服务通信机制 概述

服务通信 是一种 请求响应模式

节点 a 向 b 发送请求 b 处理请求 并将结果给 A

机器人 巡逻，控制系统分析传感器的数据发现可以的人 此时控制器向传感器发出拍照请求，之后传感器拍照 并将结果给控制器

服务器 实现两个 数字的求和

客户端 向服务器 发送两个数字

服务器 接受两个数字 计算结果 返回给服务器

服务通信的理论模型

master 是管理者
servier 是服务端 talker
client 是客户端   server

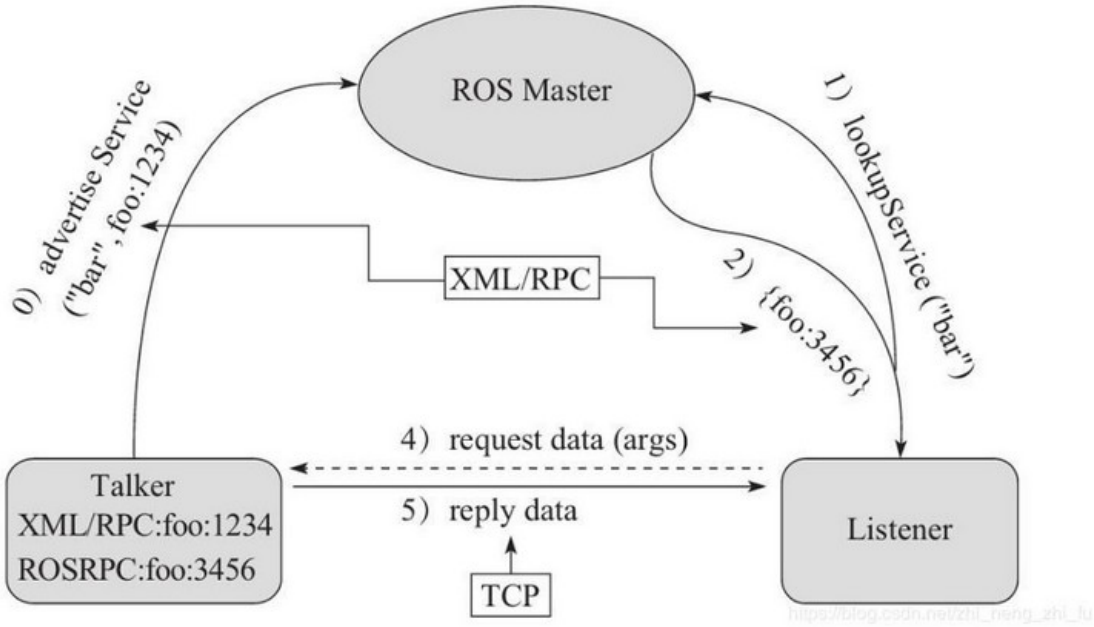
master 通过话题 实现 client 和 server 的连接

第 0 步 server 也就是 talker 建立话题、和 RPBC 址 foo:1234 这里还有一个 ROS RPC 地址 这个是 ros 的一个对于 RPC 地址的封装
第 1 步 client 也就是 client 在 masrer 建立联系 并说明 话题 给 master
第 2 步 master 根据话题 进行匹配， 将 talker 就算 server 的的 ROS RPC 地址给 listener
第 3 步 对应 4） listener 根据 ROS RPC 地址与 talker 建立连接 并将数据发送给 talker
第 4 步 对应 5） talker 通过 TCP 地址 向 listener 发送处理后的 data

服务通信较之于话题通信更简单些，理论模型如下图所示，该模型中涉及到三个角色:

- ROS master(管理者)
- Server(服务端)
- Client(客户端)

ROS Master 负责保管 Server 和 Client 注册的信息，并匹配话题相同的 Server 与 Client ，帮助 Server 与 Client 建立连接，连接建立后，Client 发送请求信息，Server 返回响应信息。



注意
1：client 发送请求时 要服务端 已经创建了节点
2：客户端可以存在多个 就算多个 listener
3、master 会自动帮助 talker 和 listener 建立个连接
4、服务端接受到数据如何处理是关键
5、建立的关系的话题要保证 唯一性 在 ros 空间中
6、客户端如何发送数据 、

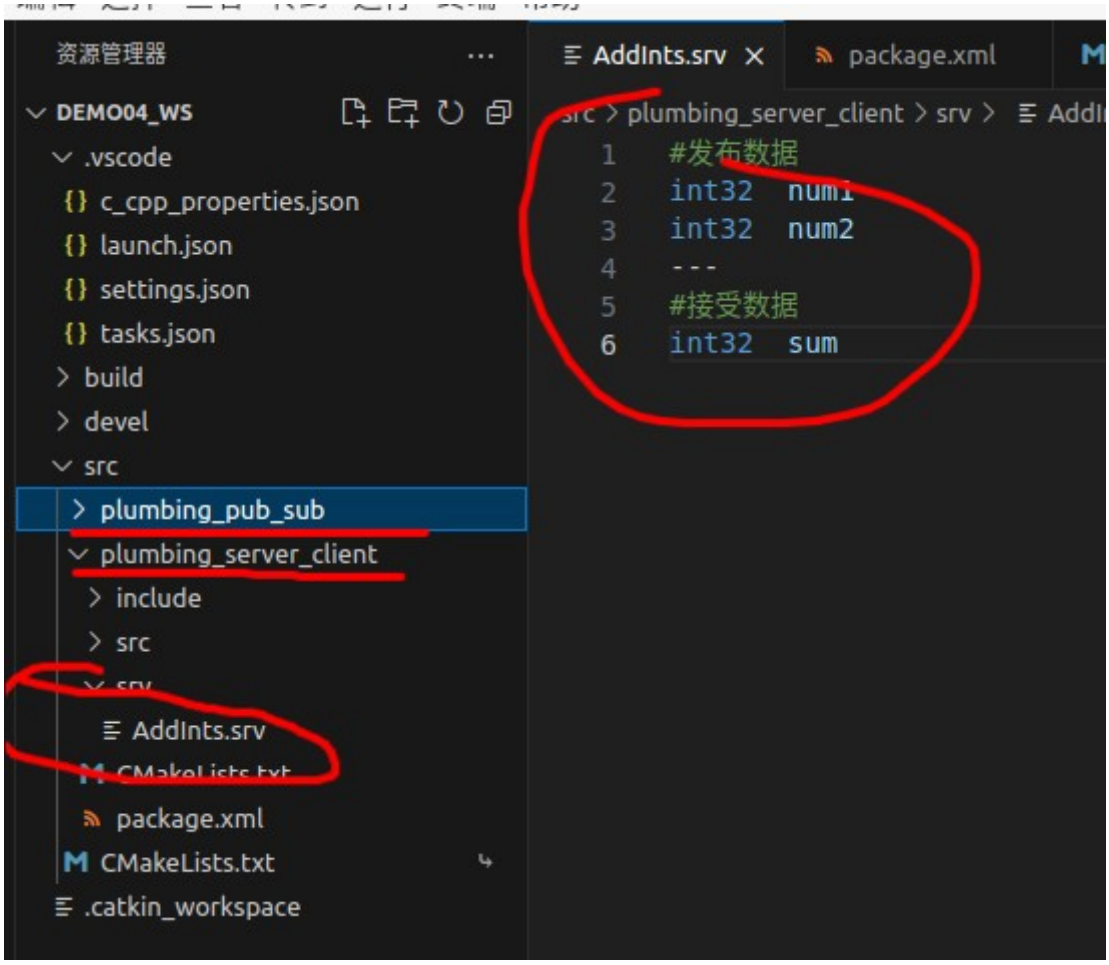
7、数据载体 可能需要自定义

自定义 服务通信的 信息

1、按照固定格式创建文件
2、编辑配置文件
3、编译过程的文件
srv 的文件是分为两个部分
1、客户端发送的数据
2、服务端发送的返回数据

下属图片 看一下几点
1、基于发布订阅的功能包 的空间 创建了一个新的功能包
2、这个包的名字是 plumbing_server_client
3、之后 srv 的文件在 srv 文件夹下 创建一个 srv 文件

num1 和 num2 是发布的数据
sum 是响应的文件



之后我门需要在

package.xml 和 cmakelist 中 更改编译文件

在 package.xml 中 是个发布订阅摸索的 package 是一样的 我们自定义了 message 告诉他我们编译的依赖和 输出 也就是运行时的依赖

```
<!--    <doc_depend>doxygen</doc_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>message_generation</build_depend>


<build_export_depend>roscpp</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>

<exec_depend>roscpp</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
<exec_depend>message_runtime</exec_depend>

<!-- The export tag contains other, unspecified, tags -->
<export>
  <!-- Other tools can request additional information be placed

</export>
</package>
```

在 cmakelist 的文件

1、 构建包时候的 find_package 加入 message_generation

```
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)
```

2 是找到 add_service_files

原本的文件

```
## Generate services in th
add_service_files(
  FILES
  Service1.srv
  Service2.srv
)
```

```
57  ## Generate services in the 'srv' fo
58  add_service_files(
59    FILES
60    AddInts.srv
61  )
62
63  ## Generate actions in the 'action'
64  # add_action_files(
```

自带的 Service1 和 2 .srv 文件都删除了 这是给的例子

这些 service 都基于 std_msgs 生成的 放开 generate_message

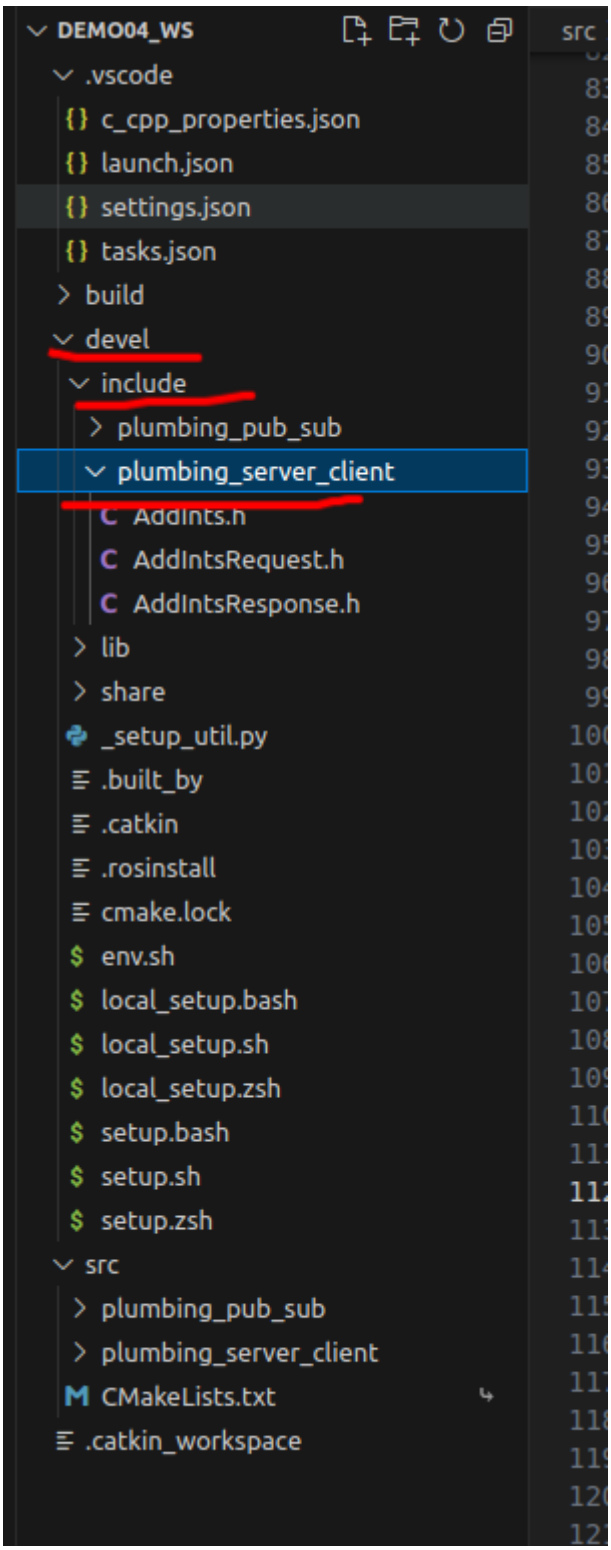
```
69
70  ## Generate added messages and services with a
71  generate_messages(
72    DEPENDENCIES
73    std_msgs
74  )
75
76  ################################################
77  ## Declare ROS dynamic reconfigure parameters
78  ################################################
```

catkin_package 是你编译的包 依赖的包

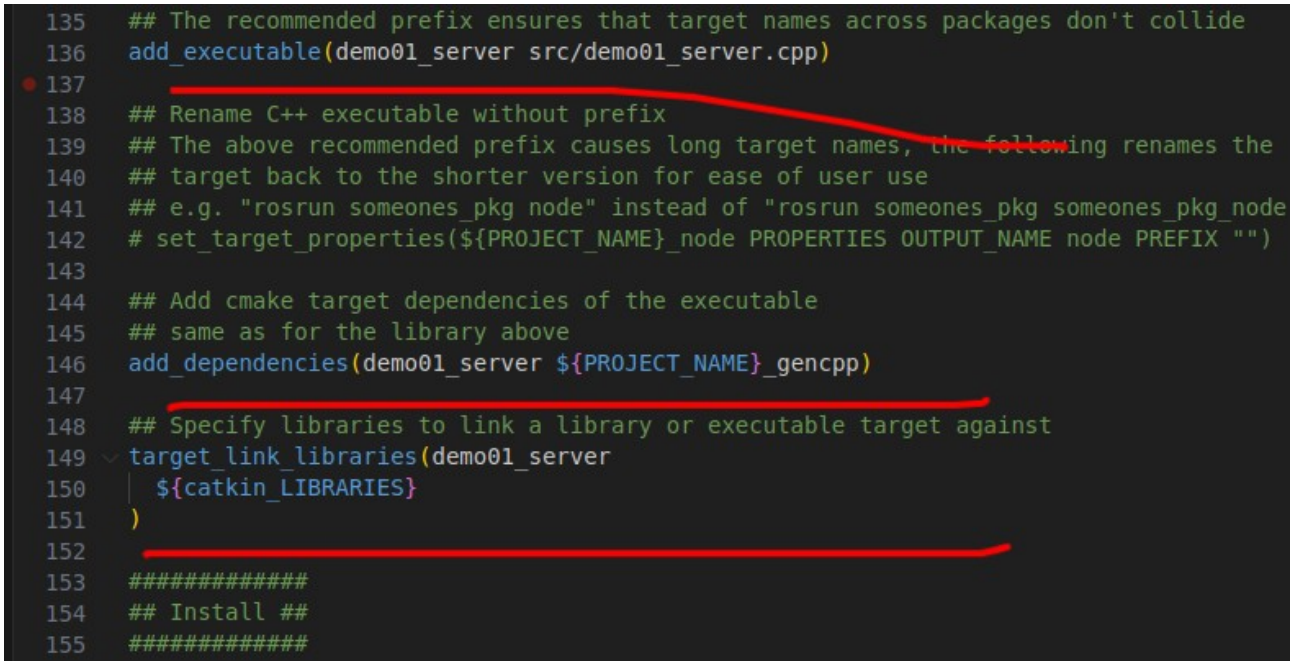
编译结果

```
Scanning dependencies of target plumbing_server_client_generate_messages_py
Scanning dependencies of target plumbing_server_client_generate_messages_eus
Scanning dependencies of target plumbing_server_client_generate_messages_nodejs
Scanning dependencies of target plumbing_server_client_generate_messages_cpp
[ 22%] Generating EusLisp code from plumbing_server_client/AddInts.srv
[ 22%] Generating C++ code from plumbing_server_client/AddInts.srv
[ 27%] Generating Lisp code from plumbing_server_client/AddInts.srv
[ 36%] Generating EusLisp manifest code for plumbing_server_client
[ 36%] Generating Javascript code from plumbing_server_client/AddInts.srv
[ 40%] Generating Python code from SRV plumbing_server_client/AddInts
[ 40%] Built target plumbing_server_client_generate_messages_lisp
[ 40%] Built target plumbing_server_client_generate_messages_nodejs
[ 50%] Built target plumbing_pub_sub_generate_messages_cpp
[ 50%] Built target plumbing_pub_sub_generate_messages_lisp
[ 59%] Built target plumbing_pub_sub_generate_messages_py
[ 63%] Built target plumbing_pub_sub_generate_messages_nodejs
[ 72%] Built target plumbing_pub_sub_generate_messages_eus
[ 72%] Built target plumbing_pub_sub_generate_messages
[ 81%] Built target demo03_pub
[ 90%] Built target demo04_sub
[ 90%] Built target plumbing_server_client_generate_messages_cpp
[ 95%] Generating Python srv __init__.py for plumbing_server_client
[ 95%] Built target plumbing_server_client_generate_messages_eus
[ 95%] Built target plumbing_server_client_generate_messages_py
Scanning dependencies of target plumbing_server_client_generate_messages
[ 95%] Built target plumbing_server_client_generate_messages
[100%] Linking CXX executable /home/qinghuan/env_cv/demo04_ws/devel/lib/plumbing_pub_sub/demo02_sub
[100%] Built target demo02_sub
 *  终端将被任务重用，按任意键关闭。
```

我们看开发空间下 我们的功能包 我们能看到 addints 的文件
devel 这个是 development 文件下的 我们的命名空间下的 h 文件



在 c++的配置
add_executable
add_dependency
target_link_libraries



```
135  ## The recommended prefix ensures that target names across packages don't collide
136  add_executable(demo01_server src/demo01_server.cpp)
137
138  ## Rename C++ executable without prefix
139  ## The above recommended prefix causes long target names, the following renames the
140  ## target back to the shorter version for ease of user use
141  ## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_node
142  # set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")
143
144  ## Add cmake target dependencies of the executable
145  ## same as for the library above
146  add_dependencies(demo01_server ${PROJECT_NAME}_gencpp)
147
148  ## Specify libraries to link a library or executable target against
149  target_link_libraries(demo01_server
150    ${catkin_LIBRARIES}
151  )
152
153  #############
154  ## Install ##
155  #############
```