RRM

ROS komunikácia
Service Client

Marek Čornák
marek.cornak@stuba.sk

Michal Dobiš
michal.dobis@stuba.sk

Jakub Ivan
jakub.ivan@stuba.sk

Obsah

1. Úvod do C++ - std::vector a operátor &
2. ROS Service
3. ROS Service Client implementácia
4. Samostatná úloha

#include <vector>

Inicializácia cez list

```
std::vector<double> vec = {1.0, 2.0};
```

Priradovaním po prvkoch

```
std::vector<double> vec;
vec.resize(2);
vec[0] = 1.0;
vec[1] = 2.0;
```

Vkladaním (vhodné vopred rezervovať pamäť)

```
std::vector<double> vec;
vec.reserve(2);
vec.push_back(1.0);
vec.push_back(2.0);
```

Priradovaním po prvkoch

```
std::vector<double> vec(2);
vec[0] = 1.0;
vec[1] = 2.0;
```

#include <vector>

Prístup cez operátor []

```
std::vector<double> vec(2);
auto x = vec[0];
auto y = vec[1];
auto z = vec[2];
```

Prístup cez metódu at() - bezpečnejšie

```
std::vector<double> vec(2);
auto x = vec.at(0);
auto y = vec.at(1);
auto z = vec.at(2);
```

Prístup cez cyklus

```
for (const auto& v : vec) {
    std::cout << v << std::endl;
}
```

#include <vector>

Prístup cez operátor []

```
std::vector<double> vec(2);
auto x = vec[0];
auto y = vec[1];
auto z = vec[2];
```

Prístup cez metódu at() - bezpečnejšie

```
std::vector<double> vec(2);
auto x = vec.at(0);
auto y = vec.at(1);
auto z = vec.at(2);// hodí výnimku (exception)
```

Prístup cez cyklus

```
for (const auto& v : vec) {
    std::cout << v << std::endl;
}
```

auto - automatická deklarácia objektu

& - referencia

Operátor &

Prístup k adrese

```
int x = 10;
int* ptr = &x; // ptr teraz obsahuje adresu premennej
```

Referencia

```
int y = 20;
int& ref = y;  // ref je referencia na y
ref = 30;  // Mení hodnotu y na 30
```

Využitie referencie vo funkciách

```
void func(std::vector<double> vec);
```

```
void func(std::vector<double>& vec);
```

```
void func(const std::vector<double>&
vec);
```

Operátor &

Prístup k adrese

```
int x = 10;
int* ptr = &x; // ptr teraz obsahuje adresu premennej x
```

Referencia

```
int y = 20;
int& ref = y;  // ref je referencia na y
ref = 30;  // Mení hodnotu y na 30
```

Využitie vo funkciách

```
void func(std::vector<double> vec);
```

Nevhodné!!!

Kopíruje pamäť, zbytočné

```
void func(std::vector<double>& vec);
```

Nevhodné!

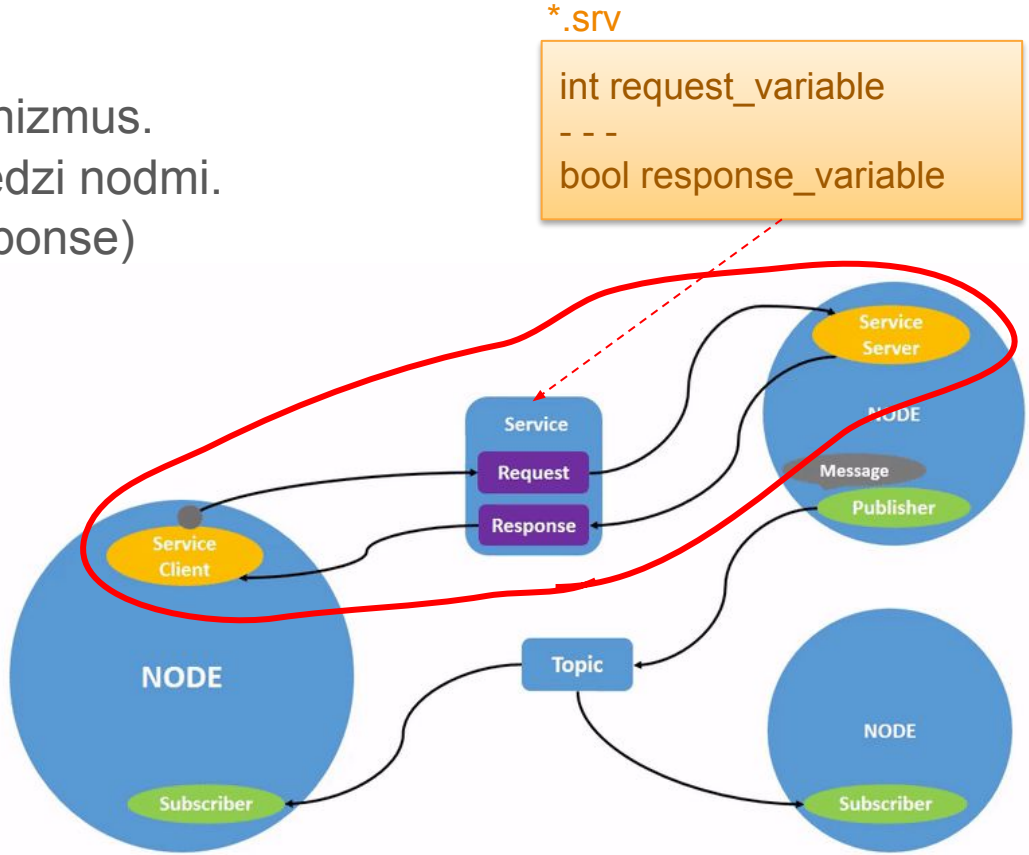
Vhodnejšie, ale funkcia môže zmeniť vstup
Využíva sa ako vstupno výstupný argument

```
void func(const std::vector<double>& vec);
```

Užívateľ funkcie má zaručené, že sa mu argument nezmení a zároveň nemusí kopírovať pamäť

ROS Service

- Synchronný komunikačný mechanizmus.
- Dvojcestná výmena informácií medzi nodmi.
- Model client - server (request-response)
- Client
  - Iniciuje komunikáciu (request)
  - Čaká na odpoveď (response)
- Server
  - Spracováva požiadavky (request)
  - Posiela odpoveď Clientovi
- Služba (Service)
  - Definuje štruktúru komunikácie
  - štruktúra daná .srv súborom



ROS Service - balík rrm_sim

- Balík rrm_sim ponúka servis /move_command:

```
marek@marek-ASUS-TUF-Gaming-A15-FA507NV-FA507NV:~/xcornak_rrm$ ros2 service list
/move_command
/robot_state_publisher/describe_parameters
/robot_state_publisher/get_parameter_types
/robot_state_publisher/get_parameters
```

- Servis /move_command má nasledovný interface:

```
marek@marek-ASUS-TUF-Gaming-A15-FA507NV-FA507NV:~/xcornak_rrm$ ros2 service info /move_command
Type: rrm_msgs/srv/Command
Clients count: 0
Services count: 1
marek@marek-ASUS-TUF-Gaming-A15-FA507NV-FA507NV:~/xcornak_rrm$ ros2 interface show rrm_msgs/srv/Command
float64[] positions
float64[] velocities
---
int32 result_code
string message
```

Nezabudnite si balík rrm_sim spustiť (cvičenie 2) **ros2 launch rrm_sim rrm_sim.launch.xml**

ROS Service - balík rrm_sim

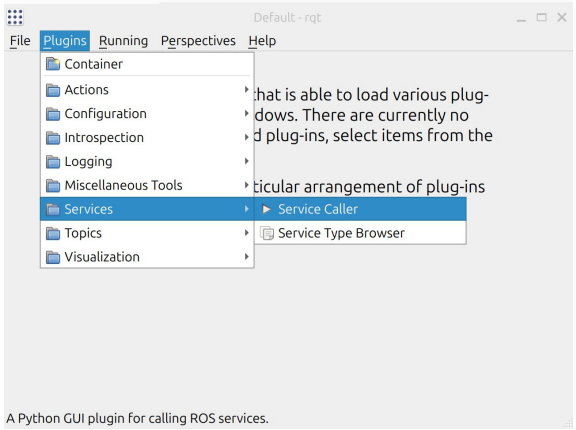
- Zavolanie servisu /move_command:

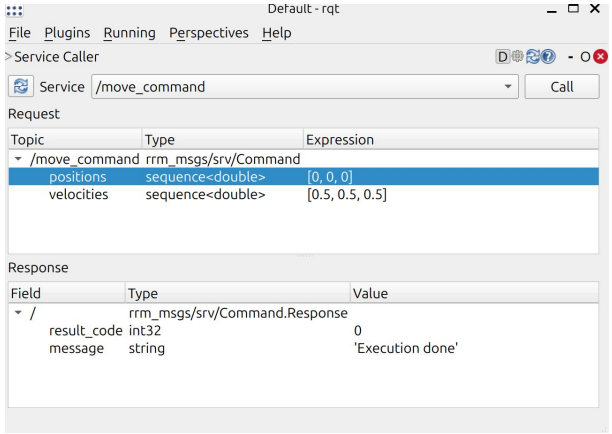
ros2 service call /move_command rrm_msgs/srv/Command "{positions: [1.0, 0.5, -0.3], velocities: [0.1, 0.2, 0.15]}"

```
marek@marek-ASUS-TUF-Gaming-A15-FA507NV-FA507NV:~/xcornak_rrm$ ros2 service call /move_command rrm_msgs/srv/Command
"{positions: [1.0, 0.5, -0.3], velocities: [0.1, 0.2, 0.15]}"
requester: making request: rrm_msgs.srv.Command_Request(positions=[1.0, 0.5, -0.3], velocities=[0.1, 0.2, 0.15])

response:
rrm_msgs.srv.Command_Response(result_code=0, message='Execution done')
```

Zavolanie servisu cez **rqt**:





ROS Client - implementácia

- Pridanie header file pre srv interface

```
#include "rrm_msgs/srv/command.hpp"    //header file pre msg interface
```

- Vytvorenie objektu klient v triede Teleop:

```
private:
      rclcpp::Client<rrm_msgs::srv::Command>::SharedPtr client_;
```

- Inicializácia klienta v konštruktore triedy Teleop:

```
client_ = this->create_client<rrm_msgs::srv::Command>("move_command");
```

- Čakanie na servis server ak nie je dostupný

```
while (!client_->wait_for_service(std::chrono::seconds(1))) {
  if (!rclcpp::ok()) {
    RCLCPP_ERROR(this->get_logger(), "Interrupted while waiting for the service. Exiting." );
    return;
  }
  RCLCPP_INFO(this->get_logger(), "Service not available, waiting again..." );
}
```

ROS Client - implementácia

- Použite klienta vo vašej funkcii

```
//Deklaracia premennej pointer na poziadavku/request
//std::shared_ptr<rrm_msgs::srv::Command::Request> request = std::make_shared<rrm_msgs::srv::Command::Request>();
 auto request = std::make_shared<rrm_msgs::srv::Command::Request>();
 request->positions = {0.5, 1.0, 1.5};
 request->velocities = {0.5, 0.5, 0.5};
```

- Odoslanie požiadavky - request

```
 //std::shared_future<std::shared_ptr<rrm_msgs::srv::Command::Response>> result = client_->async_send_request(request);
   auto result = client_->async_send_request(request);
```

- Prijatie a vypísanie odpovede - response

```
if (rclcpp::spin_until_future_complete(this->get_node_base_interface(), result) !=
rclcpp::FutureReturnCode::SUCCESS) {
     RCLCPP_ERROR(this->get_logger(), "Service call failed"); // Log an error if the service call failed
      return false;
 }
//std::shared_ptr<rrm_msgs::srv::Command::Response> response = result.get(); // Get the response from the service
auto response = result.get();
RCLCPP_INFO(this->get_logger(), "Service call succeeded: result_code = %d, message = %s",
               response->result_code, response->message.c_str()); // Log the response
return true;
```

ROS Client - implementácia

- Funkcia spin_until_future_complete - blokuje do nekonečna.

```
rclcpp::spin_until_future_complete(this->get_node_base_interface(), result);
```

- Vhodnejšie nastaviť timeout, pre prípad, že service server spadol počas behu.

```
rclcpp::spin_until_future_complete(this->get_node_base_interface(), result,  std::chrono::seconds(5));
```