

---

# 「ROS」の基礎と ROS 2プログラミングの実践

## 2. ROSの基本的な仕組み

高瀬 英希

(京都大学／JSTさきがけ)



# プログラム

---

## 2. ROSの基本的な仕組み [day1 14:15-17:00]

- ROSの通信モデル
- サンプルプログラムによるROS通信の理解 [実習]
- ROS 1プログラムのC++による開発 [実習]
- Pythonによるコード例
- ROSの便利な様々な機能

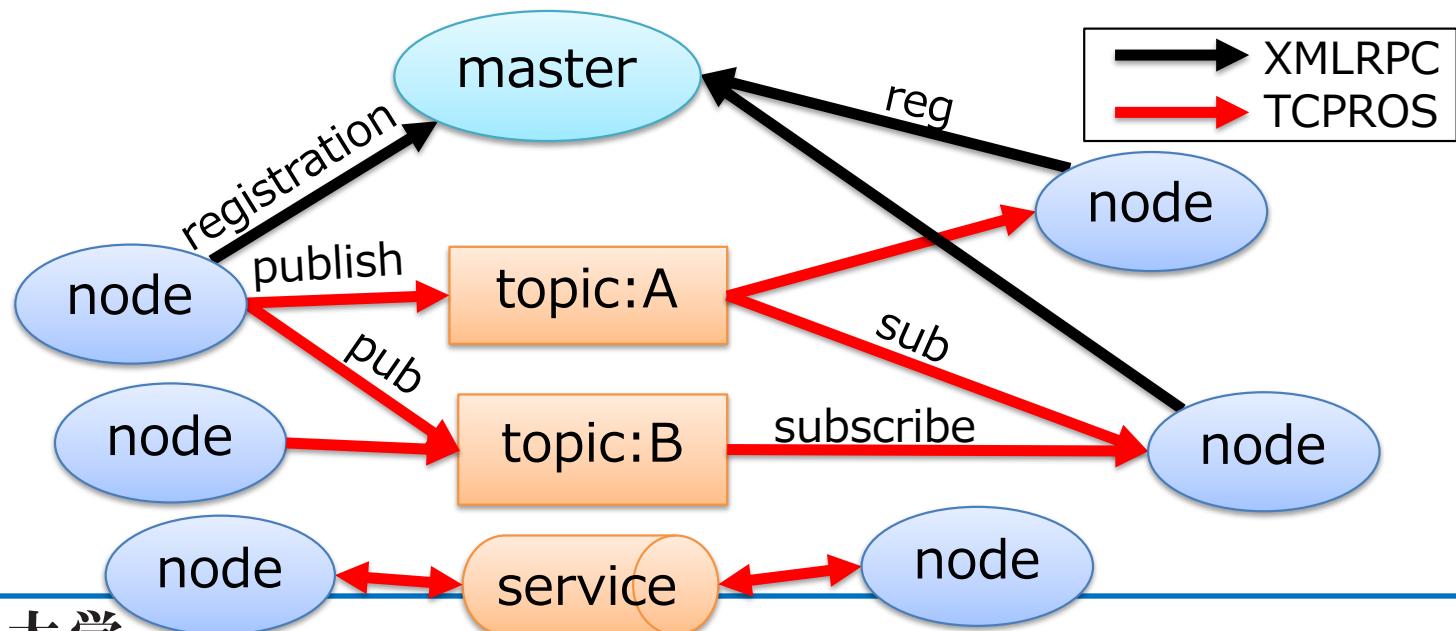
# ROSの通信モデル

---

- トピック topic
  - ROS通信の基本
  - 多対多・非同期型の出版購読通信 (Pub/Sub)
- サービス service
  - 1対1・同期型のRPC通信
- アクション action   ※ROS 1実習では割愛
  - フィードバック付きRPC通信
  - 非同期と同期の組み合わせ
- パラメータ parameter
  - 多変量辞書 (key value store)

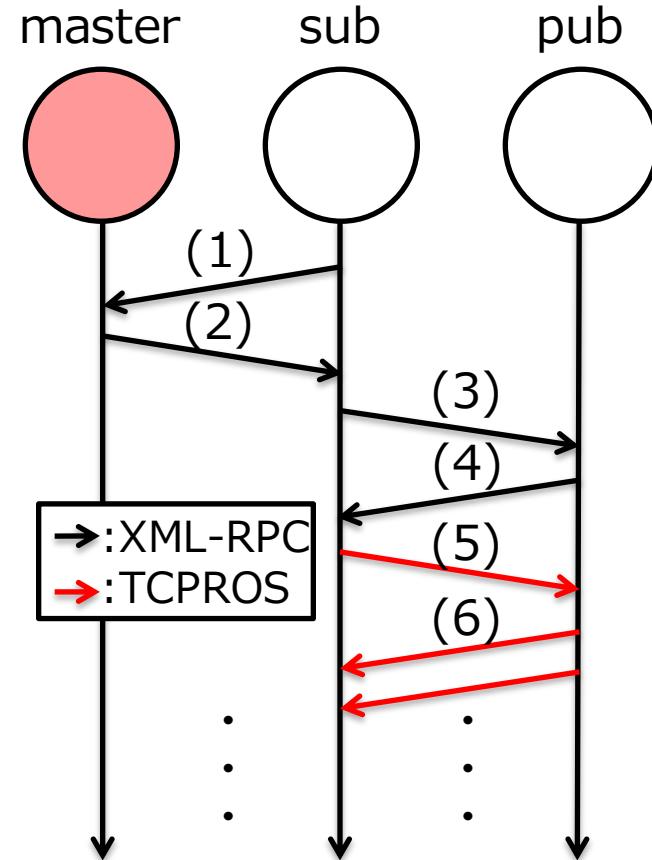
# ROSの通信モデル

- roscore : ROS基本機能の集合
  - ROS Master : ノードの登録と名前空間の管理
  - Parameter Server : 多変量辞書の格納先サーバ
  - rosout : ログ出力用のサーバ



# pub/subのプロトコル

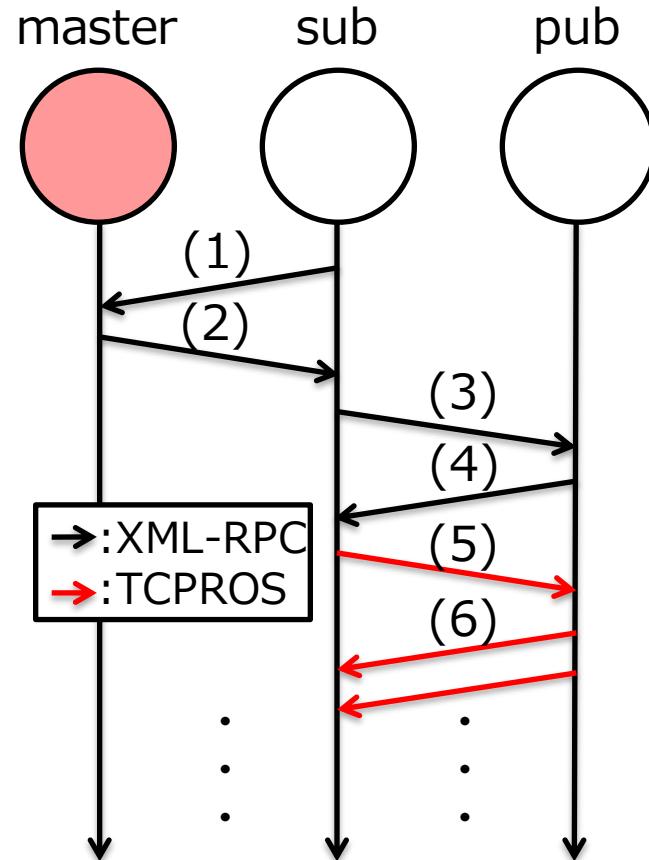
- XML-RPC：
  - masterへのノードの登録
  - トピックを介したノード間の接続の確立
- TCPROS
  - TCP/IP上での非同期通信
  - UDPROSもある



# pub/subの仕組み

- 出版者側 (publisher) :
  - ROS環境とノードの生成
  - roscoreとtopicへの登録
  - メッセージの出版
- 購読者側 (subscriber) :
  - ROS環境とノードの生成
  - roscoreとtopicへの登録
  - (メッセージの購読時)  
コールバック関数の実行
- API仕様の詳細 :

<http://docs.ros.org/api/roscpp/html/>



# 実習の概要と進め方

---

- 概要：ROS1 Melodicでの開発方法の理解
  - まずはpub/subを動かしてみる
  - ワークスペースの設定・topicによる通信
  - 独自定義型のメッセージによるtopic通信
  - serviceによる通信
  - parameterによる通信
- 進め方
  - 進捗が早い方は **演習** に取り組んでみてください
  - ページ下部の **X-Y** は Git Branch番号 に対応します
    - ✓ 適宜でcheckoutして参照してください

# まずは動かしてみる

- ターミナルを3つ開いて、それぞれ下記を実行

```
# roscore(master) の起動  
$ ros1setup  
$ roscore
```

```
# 出版者(C++版)の実行  
$ ros1setup  
$ rosrun roscpp_tutorials talker
```

```
# 購読者(Python版)の実行  
$ ros1setup  
$ rosrun rospy_tutorials listener
```

- 終了はCtrl+Cです
- TIPS: 環境設定 (`~/.bashrc`)

```
120 ## ROS environment  
121 function ros1setup() {  
122     export CHOOSE_ROS_DISTRO=melodic  
123     source /opt/ros/${CHOOSE_ROS_DISTRO}/setup.bash  
124     #export ROS_MASTER_URI=http://192.168.11.2:11311  
125     #export ROS_HOSTNAME=192.168.11.2  
126 }  
127 alias ros1setup=ros1setup  
128
```

接続先(ROSマスター)を  
指定する場合の環境変数

# まずは動かしてみる

```
roscore http://takase-VirtualBox:11311/
roscore http://takase-VirtualBox:11311/_167x22

started roslaunch server http://takase-VirtualBox:33089/
ros_comm version 1.14.3

SUMMARY
=====
PARAMETERS
* /rosdistro: melodic
* /rosversion: 1.14.3

NODES

auto-starting new master
process[master]: started with pid [3629]
ROS_MASTER_URI=http://takase-VirtualBox:11311/

setting /run_id to fe8f2270-a517-11e9-be2b-08002728af08
process[rosout-1]: started with pid [3640]
started core service [/rosout]

takase@takase-VirtualBox:~ takase@takase-VirtualBox: ~ 82x22
$ ros1setup
takase@takase-VirtualBox:~
$ rosrun roscpp_tutorials talker
[INFO] [1562985838.476045380]: hello world 0
[INFO] [1562985838.585578586]: hello world 1
[INFO] [1562985838.683594427]: hello world 2
[INFO] [1562985838.776550981]: hello world 3
[INFO] [1562985838.878487139]: hello world 4
[INFO] [1562985838.976405356]: hello world 5
[INFO] [1562985839.076733400]: hello world 6
[INFO] [1562985839.177844135]: hello world 7
[INFO] [1562985839.281129353]: hello world 8
[INFO] [1562985839.382833520]: hello world 9
[INFO] [1562985839.483318553]: hello world 10
[INFO] [1562985839.577040577]: hello world 11
[INFO] [1562985839.677073479]: hello world 12
[INFO] [1562985839.776766359]: hello world 13
[INFO] [1562985839.877166366]: hello world 14
[INFO] [1562985839.977312921]: hello world 15
[INFO] [1562985840.077610640]: hello world 16
[INFO] [1562985840.176613575]: hello world 17

takase@takase-VirtualBox:~ takase@takase-VirtualBox: ~ 83x22
$ rosrun rospy_tutorials listener
[INFO] [1562985838.586634]: /listener_3701_1562985827550I heard hello world 1
[INFO] [1562985838.684177]: /listener_3701_1562985827550I heard hello world 2
[INFO] [1562985838.777034]: /listener_3701_1562985827550I heard hello world 3
[INFO] [1562985838.879685]: /listener_3701_1562985827550I heard hello world 4
[INFO] [1562985838.976870]: /listener_3701_1562985827550I heard hello world 5
[INFO] [1562985839.077414]: /listener_3701_1562985827550I heard hello world 6
[INFO] [1562985839.178300]: /listener_3701_1562985827550I heard hello world 7
[INFO] [1562985839.281598]: /listener_3701_1562985827550I heard hello world 8
[INFO] [1562985839.383279]: /listener_3701_1562985827550I heard hello world 9
[INFO] [1562985839.483882]: /listener_3701_1562985827550I heard hello world 10
[INFO] [1562985839.577526]: /listener_3701_1562985827550I heard hello world 11
[INFO] [1562985839.677573]: /listener_3701_1562985827550I heard hello world 12
[INFO] [1562985839.777257]: /listener_3701_1562985827550I heard hello world 13
[INFO] [1562985839.877666]: /listener_3701_1562985827550I heard hello world 14
[INFO] [1562985839.977866]: /listener_3701_1562985827550I heard hello world 15
[INFO] [1562985840.078102]: /listener_3701_1562985827550I heard hello world 16
[INFO] [1562985840.177798]: /listener_3701_1562985827550I heard hello world 17
[INFO] [1562985840.278416]: /listener_3701_1562985827550I heard hello world 18
```

# 実習の概要と進め方

---

- 概要：ROS1 Melodicでの開発方法の理解
  - まずはpub/subを動かしてみる
  - ワークスペースの設定・topicによる通信
  - 独自定義型のメッセージによるtopic通信
  - serviceによる通信
  - parameterによる通信
- 進め方
  - 進捗が早い方は **演習** に取り組んでみてください
  - ページ下部の **X-Y** は Git Branch番号 に対応します
    - ✓ 適宜でcheckoutして参照してください

# ワークスペースの設定

- catkin用ワークスペースの作成と設定

```
# ROS 1 Melodicの環境設定  
$ ros1setup  
# ワークスペース用ディレクトリの作成  
$ mkdir -p ~/catkin_ws/src  
# ワークスペース用ディレクトリへの移動  
$ cd ~/catkin_ws/  
# ワークスペースの作成  
$ catkin_make  
# ワークスペースの環境設定  
$ source devel/setup.bash
```

新しいターミナルを  
開くたびに必要

新しいターミナルを開くたび／  
新しいパッケージをビルドした  
のちに都度で必要

# パッケージの作成

- topic通信パッケージ pubsub\_topic の作成

```
# パッケージの作成  
$ cd ~/catkin_ws/src  
$ catkin_create_pkg pubsub_topic std_msgs roscpp
```

- 第1引数：作成するパッケージ名
- 第2引数以降：依存するパッケージの指定（複数可）
  - ✓ -V: パッケージのバージョン番号
  - ✓ -D: パッケージの説明の記述
  - ✓ -l: パッケージのライセンス（BSD, MIT, GPLv3など）
  - ✓ -a: パッケージの開発者の記述
  - ✓ -m: パッケージの管理者の記述
  - ✓ --rosdistro: ROS distroの指定（デフォルトは環境変数\$ROS\_DISTRO）
  - ✓ など



# パッケージの設定

- package.xml : パッケージ情報の定義ファイル
  - パッケージの各種情報

```
1 <?xml version="1.0"?>
2 <package format="2">
3   <name>pubsub_topic</name>
4   <version>0.0.0</version>
5   <description>The pubsub_topic package</description>
6
7   <!-- One maintainer tag required, multiple allowed, one per
8       person per tag -->
9   <!-- Example:  -->
10  <!-- <maintainer email="jane.doe@example.com">Jane Doe</mai-
11      ntainer> -->
12  <maintainer email="takase@todo.todo">takase</maintainer>
```

演習

きちんと情報を  
記述してみましょう

- ビルドツールの指定と依存パッケージの情報

```
51   <buildtool_depend>catkin</buildtool_depend>
52   <build_depend>roscpp</build_depend>
53   <build_depend>std_msgs</build_depend>
54   <build_export_depend>roscpp</build_export_depend>
55   <build_export_depend>std_msgs</build_export_depend>
56   <exec_depend>roscpp</exec_depend>
57   <exec_depend>std_msgs</exec_depend>
58
59
60   <!-- The export tag contains other, unspecified, tags -->
61   <export>
62     <!-- Other tools can request additional information be pl-
63         aced here -->
64   </export>
65 </package>
```

# パッケージの設定

- CMakeLists.txt : cmake用の設定ファイル
  - cmakeバージョンとパッケージ名

```
1 cmake_minimum_required(VERSION 2.8.3)
2 project(pubsub_topic)
3
```

- 依存パッケージの走査

```
10 find_package(catkin REQUIRED COMPONENTS
11   roscpp
12   std_msgs
13 )
```

- include dirsの指定

```
117 include_directories(
118 # include
119 ${catkin_INCLUDE_DIRS}
120 )
121
```



# パッケージの実装

- topic通信パッケージの実装
  - ~/catkin\_ws/src/pubsub\_topic/src に下記 2 個をダウンロード  
[src/talker.cpp](#)   [src/listener.cpp](#)
    - ✓ 保存時は拡張子に注意！.cpp として保存する
  - pubsub\_topic/CMakeLists.txt を修正

```
133 ## With catkin_make all packages are built within a single CMake context
134 ## The recommended prefix ensures that target names across packages don't collide
135 # add_executable(${PROJECT_NAME}_node src/pubsub_topic_node.cpp)
136 add_executable(talker src/talker.cpp)
137 add_executable(listener src/listener.cpp)
138
139 ## Rename C++ executable without prefix
140 ## The above recommended prefix causes long target names, the following renames the
141 ## target back to the shorter version for ease of user use
```

実行ファイル名と  
対応するソースの指定

```
150 # target_link_libraries(${PROJECT_NAME}_node
151 #   ${catkin_LIBRARIES}
152 # )
153 target_link_libraries(talker ${catkin_LIBRARIES})
154 target_link_libraries(listener ${catkin_LIBRARIES})
155
156 #####
```

ライブラリの指定

# パッケージのビルドと実行

- ・パッケージのビルド

```
$ cd ~/catkin_ws  
$ catkin_make
```

- ・実行

```
# roscore(master) の起動  
$ ros1setup  
$ roscore
```

演習

chatter\_pub に対して  
複数の talker/listener を  
同時に実行してみましょう

```
# 出版者の実行  
$ ros1setup  
$ source ~/catkin_ws/devel/setup.bash  
$ rosrun pubsub_topic talker
```

```
# 購読者の実行  
$ ros1setup  
$ source ~/catkin_ws/devel/setup.bash  
$ rosrun pubsub_topic listener
```

# Code Viewing

```
.  
├── build  
├── devel  
│   ├── _setup_util.py  
│   └── env.sh  
└── lib  
    ├── setup.bash  
    ├── setup.sh  
    └── setup.zsh  
    └── share  
        └── src  
            ├── CMakeLists.txt -> /opt/ros/melodic/share/catkin/cmake/toplevel.cmake  
            └── pubsub_topic  
                ├── CMakeLists.txt  
                ├── include  
                │   └── pubsub_topic  
                ├── package.xml  
                └── src  
                    ├── listener.cpp  
                    └── talker.cpp
```

# talker.cppの解説

- ライブドリと関数宣言

```
29 #include "ros/ros.h"
30 // %EndTag(ROS_HEADER)%
31 // %Tag(MSG_HEADER)%
32 #include "std_msgs/String.h"
33 // %EndTag(MSG_HEADER)%
34
35 #include <iostream>
36
37 /**
38  * This tutorial demonstrates simple sending of messages over the ROS system.
39 */
40 int main(int argc, char **argv)
41 {
```



# talker.cppの解説

- ROS環境の初期化とノードの名前付け

```
52 // %Tag(INIT)%  
53   ros::init(argc, argv, "talker");  
54 // %EndTag(INIT)%  
55
```

- ROSノードの生成

```
60 //  
61 // %Tag(NODEHANDLE)%  
62   ros::NodeHandle n;  
63 // %EndTag(NODEHANDLE)%  
64
```



# talker.cppの解説

- 出版者ノードとしてroscoreとトピックに登録
  - 第1引数：配信先トピック名
  - 第2引数：メッセージキューのサイズ
  - [TIPS] 第3引数：latch（省略可， デフォルトfalse）
    - ✓ trueにすると， 接続が確立されるまでに配信されたメッセージの最後のものを保存する（購読ノードの接続確立後に配信される）

```
81 */  
82 // %Tag(PUBLISHER)%  
83 ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);  
84 // %EndTag(PUBLISHER)%  
85
```

- ループ周期の設定（単位はHz）

```
86 // %Tag(LOOP_RATE)%  
87 ros::Rate loop_rate(10);  
88 // %EndTag(LOOP_RATE)%  
89
```

# talker.cppの解説

- ループとカウント値の生成（ノード実行が正常か）

```
93  */
94 // %Tag(ROS_OK)%
95 int count = 0;
96 while (ros::ok())
97 {
98 // %EndTag(ROS_OK)%
99 /**
```

- 配信用のメッセージの型宣言と作成

```
102 /*
103 // %Tag(FILL_MESSAGE)%
104 std_msgs::String msg;
105 std::stringstream ss;
106 ss << "hello world " << count;
107 msg.data = ss.str();
108 // %EndTag(FILL_MESSAGE)%
109 */
```

- メッセージのログ表示

```
110 // %Tag(ROSCONSOLE)%
111 ROS_INFO("%s", msg.data.c_str());
112 // %EndTag(ROSCONSOLE)%
113 */
```



# talker.cppの解説

- メッセージの出版

```
120 // %Tag(PUBLISH)%  
121     chatter_pub.publish(msg);  
122 // %EndTag(PUBLISH)%
```

- コールバック関数の実行などのイベント待ち（1回のみ）

```
124 // %Tag(SPINONCE)%  
125     ros::spinOnce();  
126 // %EndTag(SPINONCE)%
```

- 設定された時間の経過待ち

```
128 // %Tag(RATE_SLEEP)%  
129     loop_rate.sleep();  
130 // %EndTag(RATE_SLEEP)%  
131     ++count;  
132 }133  
134  
135     return 0;  
136 }  
137 // %EndTag(FULLTEXT)%
```



# listener.cppの解説

- コールバック関数の定義

```
35 // %tag(CALLBACK)%  
36 void chatterCallback(const std_msgs::String::ConstPtr& msg)  
37 {  
38     ROS_INFO("I heard: [%s]", msg->data.c_str());  
39 }  
40 // %EndTag(CALLBACK)%
```

- ROS環境の初期化とノードの名前付け

```
53 /  
54     ros::init(argc, argv, "listener");  
55 //
```

- ROSノードの生成

```
60 */  
61 ros::NodeHandle n;  
62
```



# listener.cppの解説

- 購読者ノードとしてroscoreとトピックに登録
  - 第1引数：配信先トピック名
  - 第2引数：メッセージキューのサイズ
  - 第3引数：購読時に実行されるコールバック関数

```
78 // %Tag(SUBSCRIBER)%  
79   ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);  
80 // %EndTag(SUBSCRIBER)%
```

- コールバック関数の実行などのイベントを無限待ち (Ctrl+Cまで)

```
86  */  
87 // %Tag(SPIN)%  
88   ros::spin();  
89 // %EndTag(SPIN)%  
90  
91   return 0;  
92 }  
93 // %EndTag(FULLTEXT)%
```

# 実習の概要と進め方

---

- 概要：ROS1 Melodicでの開発方法の理解
  - まずはpub/subを動かしてみる
  - ワークスペースの設定・topicによる通信
  - 独自定義型のメッセージによるtopic通信
  - serviceによる通信
  - parameterによる通信
- 進め方
  - 進捗が早い方は **演習** に取り組んでみてください
  - ページ下部の **X-Y** は Git Branch番号 に対応します
    - ✓ 適宜でcheckoutして参照してください

# ROS通信の型

---

- ロボット通信に頻出する型が定義されている
  - <http://wiki.ros.org/msg>
  - [http://wiki.ros.org/std\\_msgs](http://wiki.ros.org/std_msgs)
- Primitive Type
  - [Bool](#) / [Byte](#) / [Char](#) / [Duration](#) / [String](#) / [Time](#) / [Float32](#) / [Float64](#) / [Int8](#) / [Int16](#) / [Int32](#) / [Int64](#) / [UInt8](#) / [UInt16](#) / [UInt32](#) / [UInt64](#) / [Empty](#)
- Array Type
  - [Header](#) / [ByteMultiArray](#) / [ColorRGBA](#) / [MultiArrayDimension](#) / [MultiArrayLayout](#) / [Float32MultiArray](#) / [Float64MultiArray](#) / [Int8MultiArray](#) / [Int16MultiArray](#) / [Int32MultiArray](#) / [Int64MultiArray](#) / [UInt8MultiArray](#) / [UInt16MultiArray](#) / [UInt32MultiArray](#) / [UInt64MultiArray](#)

# ROS通信の型

- 独自の型のメッセージを定義することができる
  - 例：車輪の角速度と回転量，現在位置の3次元座標
  - 階層構造や配列を含むこともできる
  - msg/\*.msg ファイルで定義する
    - ✓ ファイル名の最初と単語(意味)区切りは大文字
    - ✓ \_ (アンダースコア) は利用不可
- 例題：人物の情報を出版して購読側でBMIを計算
  - Human型の定義：  
string name, uint16 height, float32 weight
  - BMI計算 = (体重[kg]) / (身長[m])<sup>2</sup>

# パッケージの作成

- 独自メッセージ通信パッケージ pubsub\_custom の作成

```
# ワークスペースに移動して環境設定  
$ cd ~/catkin_ws  
$ source devel/setup.bash  
# パッケージの作成  
$ cd src/  
$ catkin_create_pkg pubsub_custom roscpp std_msgs message_generation
```

- message\_generation: 独自型を生成するためのパッケージ

# 独自メッセージの作成

- 独自メッセージの定義ファイルの作成

```
# パッケージのディレクトリに移動  
$ roscd pubsub_custom  
# 定義ファイルを作成  
$ mkdir msg  
$ touch msg/Human.msg
```

- msg/Human.msg を編集（作成）

```
1 string name  
2 uint16 height  
3 float32 weight
```

# 独自メッセージの作成

- CMakeLists.txt を編集

```
48
49 ## Generate messages in the 'msg' folder
50 add_message_files(
51   FILES
52   Human.msg
53 )
54
55 ## Generate services in the 'srv' folder

68
69 ## Generate added messages and services with any dependencies listed here
70 generate_messages(
71   DEPENDENCIES
72   std_msgs
73 )
74
75 #####
```

- ビルド

```
# ワークスペースのディレクトリに移動
$ cd ~/catkin_ws
# ビルド（定義ファイルを生成）
$ catkin_make
```

# 独自メッセージの作成

- 生成されたヘッダファイルを確認してみる
  - ~/catkin\_ws/devel/include/pubsub\_custom/Human.h

```
18
19 namespace pubsub_custom
20 {
21 template <class ContainerAllocator>
22 struct Human_
23 {
24     typedef Human_<ContainerAllocator> Type;
25
26     Human_()
27         : name()
28         , height(0)
29         , weight(0.0)  {
30     }
31     Human_(const ContainerAllocator& _alloc)
32         : name(_alloc)
33         , height(0)
34         , weight(0.0)  {
35     (void)_alloc;
36     }
37
38 }
```

# パッケージの実装

- 独自メッセージ通信パッケージの実装

```
# ソースを pubsub_topicからコピー  
$ roscd pubsub_custom  
$ cp ..../pubsub_topic/src/* src/
```

- コピーしてきたファイルを次ページ以降に従って編集
- または、正解は下記（ダウンロード時は拡張子に注意）
  - ✓ [CMakeLists.txt](#)
  - ✓ [src/talker.cpp](#)
  - ✓ [src/listener.cpp](#)

# パッケージの実装

- CMakeLists.txt の編集

```
134 ## The recommended prefix ensures that target names across packages don't collide
135 # add_executable(${PROJECT_NAME}_node src/pubsub_custom_node.cpp)
136 add_executable(bmi_talker src/talker.cpp)
137 add_executable(bmi_listener src/listener.cpp)
138 add_dependencies(bmi_talker ${PROJECT_NAME}_generate_messages_cpp)
139 add_dependencies(bmi_listener ${PROJECT_NAME}_generate_messages_cpp)
140
141 ## Rename C++ executable without prefix
142 ## The above recommended prefix causes long target names, the following
```

```
151 ## target_link_libraries(${PROJECT_NAME}_node
152 ##   ${catkin_LIBRARIES}
153 ## )
154 #
155 target_link_libraries(bmi_talker ${catkin_LIBRARIES})
156 target_link_libraries(bmi_listener ${catkin_LIBRARIES})
157
158 #####
```

# talker.cppの実装

- ライブラリの読み込み

```
27 // %Tag(FULLTEXT)%  
28 // %Tag(ROS_HEADER)%  
29 #include "ros/ros.h"  
30 // %EndTag(ROS_HEADER)%  
31 // %Tag(MSG_HEADER)%  
32 #include "pubsub_custom/Human.h"  
33 // %EndTag(MSG_HEADER)%  
34  
35 #include <iostream>  
36 using namespace std;  
37
```

- ROS環境の初期化とノードの名前付け

```
53 // %Tag(INIT)%  
54   ros::init(argc, argv, "bmi_talker");  
55 // %EndTag(INIT)%
```

- 出版者ノードとしてroscoreとトピックに登録

```
83 // %Tag(PUBLISHER)%  
84   ros::Publisher chatter_pub = n.advertise<pubsub_custom::Human>("chatter", 1000);  
85 // %EndTag(PUBLISHER)%
```



# talker.cppの実装

- 配信用のメッセージの型宣言と作成

```
102  /*
103 // %Tag(FILL_MESSAGE)%
104     pubsub_custom::Human msg;
105
106     cout << "Enter Name [str]: " << endl;
107     cin >> msg.name;
108     cout << "Enter Height [int/cm]: " << endl;
109     cin >> msg.height;
110     cout << "Enter Weight [float/kg]: " << endl;
111     cin >> msg.weight;
112 // %EndTag(FILL_MESSAGE)%
113
114 // %Tag(ROSCONSOLE)%
115     ROS_INFO("[%02d] name: %s height: %d weight: %.2f",
116             count, msg.name.c_str(), msg.height, msg.weight);
117 // %EndTag(ROSCONSOLE)%
118
119 */
```



# talker.cppの実装

```
▼ 23  ros1_melodic/catkin_ws/src/pubsub_custom/src/talker.cpp □
    @@ -29,10 +29,11 @@
 29   29      #include "ros/ros.h"
 30   30      // %EndTag(ROS_HEADER)%
 31   31      // %Tag(MSG_HEADER)%
 32 -  32      - #include "std_msgs/String.h"
 32 +  32      + #include "pubsub_custom/Human.h"
 33   33      // %EndTag(MSG_HEADER)%
 34   34
 35 -  35      - #include <sstream>
 35 +  35      + #include <iostream>
 36 +  36      + using namespace std;
 36
 37   38      /**
 38   39          * This tutorial demonstrates simple sending of messages over the ROS system.
    @@ -50,7 +51,7 @@
    ━━━━━━━━━━━━━━━━
 50   51          * part of the ROS system.
 51   52          */
 52   53      // %Tag(INIT)%
 53 -  53      - ros::init(argc, argv, "talker");
 54 +  54      + ros::init(argc, argv, "bmi_talker");
 54   55      // %EndTag(INIT)%
  --
```



# talker.cppの実装

```
@@ -80,7 +81,7 @@ int main(int argc, char **argv)
80    81      * buffer up before throwing some away.
81    82      */
82    83      // %Tag(PUBLISHER)%
83      -  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
84      +  ros::Publisher chatter_pub = n.advertise<pubsub_custom::Human>("chatter", 1000);
84    85      // %EndTag(PUBLISHER)%
85    86
86    87      // %Tag(LOOP_RATE)%
```

---

```
@@ -100,15 +101,19 @@ int main(int argc, char **argv)
100   101      * This is a message object. You stuff it with data, and then publish it.
101   102      */
102   103      // %Tag(FILL_MESSAGE)%
103      -  std_msgs::String msg;
104      +  pubsub_custom::Human msg;
104   105
```



# talker.cppの実装

```
104    105
105    -     std::stringstream ss;
106    -     ss << "hello world " << count;
107    -     msg.data = ss.str();
106    +     cout << "Enter Name [str]: " << endl;
107    +     cin >> msg.name;
108    +     cout << "Enter Height [int/cm]: " << endl;
109    +     cin >> msg.height;
110    +     cout << "Enter Weight [float/kg]: " << endl;
111    +     cin >> msg.weight;
108    112     // %EndTag(FILL_MESSAGE)%
109    113
110    114     // %Tag(ROSCONSOLE)%
111    -     ROS_INFO("%s", msg.data.c_str());
115    +     ROS_INFO("[%02d] name: %s height: %d weight: %.2f",
116    +                 count, msg.name.c_str(), msg.height, msg.weight);
112    117     // %EndTag(ROSCONSOLE)%
113    118
114    119     /**
  ↵
```

# listener.cppの実装

- ライブラリの読み込み

```
28 // %Tag(FULLTEXT)%  
29 #include "ros/ros.h"  
30 #include "pubsub_custom/Human.h"  
31
```

- コールバック関数の定義

```
34 */  
35 // %Tag(CALLBACK)%  
36 void chatterCallback(const pubsub_custom::Human msg)  
37 {  
38     float bmi = msg.weight / (msg.height/100.0) / (msg.height/100.0);  
39     ROS_INFO("%s's BMI is %.2f", msg.name.c_str(), bmi);  
40 }  
41 // %EndTag(CALLBACK)%  
42
```

- ROS環境の初期化とノードの名前付け

```
53     * part of the ROS system.  
54     */  
55     ros::init(argc, argv, "bmi_listener");  
56  
57     /**
```

# listener.cppの実装

```
▼ 9  ros1_melodic/catkin_ws/src/pubsub_custom/src/listener.cpp □
```

	@@ -27,15 +27,16 @@
27	27
28	// %Tag(FULLTEXT)%
29	#include "ros/ros.h"
30	- #include "std_msgs/String.h"
30	+ #include "pubsub_custom/Human.h"
31	31
32	32 /**
33	33 * This tutorial demonstrates simple receipt of messages over the ROS system.
34	34 */
35	35 // %Tag(CALLBACK)%
36	- void chatterCallback(const std_msgs::String::ConstPtr& msg)
36	+ void chatterCallback(const pubsub_custom::Human msg)
37	37 {
38	- ROS_INFO("I heard: [%s]", msg->data.c_str());
38	+ float bmi = msg.weight / (msg.height/100.0) / (msg.height/100.0);
39	+ ROS_INFO("%s's BMI is %.2f", msg.name.c_str(), bmi);
39	40 }
40	41 // %EndTag(CALLBACK)%
41	42
53	54 */
54	- ros::init(argc, argv, "listener");
55	+ ros::init(argc, argv, "bmi_listener");
55	56

# パッケージのビルドと実行

- ・パッケージのビルド

```
$ cd ~/catkin_ws  
$ catkin_make
```

- ・実行

```
# roscore(master) の起動  
$ ros1setup  
$ roscore
```

演習

独自型メッセージHumanに、  
ループのcount値(pub回数)を  
idとして追加してみましょう

2-6

```
# 出版者の実行  
$ ros1setup  
$ source ~/catkin_ws/devel/setup.bash  
$ rosrun pubsub_custom bmi_talker
```

```
# 購読者の実行  
$ ros1setup  
$ source ~/catkin_ws/devel/setup.bash  
$ rosrun pubsub_custom bmi_listener
```

2-5

# 実習の概要と進め方

---

- 概要：ROS1 Melodicでの開発方法の理解
  - まずはpub/subを動かしてみる
  - ワークスペースの設定・topicによる通信
  - 独自定義型のメッセージによるtopic通信
  - **service**による通信
  - parameterによる通信
- 進め方
  - 進捗が早い方は **演習** に取り組んでみてください
  - ページ下部の **X-Y** は Git Branch番号 に対応します
    - ✓ 適宜でcheckoutして参照してください

# service通信

---

- 同期型の通信方式
  - サービスを介したサーバ・クライアント型の通信方式
  - `srv/*.srv` ファイルで独自のサービスを定義できる
- 例題：人物の情報を送信してBMI計算値を受ける
  - これまでのpub/subパッケージをserviceで実装する
  - `human`型の定義：  
`string name, uint16 height, float32 weight,`  
**`float32 bmi`**
  - BMI計算 = (体重[kg]) / (身長[m])<sup>2</sup>

# パッケージの作成

- 独自サービス通信パッケージ service\_custom の作成

```
# ワークスペースに移動して環境設定  
$ cd ~/catkin_ws  
$ source devel/setup.bash  
# パッケージの作成  
$ cd src/  
$ catkin_create_pkg service_custom roscpp std_msgs message_generation
```



# 独自サービスの作成

- 独自サービスの定義ファイルの作成

```
# パッケージのディレクトリに移動  
$ roscd service_custom  
# 定義ファイルを作成  
$ mkdir srv  
$ touch srv/Human.srv
```

- srv/Human.srv を編集（作成）

```
1 string name  
2 uint16 height  
3 float32 weight  
4 --  
5 float32 bmi
```

requestの設定  
(serviceへの送信値)

responseの設定  
(serviceからの返送値)

# 独自サービスの作成

- CMakeLists.txt を編集

```
54
55 ## Generate services in the 'srv' folder
56 add_service_files(
57   FILES
58   Human.srv
59 )
60
61 ## Add message and service files here
```

```
67
68 ## Generate added messages and services with any dependencies listed here
69 generate_messages(
70   DEPENDENCIES
71 )
72
```

- ビルド

```
# ワークスペースのディレクトリに移動
$ cd ~/catkin_ws
# ビルド（定義ファイルを生成）
$ catkin_make
```

演習

生成されたヘッダファイルを  
確認してみましょう

# パッケージの実装

- 独自サービス通信パッケージの実装

```
# ソースを pubsub_customからコピー  
$ roscd service_custom  
$ rosdp pubsub_custom talker.cpp src/server.cpp  
$ rosdp pubsub_custom listener.cpp src/client.cpp
```

- 次ページ以降に従って編集

- ✓ 変更点 :

- [https://github.com/takasehideki/ros\\_study/commit/  
c7a6130827c020b012bb2170b7873c683a6c4530](https://github.com/takasehideki/ros_study/commit/c7a6130827c020b012bb2170b7873c683a6c4530)

- または、正解は下記

- ✓ [CMakeLists.txt](#)
    - ✓ [src/server.cpp](#)
    - ✓ [src/client.cpp](#)

# パッケージの実装

- CMakeLists.txt の編集

```
134 ## The recommended prefix ensures that target names across packages don't collide
135 # add_executable(${PROJECT_NAME}_node src/service_custom_node.cpp)
136 add_executable(bmi_server src/server.cpp)
137 add_executable(bmi_client src/client.cpp)
138 add_dependencies(bmi_server ${PROJECT_NAME}_generate_messages_cpp)
139 add_dependencies(bmi_client ${PROJECT_NAME}_generate_messages_cpp)
140
141 ## Rename C++ executable without prefix
142 ## This is required for ROS to find the correct executable
143 # ${catkin_LIBRARIES}
153 #
154 #
155 target_link_libraries(bmi_server ${catkin_LIBRARIES})
156 target_link_libraries(bmi_client ${catkin_LIBRARIES})
157
158 #####
```

# server.cppの実装

- ライブラリの読み込み

```
27 // %Tag(FULLTEXT)%  
28 // %Tag(ROS_HEADER)%  
29 #include "ros/ros.h"  
30 // %EndTag(ROS_HEADER)%  
31 // %Tag(MSG_HEADER)%  
32 #include "service_custom/Human.h"  
33 // %EndTag(MSG_HEADER)%  
34
```

- サービスとして実行する関数の記述

- Human.srv で定義されたrequestの値を読み込んで処理し、responseの値として返送する

```
34  
35 bool calc_bmi(service_custom::Human::Request &req,  
36                  service_custom::Human::Response &res)  
37 {  
38     res.bmi = req.weight / (req.height/100.0) / (req.height/100.0);  
39     ROS_INFO("request: name: %s height: %d weight: %.2f",  
40               req.name.c_str(), req.height, req.weight);  
41     ROS_INFO("sending back response: bmi = %.2f", res.bmi);  
42     return true;  
43 }  
44
```

# server.cppの実装

- ROS環境の初期化とノードの名前付け

```
58  * part of the ROS system.  
59  */  
60 // %Tag(INIT)%  
61   ros::init(argc, argv, "bmi_server");  
62 // %EndTag(INIT)%  
63
```

- ROSノードの生成

```
68  */  
69 // %Tag(NODEHANDLE)%  
70   ros::NodeHandle n;  
71 // %EndTag(NODEHANDLE)%  
72
```

- サーバノードとしてroscoreとサービスに登録

```
75  */  
76 // %Tag(SERVICESERVER)%  
77   ros::ServiceServer service = n.advertiseService("human_info", calc_bmi);  
78 // %EndTag(SERVICESERVER)%  
79
```

```
80 // %EndTag(SERVICESERVER)%  
81  
82   ROS_INFO("Ready to calc human's BMI.");  
83  
84 // %Tag(SPIN)%  
85   ros::spin();  
86 // %EndTag(SPIN)%  
87  
88 // %EndTag(FULLTEXT)%
```



# client.cppの実装

- ライブラリの読み込み

```
28 // %Tag(FULLTEXT)%  
29 #include "ros/ros.h"  
30 #include "service_custom/Human.h"  
31
```

- ROS環境の初期化とノードの名前付け (+引数チェック)

```
42 * part of the ROS system.  
43 */  
44 ros::init(argc, argv, "bmi_client");  
45  
46 if (argc != 4)  
47 {  
48     ROS_INFO("usage: bmi_client [Name(str)] [Height(uint/cm)] [Weight(float/kg)]")  
49 ;  
50     return 1;  
51 }
```

# client.cppの実装

- ROSノードの生成

```
55     * NodeHandle destructed will close down the node.  
56     */  
57     ros::NodeHandle n;  
58
```

- クライアントノードとしてroscoreとサービスに登録

```
62 // %Tag(SERVICECLIENT)%  
63     ros::ServiceClient client = n.serviceClient<service_custom::Human>("human_info");  
64 // %EndTag(SERVICECLIENT)%  
65
```

- サービスに送信する値の作成 (argvから取得)

```
65  
66     service_custom::Human srv;  
67     srv.request.name = argv[1];  
68     srv.request.height = atoi(argv[2]);  
69     srv.request.weight = atof(argv[3]);  
70  
71     if (client.call(srv))
```

- サービスの呼び出しと返送値の取得

```
70     if (client.call(srv))  
71     {  
72         ROS_INFO("%s's BMI is %.2f", srv.request.name.c_str(), srv.response.bmi);  
73     }  
74     else  
75     {  
76         ROS_ERROR("Failed to call service human_info.");  
77         return 1;  
78     }  
79     }  
80  
81     return 0;  
82 }  
83 // %EndTag(FULLTEXT)%
```

# パッケージのビルドと実行

- ・ パッケージのビルド

```
$ cd ~/catkin_ws  
$ catkin_make
```

- ・ 実行

```
# roscore(master) の起動  
$ ros1setup  
$ roscore
```

```
# サーバの実行  
$ ros1setup  
$ source ~/catkin_ws/devel/setup.bash  
$ rosrun service_custom bmi_servier
```

```
# クライアントの実行  
$ ros1setup  
$ source ~/catkin_ws/devel/setup.bash  
$ rosrun service_custom bmi_client ¥  
bob 183 64.4
```

# 実習の概要と進め方

---

- 概要：ROS1 Melodicでの開発方法の理解
  - まずはpub/subを動かしてみる
  - ワークスペースの設定・topicによる通信
  - 独自定義型のメッセージによるtopic通信
  - serviceによる通信
  - parameterによる通信
- 進め方
  - 進捗が早い方は **演習** に取り組んでみてください
  - ページ下部の **X-Y** は Git Branch番号 に対応します
    - ✓ 適宜でcheckoutして参照してください

# parameter

---

- 多変量辞書 (key value store)
- サーバに対して(非同期に)任意のタイミングで値のset, getなどができる
- 環境変数や設定値のやりとりに使われる
  - 利用可能な型：  
32-bit integers / booleans / strings / doubles /  
iso8601 dates / lists / base64-encoded binary data
- 例題：
  - /foo を5秒毎に読み込み、整数型であれば正常表示する

# パッケージの作成

- parameter利用パッケージ param\_comm の作成

```
# ワークスペースに移動して環境設定  
$ cd ~/catkin_ws  
$ source devel/setup.bash  
# パッケージの作成  
$ cd src/  
$ catkin_create_pkg param_comm roscpp
```



# パッケージの実装

- parameter利用パッケージの実装
  - ~/catkin\_ws/src/param\_comm/src に下記をダウンロード  
[src/getter.cpp](#)
    - ✓ 保存時は拡張子に注意
  - param\_comm/CMakeLists.txt を修正

```
133 ## The recommended prefix ensures that target names across packages don't collide
134 # add_executable(${PROJECT_NAME}_node src/param_comm_node.cpp)
135 add_executable(param_getter src/getter.cpp)
136
137 ## Rename C++ executable without prefix
138 #include <ros/ros.h>
139 #include <string>
140 #include <ros/node_handle.h>
141
142 #include <ros/time.h>
143 #include <ros/console.h>
144
145 #include <ros/param.h>
146
147 #include <boost/thread.hpp>
148
149 #include <${catkin_LIBRARIES}>
150 #
151 target_link_libraries(param_getter ${catkin_LIBRARIES})
152
153 #####
154 ## To be added ####
```

```
getter.cpp
1 #include <ros/ros.h>
2
3 int main(int argc, char** argv) {
4     ros::init(argc, argv, "getter");
5     ros::NodeHandle n;
6
7     ros::Rate loop_rate(0.2);
8
9     while (ros::ok()) {
10         if (n.hasParam("/foo")) {
11             int param;
12             if (n.getParam("/foo", param)) {
13                 ROS_INFO("Value of /foo is %d", param);
14             } else {
15                 ROS_ERROR("Type of /foo is not integer");
16             }
17         } else {
18             ROS_WARN("Key /foo is not set now");
19         }
20
21         loop_rate.sleep();
22     }
23
24     return 0;
25 }
```

# パッケージのビルドと実行

- ・パッケージのビルド

```
$ cd ~/catkin_ws  
$ catkin_make
```

- ・実行

```
# roscore(master) の起動  
$ ros1setup  
$ roscore
```

```
# param読み込みノードの実行  
$ ros1setup  
$ source ~/catkin_ws/devel/setup.bash  
$ rosrun param_comm param_getter
```

```
# parameterへの書き込み  
$ ros1setup  
$ rosparam set /foo 3  
$ rosparam set /foo -5  
$ rosparam set /foo "hoge"
```

# [参考] Pythonパッケージの例

- topic通信パッケージ pubsub\_topic\_py の作成

```
# パッケージの作成
$ cd ~/catkin_ws/src
$ catkin_create_pkg pubsub_topic_py std_msgs rospy
# pythonソースのディレクトリを作成
$ roscd pubsub_topic_py
$ mkdir scripts
```

- topic通信パッケージの実装
  - pubsub\_topic\_py/scripts に下記2ファイルをダウンロード
    - ✓ [scripts/talker.py](#)
    - ✓ [scripts/listener.py](#)

# Pythonパッケージのビルドと実行

- パッケージのビルド

```
$ cd ~/catkin_ws  
$ catkin_make
```

- 実行

```
# roscore(master) の起動  
$ ros1setup  
$ roscore
```

```
# 出版者の実行  
$ ros1setup  
$ source ~/catkin_ws/devel/setup.bash  
$ rosrun pubsub_topic_pub talker.py
```

```
# 購読者の実行  
$ ros1setup  
$ source ~/catkin_ws/devel/setup.bash  
$ rosrun pubsub_topic_py listener.py
```



# ROSの便利な様々な機能

---

- roslaunch
- 機能コマンド
- VSCode ROSプラグイン

# roslaunch

---

- 複数のノードを同時に起動する仕組み
  - roscore が立ち上がっていなければ同時に起動される
  - launch/\*.launchにXML形式で記述する
    - ✓ ノード起動時間をずらしたい（遅延させたい）  
場合は [timed\\_roslaunch](#) を使うこともできる
- 例題：service\_customに対するlaunch実行

# launchファイルの追加

- service\_custom に対するlaunchファイルの作成

```
# ワークスペースに移動  
$ roscd service_custom  
# launch用ディレクトリとファイルの作成  
$ mkdir launch/  
$ touch launch/server_client.launch
```

- server\_client.launchの記述

```
1 <launch>  
2   <node pkg="service_custom" name="bmi_server" type="bmi_server" output="screen" />  
3   <node pkg="service_custom" name="bmi_client1" type="bmi_client" output="screen" a  
rgs="$(arg c1n) $(arg c1h) $(arg c1w)" />  
4 </launch>
```

- pkg: パッケージ名
- name: (launch実行時の)ノード名
- type: ノードの実行ファイル名
- output: 出力を(screenに)表示
- args: 実行時の引数

# launchファイルの追加

- CMakeLists.txt の編集

```
188 " ")
189
190 ## Mark other files for installation (e.g. launch and bag files, etc.)
191 install(FILES
192   launch/server_client.launch
193   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}/Launch
194 )
195
196 #####
197 ## Testing ##
```



# roslaunchの実行

```
# ワークスペースに移動してビルド (launchファイルをinstall)  
$ cd ~/catkin_ws  
$ catkin_make  
# launchファイルの実行  
$ rosrun service_custom server_client.launch  
c1n:=bob c1h:=183 c1w:=64.4
```

## 演習

複数のクライアントを同時に  
実行できるようにしてみましょう

## 演習

pubsub\_topicでも  
launch実行できるように  
してみましょう



# 機能コマンド

コマンド	機能の説明
rospack	ROSパッケージの関連情報の出力
roscd	パッケージのディレクトリへの移動
rosls	パッケージ内のファイルを表示 (ls)
roscp	パッケージからファイルをコピー
rosmsg	メッセージ定義に関連した情報の出力
rossrv	サービス定義に関連した情報の出力
rosnode	ノードに関する操作や情報の出力 (list, info, ping, kill等)
rostopic	トピックに関する操作や情報の出力 (list, info, find, pub, echo等)
rosparam	パラメータに関する操作や情報の出力 (list, load, set, get, dump等)
rqt	デバッグ・可視化ツールの実行 (rqt_console, _graph, _plot等)

# vsCode ROS plugin

