

Моделирование колесных роботов

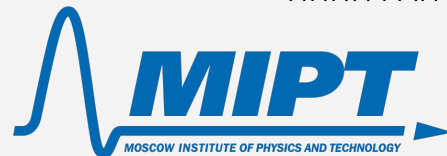
Семинар 2. Файловая система ROS,
создание пакета, типы коммуникации

Николай Жердев

Москва, 2023



ИППИ РАН

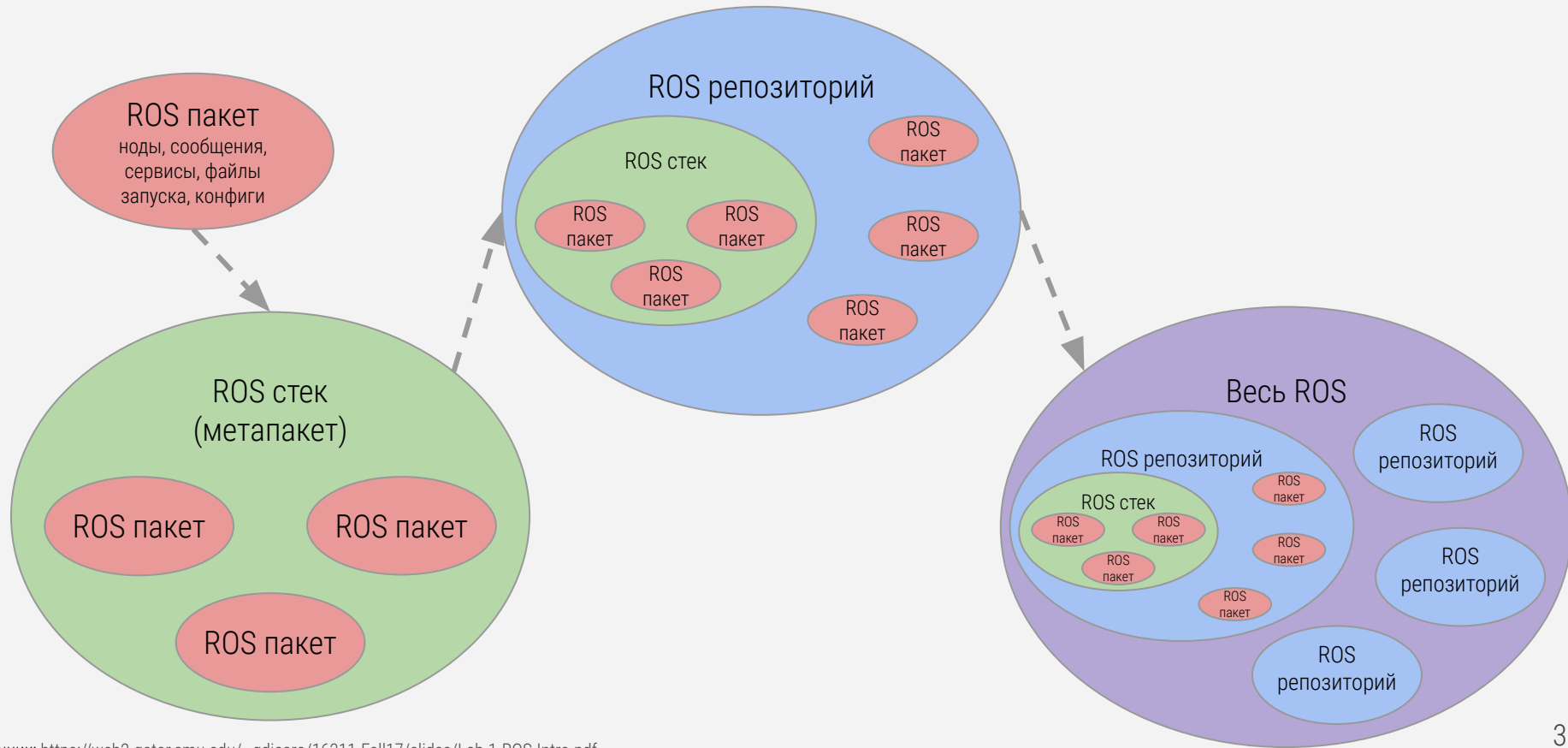




СОДЕРЖАНИЕ ЛЕКЦИИ

1. Файловая система ROS
2. Создание и компиляция пакета
 - a. CMakeLists.txt
 - b. package.xml
3. Типы коммуникации в ROS
4. Написание простых нодов: Publisher и Subscriber
 - a. Использование стандартных типов сообщений
 - b. Написание своего msg-файла

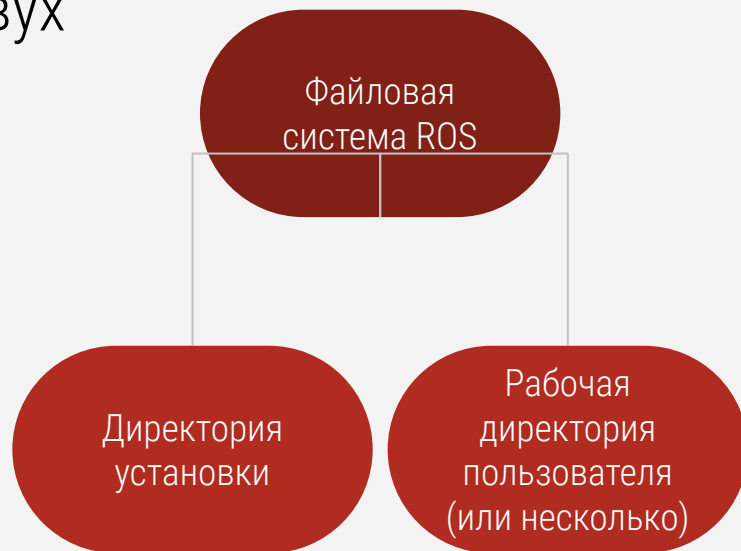
НАЗАД К СТРУКТУРЕ...



ФАЙЛОВАЯ СИСТЕМА ROS

<https://www.ros.org/reps/rep-0122.html>

- ❑ Файловая система ROS состоит из двух компонентов (директория)
 - ❑ Директория установки (обычно `/opt/ros/<дистрибутив>`)
 - ❑ Пользовательская рабочая директория (*workspace*)



ДИРЕКТОРИЯ УСТАНОВКИ ROS

- Обычно /opt/ros/<дистрибутив>
 - /bin — исполняемые бинарные файлы
 - /etc — файлы конфигурации ROS и Catkin
 - /include — файлы заголовков (header files)
 - /lib — файлы библиотек
 - /share — ROS пакеты
 - setup.* — скрипты для конфигурации среды

```
shipitko@devel-Latitude-5491: ~  
shipitko@devel-Latitude-5491: ~ 49x26  
→ ~ tree -L 1 /opt/ros/kinetic  
/opt/ros/kinetic  
├── bin  
├── env.sh  
├── etc  
├── include  
├── lib  
├── local_setup.bash  
├── local_setup.sh  
├── local_setup.zsh  
├── setup.bash  
├── setup.sh  
├── _setup_util.py  
├── setup.zsh  
└── share  
  
5 directories, 8 files  
→ ~
```

ПОЛЬЗОВАТЕЛЬСКАЯ РАБОЧАЯ ДИРЕКТОРИЯ

<http://wiki.ros.org/catkin/workspaces>

<https://www.ros.org/reps/rep-0128.html>

user_catkin_workspace_folder/
src/

CMakeLists.txt

package_1/

CMakeLists.txt

package.xml

...

package_n/

CMakeLists.txt

package.xml

build/

devel/

install/

– РАБОЧАЯ ДИРЕКТОРИЯ

– ДИРЕКТОРИЯ С ИСХОДНЫМИ ФАЙЛАМИ

– ФАЙЛ СМАКЕ ВЕРХНЕГО УРОВНЯ

– ФАЙЛ СМАКЕ ПАКЕТА package_1

– ФАЙЛ-МАНИФЕСТ ПАКЕТА package_1

– ФАЙЛ СМАКЕ ПАКЕТА package_n

– ФАЙЛ-МАНИФЕСТ ПАКЕТА package_n

– РАЗЛИЧНЫЕ ФАЙЛЫ СБОРКИ, КЭШ

– СКОМПИЛИРОВАННЫЕ ФАЙЛЫ ЗАПУСКА, HEADER ФАЙЛЫ,
ПОЛЬЗОВАТЕЛЬСКИЕ БИБЛИОТЕКИ, MSG И SRV ФАЙЛЫ

– ДИРЕКТОРИЯ ДЛЯ УСТАНОВКИ ПАКЕТОВ ПОСЛЕ СБОРКИ

ИНИЦИАЛИЗИРУЕМ РАБОЧУЮ ДИРЕКТОРИЮ

<http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment>

- ❑ Для того, чтобы автоматически создать новую рабочую директорию необходимо выполнить набор команд:

```
mkdir -p ~/my_ros_ws/src  
cd ~/my_ros_ws  
catkin_make
```

Имя рабочей директории может
быть любым.
Стандартный выбор — **catkin_ws**

Создать родительские
директории, если они
не существуют

- ❑ После инициализации рабочей директории (и перед каждым запуском ROS в новом терминале) необходимо выполнять команду:

```
source ~/my_ros_ws/devel/setup.bash
```

Она конфигурирует переменные окружения и добавляет рабочую директорию в **\$ROS_PACKAGE_PATH**

- ❑ Чтобы убедиться, что ROS ищет пакеты в нашей рабочей директории, можно выполнить команду:

```
echo $ROS_PACKAGE_PATH
```

В выводе должна содержаться созданная нами директория **~/my_ros_ws/src**

СОЗДАЕМ СВОЙ ПЕРВЫЙ ПАКЕТ

<http://wiki.ros.org/ROS/Tutorials/CreatingPackage>

- ❑ Для создания пользовательского ROS пакета необходимо перейти в директорию с исходными файлами, рабочей директории:

```
cd ~/my_ros_ws/src
```

- ❑ Для автоматического создания пакет и указания зависимостей воспользоваться инструментом **catkin_create_pkg**:

```
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

```
catkin_create_pkg test_package rospy std_msgs
```

- ❑ Пакет также можно создать вручную. Для этого необходимо создать директорию пакета, а также файлы **CMakeLists.txt** и **package.xml**

СТРУКТУРА ПАКЕТА

- ❑ Обычно .../<catkin workspace>/src/<package name>
 - ❑ `/include` — заголовочные (`.h`, `.hpp`) файлы пакета
 - ❑ `/node` (`/scripts`) — python скрипты
 - ❑ `/launch` — файлы запуска (`.launch`), используемые с `roslaunch`
 - ❑ `/msg` — файлы описания сообщений (`.msg`)
 - ❑ `/src` — исходные файлы
 - ❑ `/srv` — файлы описания сервисов (`.srv`)
 - ❑ `/action` — файлы описания действий (`.action`)
 - ❑ `CMakeLists.txt` — файл конфигурации сборки
 - ❑ `package.xml` — файл описания пакета (манифест)
 - ❑ (опционально) `setup.py` — файл описания установки python-модулей

PACKAGE.XML

<http://wiki.ros.org/catkin/package.xml>

<https://www.ros.org/repos/rep-0140.html>

- ❑ Файл-манифест (package.xml) определяет необходимую информацию о пакете: имя, версию, авторов, кто поддерживает пакет в текущий момент, зависимости пакета от других пакетов.

- ❑ Описание пакетов на сайте <http://wiki.ros.org/> сгенерированно из из файлов-манифестов.

Минимальный пример package.xml

```
<package format="2">
  <name>foo_core</name>
  <version>1.2.4</version>
  <description>
    This package provides foo
    capability.
  </description>
  <maintainer
    email="ivana@osrf.org">Ivana
    Bildbotz</maintainer>
  <license>BSD</license>
  <buildtool_depend>catkin</buildtool_depend>
</package>
```

CATKIN

http://wiki.ros.org/catkin/conceptual_overview

- ❑ **catkin** — система автоматизации сборки ПО из исходного кода, созданная для ROS. Позволяет генерировать “цели” сборки (библиотеки, исполняемые файлы, генерируемые скрипты и т.д.) из исходного кода.
- ❑ **catkin** по своей сути является набором CMake-макросов и python-скриптов и расширяет возможности CMake.



CMAKELISTS.TXT

<http://wiki.ros.org/catkin/CMakeLists.txt>

- ❑ **Важно!** Порядок инструкций в CMakeLists.txt имеет значение.
- ❑ Конкретный набор инструкций в CMakeLists.txt может меняться в зависимости от содержимого пакета, однако он должен следовать определенному шаблону:
 1. Необходимая версия CMake (`cmake_minimum_required()`)
 2. Имя пакета (`project()`)
 3. Поиск других CMake/Catkin пакетов, необходимых для сборки (`find_package()`)
 4. Включение импорта python-модулей (`catkin_python_setup()`)
 5. Описание файлов message/service/action для генерации (`add_message_files()`, `add_service_files()`, `add_action_files()`)
 6. Запуск генерации message/service/action (`generate_messages()`)
 7. Specify package build info export (`catkin_package()`)
 8. Сборка библиотек и исполняемых файлов (`add_library()/add_executable()/target_link_libraries()`)
 9. Сборка тестов (`catkin_add_gtest()`)
 10. Правила установки пакетов (`install()`)

CMAKELISTS.TXT

<http://wiki.ros.org/catkin/CMakeLists.txt>

Минимальная необходимая версия CMake

Имя проекта (пакета). Должно совпадать с именем, указанным в package.xml. После объявления хранится в переменной
`${PROJECT_NAME}`

Поиск и подключение пакетов, необходимых для сборки нашего пакета. Для всех ROS пакетов необходимой зависимостью является catkin.

Поиск и подключение ROS-независимых библиотек

```
cmake_minimum_required (VERSION 2.8.3)
project (my_first_ros_pkg)
find_package (catkin REQUIRED COMPONENTS
    roscpp
    std_msgs
)
find_package (Boost REQUIRED COMPONENTS
system)
catkin_python_setup ()
add_message_files (
    FILES
    Message1.msg
    Message2.msg
)
add_service_files (
    FILES
    Service1.srv
    Service2.srv
)
generate_messages (
    DEPENDENCIES
    std_msgs
)
catkin_package (
    INCLUDE_DIRS include
    LIBRARIES my_first_ros_pkg
    CATKIN_DEPENDS roscpp std_msgs message_runtime
    DEPENDS system_lib
)
```

FIND_PACKAGE()

<http://wiki.ros.org/catkin/CMakeLists.txt>

- ❑ Нахождения пакета CMake через команду `find_package()` приводит к созданию нескольких переменных окружения CMake, которые могут быть использованы в последующих командах CMake файла.
- ❑ Имена этих переменных всегда следуют определенному формату `<PACKAGE_NAME>_<PROPERTY>` :
 - ❑ `<NAME>_FOUND` — устанавливается в `True`, если пакет найден
 - ❑ `<NAME>_INCLUDE_DIRS` или `<NAME>_INCLUDES` — путь до include директории пакета
 - ❑ `<NAME>_LIBRARIES` или `<NAME>_LIBS` — библиотеки, экспортируемые пакетом
- ❑ Почему ROS пакеты добавляются в проект как **компоненты** `catkin`? Это сделано для удобства т.к. в таком случае все найденные пакеты добавляются в переменные окружения, связанные с `catkin` (например, `catkin_INCLUDE_DIRS`)

CMAKELISTS.TXT

<http://wiki.ros.org/catkin/CMakeLists.txt>

Команда необходима, если модуль предоставляет python модули для импорта другими пакетами. При этом предварительно нужно создать файл **setup.py**. Должна быть вызвана **ДО** команд `generate_messages()` и `catkin_package()`.

Добавляем пользовательские .msg, .srv, .action файлы для генерации их программных файлов-описаний для разных языков программирования. И запускаем генерацию.

Необходимо вызывать **ДО** команды `catkin_package()`.

Передаёт catkin-специфичную информацию системе сборки (CMake). Должна быть вызвана **ДО** задания целей сборки: `add_library()` or `add_executable()`.

```
cmake_minimum_required(VERSION 2.8.3)
project(my_first_ros_pkg)
find_package(catkin REQUIRED COMPONENTS
    roscpp
    std_msgs
)
find_package(Boost REQUIRED COMPONENTS
system)
catkin_python_setup()
add_message_files(
    FILES
    Message1.msg
    Message2.msg
)
add_service_files(
    FILES
    Service1.srv
    Service2.srv
)
generate_messages(
    DEPENDENCIES
    std_msgs
)
catkin_package(
    INCLUDE_DIRS include
    LIBRARIES my_first_ros_pkg
    CATKIN_DEPENDS roscpp std_msgs message_runtime
    DEPENDS system_lib
)
```

SETUP.PY

http://docs.ros.org/api/catkin/html/user_guide/setup_dot_py.htm

- ❑ `setup.py` необходимо использовать, если в ROS пакете есть модули и скрипты, которые необходимо устанавливать в систему, например, для использования в других пакетах. В python для этих целей используются библиотеки `distutils` и `setuputils`.
- ❑ Если в `CMakeLists.txt` указана команда `catkin_python_setup()` `catkin` выполнит скрипт `setup.py`, лежащий в корне проекта. При этом `setup.py` сможет переиспользовать информацию из `CMakeLists.txt`.
- ❑ Функция `generate_distutils_setup()` позволяет переиспользовать информацию из `package.xml`.

```
from setuptools import setup
from catkin_pkg.python_setup
import generate_distutils_setup

d = generate_distutils_setup(
    packages=['mypkg'],
    scripts=['bin/myscript'],
    package_dir={'': 'src'})

setup(**d)
```



CMAKELISTS.TXT

<http://wiki.ros.org/catkin/CMakeLists.txt>

Создание целей сборки, указания зависимостей для правильного порядка генерации сообщений/сервисов и целей сборки, указание зависимости цели сборки от других библиотек.

(Опционально) Добавление модульных тестов.

(Опционально) Установка собранных пакетов и исполняемых python-скриптов.



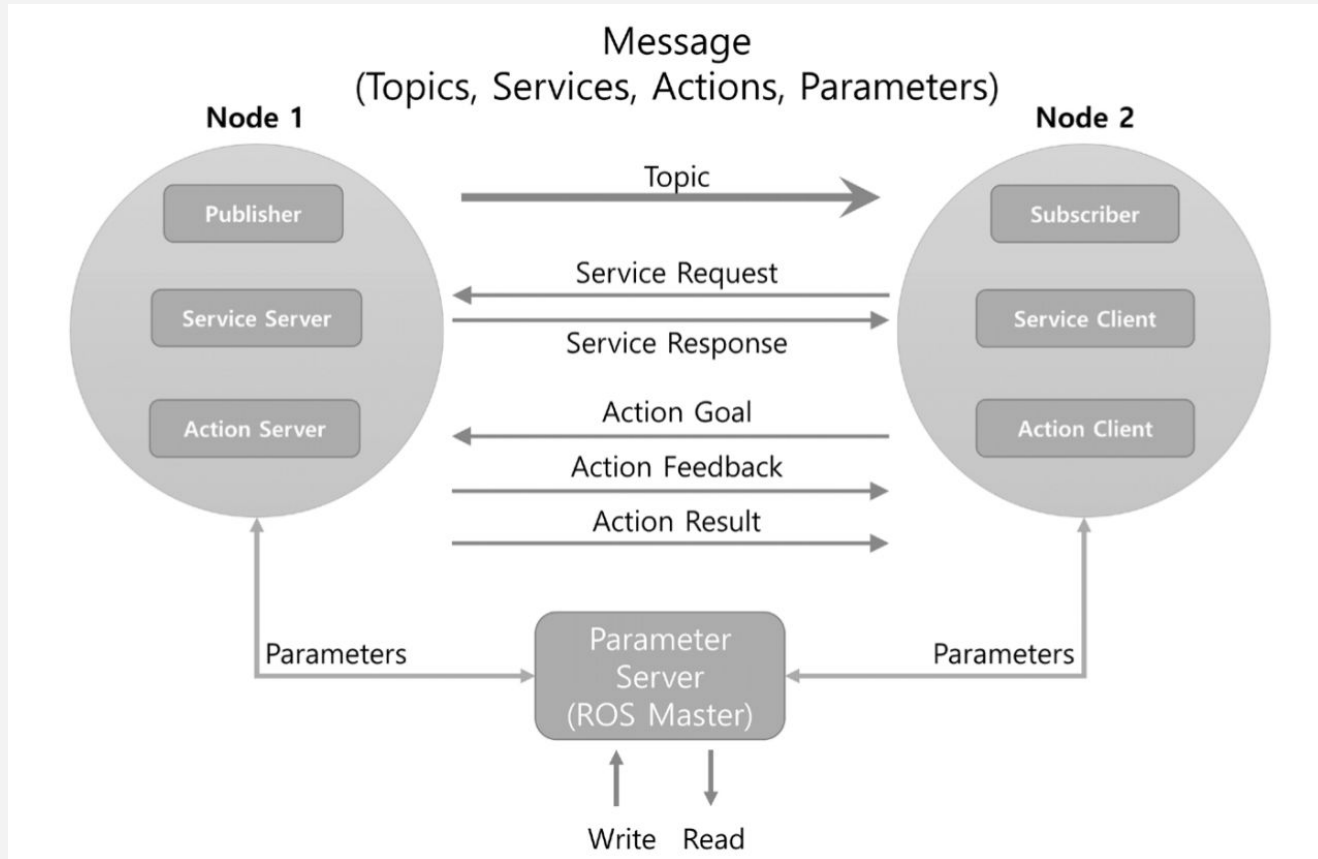
```
add_executable(my_first_ros_pkg_node src/main.cpp)
add_dependencies(my_first_ros_pkg_node
    ${${PROJECT_NAME}_EXPORTED_TARGETS}
    ${catkin_EXPORTED_TARGETS}
)
target_link_libraries(my_first_ros_pkg_node
    ${catkin_LIBRARIES}
)

if(CATKIN_ENABLE_TESTING)
    catkin_add_gtest(myUnitTest test/utest.cpp)
endif()

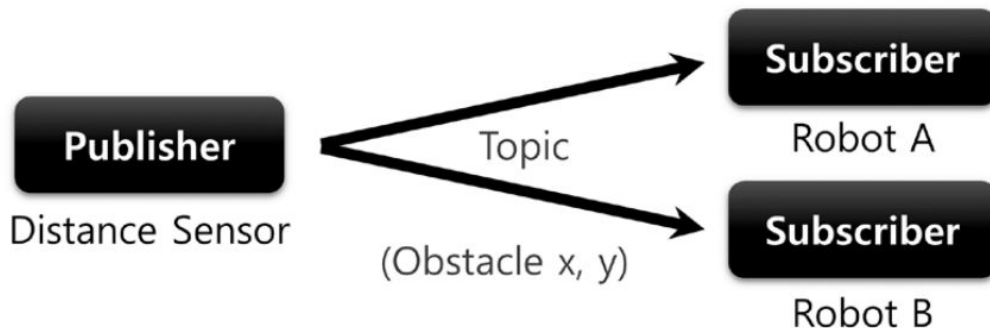
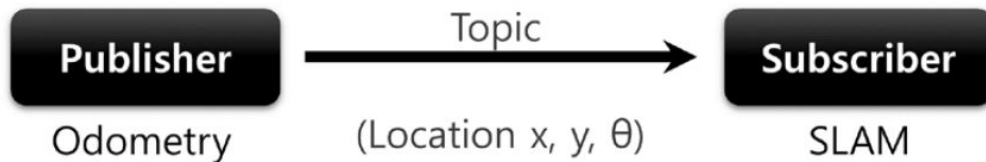
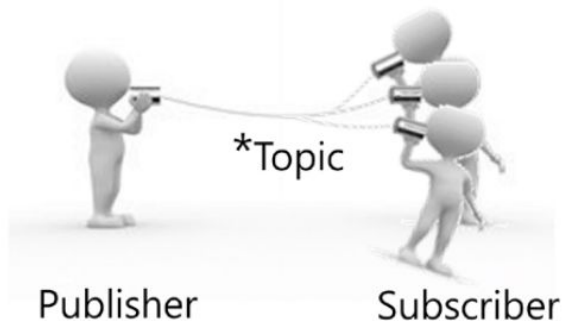
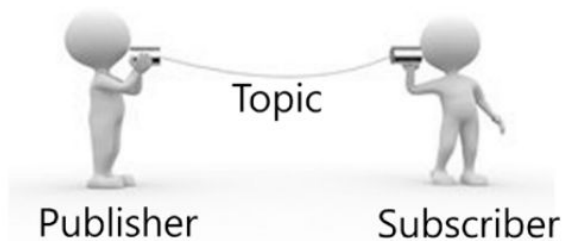
install(TARGETS ${PROJECT_NAME}
    ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
    LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
    RUNTIME DESTINATION ${CATKIN_GLOBAL_BIN_DESTINATION}
)

catkin_install_python(PROGRAMS scripts/myscript
    DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

ТИПЫ КОММУНИКАЦИИ В ROS

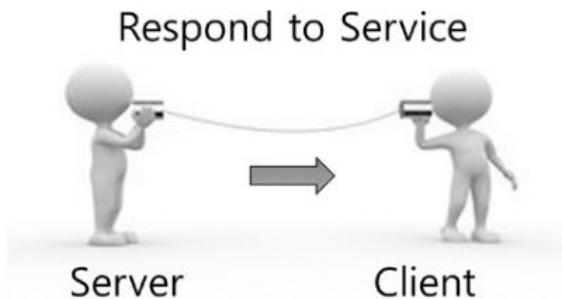
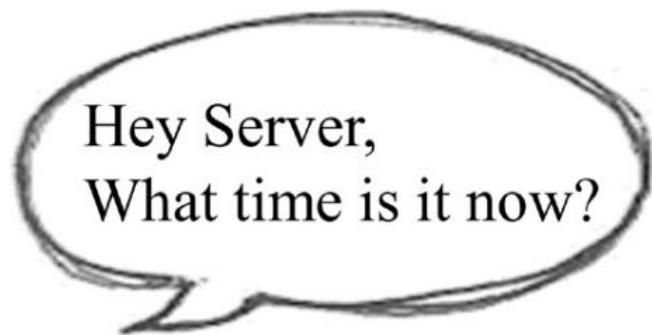
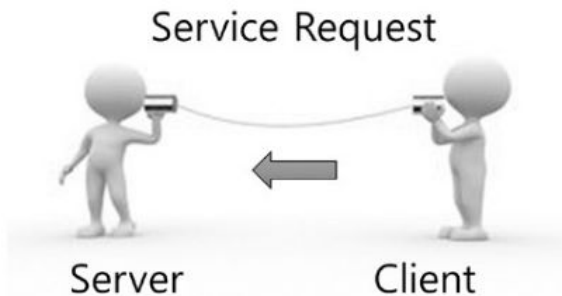
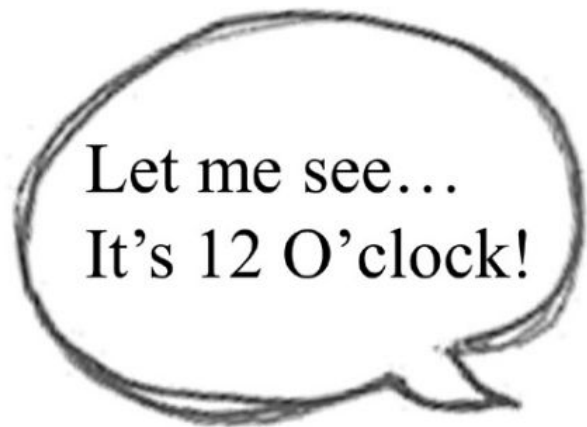


ТОПИКИ (TOPICS)

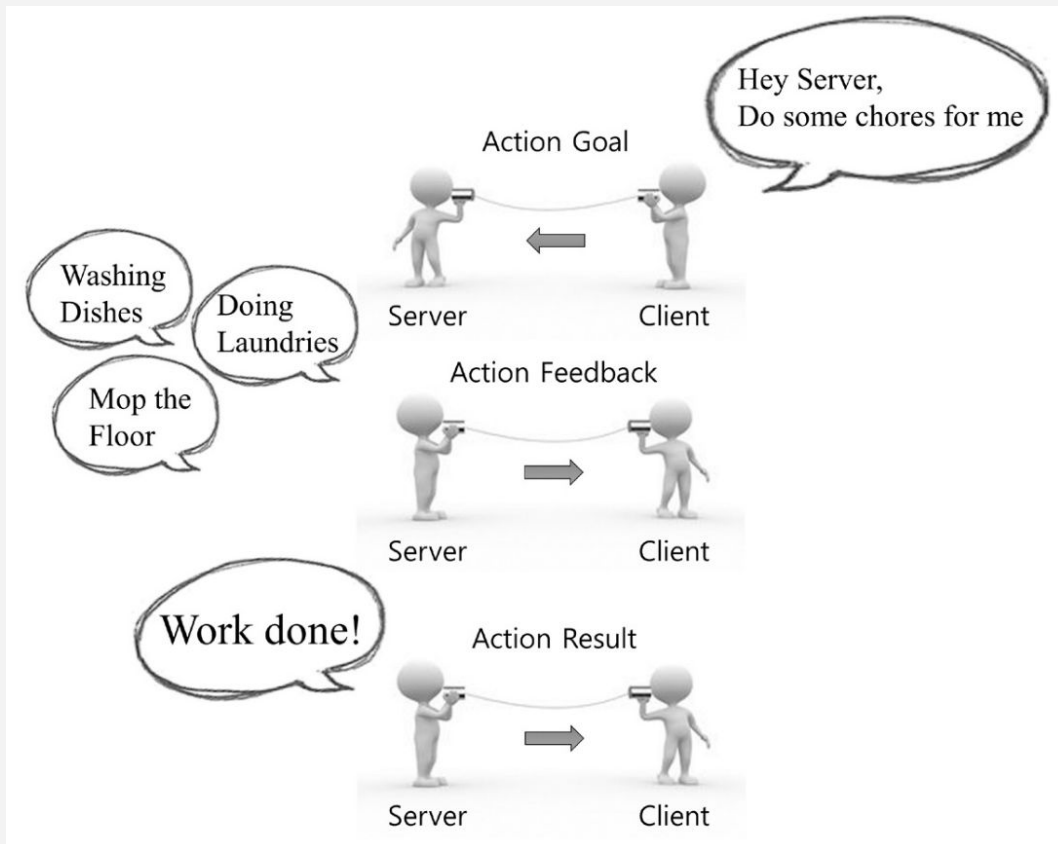


- ❑ Механизм топиков позволяет организовывать не только 1 – 1 коммуникацию, но и N – N

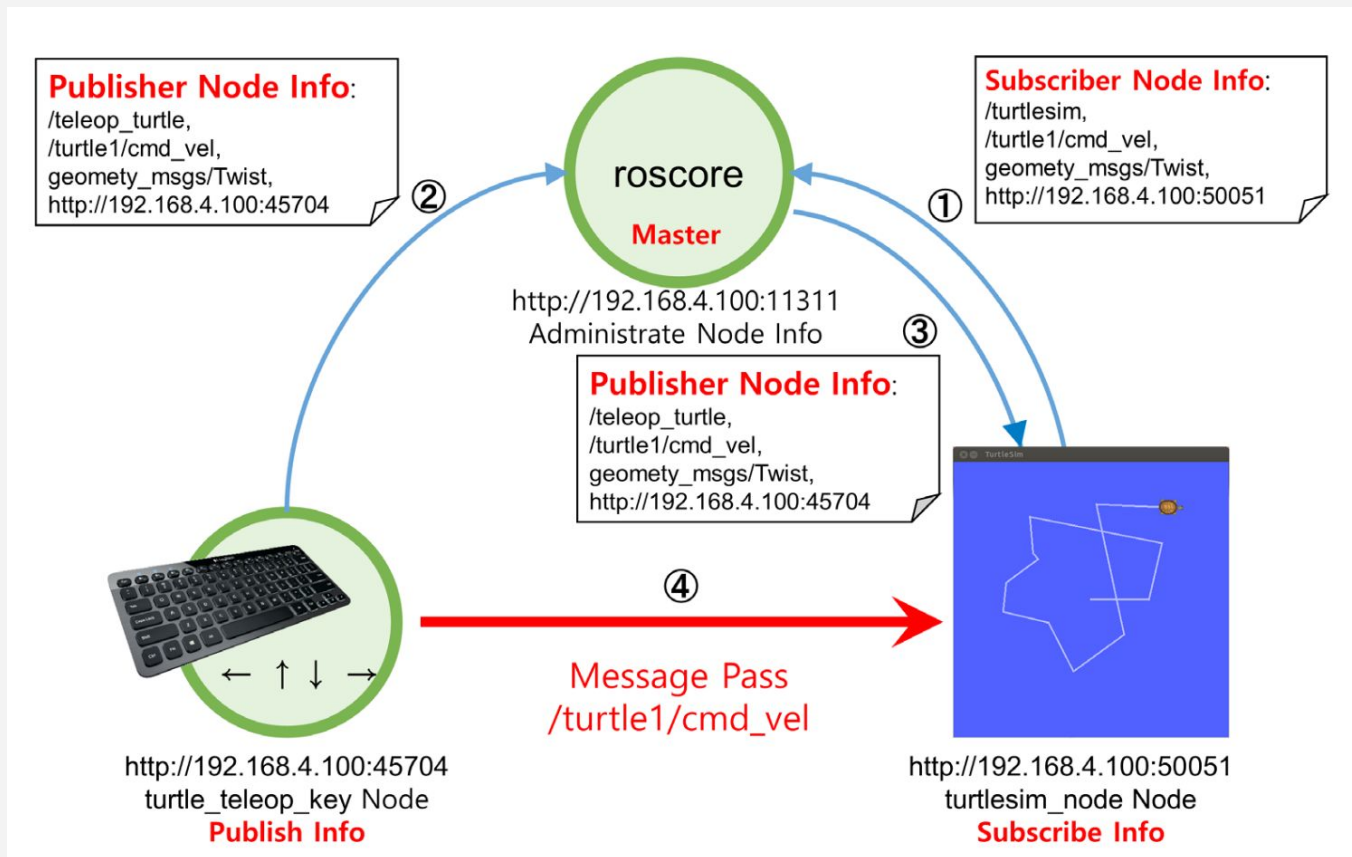
СЕРВИСЫ (SERVICES)



ДЕЙСТВИЯ (ACTIONS)



КОММУНИКАЦИЯ В ROS



ТИПЫ КОММУНИКАЦИИ В ROS

Тип коммуникации	Особенности	Когда применяется?
Топик	Асинхронная однонаправленная передача	Используется для непрерывной передачи данных
Сервис	Синхронная двунаправленная передача	Используется для коммуникации типа запрос-ответ с быстрой обработкой запроса
Действие	Асинхронная двунаправленная передача	Используется, когда механизм сервисов неприменим, из-за долгого процесса обработки запроса или когда необходима обратная связь в процессе обработки запроса

СООБЩЕНИЯ (MESSAGES)

<http://wiki.ros.org/msg>

- ❑ В ROS используется простой язык описания сообщений. Из описаний сообщений автоматически генерируются программные описания сообщений для нескольких целевых языков (python, C++, lisp)
- ❑ Пользовательские описываются в файлах .msg, которые обычно хранятся в директории пакета /msg
- ❑ Сообщения могут содержать 2 части:
 - ❑ **Поля данных** (обязательная часть) — описание информационных полей = тип + имя
 - ❑ **Константы** — вспомогательные константы для интерпретации полей данных (например, как enum в C++)

СООБЩЕНИЯ (MESSAGES)

<http://wiki.ros.org/msg>

- ❑ **Тип данных** — может быть встроенным типом (например, `float64`), типом уже существующего сообщения (`geometry_msgs/Quaternion`), массивом фиксированной или динамической длины (`float64[]` или `float64[9]` `orientation_covariance`), специальным типом `Header` (раскрывается в `std_msgs/Header`)
- ❑ **Константы** — могут иметь только встроенные типы (кроме `time` и `duration`)

```
sensor_msgs/Imu
```

```
Header header
geometry_msgs/Quaternion orientation
float64[9] orientation_covariance
# Row major about x, y, z axes

geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance
# Row major about x, y, z axes

geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance
# Row major x, y, z
```

```
# Constants example
int32 X=123
string FOO=foo
```

ГЕНЕРАЦИЯ СООБЩЕНИЙ, ЧАСТЫЕ ПРОБЛЕМЫ

<http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv>

❑ Если вы добавляете генерацию сообщений в свой пакет:

❑ Не забудьте обновить зависимости в package.xml

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

❑ Добавьте message_generation В СПИСОК необходимых пакетов

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)
```

❑ Также добавьте ЗАВИСИМОСТЬ ОТ message_runtime

```
catkin_package(
  ...
  CATKIN_DEPENDS message_runtime ...
  ...
)
```

❑ Добавьте файлы описания сообщений

```
add_message_files(
  FILES
  Num.msg
)
```

❑ Добавьте команду для генерации сообщений и файлов-описания

```
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

ROSPY API

<http://wiki.ros.org/rospy>

- ❑ Импорт клиентской библиотеки ROS для языка python

```
import rospy
```

- ❑ Импорт сообщения типа Float32 из пакет std_msgs. **Внимание!** При импорте сообщений из пакетов, не забывайте добавлять .msg к имени пакета

```
from std_msgs.msg import Float32  
from <package>.msg import <Message>
```

- ❑ Регистрация подписки на топик с указанием его имени, типа передаваемого сообщения и функции обработки (callback)

```
rospy.Subscriber("signal", Float32, signal_callback)  
rospy.Subscriber(name, data_class, callback=None, callback_args=None, queue_size=None,  
buff_size=65536, tcp_nodelay=False)
```

ROSPY API

<http://wiki.ros.org/rospy>

- ❑ Регистрация публикации топика с указанием его имени, типа передаваемого сообщения, размера очереди сообщений

```
rospy.Publisher("filtered_signal", Float32, queue_size=10)
rospy.Publisher(name, data_class, subscriber_listener=None, tcp_nodelay=False,
latch=False, headers=None, queue_size=None)
```

- ❑ Логирование сообщений. Можно использовать разные уровни логирования: `.logdebug`, `.logwarn`, `.logerr`, `.logfatal`

```
rospy.loginfo("I've got {}".format(signal.data))
```

- ❑ Инициализация ноды с заданным именем

```
rospy.init_node("signal_filter")
rospy.init_node(name, argv=None, anonymous=False, log_level=2,
disable_rostime=False, disable_rosout=False, disable_signals=False)
```

СОХРАНЕНИЕ ИЗМЕНЕНИЙ В КОНТЕЙНЕРЕ

<https://docs.docker.com/engine/reference/commandline/commit/>

<https://docs.docker.com/storage/volumes/>

❑ Существует несколько способов сохранить модифицированные данные в контейнере:

❑ Использовать:

```
docker commit <container-id> USER_NAME/IMAGE_NAME
```

чтобы создать новую версию образа с сохраненными изменениями

❑ Использовать:

```
docker cp CONTAINER:SRC_PATH DEST_PATH
```

чтобы скопировать данные из контейнера на машину-хост

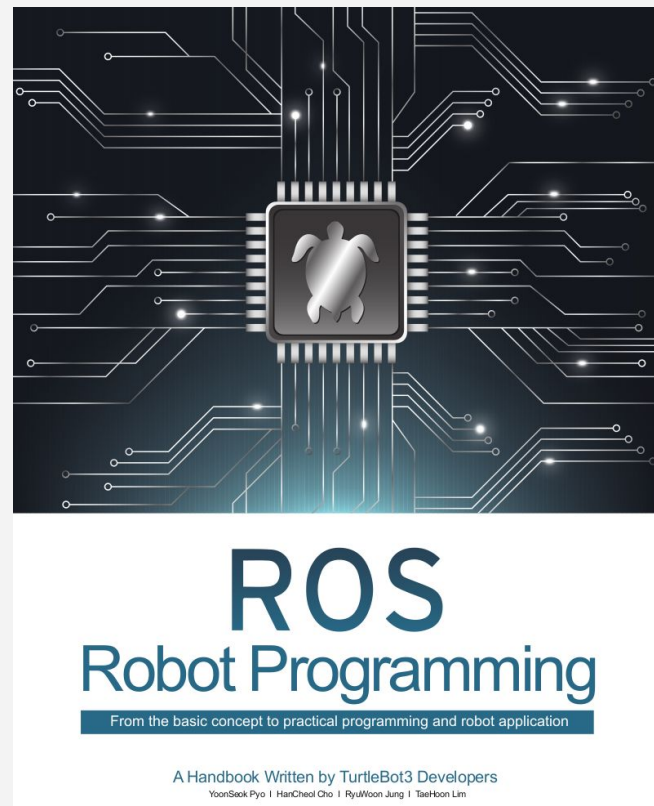
❑ Использовать:

```
sudo docker run -v [-- volume] HOST_FOLDER:CONTAINER_VOLUME_NAME
```

чтобы сделать директорию хост-машины доступной из контейнера (или создать том данных для docker-контейнера)

ДОПОЛНИТЕЛЬНЫЕ ИСТОЧНИКИ

1. Книга: ROS Robot Programming.
YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim (Eng)
2. Обучающие инструкции ROS (Eng)
3. Введение в ROS от Voltbro (Rus)
4. Clearpath Robotics ROS Tutorial (Eng)



ИНФОРМАЦИЯ О ПРЕЗЕНТАЦИИ

Эта презентация была подготовлена Олегом Шипитько в рамках курса “Моделирование колесных роботов” кафедры когнитивных технологий Московского физико-технического института (МФТИ). Автор выражает благодарность, авторам, чьи материалы были использованы в презентации. В случае, если вы обнаружили в презентации свои материалы, свяжитесь со мной, для включения в список авторов заимствованных материалов.

This presentation was prepared by Oleg Shipitko as part of the “Mobile Robotics” course at the Department of Cognitive Technologies, Moscow Institute of Physics and Technology. The author is grateful to the authors whose materials were used in the presentation. If you find your materials in a presentation, contact me to be included in the list of contributing authors.