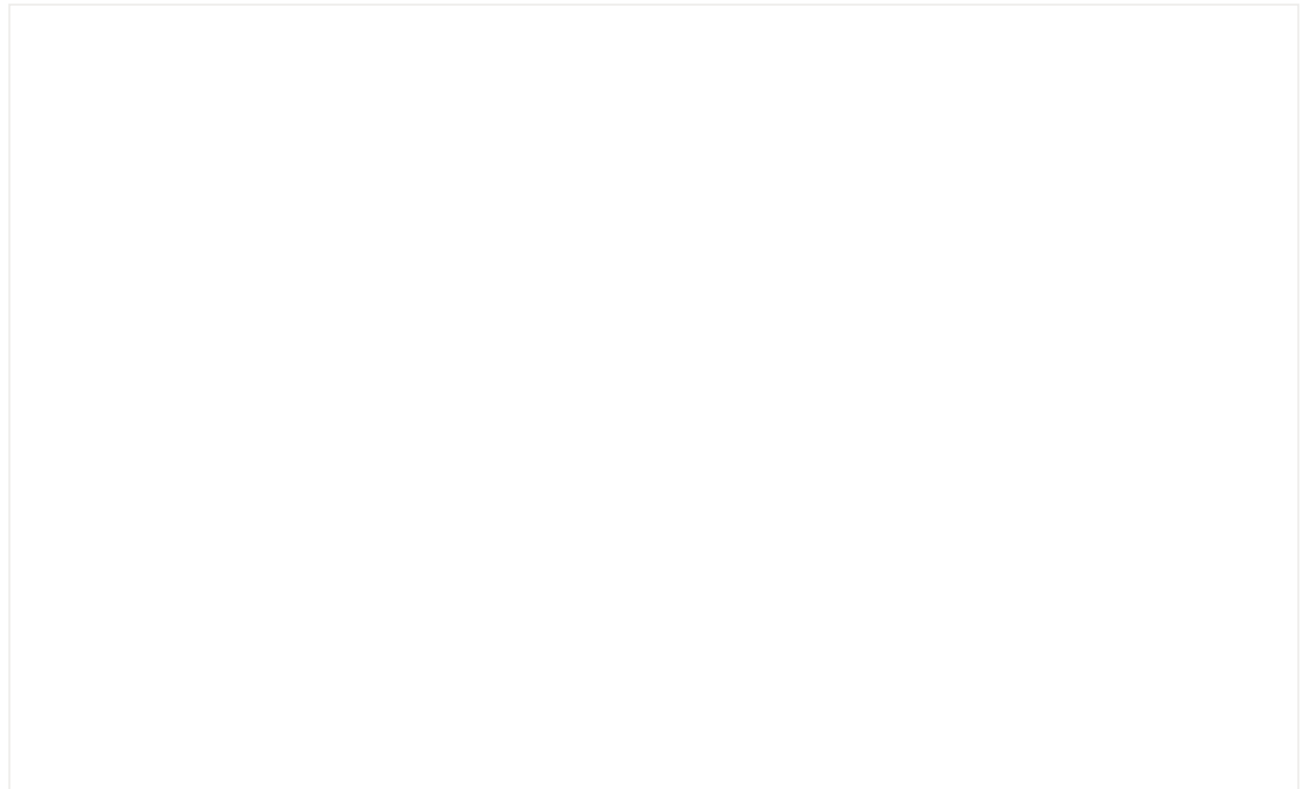
**1、回调函数：**subscriber作为信息的接收方有一个回调函数，回调函数定义了它接收到的每一帧信息如何使用；上图listener回调函数比较简单，它接收到信息后只是进行了打印处理。Publisher没有回调函数，它不需要对消息进行处理。

**2、声明的时候：**subscriber把回调函数传入到对应的node初始化程序里面。publisher声明的时候只需要注册要往哪一个topic上去发信息，同时还设置队列长度。

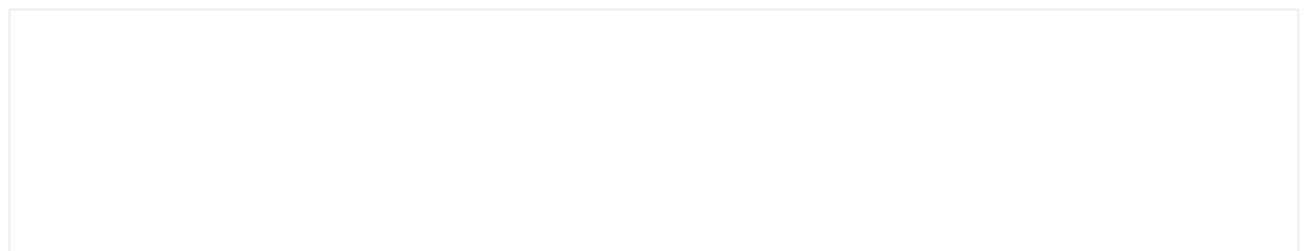
**3、Rosspin：**在ROS构架里所有的回调函数都不是主动触发的。Rosspin是阻塞性的，声明Rosspin之后，就阻塞在此，程序不会退出，它会一直监听自己对应的队列里面是否有新消息的到达，若有新消息到达就会触发回调函数处理。

如果Subscriber的主程序里除了订阅消息之外还有其他的功能则可以采用rospin once，对所有已达消息进行回调函数的处理。同时可以写一个while循环，rospin once按照一定的频率去处理回调函数的消息。ROS提供Rosspin这两种方式，就是为了满足这两种场景。第一种是阻塞，只有一个回调函数进行处理，第二种是订阅回调函数消息以外，他还进行了一些封装和处置。

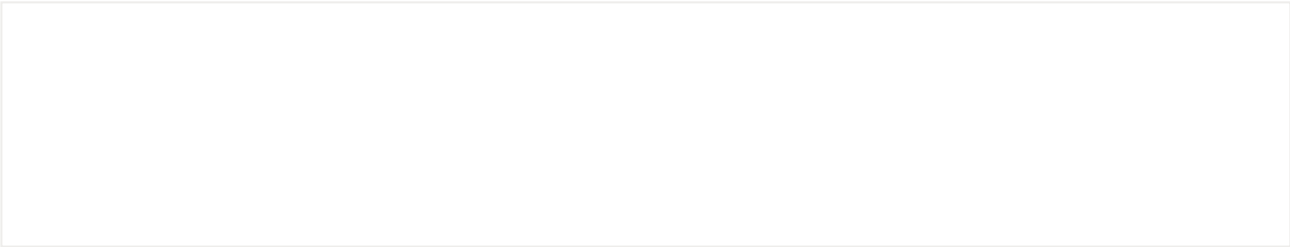
看似很复杂的自动驾驶，节点整体写下来都是比较简单的，就是按照上图Subscriber与Publisher方式来写。但是在实际的自动驾驶系统里面，所有的模块都不是简单的一个角色，它可能既是消息的订阅者也是消息的发送者，是一个复杂交互的功能，甚至是用到很多数据融合或者是消息对齐。




ROS节点之间的通信除了基于message消息订阅和发布模型之外还有另外两种方式，虽然另外两种方式使用的比较少，但是在某些特定的场合是比较有用的。



节点可以启动service去注册一项服务，另外一个节点在使用这项服务的时候可以直接call service完成一些实时的数据通讯交互。Message是一个被动的消息行为，发送者发送消息的时候并不知道消息会被谁去消费，接收者在接收消息的时候也不知道目前有几个发送节点在发送，发送和接收之间是一个什么状态也是不知道的，他们是一个松耦合和透明的关系。Service弥补了这种通信方式的不足，它需要及时回应。Client向server去发送service请求的时候，需要实时等待一个response，根据响应做出下一步的行为指示。




Parameter通信方式借鉴了service的原理。它启动了Parameter service，Parameter service是一个全局的服务器，各个节点在进行参数设置和获取的时候可以通过Parameter service的方式轻易完成。因为Parameter不像基于message消息通讯方式那么频繁，一个参数在设置完成之后，在整个网络拓扑运行期间所有的节点只需要在一个地方取此参数就行或者某个节点根据自己的运行状态去改变这个参数。



Parameter对应有一套ROS所使用的基本命令行工具—rosparam。

rosparam其它工具相比，有两个不同的地方：get和set。get是get某一个全局参数的值；set是设置某一个全局参数的值。



上图展示的例子是通过node所提供的nodeHandler指针去调用它的getParam，得到某个参数的数值。当它进行一些运算之后，也可以通过setparam去设置参数的值。设置之后，整个系统参数服务器里面对应的这个参数就会被设置成对应的值，其他节点在得到这个值之后再作出相应的处理。



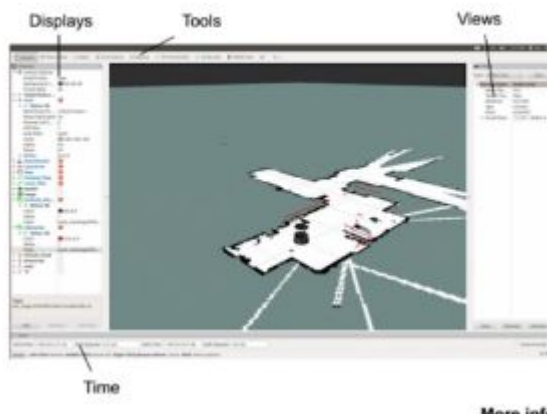
# ROS的可视化工具RViz

## RViz

- 3D visualization tool for ROS
- Subscribes to topics and visualizes the message contents
- Different camera views (orthographic, top-down, etc.)
- Interactive tools to publish user information
- Save and load setup as RViz configuration
- Extensible with plugins

Run RViz with

```
> rosrunc rviz rviz
```

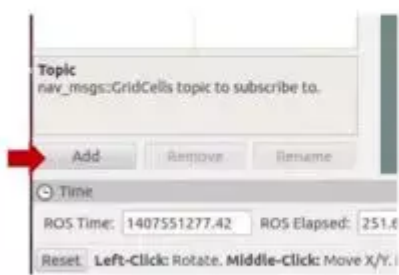


自动驾驶节点比较多，网络拓扑也比较复杂，每个节点在进行消息通讯的时候有很多channel同时运行，如果只是通过命令行工具去查看节点的状态和节点之间的拓扑，会很麻烦。ROS提供了一些比较好的可视化工具立体化展示某一个拓扑结构里面的拓扑网络，RViz就是其中之一。

RViz在整个ROS生态里可以看成是一个节点，它定义了整个拓扑结构里面所有的消息，然后按照固定的格式进行图形化展示，同时提供很多debug相关的功能。因为RViz也是一个普通的节点，所以在启动的时候可以通过roslaunch命令的方式去启动RViz相关的功能。

## RViz

### Display Plugins



RViz也提供了很多插件可以放到诸如eclipse这样的功能插件里面，在进行eclipse开发时可以通过eclipse的plugin去调取RViz的相关功能，进行可视化调试。