***roscpp overview (/roscpp/Overview)**: Initialization and Shutdown (/roscpp/Overview/Initialization%20and%20Shutdown) | 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 | 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

The `ros::NodeHandle` class serves two purposes. First, it provides RAII (http://en.wikipedia.org/wiki/Resource_Acquisition_Is_Initialization)-style startup and shutdown of the internal node inside a roscpp (/roscpp) program. Second, it provides an extra layer of namespace resolution that can make writing subcomponents easier.

1. Automatic Startup and Shutdown

As explained in the Initialization and Shutdown roscpp overview (/roscpp/Overview/Initialization%20and%20Shutdown), `ros::NodeHandle` manages an internal reference count to make starting and shutting down a node as simple as:

切换行号显示

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

On creation, if the internal node has not been started already, `ros::NodeHandle` will start the node. Once all `ros::NodeHandle` instances have been destroyed, the node will be automatically shutdown.

2. Namespaces

See also: ROS names documentation (/Names)

`NodeHandle`s let you specify a namespace to their constructor:

切换行号显示

```
1 ros::NodeHandle nh("my_namespace");
```

This makes any *relative* name (/Names) used with that `NodeHandle` relative to `<node_namespace>/my_namespace` instead of just `<node_namespace>`.

You can also specify a parent `NodeHandle` and a namespace to append:

切换行号显示

```
1 ros::NodeHandle nh1("ns1");
2 ros::NodeHandle nh2(nh1, "ns2");
```

This puts `nh2` into the `<node_namespace>/ns1/ns2` namespace.

Global Names

If you really want you can specify a global name (/Names):

切换行号显示

```
1 ros::NodeHandle nh("/my_global_namespace");
```

This is generally discouraged, as it prevents nodes from being pushed into namespaces (e.g. by roslaunch). There are times, however, when using global names in code can be useful.

Private Names

Using private names is a little bit tricker than calling a `NodeHandle` method with a private name ("~name") directly. Instead, you must create a new `NodeHandle` located inside a private namespace:

切换行号显示

```
1 ros::NodeHandle nh("~my_private_namespace");
2 ros::Subscriber sub = nh.subscribe("my_private_topic", ...);
```

The above example will subscribe to `<node_name>/my_private_namespace/my_private_topic`

Wiki: roscpp/Overview/NodeHandles (2009-11-17 03:12:32由JoshFaust (/JoshFaust)编辑)