编写简单的服务器和客户端（C++）

AI 航 团队

1 编写 Service 节点

这里，我们将创建一个简单的 service 节点("add_two_ints_server")，该节点将接收到两个整形数字，并返回它们的和。

进入先前你在 catkin workspace 教程中所创建的 beginner_tutorials 包所在的目录：

```
$cd ~/catkin_ws/src/beginner_tutorials
```

请确保已经按照 creating the AddTwoInts.srv 教程的步骤创建了本教程所需要的 srv（确保选择了对应的编译系统“catkin”和“rosbuild”）。

1.1 代码

在 beginner_tutorials 包中创建 src/add_two_ints_server.cpp 文件，并复制粘贴下面的代码：

~~~~~~

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
bool add(beginner_tutorials::AddTwoInts::Request  &req,
         beginner_tutorials::AddTwoInts::Response &res)
{
  res.sum = req.a + req.b;
  ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
  ROS_INFO("sending back response: [%ld]", (long int)res.sum);
  return true;
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "add_two_ints_server");
```
~~~~~~

```
  ros::NodeHandle n;


  ros::ServiceServer service = n.advertiseService("add_two_ints", add);
  ROS_INFO("Ready to add two ints.");
  ros::spin();


  return 0;
}
```

~~~~~~

1.2 代码解释

现在，让我们来逐步分析代码。

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
```

beginner_tutorials/AddTwoInts.h 是由编译系统自动根据我们先前创建的 srv 文件生成的对应该 srv 文件的头文件。

```
bool add(beginner_tutorials::AddTwoInts::Request  &req,
         beginner_tutorials::AddTwoInts::Response &res)
```

这个函数提供两个 int 值求和的服务，int 值从 request 里面获取，而返回数据装入 response 内，这些数据类型都定义在 srv 文件内部，函数返回一个 boolean 值。

```
{
  res.sum = req.a + req.b;
  ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
  ROS_INFO("sending back response: [%ld]", (long int)res.sum);
  return true;
}
```

现在，两个 int 值已经相加，并存入了 response。然后一些关于 request 和 response 的信息被记录下来。最后，service 完成计算后返回 true 值。

```
ros::ServiceServer service = n.advertiseService("add_two_ints", add);
```

这里，service 已经建立起来，并在 ROS 内发布出来。
~~~~~~

2、编写 Client 节点

2.1 代码

在 beginner_tutorials 包中创建 src/add_two_ints_client.cpp 文件，并复制粘贴下面的代码：

~~~~~~

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
#include <cstdlib>

int main(int argc, char **argv)
{
 ros::init(argc, argv, "add_two_ints_client");
 if (argc != 3)
 {
  ROS_INFO("usage: add_two_ints_client X Y");
  return 1;
 }
 ros::NodeHandle n;
 ros::ServiceClient client =
n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");
 beginner_tutorials::AddTwoInts srv;
 srv.request.a = atoll(argv[1]);
 srv.request.b = atoll(argv[2]);
 if (client.call(srv))
 {
  ROS_INFO("Sum: %ld", (long int)srv.response.sum);
 }
```
~~~~~~

```
  else
  {
    ROS_ERROR("Failed to call service add_two_ints");
    return 1;
  }

  return 0;
}
```

2.2 代码解释

```
ros::ServiceClient client = n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");
```

这段代码为 add_two_ints service 创建一个 client。ros::ServiceClient 对象待会用来调用 service。

```
beginner_tutorials::AddTwoInts srv;
  srv.request.a = atoll(argv[1]);
  srv.request.b = atoll(argv[2]);
```

这里，我们实例化一个由 ROS 编译系统自动生成的 service 类，并给其 request 成员赋值。一个 service 类包含两个成员 request 和 response。同时也包括两个类定义 Request 和 Response。

```
  if (client.call(srv))
```

这段代码是在调用 service。由于 service 的调用是模态过程（调用的时候占用进程阻止其他代码的执行），一旦调用完成，将返回调用结果。如果 service 调用成功，call()函数将返回 true，srv.response 里面的值将是合法的值。如果调用失败，call()函数将返回 false，srv.response 里面的值将是非法的。

3、编译

再来编辑一下 beginner_tutorials 里面的 CMakeLists.txt，文件位于 ~/catkin_ws/src/beginner_tutorials/CMakeLists.txt，并将下面的代码添加在文件末尾：

再来编辑一下 beginner_tutorials 里面的 CMakeLists.txt，文件位于 ~/catkin_ws/src/beginner_tutorials/CMakeLists.txt，并将下面的代码添加在文件末尾：

```
add_executable(add_two_ints_server src/add_two_ints_server.cpp)
target_link_libraries(add_two_ints_server ${catkin_LIBRARIES})
add_dependencies(add_two_ints_server beginner_tutorials_gencpp)

add_executable(add_two_ints_client src/add_two_ints_client.cpp)
target_link_libraries(add_two_ints_client ${catkin_LIBRARIES})
add_dependencies(add_two_ints_client beginner_tutorials_gencpp)
```

这段代码将生成两个可执行程序"add_two_ints_server"和"add_two_ints_client"，这两个可执行程序默认被放在你的 devel space 下的包目录下，默认为 ~/catkin_ws/devel/lib/share/<package name>。你可以直接调用可执行程序，或者使用 rosrun 命令去调用它们。它们不会被装在<prefix>/bin 目录下，因为当你在你的系统里安装这个包的时候，这样做会污染 PATH 变量。如果你希望在安装的时候你的可执行程序在 PATH 变量里面，你需要设置一下 install target，请参考：catkin/CMakeLists.txt

关于 CMakeLists.txt 文件更详细的描述请参考：catkin/CMakeLists.txt

现在运行 catkin_make 命令：

```
yt@yt-UNO-2483G-453AE:~/catkin_ws$ catkin_make
Base path: /home/yt/catkin_ws
Source space: /home/yt/catkin_ws/src
Build space: /home/yt/catkin_ws/build
Devel space: /home/yt/catkin_ws/devel
Install space: /home/yt/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/yt/catkin_ws/bui
ld"
####
-- Using CATKIN_DEVEL_PREFIX: /home/yt/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/yt/catkin_ws/devel;/opt/ros/kinetic
-- This workspace overlays: /home/yt/catkin_ws/devel;/opt/ros/kinetic
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/yt/catkin_ws/build/test_results
-- Found gmock sources under '/usr/src/gmock': gmock will be built
```