「ROS」の基礎と ROS 2プログラミングの実践
4. ROS 2によるプログラミング

高瀬 英希

(京都大学／JSTさきがけ)

takase@i.kyoto-u.ac.jp

プログラム・スケジュール

4. ROS 2プログラミング [day2 10:00-12:30]

- ROS 2によるロボットソフトウェアの開発［実習］
- ROS 1のプログラミングモデルとの違い

実習の概要と進め方

- 概要：ROS2 Dashing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による実装
  - 独自定義型のライブラリの実装
  - serviceによる通信
  - actionによる通信
  - コンポーネントとコンポジションによるノードの実装
- 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**新しく2つ**開いて，それぞれ下記を実行
  - ROS 1環境の設定されたターミナルでは実行しないでください
  - ROS 1/2環境が混在するとトラブルを招くことが多いです

```
# 出版者(C++版)の実行
$ ros2setup
$ ros2 run demo_nodes_cpp talker
```

```
# 購読者(Python版)の実行
$ ros2setup
$ ros2 run demo_nodes_py listener
```

  - 終了はCtrl+Cです

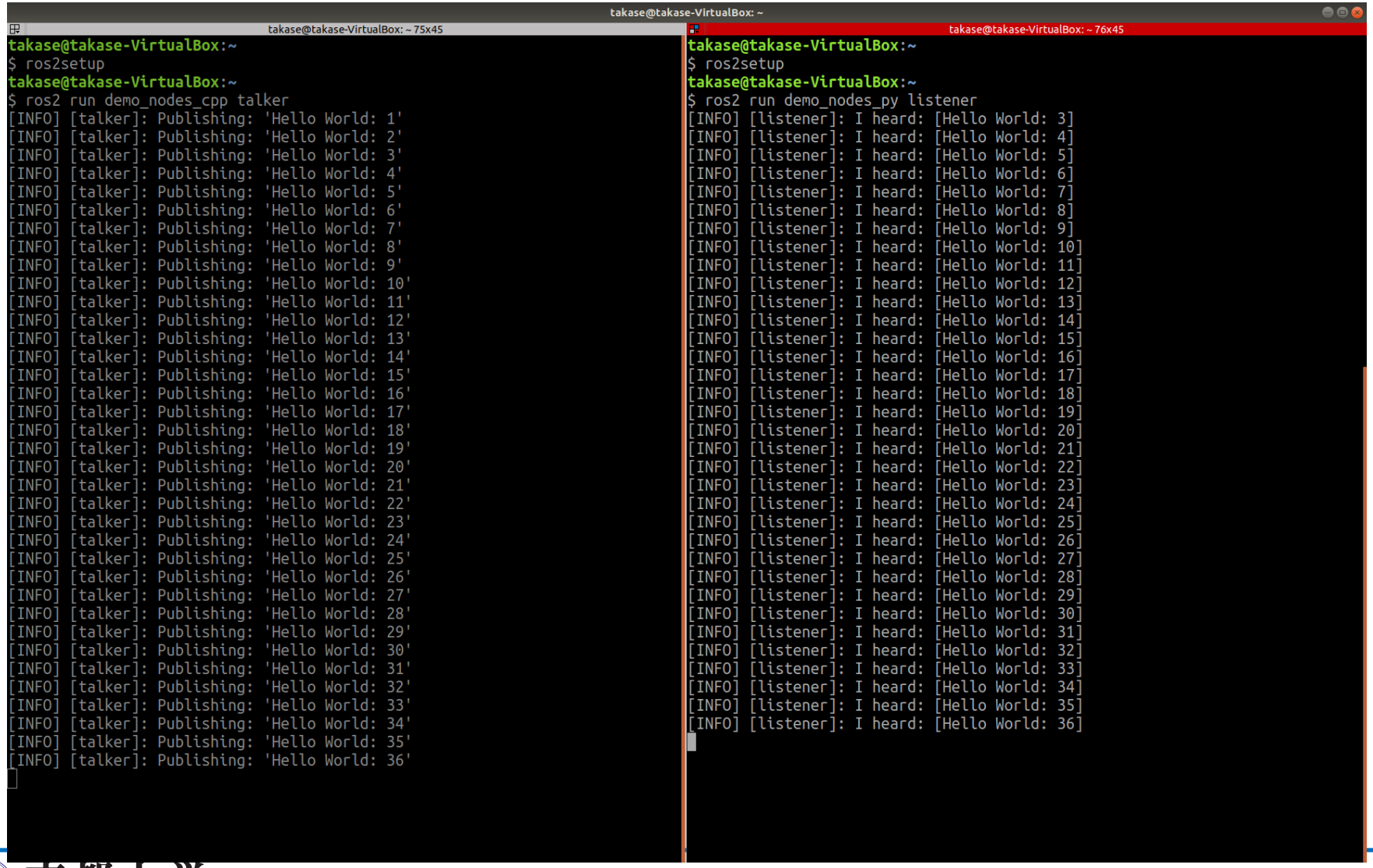
- TIPS: 環境設定（~/.bashrc）

```
129
130 function ros2setup() {
131   export CHOOSE_ROS_DISTRO=dashing
132   source /opt/ros/${CHOOSE_ROS_DISTRO}/setup.bash
133   #export ROS_DOMAIN_ID=42
134 }
135 alias ros2setup=ros2setup
136
```

ROS_DOMAIN_ID
ROS 2通信の空間を
指定する場合の環境変数

他手段：
ROS_LOCALHOST_ONLY=1
DDS通信をlocalhost内に限定
(Eloquent以降)

まずは動かしてみる


takase@takase-VirtualBox: ~
takase@takase-VirtualBox: ~ 75x45
takase@takase-VirtualBox:~
$ ros2setup
takase@takase-VirtualBox:~
$ ros2 run demo_nodes_cpp talker
[INFO] [talker]: Publishing: 'Hello World: 1'
[INFO] [talker]: Publishing: 'Hello World: 2'
[INFO] [talker]: Publishing: 'Hello World: 3'
[INFO] [talker]: Publishing: 'Hello World: 4'
[INFO] [talker]: Publishing: 'Hello World: 5'
[INFO] [talker]: Publishing: 'Hello World: 6'
[INFO] [talker]: Publishing: 'Hello World: 7'
[INFO] [talker]: Publishing: 'Hello World: 8'
[INFO] [talker]: Publishing: 'Hello World: 9'
[INFO] [talker]: Publishing: 'Hello World: 10'
[INFO] [talker]: Publishing: 'Hello World: 11'
[INFO] [talker]: Publishing: 'Hello World: 12'
[INFO] [talker]: Publishing: 'Hello World: 13'
[INFO] [talker]: Publishing: 'Hello World: 14'
[INFO] [talker]: Publishing: 'Hello World: 15'
[INFO] [talker]: Publishing: 'Hello World: 16'
[INFO] [talker]: Publishing: 'Hello World: 17'
[INFO] [talker]: Publishing: 'Hello World: 18'
[INFO] [talker]: Publishing: 'Hello World: 19'
[INFO] [talker]: Publishing: 'Hello World: 20'
[INFO] [talker]: Publishing: 'Hello World: 21'
[INFO] [talker]: Publishing: 'Hello World: 22'
[INFO] [talker]: Publishing: 'Hello World: 23'
[INFO] [talker]: Publishing: 'Hello World: 24'
[INFO] [talker]: Publishing: 'Hello World: 25'
[INFO] [talker]: Publishing: 'Hello World: 26'
[INFO] [talker]: Publishing: 'Hello World: 27'
[INFO] [talker]: Publishing: 'Hello World: 28'
[INFO] [talker]: Publishing: 'Hello World: 29'
[INFO] [talker]: Publishing: 'Hello World: 30'
[INFO] [talker]: Publishing: 'Hello World: 31'
[INFO] [talker]: Publishing: 'Hello World: 32'
[INFO] [talker]: Publishing: 'Hello World: 33'
[INFO] [talker]: Publishing: 'Hello World: 34'
[INFO] [talker]: Publishing: 'Hello World: 35'
[INFO] [talker]: Publishing: 'Hello World: 36'
takase@takase-VirtualBox: ~ 76x45
takase@takase-VirtualBox:~
$ ros2setup
takase@takase-VirtualBox:~
$ ros2 run demo_nodes_py listener
[INFO] [listener]: I heard: [Hello World: 3]
[INFO] [listener]: I heard: [Hello World: 4]
[INFO] [listener]: I heard: [Hello World: 5]
[INFO] [listener]: I heard: [Hello World: 6]
[INFO] [listener]: I heard: [Hello World: 7]
[INFO] [listener]: I heard: [Hello World: 8]
[INFO] [listener]: I heard: [Hello World: 9]
[INFO] [listener]: I heard: [Hello World: 10]
[INFO] [listener]: I heard: [Hello World: 11]
[INFO] [listener]: I heard: [Hello World: 12]
[INFO] [listener]: I heard: [Hello World: 13]
[INFO] [listener]: I heard: [Hello World: 14]
[INFO] [listener]: I heard: [Hello World: 15]
[INFO] [listener]: I heard: [Hello World: 16]
[INFO] [listener]: I heard: [Hello World: 17]
[INFO] [listener]: I heard: [Hello World: 18]
[INFO] [listener]: I heard: [Hello World: 19]
[INFO] [listener]: I heard: [Hello World: 20]
[INFO] [listener]: I heard: [Hello World: 21]
[INFO] [listener]: I heard: [Hello World: 22]
[INFO] [listener]: I heard: [Hello World: 23]
[INFO] [listener]: I heard: [Hello World: 24]
[INFO] [listener]: I heard: [Hello World: 25]
[INFO] [listener]: I heard: [Hello World: 26]
[INFO] [listener]: I heard: [Hello World: 27]
[INFO] [listener]: I heard: [Hello World: 28]
[INFO] [listener]: I heard: [Hello World: 29]
[INFO] [listener]: I heard: [Hello World: 30]
[INFO] [listener]: I heard: [Hello World: 31]
[INFO] [listener]: I heard: [Hello World: 32]
[INFO] [listener]: I heard: [Hello World: 33]
[INFO] [listener]: I heard: [Hello World: 34]
[INFO] [listener]: I heard: [Hello World: 35]
[INFO] [listener]: I heard: [Hello World: 36]

ros2コマンド

コマンド	概要	操作
run	ノードの実行	
pkg	パッケージの生成と情報表示	create, list, prefix, executables
node	ノードの情報表示	info, list
topic	トピックの操作と情報表示	pub, echo, info, list, 等
msg	メッセージ型の情報表示	list, show, package, packages
service	サービスの起動と情報表示	call, list
srv	サービス型の情報表示	list, show, package, packages
action	アクションの操作と情報表示	send_goal, info, list, show
component	コンポーネントの操作と情報表示	standalone, load, list, types, 等
lifecycle	ライフサイクルの操作と情報表示	get, set, list, nodes
param	パラメータの操作と情報表示	get, set, delete, list
…		

実習の概要と進め方

- 概要：ROS2 Dashing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による実装**
  - 独自定義型のライブラリの実装
  - serviceによる通信
  - actionによる通信
  - コンポーネントとコンポジションによるノードの実装
- 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

- colcon用ワークスペースの作成と設定

```
# ROS 2 Dashingの環境設定
$ ros2setup
# ワークスペース用ディレクトリの作成
$ mkdir -p ~/ros2_ws/src
# ワークスペース用ディレクトリへの移動
$ cd ~/ros2_ws/
# ワークスペースの作成
$ colcon build
# ワークスペースの環境設定
$ source install/local_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 ~/ros2_ws/src
$ ros2 pkg create pubsub_topic --dependencies std_msgs rclcpp
```

  - 第1引数：ROS2コマンドの指定
  - 第2引数：ros2 pkgの操作（create, executables, list, prefix）
  - 第3引数：作成するパッケージ名
  - 第4引数以降：ros2 pkg createに関する各種のオプション
    - ✓ --dependencies: 依存するパッケージの指定（複数可）
    - ✓ --description: パッケージの説明の記述
    - ✓ --license: パッケージのライセンス
    - ✓ --maintainer-name: パッケージの開発者・管理者の記述
    - ✓ などなど，，，

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 <?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="
  http://www.w3.org/2001/XMLSchema"?>
3 <package format="3">
4   <name>pubsub_topic</name>
5   <version>0.0.0</version>
6   <description>TODO: Package description</description>
7   <maintainer email="takase@i.kyoto-u.ac.jp">takase</maintainer>
8   <license>TODO: License declaration</license>
9
```

演習

きちんと情報を
記述してみましょう

パッケージの設定

- package.xml：パッケージ情報の定義ファイル
  - ビルドツールの指定と依存パッケージの情報

```
10  <buildtool_depend>ament_cmake</buildtool_depend>
11
12  <depend>std_msgs</depend>
13  <depend>rclcpp</depend>
14
15  <test_depend>ament_lint_auto</test_depend>
16  <test_depend>ament_lint_common</test_depend>
17
18  <export>
19    <build_type>ament_cmake</build_type>
20  </export>
21 </package>
```

  - TIPS: <depend></> は下記３つを兼ねる
    <build_depend></>　：ビルドに依存するパッケージ
    <test_depend></>　：テストに～
    <exec_depend></>　：実行時に～

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

```
cmake_minimum_required(VERSION 3.5)
project(pubsub_topic)

```

  - コンパイラのオプション等の指定

```

# Default to C99
if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 99)
endif()

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()
```

  - 依存パッケージの走査

```

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(std_msgs REQUIRED)
find_package(rclcpp REQUIRED)

```

パッケージの実装

- topic通信パッケージの実装
  - ROS 1ソースから編集していくことで，APIの違いなどを学ぶ

```
# ソースを ROS 1のpubsub_topicからコピー
$ cd ~/ros2_ws/src/pubsub_topic/src
$ cp ~/catkin_ws/src/pubsub_topic/src/* .
```

  - 次ページ以降に従って編集
  - または，差分は下記の通り
    https://github.com/takasehideki/ros_study/commit/b7ee4f9390d86ba0633db42abcb9323172e28175
    - ✓ ROS 1とのAPIとプログラミングスタイルの違いが分かる

パッケージの実装

- CMakeLists.txt の編集
  - ノードの実行ファイル名とソースファイル名の指定
  - 依存パッケージ情報の記述

```
22
23 add_executable(talker src/talker.cpp)
24 ament_target_dependencies(talker rclcpp std_msgs)
25 add_executable(listener src/listener.cpp)
26 ament_target_dependencies(listener rclcpp std_msgs)
27
```

  - ノードの実行ファイルのインストールの指示
    - ✓ devel spaceの作成（ビルド済みファイルのソフトリンク）

```
27
28 install(TARGETS
29   talker
30   listener
31   DESTINATION lib/${PROJECT_NAME}
32 )
33
```

talker.cppの実装と解説

- ライブラリと関数宣言
  - rclcpp: ROS 2 Client Library for C++
  - chrono: C++11の時間ユーティリティ
    - ✓ ROS 2 rclcppではC++11/14機能の積極利用を推奨
    - ✓ C++17, 20にも対応予定

```
28 // %Tag(ROS_HEADER)%
29 #include "rclcpp/rclcpp.hpp"
30 // %EndTag(ROS_HEADER)%
31 // %Tag(MSG_HEADER)%
32 #include "std_msgs/msg/string.hpp"
33 // %EndTag(MSG_HEADER)%
34
35 #include <iostream>
36 #include <chrono>
37 using namespace std::chrono_literals;
38
39 /**
40  * This tutorial demonstrates simple sending of messages over the ROS system.
41  */
42 int main(int argc, char **argv)
43 {
```

talker.cppの実装と解説

- ROS環境の初期化

```
53    */
54 // %Tag(INIT)%
55   rclcpp::init(argc, argv);
56 // %EndTag(INIT)%
57
```

- ROSノードの生成とノードの名前付け

```
62    */
63 // %Tag(NODEHANDLE)%
64   auto n = rclcpp::Node::make_shared("talker");
65 // %EndTag(NODEHANDLE)%
66
```

  - 環境とノードの初期化は別々に行う
  - ノードを共有ライブラリとして実装し，
    複数ノードを同一プロセス内に生成することもできる

talker.cppの実装と解説

- 出版者ノードとしてトピックに登録
  - ✓auto によって型が自動推論されている
  - 第1引数：配信先トピック名
  - 第2引数：QoSのDepth (Historyキューのサイズ)
    - ✓ROS 1ではメッセージキューのサイズ

```
84 // %Tag(PUBLISHER)%
85   auto chatter_pub = n->create_publisher<std_msgs::msg::String>("chatter", 1000);
86 // %EndTag(PUBLISHER)%
```

- ループ周期の設定
  - chronoによる時間単位の記述
  - ROS 1と同様に数値での指定も可能（Hz単位）

```
88 // %Tag(LOOP_RATE)%
89   rclcpp::WallRate loop_rate(100ms);
90 // %EndTag(LOOP_RATE)%
```

talker.cppの実装と解説

- ループとカウント値の生成（ノード実行が正常か）

```
 96 // %Tag(ROS_OK)%
 97   auto count = 0;
 98   while (rclcpp::ok())
 99   {
100 // %EndTag(ROS_OK)%
```

- 配信用のメッセージの型宣言と作成
  - C++11関数で明示的に型変換

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

- メッセージのログ表示
  - ノード情報もログに渡される

```
112 // %Tag(ROSCONSOLE)%
113     RCLCPP_INFO(n->get_logger(), "%s", msg.data.c_str());
114 // %EndTag(ROSCONSOLE)%
```

talker.cppの実装と解説

- メッセージの出版

```
122 // %Tag(PUBLISH)%
123     chatter_pub->publish(msg);
124 // %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）

```
126 // %Tag(SPINONCE)%
127     rclcpp::spin_some(n);
128 // %EndTag(SPINONCE)%
```

- 設定された時間の経過待ち

```
130 // %Tag(RATE_SLEEP)%
131     loop_rate.sleep();
132 // %EndTag(RATE_SLEEP)%
133     ++count;
134   }
```

- ROS 2の実行を終了

```
136   rclcpp::shutdown();
137   return 0;
138 }
139 // %EndTag(FULLTEXT)%
```

listener.cppの実装と解説

- ROSノードのためのスマートポインタを生成
  - 対象のポインタへの所有権を共有するポインタ

```
28 // %Tag(FULLTEXT)%
29 #include "rclcpp/rclcpp.hpp"
30 #include "std_msgs/msg/string.hpp"
31
32 rclcpp::Node::SharedPtr n = nullptr;
33
```

- コールバック関数の定義

```
37 // %Tag(CALLBACK)%
38 void chatterCallback(const std_msgs::msg::String::SharedPtr msg)
39 {
40   RCLCPP_INFO(n->get_logger(), "I heard: [%s]", msg->data.c_str());
41 }
42 // %EndTag(CALLBACK)%
```

- ROS環境の初期化

```
55    */
56   rclcpp::init(argc, argv);
57
```

- ROSノードの生成とノードの名前付け

```
62    */
63   n = rclcpp::Node::make_shared("listener");
64
```

listener.cppの実装と解説

- 購読者ノードとしてトピックに登録
  - 第1引数：配信先トピック名
  - 第2引数：QoSのDepth (History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

```
80 // %Tag(SUBSCRIBER)%
81   auto sub = n->create_subscription<std_msgs::msg::String>("chatter", 1000, chatterC
   allback);
82 // %EndTag(SUBSCRIBER)%
```

- コールバック関数の実行を無限待ち（Ctrl+Cまで）

```
89 // %Tag(SPIN)%
90   rclcpp::spin(n);
91 // %EndTag(SPIN)%
```

- ROS 2の実行を終了
  - スマートポインタを解放（初期化）

```
92
93   rclcpp::shutdown();
94   sub = nullptr;
95   n = nullptr;
96   return 0;
97 }
```

パッケージのビルドと実行

- バッケージのビルド

```
$ cd ~/ros2_ws
$ colcon build --packages-select pubsub_topic
```

- 実行

```
# 出版者の実行
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run pubsub_topic talker
```

```
# 購読者の実行
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run pubsub_topic listener
```

. (ドット)のあとにスペースが必要 sourceと同義
新しいパッケージをビルドしたのちに都度で必要

演習

chatter_pubに対する
複数のtalker/listenerを同時に
実行できるようにしましょう

実習の概要と進め方

- 概要：ROS2 Dashing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による実装
  - **独自定義型のライブラリの実装**
  - serviceによる通信
  - actionによる通信
  - コンポーネントとコンポジションによるノードの実装
- 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

- ROS 1と同様
  - Primitive Type
  - Array Type

- 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
  - ネスト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

独自定義型のライブラリ

- ROS 2では，独自定義のmsg (srv, action)は
  ノード実装のパッケージとは別にライブラリ化する
  - パッケージごとの保守性，移植性が良くなる
  - colconビルド時のトラブルも抑止できる

- 例題：独自定義型のライブラリ ros_study_types
  - Human型msgの定義：
    string name, uint16 height, float32 weight
  - 後段のsrv, actionも同パッケージにまとめる

パッケージの作成

- 独自定義型のライブラリ ros_study_types の作成

```
# ワークスペースに移動して環境設定
$ cd ~/ros2_ws
$ source install/local_setup.bash
# パッケージの作成
$ cd src/
$ ros2 pkg create ros_study_types --dependencies rosidl_default_generators
# 使用しないディレクトリ (src, include) を削除
$ rm -rf ros_study_types/src ros_study_types/include
```

  - rosidl_default_generators: 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
$ cd ros_study_types
# 定義ファイルを作成
$ mkdir msg
$ touch msg/Human.msg
```

- msg/Human.msg を編集（作成）

  - TIPS: 文字列の最大長や配列の最大要素数，初期値を指定できる
    - ✓ people<=10[<=2]　　# 10字以下で2要素までの配列
    - ✓ string name “Takase”　　# 初期値を指定

独自メッセージの作成

- package.xml を編集
  - ROS IDL用の各種設定を追加する

```
10  <buildtool_depend>ament_cmake</buildtool_depend>
11
12  <buildtool_depend>rosidl_default_generators</buildtool_depend>
13  <exec_depend>rosidl_default_runtime</exec_depend>
14  <member_of_group>rosidl_interface_packages</member_of_group>
15
16  <test_depend>ament_lint_auto</test_depend>
```

独自メッセージの作成

- CMakeLists.txt を編集
  - ROS IDLメッセージ生成用の設定，対象ファイルの指定，ライブラリ作成のためのパッケージ指定，を追加

```
18 # find dependencies
19 find_package(ament_cmake REQUIRED)
20 find_package(rosidl_default_generators REQUIRED)
21
22 set(msg_files
23   "msg/Human.msg"
24 )
25
26 rosidl_generate_interfaces(${PROJECT_NAME}
27   ${msg_files}
28 )
29
30 ament_export_dependencies(rosidl_default_runtime)
31
32 if(BUILD_TESTING)
```

- ビルド

```
# ワークスペースのディレクトリに移動
$ cd ~/ros2_ws
# ビルド（定義ファイルを生成）
$ colcon build --packages-select ros_study_types
```

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
  - ~/ros2_ws/install/ros_study_types/include/ros_study_types/msg/*

human.hpp

```
1 // generated from rosidl_generator_cpp/resource/idl.hpp.em
2 // generated code does not contain a copyright notice
3
4 #ifndef PUBSUB_CUSTOM__MSG__HUMAN_HPP_
5 #define PUBSUB_CUSTOM__MSG__HUMAN_HPP_
6
7 #include "pubsub_custom/msg/human__struct.hpp"
8 #include "pubsub_custom/msg/human__traits.hpp"
9
10 #endif  // PUBSUB_CUSTOM__MSG__HUMAN_HPP_
```

```
32
33 namespace pubsub_custom
34 {
35
36 namespace msg
37 {
38
39 // message struct
40 template<class ContainerAllocator>
41 struct Human_
42 {
43   using Type = Human_<ContainerAllocator>;
44
45   explicit Human_(rosidl_generator_cpp::MessageInitialization _init = rosidl_gener
   ator_cpp::MessageInitialization::ALL)
46   {
47     if (rosidl_generator_cpp::MessageInitialization::ALL == _init ||
48       rosidl_generator_cpp::MessageInitialization::ZERO == _init)
49     {
50       this->name = "";
51       this->height = 0;
52       this->weight = 0.0f;
53     }
54   }
55
56   explicit Human_(const ContainerAllocator & _alloc, rosidl_generator_cpp::Message
   Initialization _init = rosidl_generator_cpp::MessageInitialization::ALL)
57   : name(_alloc)
```

パッケージの作成

- 独自定義型を利用する pubsub_custom の作成

```
# ワークスペースに移動して環境設定
$ cd ~/ros2_ws
$ source install/local_setup.bash
# パッケージの作成
$ cd src/
$ ros2 pkg create pubsub_custom --dependencies rclcpp ros_study_types
```

  - 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])^2

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
$ cd ~/ros2_ws/src/pubsub_custom
$ cp ../pubsub_topic/src/* src/
```

  - 下記の差分に従って編集
    - ✓ https://github.com/takasehideki/ros_study/commit/fd1c8743ba358cc4738c27723387133da595f98e
  - 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
  - 独自定義型ライブラリの走査（pkg create時に自動追加）

```
18 # find dependencies
19 find_package(ament_cmake REQUIRED)
20 find_package(rclcpp REQUIRED)
21 find_package(ros_study_types REQUIRED)
22
```

  - 依存パッケージ情報にも追加

```
22
23 add_executable(bmi_talker src/talker.cpp)
24 ament_target_dependencies(bmi_talker rclcpp ros_study_types)
25 add_executable(bmi_listener src/listener.cpp)
26 ament_target_dependencies(bmi_listener rclcpp ros_study_types)
27
28 install(TARGETS
29   bmi_talker
30   bmi_listener
31   DESTINATION lib/${PROJECT_NAME}
32 )
33
34 if(BUILD_TESTING)
```

talker.cppの実装

- 独自定義型のライブラリ ros_study_types の読み込み

```
28 // %Tag(ROS_HEADER)%
29 #include "rclcpp/rclcpp.hpp"
30 // %EndTag(ROS_HEADER)%
31 // %Tag(MSG_HEADER)%
32 #include "ros_study_types/msg/human.hpp"
33 // %EndTag(MSG_HEADER)%
34
```

- ROSノードの名前付け

```
63 // %Tag(NODEHANDLE)%
64   auto n = rclcpp::Node::make_shared("bmi_talker");
65 // %EndTag(NODEHANDLE)%
```

- 出版者ノードとしてトピックに登録

```
84 // %Tag(PUBLISHER)%
85   auto chatter_pub = n->create_publisher<ros_study_types::msg::Human>("chatter", 100
   0);
86 // %EndTag(PUBLISHER)%
```

talker.cppの実装

- 配信用のメッセージの型宣言と作成

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

listener.cppの実装

- ライブラリの読み込み

```
28 // %Tag(FULLTEXT)%
29 #include "rclcpp/rclcpp.hpp"
30 #include "ros_study_types/msg/human.hpp"
31
```

- コールバック関数の定義

```
37 // %Tag(CALLBACK)%
38 void chatterCallback(const ros_study_types::msg::Human::SharedPtr msg)
39 {
40   float bmi = msg->weight / (msg->height/100.0) / (msg->height/100.0);
41   RCLCPP_INFO(n->get_logger(), "%s's BMI is %.2f", msg->name.c_str(), bmi);
42 }
43 // %EndTag(CALLBACK)%
```

- ROSノードの名前付け

```
63    */
64   n = rclcpp::Node::make_shared("bmi_listener");
65
66   /**
```

- 購読者ノードとして登録

```
81 // %Tag(SUBSCRIBER)%
82   auto sub = n->create_subscription<ros_study_types::msg::Human>("chatter", 1000, ch
   atterCallback);
83 // %EndTag(SUBSCRIBER)%
```

パッケージのビルドと実行

- バッケージのビルド

```
$ cd ~/ros2_ws
$ colcon build --packages-select pubsub_custom
```

- 実行

```
# 出版者の実行
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run pubsub_custom bmi_talker
```

```
# 購読者の実行
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run pubsub_custom bmi_listener
```

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

4-6

実習の概要と進め方

- 概要：ROS2 Dashing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
  - 独自定義型のライブラリの実装
  - **serviceによる通信**
  - actionによる通信
  - コンポーネントとコンポジションによるノードの実装
- 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

- 内容
  - ROS 1でのservice実装をROS 2に変更する
  - 独自定義のserviceを ros_study_types に追加する

- 例題：人物の情報を送信してBMI計算値を受ける
  - human型の定義：
    string name, uint16 height, float32 weight,
    float32 bmi
  - BMI計算 = (体重[kg]) / (身長[m])^2

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

```
# 独自定義型のライブラリのディレクトリに移動
$ cd ros_study_types
# 定義ファイルを作成
$ mkdir srv
$ touch srv/Human.srv
```

- srv/Human.srv を編集（作成）

独自サービスの作成

- CMakeLists.txt を編集
  - 対象ファイルの指定に .srv ファイル群を追加

```
22 set(msg_files
23   "msg/Human.msg"
24 )
25
26 set(srv_files
27   "srv/Human.srv"
28 )
29
30 rosidl_generate_interfaces(${PROJECT_NAME}
31   ${msg_files}
32   ${srv_files}
33 )
34
35 ament_export_dependencies(rosidl_default_runtime)
```

- ビルド

```
# ワークスペースのディレクトリに移動
$ cd ~/ros2_ws
# ビルド（定義ファイルを生成）
$ colcon build --packages-select ros_study_types
```

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 ~/ros2_ws
$ source install/local_setup.bash
# パッケージの作成
$ cd src/
$ ros2 pkg create service_custom --dependencies rclcpp ros_study_types
```

パッケージの実装

- 独自サービス通信パッケージの実装
  - ROS 1ソースから編集していくことで，APIの違いなどを学ぶ

```
# ソースを ROS 1のservice_customからコピー
$ cd ~/ros2_ws/src/service_custom/src
$ cp ~/catkin_ws/src/service_custom/src/* .
```

  - 下記の差分に従って編集
    https://github.com/takasehideki/ros_study/commit/ed472a54cd79fbaa7e5dc5d9cdb1ad7ac0ee8ad5
    - ✓ ROS 1とのAPIとプログラミングスタイルの違いが分かる
  - 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/server.cpp
    - ✓ src/client.cpp

パッケージの実装

- CMakeLists.txt の編集
  - 独自定義型ライブラリの走査（pkg create時に自動追加）

```
18 # find dependencies
19 find_package(ament_cmake REQUIRED)
20 find_package(rclcpp REQUIRED)
21 find_package(ros_study_types REQUIRED)
22
```

  - 依存パッケージ情報にも追加

```
22
23 add_executable(bmi_server src/server.cpp)
24 ament_target_dependencies(bmi_server rclcpp ros_study_types)
25 add_executable(bmi_client src/client.cpp)
26 ament_target_dependencies(bmi_client rclcpp ros_study_types)
27
28 install(TARGETS
29   bmi_server
30   bmi_client
31   DESTINATION lib/${PROJECT_NAME}
32 )
33
34 if(BUILD_TESTING)
```

server.cppの実装

- ライブラリの読み込み

```
28 // %Tag(ROS_HEADER)%
29 #include "rclcpp/rclcpp.hpp"
30 // %EndTag(ROS_HEADER)%
31 // %Tag(MSG_HEADER)%
32 #include "ros_study_types/srv/human.hpp"
33 // %EndTag(MSG_HEADER)%
```

- ROSノードのためのスマートポインタを生成

```
34
35 rclcpp::Node::SharedPtr n = nullptr;
36
```

- サービスとして実行する関数の記述
  - 42行目はビルド時警告の抑止のため
  - requestの値を読み込んで処理し，responseの値として返送する

```
36
37 void calc_bmi(
38   const std::shared_ptr<rmw_request_id_t> req_header,
39   const std::shared_ptr<ros_study_types::srv::Human::Request>  req,
40   const std::shared_ptr<ros_study_types::srv::Human::Response> res)
41 {
42   (void)req_header;
43   res->bmi = req->weight / (req->height/100.0) / (req->height/100.0);
44   RCLCPP_INFO(n->get_logger(), "request: name: %s height: %d weight: %.2f",
45     req->name.c_str(), req->height, req->weight);
46   RCLCPP_INFO(n->get_logger(), "sending back response: bmi = %.2f", res->bmi);
47 }
48
```

server.cppの実装

- ROS環境の初期化

```
63    */
64 // %Tag(INIT)%
65   rclcpp::init(argc, argv);
66 // %EndTag(INIT)%
67
```

- ROSノードの生成とノードの名前付け

```
72    */
73 // %Tag(NODEHANDLE)%
74   n = rclcpp::Node::make_shared("bmi_server");
75 // %EndTag(NODEHANDLE)%
76
```

- サーバノードとしてサービスに登録

```
80 // %Tag(SERVICESERVER)%
81   auto service = n->create_service<ros_study_types::srv::Human>("human_info", calc_b
   mi);
82 // %EndTag(SERVICESERVER)%
```

```
83
84   RCLCPP_INFO(n->get_logger(), "Ready to calc human's BMI.");
85
86 // %Tag(SPIN)%
87   rclcpp::spin(n);
88 // %EndTag(SPIN)%
89
90   rclcpp::shutdown();
91   n = nullptr;
92
93   return 0;
94 }
95 // %EndTag(FULLTEXT)%
```

client.cppの実装

- ライブラリの読み込み

```
28 // %Tag(FULLTEXT)%
29 #include "rclcpp/rclcpp.hpp"
30 #include "ros_study_types/srv/human.hpp"
31 #include <chrono>
32 using namespace std::chrono_literals;
33
```

- ROS環境の初期化

```
44    * part of the ROS system.
45    */
46   rclcpp::init(argc, argv);
47
48   /**
```

- ROSノードの生成と名前付け（＋引数チェック）

```
51    * NodeHandle destructed will close down the node.
52    */
53   auto n = rclcpp::Node::make_shared("bmi_client");
54
55   if (argc != 4)
56   {
57     RCLCPP_INFO(n->get_logger(), "usage: bmi_clinent [Name(str)] [Height(uint/cm)] [
   Weight(float/kg)]");
58     return 1;
59   }
60
61   /**
```

client.cppの実装

- クライアントノードとしてサービスに登録

```
63    */
64  // %Tag(SERVICECLIENT)%
65    auto client = n->create_client<ros_study_types::srv::Human>("human_info");
66  // %EndTag(SERVICECLIENT)%
67
```

- サービスとの接続確立を待つ

```
67
68    while (!client->wait_for_service(1s)) {
69      if (!rclcpp::ok()) {
70        RCLCPP_ERROR(n->get_logger(), "client interrupted while waiting for service to
    appear.");
71        return 1;
72      }
73      RCLCPP_INFO(n->get_logger(), "waiting for service to appear...");
74    }
75
```

- サービスに送信する値の作成（argvから取得）

```
75
76    auto request = std::make_shared<ros_study_types::srv::Human::Request>();
77    request->name = argv[1];
78    request->height = atoi(argv[2]);
79    request->weight = atof(argv[3]);
80
```

client.cppの実装

- サービスの呼び出しと返送値の取得

```
80
81   auto response_future = client->async_send_request(request);
82   if (rclcpp::spin_until_future_complete(n, response_future) ==
83     rclcpp::executor::FutureReturnCode::SUCCESS)
84   {
85     auto response = response_future.get();
86     RCLCPP_INFO(n->get_logger(), "%s's BMI is %.2f", request->name.c_str(), response
   ->bmi);
87   }
88   else
89   {
90     RCLCPP_ERROR(n->get_logger(), "Failed to call service human_info.");
91     return 1;
92   }
93
94   rclcpp::shutdown();
95   return 0;
96 }
97 // %EndTag(FULLTEXT)%
```

パッケージのビルドと実行

- バッケージのビルド

```
$ cd ~/ros2_ws
$ colcon build --packages-select service_custom
```

- 実行

```
# サーバの実行
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run service_custom bmi_server
```

```
# クライアントの実行
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run service_custom bmi_client ¥
  bob 183 64.4
```

roslaunch

- 複数のノードを同時に起動する仕組み
  - ROS 2 Dashing以前ではPython対応が進んでいた
  - EloquentでXML/YAML対応に(回帰？)
    - ✓composition や lifecycle など未対応のものも
- 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
$ cd ~/ros2_ws/src/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.py
```

launchファイルの追加

- server_client.launch.py の編集
  - または上記リンクよりダウンロード（拡張子に注意）

```python
import launch
import launch_ros.actions


def generate_launch_description():
  bmi_server = launch_ros.actions.Node(
      package='service_custom',
      node_executable='bmi_server',
      output='screen'
      )
  bmi_client1 = launch_ros.actions.Node(
      package='service_custom',
      node_executable='bmi_client',
      output='screen',
      arguments=['Bob', '183', '63.3']
      )

  return launch.LaunchDescription([
      bmi_server,
      bmi_client1,
      launch.actions.RegisterEventHandler(
        event_handler=launch.event_handlers.OnProcessExit(
          target_action=bmi_server,
          on_exit=[launch.actions.EmitEvent(event=launch.events.Shutdown())],
          )),
  ])
```

launchファイルの追加

- CMakeLists.txt の編集

```
32 )
33
34 install(DIRECTORY
35   launch
36   DESTINATION share/${PROJECT_NAME}/
37 )
38
39 if(BUILD_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 ~/ros2_ws
$ colcon build --packages-select service_custom
$ . install/local_setup.bash
# launchファイルの実行
$ ros2 launch service_custom server_client.launch.py
```

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

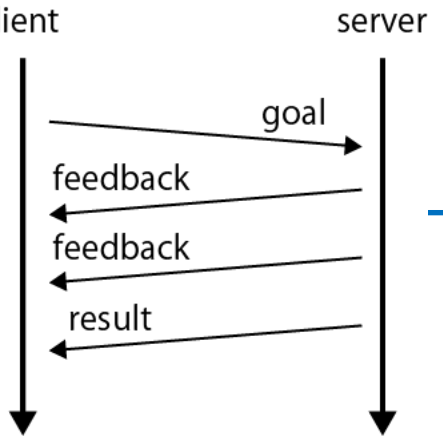
演習

pubsub_topicでも
launch実行できるようにして
みましょう

実習の概要と進め方

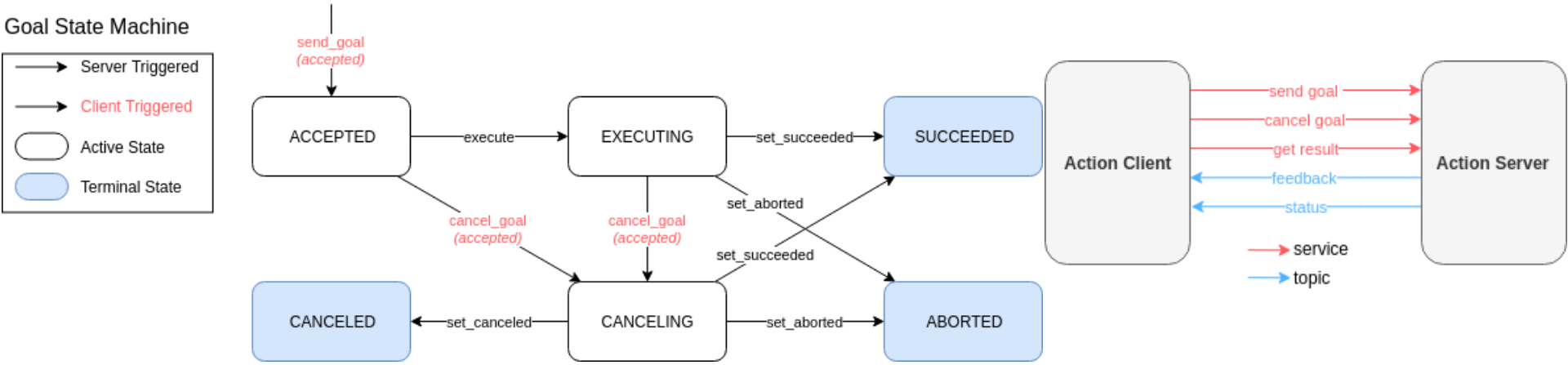
- 概要：ROS2 Dashingでの開発方法の理解
  - まずはpub/subを動かしてみる
  - ワークスペースの設定・topicによる通信
  - 独自定義型のライブラリの実装
  - serviceによる通信
  - **actionによる通信**
  - コンポーネントとコンポジションによるノードの実装
- 進め方
  - 進捗が早い方は 演習 に取り組んでみてください
  - ページ下部の X-Y はGit Branch番号に対応します
    - ✓適宜でcheckoutして参照してください

Actions



- 非同期送信／同期受信の組合せ
  - Entities: action server / client
  - Interface Definition: Goal / Result / Feedback

→ 今回は例題でプログラミングモデルと動作を確認



http://design.ros2.org/articles/actions.html

アクションの定義

- アクションの定義ファイルの作成

```
# 独自定義型のライブラリのディレクトリに移動
$ cd ros_study_types
# 定義ファイルを作成
$ mkdir action
$ touch action/Fibonacci.action
```

- action/Fibonacci.action を編集（作成）

```
1 int32 order
2 ---
3 int32[] sequence
4 ---
5 int32[] sequence
```

上から，Goal, Result, Feedback

アクションの作成

- CMakeLists.txt を編集
  - 対象ファイルの指定に .action ファイル群を追加

```
26 set(srv_files
27   "srv/Human.srv"
28 )
29
30 set(action_files
31   "action/Fibonacci.action"
32 )
33
34 rosidl_generate_interfaces(${PROJECT_NAME}
35   ${msg_files}
36   ${srv_files}
37   ${action_files}
38 )
39
40 ament_export_dependencies(rosidl_default_runtime)
```

- ビルド

```
# ワークスペースのディレクトリに移動
$ cd ~/ros2_ws
# ビルド（定義ファイルを生成）
$ colcon build --packages-select ros_study_types
```

演習

生成されたヘッダファイルを確認してみましょう

パッケージの作成

- アクション通信パッケージ action_custom の作成

```
# ワークスペースに移動して環境設定
$ cd ~/ros2_ws
$ source install/local_setup.bash
# パッケージの作成
$ cd src/
$ ros2 pkg create action_custom --dependencies rclcpp rclcpp_action ros_study_types
```

パッケージの実装

- 今回は用意してある例題のソースを動かす

```
$ git checkout 4-a
```

  - または，下記からダウンロードして配置する
    - ✓ CMakeLists.txt
    - ✓ src/server.cpp
    - ✓ src/client.cpp
    - ✓ src/client_with_cancel.cpp
    - ✓ src/client_with_feedback.cpp

パッケージのビルドと実行

- バッケージのビルド

```
$ cd ~/ros2_ws
$ colcon build --packages-select action_custom
```

- 実行

```
# サーバの実行
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run action_custom servier
```

```
# クライアントの実行
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run action_custom client
```

演習

client_with_cancel,
client_with_feedback も
動かしてみましょう

演習

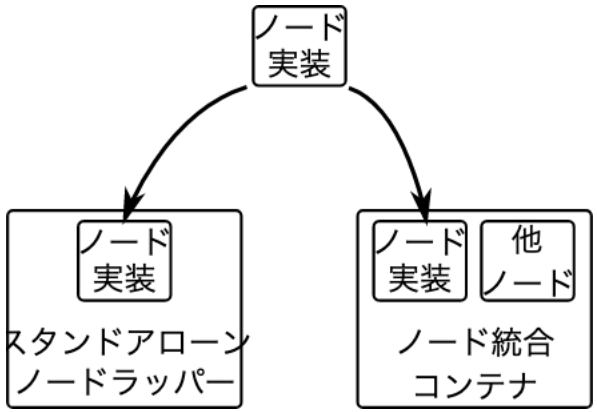
ソースを追いながら,
APIや振る舞いの違いを
確認してみましょう

実習の概要と進め方

- 概要：ROS2 Dashing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
  - 独自定義型のライブラリの実装
  - serviceによる通信
  - actionによる通信
  - **コンポーネントとコンポジションによるノードの実装**
- 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

コンポーネントによるノード実装

- ROS 2（Dashing以降）で推奨される実装方式
- ノード実装をコンポーネントとして再利用する
  - 共有ライブラリとしてノードを実装する
  - 「コンポーネントノード」とも呼ばれる
  - $ ros2 component
- 例題：pubsub_topic の talker をコンポーネントで実装



パッケージの作成

- コンポーネント実装パッケージの作成

```
# パッケージの作成
$ ros2 pkg create pubsub_component --dependencies rclcpp std_msgs class_loader
```

- コンポーネント実装パッケージの実装
  - pubsub_topic を起点として実装することで，
    ROS 2において推奨されるプログラミングモデルを学ぶ

```
# ソースを pubsub_topicからコピー
$ cd ~/ros2_ws/src/pubsub_component/src
# ソースをそれぞれコピー
$ cp ../../pubsub_topic/src/talker.cpp .
$ cp talker.cpp talker_component.cpp
```

パッケージの実装

- コンポーネント実装パッケージの実装
  - 下記の差分に従って作成・編集
    - ✓ https://github.com/takasehideki/ros_study/commit/548a8b4ec0fca9e805328b6651ac8d3394f1c418
  - または，正解は下記
    - ✓ CMakeLists.txt
    - ✓ src/talker.cpp
    - ✓ src/talker_component.cpp
    - ✓ include/pubsub_component/talker_component.hpp

パッケージの実装

- CMakeLists.txt の編集
  - パッケージの include/ を指定（設定追加）

```
22 find_package(class_loader REQUIRED)
23
24 include_directories(include)
25
26 add_library(talker_component SHARED src/talker_component.cpp)
27
```

  - コンポーネントノードの共有ライブラリをコンパイル指定

```
25
26 add_library(talker_component SHARED src/talker_component.cpp)
27
```

  - ヘッダファイル中のマクロを設定

```
28 target_compile_definitions(talker_component PRIVATE "TALKER_BUILDING_DLL")
29
```

  - 共有ライブラリの利用する他パッケージの指定

```
30 ament_target_dependencies(talker_component
31   rclcpp
32   std_msgs
33   class_loader
34 )
35
```

パッケージの実装

- CMakeLists.txt の編集
  - スタンドアロンの実行ノードのコンパイル指定等

```
36 add_executable(talker src/talker.cpp)
37 target_link_libraries(talker talker_component)
38 ament_target_dependencies(talker rclcpp std_msgs)
```

  - コンポーネントを再利用できるようにエクスポート

```
40 ament_export_include_directories(include)
41 ament_export_libraries(talker_component)
```

  - ヘッダファイルのインストール

```
43 install(DIRECTORY
44   include/pubsub_component
45   DESTINATION include
46 )
```

  - 共有ライブラリのインストール

```
48 install(TARGETS
49   talker_component
50   ARCHIVE DESTINATION lib
51   LIBRARY DESTINATION lib
52   RUNTIME DESTINATION bin
53 )
```

  - 実行ノードのインストール

```
55 install(TARGETS
56   talker
57   DESTINATION lib/${PROJECT_NAME}
58 )
```

talker.cppの実装

- ライブラリと関数宣言

```
28 // %Tag(ROS_HEADER)%
29 #include "rclcpp/rclcpp.hpp"
30 // %EndTag(ROS_HEADER)%
31
32 #include "pubsub_component/talker_component.hpp"
33
```

- コンポーネントからノードの生成
  - メモリ確保を同時に行うスマートポインタ

```
58 // %Tag(NODEHANDLE)%
59   auto n = std::make_shared<pubsub_component::Talker>();
60 // %EndTag(NODEHANDLE)%
```

- イベントの無限待ち

```
62 // %Tag(SPIN)%
63   rclcpp::spin(n);
64 // %EndTag(SPIN)%
65
66   rclcpp::shutdown();
67   return 0;
68 }
69 // %EndTag(FULLTEXT)%
```

talker_component.cppの実装

- ライブラリの読み出し

```
28 // %Tag(ROS_HEADER)%
29 #include "rclcpp/rclcpp.hpp"
30 // %EndTag(ROS_HEADER)%
31
32 #include "pubsub_component/talker_component.hpp"
33 #include "class_loader/register_macro.hpp"
34
```

- 名前空間の定義とノードの名前付け

```
39 namespace pubsub_component
40 {
41
42 Talker::Talker()
43 : Node("talker")
44 {
45   /**
```

- 出版者ノードとして登録

```
62 // %Tag(PUBLISHER)%
63   chatter_pub = create_publisher<std_msgs::msg::String>("chatter", 1000);
64 // %EndTag(PUBLISHER)%
```

talker_component.hppの実装

- ライブラリの読み出し

```
41
42 #include <rclcpp/rclcpp.hpp>
43 #include <std_msgs/msg/string.hpp>
44
```

- 名前空間とクラスの定義

```
45 namespace pubsub_component
46 {
47
48   class Talker : public rclcpp::Node
49   {
50     public:
51       TALKER_PUBLIC Talker();
52
53     private:
54       rclcpp::Publisher<std_msgs::msg::String>::SharedPtr chatter_pub;
55
56   };
57
58 }
59
```

パッケージのビルドと実行

- バッケージのビルド

```
$ cd ~/ros2_ws
$ colcon build --packages-select pubsub_component
```

- 実行

```
# 出版者の実行
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run pubsub_component talker
```

```
# 購読者の実行
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run pubsub_topic listener
```

演習

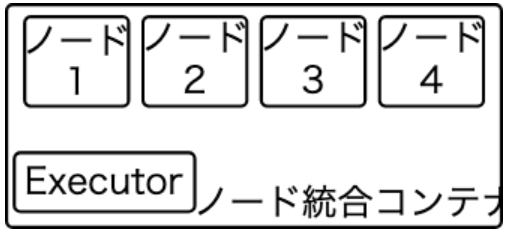
listenerもコンポーネントで
実装してみましょう

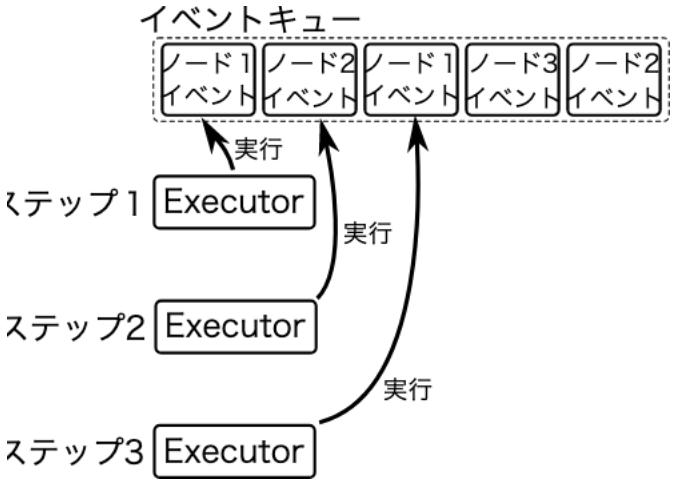
正解は 本git logを参照

コンポジションによるノード実装

- 共有ライブラリのコンポーネントを活用して複数のノードを実装する
  - 同一プロセス内で多ノードが実行できる
  - ノード間（プロセス内）通信をゼロコピーで行える
  - Executorオブジェクト内で実行される

- 例題：pubsub_component の talker をコンポジションで実行





パッケージの作成

- コンポジション実装パッケージの作成

```
# パッケージの作成
$ ros2 pkg create pubsub_composition --dependencies rclcpp pubsub_component
```

- コンポジション実装パッケージの実装

```
# ソースを pubsub_compositionからコピー
$ cd ~/ros2_ws/src/pubsub_composition/src
# ソースをコピー
$ cp ../../pubsub_component/src/talker.cpp .
```

  - 下記の差分に従って作成・編集
    - ✓ https://github.com/takasehideki/ros_study/commit/3efcae1d6d0e6f23c7081e91773c7db38286033d
  - または，正解は下記
    - ✓ CMakeLists.txt　　src/talker.cpp

パッケージの実装

- CMakeLists.txt の編集
  - コンポーネント実装を依存パッケージとして追加

```
18 # find dependencies
19 find_package(ament_cmake REQUIRED)
20 find_package(rclcpp REQUIRED)
21 find_package(pubsub_component REQUIRED)
22
23 add_executable(talker_comp src/talker.cpp)
24 ament_target_dependencies(talker_comp
25   rclcpp
26   pubsub_component
27 )
28
```

  - 実行ノードのインストール

```
27 )
28
29 install(TARGETS
30   talker_comp
31   DESTINATION lib/${PROJECT_NAME}
32 )
33
```

talker.cppの実装

- シングルスレッドのExecutorを生成

```
52
53   rclcpp::executors::SingleThreadedExecutor exec;
54
```

- コンポーネントノードのインスタンスの生成とExecutorへの登録

```
54
55   auto talker = std::make_shared<pubsub_component::Talker>();
56   exec.add_node(talker);
57
```

- Executorのイベント待ち

```
58 // %Tag(SPIN)%
59   exec.spin();
60 // %EndTag(SPIN)%
```

パッケージのビルドと実行

- バッケージのビルド

```
$ cd ~/ros2_ws
$ colcon build --packages-select pubsub_composition
```

- 実行

```
# 出版者の実行
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run pubsub_composition ¥
    talker_comp
```

```
# 購読者の実行
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run pubsub_topic listener
```

演習

listenerもコンポジションで
実装してみましょう

4-ca