RRM

ROS komunikácia
Service Server

Marek Čornák
marek.cornak@stuba.sk

Michal Dobiš
michal.dobis@stuba.sk

Jakub Ivan
jakub.ivan@stuba.sk

Obsah

1. Úvod do C++ konštruktor/deštruktor
2. Vytvorenie vlastného interface msg/srv správ
3. Service server implementácia
4. ROS launch
5. Samostatná úloha

Úvod do C++ konštruktor

C++ konštruktor s parametrom

```
class Robot {
public:
  Robot(int number_of_joints) : positions(number_of_joints){};
  Robot() : positions(???) {};
private:
  std::vector<double> positions;
};

Robot robot1(3); // inicializácia s argumentom
Robot robot2;  // Kompilátor to dovolí, ale je to správne?
```

Úvod do C++ konštruktor

Konštruktor bez argumentov sa môže zakázať používať, objekt by nebol validný

```
class Robot {
public:
  Robot(int number_of_joints) : positions(number_of_joints){};
  Robot() = delete;
private:
  std::vector<double> positions;
};

Robot robot1(3); // inicializácia s argumentom
Robot robot2;  // Kompilátor to zakáže
```

Úvod do C++ konštruktor

Explicitné označenie “default” konštruktora

```
class Robot {
public:
  Robot() = default;
private:
  std::vector<double> positions{0.0, 0.0, 0.0};
  std::string name{"robot"};
};

Robot robot; // inicializácia s default
```

1. Inicializácia prebieha zhora dole
2. Až po inicializácií všetkých premenných sa zavolá konštruktor

Úvod do C++ deštruktor

```
class Robot {
public:
  Robot() = default;
 ~Robot() { thread.joint(); };
private:
  std::vector<double> positions{0.0, 0.0, 0.0};
  std::thread thread;
};

Robot robot; // inicializácia s default
```

1. Najprv sa zavolá deštruktor
2. Deštrukcia premenných prebieha zdola hore

Úvod do C++ deštruktor - príklad

```
class TimeMeasure {
public:
  TimeMeasure() { start = std::chrono::system_clock::now(); };
  ~TimeMeasure() {
    end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end - start;
    std::cout << "Elapsed time: << elapsed_seconds.count() << std::endl;
 };
private:
std::chrono::time_point<std::chrono::system_clock> start, end;
};

int main(int argc, char **argv) {
  TimeMeasure time;
  sleep(1);        // Elapsed time ~1 seconds
}
```

ROS - rozhrania / interfaces

Definície správ, služieb a akcií, ktoré umožňujú výmenu údajov v ROS2

- Messages - Správy, komunikácia cez topics
- Services - Správy, komunikácia cez services

ROS - vytvorenie vlastného rozhrania (.srv)

1. Najskôr je potrebné vytvoriť vlastný CMake package:
   ```
   cd <vas_ros2_ws>/src
   ros2 pkg create --build-type ament_cmake <priezvisko>_interface
   ```
2. V balíčku vytvoríme priečinok pre súbory správ:
   ```
   mkdir msg srv
   ```
3. Následne v priečinku **msg** alebo **srv** definujeme náš .msg alebo .srv. Napr. pre zadanie 1.4 budete potrebovať srv:
   ```
   float64 velocity
   ---
   bool result
   string message
   ```

Link na štúdium:
https://docs.ros.org/en/jazzy/Tutorials/Beginner-Client-Libraries/Custom-ROS2-Interfaces.html

ROS - vytvorenie vlastného rozhrania CMakelists.txt

Upravenie CMakeLists.txt aby ROS2 mohol využívať interface:

```
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
 "srv/MyService.srv"
 #DEPENDENCIES geometry_msgs # Ak chcete aby vase interfaces
pouzivali aj ine zavislosti
)
```

ROS - vytvorenie vlastného rozhrania package.xml

ROS interfaces sa spoliehajú na nasledovné ROS závislosti, ktoré je potrebné uviesť v package.xml:

```
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

```
Nezabudnite zbuildovať a source-nut váš ws!
Kontrola, či existuje message

ros2 interface show <priezvisko>_interface/srv/MyService
```

ROS Service Server - vytvorenie

Server môžeme implementovať do triedy JointLogger (jointlogger.cpp):

```
#include "<priezvisko>_interface/srv/my_service.hpp" //header file pre srv interface

private:
//deklaracia premennej typu server s cust. interface
rclcpp::Service<priezvisko_interface::srv::MyService>::SharedPtr service_;

public:
//obslužná funkcia pre server
void my_service_callback(const std::shared_ptr<priezvisko_interface::srv::MyService::Request> request,
                         std::shared_ptr<priezvisko_interface::srv::MyService::Response> response);
```

ROS Service Server - vytvorenie

Definovanie servisu v konštruktore:

```
service_ = this->create_service<priezvisko_interface::srv::MyService>("my_service",
std::bind(&JointLogger::my_service_callback, this, std::placeholders::_1, std::placeholders::_2));
```

Definovanie obslužnej funkcie:

```
void JointLogger::my_service_callback(const std::shared_ptr<priezvisko_interface::srv::MyService::Request>
request, std::shared_ptr<priezvisko_interface::srv::MyService::Response> response)
{
   float velocity = request->velocity;
   //TODO: vasa funkcionalita
   RCLCPP_INFO(this->get_logger(), "Request received: %f, sending response", request->velocity);
   response->result = true;
   response->message = "Response from joint logger";
}
```

ROS Service Server - použitie

```
int main(int argc, char ** argv)
{
   rclcpp::init(argc, argv);
   std::shared_ptr<JointLogger> logger = std::make_shared<JointLogger>();
   rclcpp::spin(logger); //function that blocks the thread and allows the node to process callbacks
   rclcpp::shutdown();
   return 0;
}
```

ROS Service - CMakelists.txt

Nezabudnúť pridať závislosti do CMakelists.txt

```
find_package(priezvisko_interface REQUIRED)
ament_target_dependencies(
 logger_node
 "rclcpp"
 "sensor_msgs"
 "priezvisko_interface"
)
```

A do package.xml

```
<build_depend>priezvisko_interface</build_depend>
```

Link na štúdium: https://docs.ros.org/en/jazzy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Service-And-Client.html#

ROS Service

Spustenie servisu z CLI:

```
ros2 service call /my_service priezvisko_interface/srv/MyService "{velocity: 1.0}"
```