***roscpp overview (/roscpp/Overview)**: Initialization and Shutdown | Basics (/roscpp/Overview/Messages) | Advanced: Traits [ROS C Turtle] (/roscpp/Overview/MessagesTraits) | Advanced: Custom Allocators [ROS C Turtle] (/roscpp/Overview/MessagesCustomAllocators) | Advanced: Serialization and Adapting Types [ROS C Turtle] (/roscpp/Overview/MessagesSerializationAndAdaptingTypes) | Publishers and Subscribers (/roscpp/Overview/Publishers%20and%20Subscribers) | Services (/roscpp/Overview/Services) | Parameter Server (/roscpp/Overview/Parameter%20Server) | Timers (Periodic Callbacks) (/roscpp/Overview/Timers) | NodeHandles (/roscpp/Overview/NodeHandles) | Callbacks and Spinning (/roscpp/Overview/Callbacks%20and%20Spinning) | Logging (/roscpp/Overview/Logging) | Names and Node Information (/roscpp/Overview/Names%20and%20Node%20Information) | Time (/roscpp/Overview/Time) | Exceptions (/roscpp/Overview/Exceptions) | Compilation Options (/roscpp/Overview/Compilation%20Options) | Advanced: Internals (/roscpp/Overview/Internals) | tf/Overview (/tf/Overview) | tf/Tutorials (/tf/Tutorials) | C++ Style Guide (/CppStyleGuide)*

目录

1. Initialization

There are two levels of initialization for a roscpp Node (/Nodes):

1. Initializing the node through a call to one of the `ros::init()` functions. This provides command line arguments to ROS, and allows you to name your node and specify other options.
2. Starting the node is most often done through creation of a `ros::NodeHandle`, but in advanced cases can be done in different ways.

Each of these is described below

1.1 Initializing the roscpp Node

See also: ros::init() code API (http://docs.ros.org/api/roscpp/html/init_8h.html)

Before calling any other roscpp functions in a node you must call one of the `ros::init()` functions. The two most common `init()` invokations are:

切换行号显示

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

and

切换行号显示

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

In general, the form of `ros::init()` conforms to:

切换行号显示

```
void ros::init(<command line or remapping arguments>, std::string node_name, uint32_t options);
```

Let's go over the arguments in order:

`argc` and `argv`

ROS uses these to parse remapping arguments (/Remapping%20Arguments) from the command line. It also modifies them so that they no longer contain any remapping arguments, so that if you call `ros::init()` before processing your command line you will not need to skip those arguments yourself.

`node_name`

This is the name that will be assigned to your node unless it's overridden by one of the remapping arguments (/Remapping%20Arguments). Node names must be **unique** across the ROS system. If a second node is started with the same name as the first, the first will be shutdown automatically. In cases where you want multiple of the same node running without worrying about naming them uniquely, you may use the `init_options::AnonymousName` option described below.

`options`

This is an optional argument that lets you specify certain options that change roscpp's behavior. The field is a bitfield, so multiple options can be specified. The options are described in the Initialization Options (/roscpp/Overview/Initialization%20and%20Shutdown#InitOptions) section.

There are other forms of `ros::init()` that do not take argc/argv, instead taking explicit remapping options. Specifically, there are versions that take a `std::map<std::string, std::string>` and a `std::vector<std::pair<std::string, std::string> >`.

Initializing the node simply reads the command line arguments and environment to figure out things like the node name, namespace and remappings. It does **not** contact the master. This lets you use `ros::master::check()` and other ROS functions after calling `ros::init()` to check on the status of the master. The node only full spins up once it has been started, which is described in the Starting the roscpp Node (/roscpp/Overview/Initialization%20and%20Shutdown#Starting) section.

1.1.1 Initialization Options

See also: ros::init_options code API (http://www.ros.org/doc/api/roscpp/html/namespaceros_1_1init__options.html)

`ros::init_options::NoSigintHandler`

Don't install a SIGINT handler. You should install your own SIGINT handler in this case, to ensure that the node gets shutdown correctly when it exits. Note that the default action for

SIGINT tends to be to terminate the process, so if you want to do your own SIGTERM handling you will also have to use this option.

`ros::init_options::AnonymousName`

Anonymize the node name. Adds a random number to the end of your node's name, to make it unique.

`ros::init_options::NoRosout`

Don't broadcast rosconsole (/rosconsole) output to the `/rosout` topic.

1.1.2 Accessing Your Command Line Arguments

As mentioned above, calling `ros::init()` with argc and argv will remove ROS arguments from the command line. If you need to parse the command line before calling `ros::init()`, you can call the (*new in ROS 0.10*) ros::removeROSArgs() (http://docs.ros.org/api/roscpp/html/init_8h.html) function.

1.2 Starting the roscpp Node

The most common way of starting a roscpp node is by creating a `ros::NodeHandle`:

切换行号显示

```
ros::NodeHandle nh;
```

When the first `ros::NodeHandle` is created it will call `ros::start()`, and when the last `ros::NodeHandle` is destroyed, it will call `ros::shutdown()`. If you want to manually manage the lifetime of the node you may call `ros::start()` yourself, in which case you should call `ros::shutdown()` before your program exits.

2. Shutting Down

2.1 Shutting Down the Node

At any time you may call the `ros::shutdown()` function to shutdown your node. This will kill all open subscriptions, publications, service calls, and service servers.

By default roscpp also installs a SIGINT handler which will detect `Ctrl-C` and automatically shutdown for you.

2.2 Testing for Shutdown

There are two methods to check for various states of shutdown. The most common is `ros::ok()`. Once `ros::ok()` returns false, the node has finished shutting down. A common use of `ros::ok()`:

切换行号显示

```
while (ros::ok())
{
  ...
}
```

The other method to check for shutdown is the `ros::isShuttingDown()` method. This method will turn true as soon as `ros::shutdown()` is called, not when it is done. Use of `ros::isShuttingDown()` is generally discouraged, but can be useful in advanced cases. For example, to test inside of a prolonged service callback whether the node is requested to shut down and the callback should therefore quit now, `ros::isShuttingDown()` is needed. `ros::ok()` would not work here because the node can't finish its shutdown as long as the callback is running.

2.2.1 Custom SIGINT Handler

You can install a custom SIGINT handler that plays nice with ROS like so:

切换行号显示

```
#include <ros/ros.h>
#include <signal.h>

void mySigintHandler(int sig)
{
  // Do some custom action.
  // For example, publish a stop message to some other nodes.

  // All the default sigint handler does is call shutdown()
  ros::shutdown();
}

int main(int argc, char** argv)
{
  ros::init(argc, argv, "my_node_name", ros::init_options::NoSigintHandler);
  ros::NodeHandle nh;

  // Override the default ros sigint handler.
  // This must be set after the first NodeHandle is created.
  signal(SIGINT, mySigintHandler);

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

Wiki: roscpp/Overview/Initialization and Shutdown (2015-11-03 18:30:26由WilliamWoodall (/WilliamWoodall)编辑)