



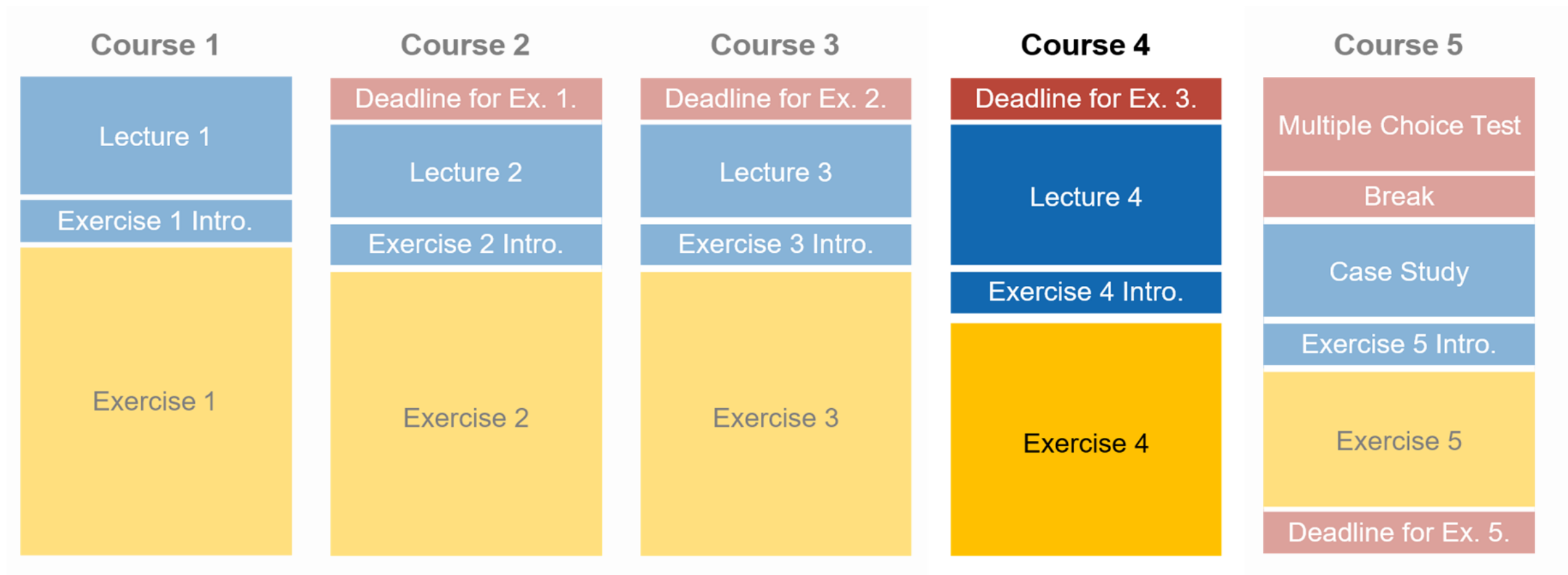
机器人编程

ROS简介—第四节课

广东工业大学DynamicX 机器人队



课程结构



课程目录

- ROS services
- ROS actions (actionlib)
- ROS time
- ROS bags
- Debugging strategies

ROS Services

- 节点之间的请求/响应通信是通过服务实现的
 - 服务器（service server）公布服务
 - 客户端（service client）访问该服务
- 在结构上与消息类似，服务被定义在*.srv文件中
- 用以下方式列出可用的服务

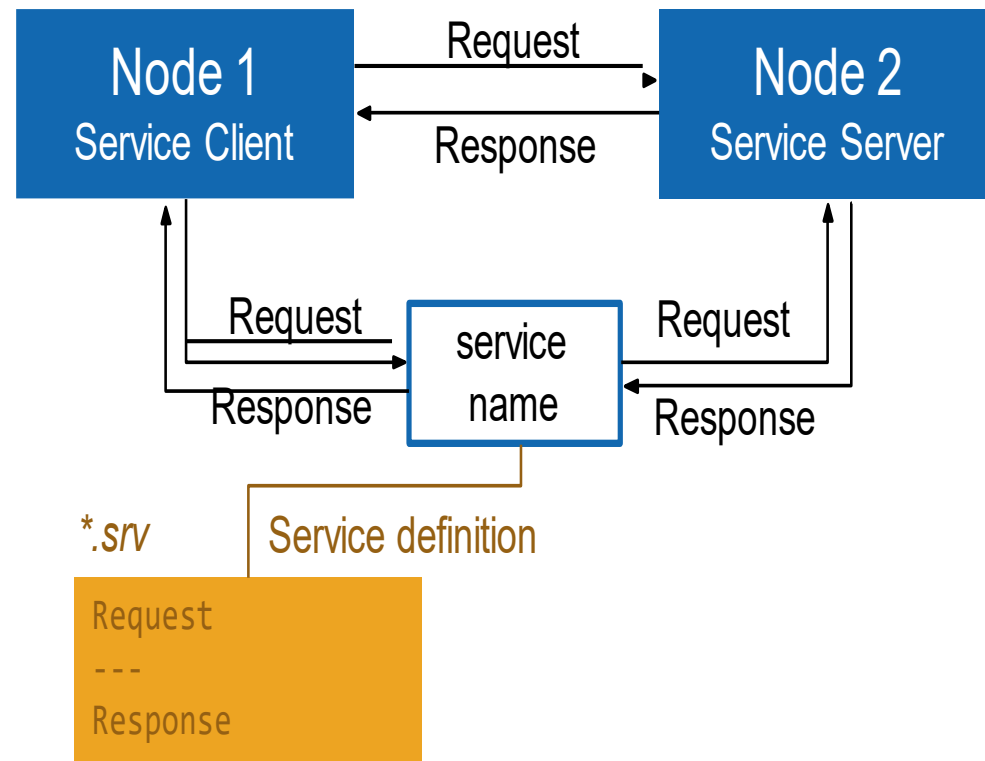
```
> rosservice list
```

显示服务类型

```
> rosservice type /service_name
```

用request contents调用一个服务，用tag自动完成

```
> rosservice call /service_name args
```



More info

<http://wiki.ros.org/tf2>

ROS Services

Examples

[std_srvs/Trigger.srv](#)

```
---  
bool success  
string message
```

Request

Response

[nav_msgs/GetPlan.srv](#)

```
geometry_msgs/PoseStamped start  
geometry_msgs/PoseStamped goal  
float32 tolerance  
---  
nav_msgs/Path plan
```

ROS Service Example

运行 *roscore* 和一个
add_two_ints_server 节点

1号控制台:

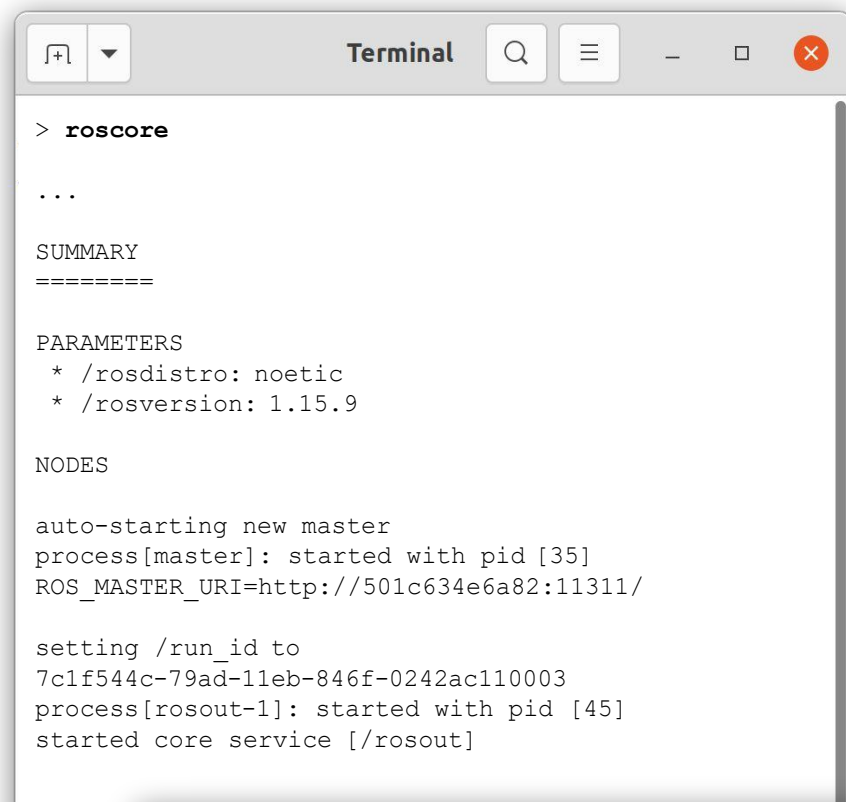
运行 *roscore* 命令用以下方式

```
> roscore
```

2号控制台:

运行 *add_two_ints_server* 节点命令用以下方式

```
> roslaunch roscpp_tutorials add_two_ints_server
```



```
> roscore

...

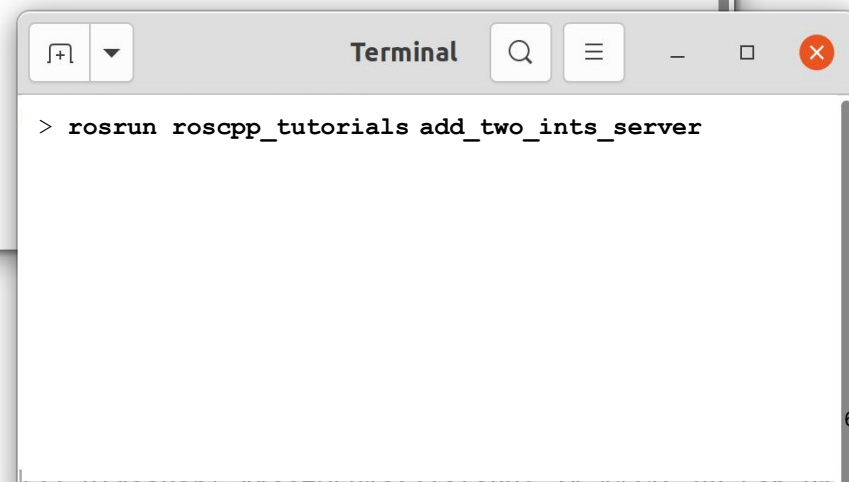
SUMMARY
=====

PARAMETERS
* /roscore: noetic
* /roscore: 1.15.9

NODES

auto-starting new master
process[master]: started with pid [35]
ROS_MASTER_URI=http://501c634e6a82:11311/

setting /run_id to
7c1f544c-79ad-11eb-846f-0242ac110003
process[roscore-1]: started with pid [45]
started core service [/roscore]
```



```
> roslaunch roscpp_tutorials add_two_ints_server
```

ROS Service Example

3号控制台 - 分析和呼叫服务

查询可用服务

```
> rosservice list
```

查询可用服务类型

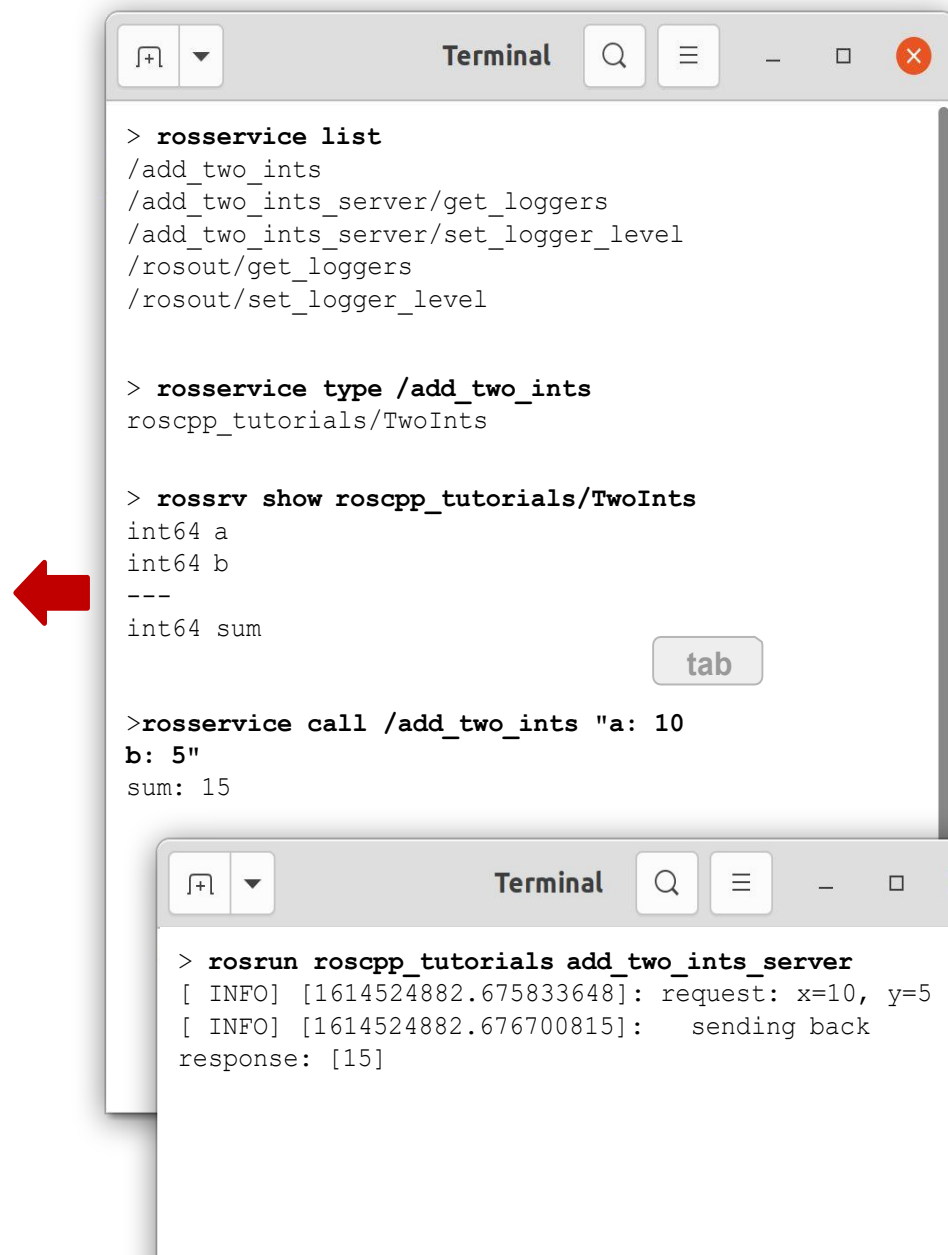
```
> rosservice type /add_two_ints
```

查询可用服务定义

```
> rossrv show roscpp_tutorials/TwoInts
```

呼叫服务（可用Tab自动完成）

```
> rosservice call /add_two_ints "a: 10  
b: 5"
```



```
Terminal

> rosservice list
/add_two_ints
/add_two_ints_server/get_loggers
/add_two_ints_server/set_logger_level
/rosout/get_loggers
/rosout/set_logger_level

> rosservice type /add_two_ints
roscpp_tutorials/TwoInts

> rossrv show roscpp_tutorials/TwoInts
int64 a
int64 b
---
int64 sum

tab

>rosservice call /add_two_ints "a: 10
b: 5"
sum: 15

Terminal

> roslaunch roscpp_tutorials add_two_ints_server
[ INFO] [1614524882.675833648]: request: x=10, y=5
[ INFO] [1614524882.676700815]: sending back
response: [15]
```

ROS C++ Client Library (*roscpp*)

服务器

- 创建服务

```
ros::ServiceServer service =
  nodeHandle.advertiseService(service_name,
    callback_function);
```

- 当服务器请求被收到, 回调函数 (callback function) 回应请求

- 将返回值写入回调函数中

- 函数返回值为true, 说明已经被正确执行

More info

wiki.ros.org/roscpp/Overview/Services

[add_two_ints_server.cpp](#) (use OO-approach in exercises)

```
#include <ros/ros.h>
#include <roscpp_tutorials/TwoInts.h>

bool add(roscpp_tutorials::TwoInts::Request &request,
         roscpp_tutorials::TwoInts::Response &response)
{
    response.sum = request.a + request.b;
    ROS_INFO("request: x=%ld, y=%ld", (long int)request.a,
              (long int)request.b);
    ROS_INFO("  sending back response: [%ld]",
              (long int)response.sum);
    return true;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "add_two_ints_server");
    ros::NodeHandle nh;
    ros::ServiceServer service =
    nh.advertiseService("add_two_ints", add);
    ros::spin();
    return 0;
}
```


ROS C++ Client Library (*roscpp*)

客户端

- 创建客户端

```
ros::ServiceClient client =
    nodeHandle.serviceClient<service_type>
        (service_name);
```

- 创建服务请求目录

- `service.request`

- 回应服务

- `client.call(service)`

- 响应服务

- `service.response`

More info

wiki.ros.org/roscpp/Overview/Services

`add_two_ints_client.cpp`

```
#include <ros/ros.h>
#include <roscpp_tutorials/TwoInts.h>
#include <cstdlib>

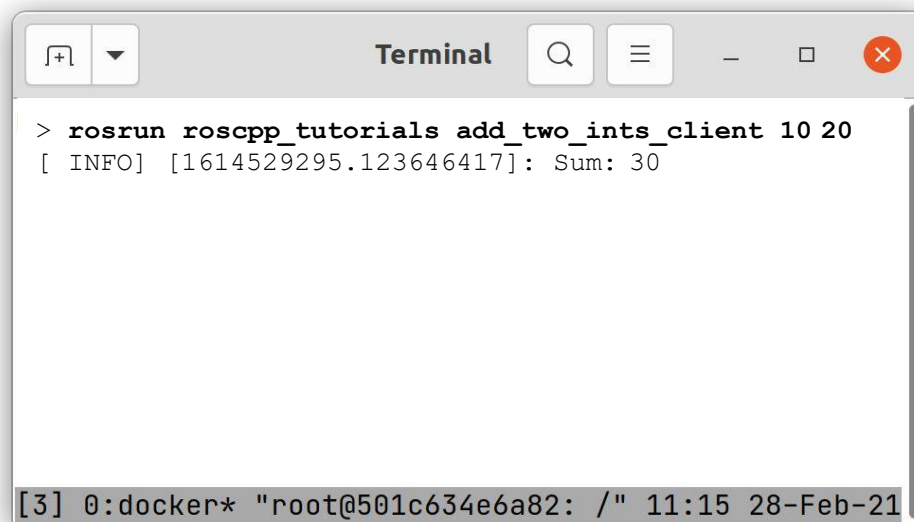
int main(int argc, char **argv) {
    ros::init(argc, argv, "add_two_ints_client");
    if (argc != 3) {
        ROS_INFO("usage: add_two_ints_client X Y");
        return 1;
    }

    ros::NodeHandle nh;
    ros::ServiceClient client =
        nh.serviceClient<roscpp_tutorials::TwoInts>("add_two_ints");
    roscpp_tutorials::TwoInts service;
    service.request.a = atoi(argv[1]);
    service.request.b = atoi(argv[2]);
    if (client.call(service)) {
        ROS_INFO("Sum: %ld", (long int)service.response.sum);
    } else {
        ROS_ERROR("Failed to call service add_two_ints");
        return 1;
    }
    return 0;
}
```



ROS C++ Client Library (*roscpp*)

客户端

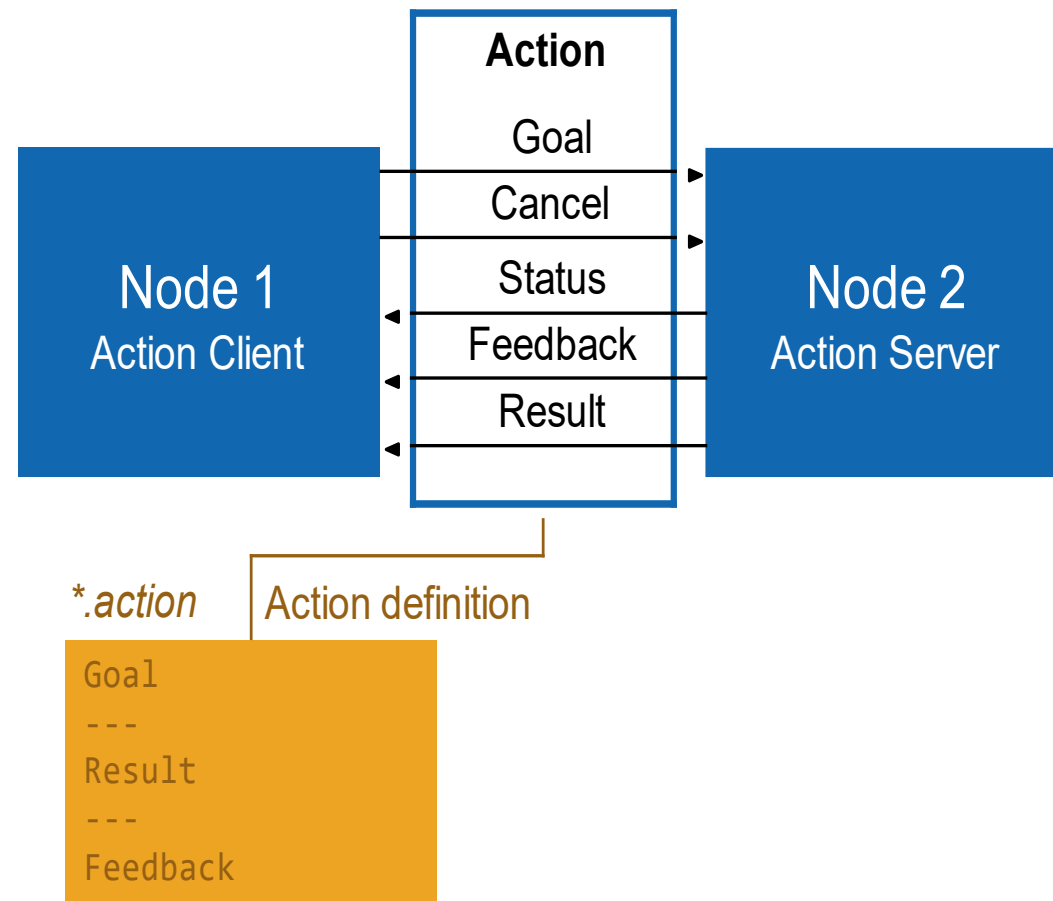
A terminal window titled "Terminal" with standard window controls (minimize, maximize, close). The terminal shows a command prompt where the user has entered `roslaunch roscpp_tutorials add_two_ints_client 10 20`. The output is `[INFO] [1614529295.123646417]: Sum: 30`. At the bottom of the terminal, a status bar shows `[3] 0:docker* "root@501c634e6a82: /" 11:15 28-Feb-21`.

```
> roslaunch roscpp_tutorials add_two_ints_client 10 20
[ INFO] [1614529295.123646417]: Sum: 30

[3] 0:docker* "root@501c634e6a82: /" 11:15 28-Feb-21
```

ROS Actions (actionlib)

- 类似于服务调用，但提供了以下可能性
 - 取消任务（抢占）。
 - 接收关于进度的反馈
- 实现延长时间的、面向目标的行为的接口的最佳方式
- 与服务的结构类似，活动被定义在*.action文件中
- 在内部，行动是通过一组topic实现的



More info

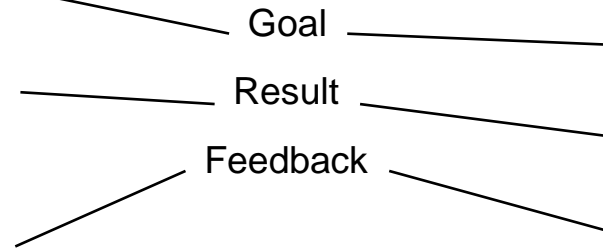
wiki.ros.org/actionlib

wiki.ros.org/actionlib/DetailedDescription

ROS Actions (actionlib)

Averaging.action

```
int32 samples
---
float32 mean
float32 std_dev
---
int32 sample
float32 data
float32 mean
float32 std_dev
```



FollowPath.action

```
navigation_msgs/Path path
---
bool success
---
float32 remaining_distance
float32 initial_distance
```

ROS Time

- 通常情况下，ROS使用PC的系统时钟作为时间源（wall time）
- 对于模拟或回放记录的数据，使用模拟的时间是很方便的（暂停、减速等）
- 要用模拟时钟工作:
 - 设置/use_sim_time参数

```
> rosparam set use_sim_time true
```

- 在topic/clock上发布时间，从
 - Gazebo（默认启用）
 - ROS包(使用选项--clock)

More info

wiki.ros.org/Clock

wiki.ros.org/roscpp/Overview/Time

- 为了利用模拟的时间，你应该始终使用ROS时间API:

- **ros::Time**

```
ros::Time begin = ros::Time::now();
double secs = begin.toSec();
```

- **ros::Duration**

```
ros::Duration duration(0.5); // 0.5s
ros::Duration passed = ros::Time().now()
                        - begin;
```

- **ros::Rate**

```
ros::Rate rate(10); // 10Hz
```

- 如果需要wall time，使用
ros::WallTime, ros::WallDuration,
and ros::WallRate



ROS Bags

- *Bag* 是一种储存信息数据的格式
- 二进制格式，文件扩展名为*.bag
- 适用于记录和录制数据集，以便日后进行可视化分析
- 记录所有topics 在一个 bag 里

```
> rosbag record --all
```

记录给定的topic

```
> rosbag record topic_1 topic_2 topic_3
```

用Ctrl+C停止录制

录像袋会以开始日期和时间作为文件名保存在当前文件夹中（例如
2019-02-07-01-27-13.bag）

显示一个bag的信息

```
> rosbag info bag_name.bag
```

阅读一个袋子并发布其内容

```
> rosbag play bag_name.bag
```

可以定义播放选项，例如

```
> rosbag play --rate=0.5 bag_name.bag
```

--rate=factor	发布频率
--clock	发布时钟时间（设置参数
use_sim_time为true)	
--loop	循环播放
etc.	

More info

wiki.ros.org/rosbag/Commandline



Debugging Strategies

用你学到的工具进行调试

- 经常编译和运行代码以尽早发现错误
- 理解编译和运行时的错误信息
- 使用分析工具检查数据流（rostopic info, rostopic echo, roswtf, rqt_graph等）。
- 可视化和绘制数据（RViz、RQT Multiplot等）。
- 将程序分成小的步骤并检查中间结果（ROS_INFO, ROS_DEBUG等）。
- 用参数和返回值检查和捕捉异常来使你的代码变得健壮。
- 只有在基本版本运行后才进行扩展和优化
- 如果事情没有意义，清理你的工作空间

```
> catkin clean --all
```

学习新的工具

- 在调试模式下构建并使用GDB

```
> catkin config --cmake-args  
-DCMAKE_BUILD_TYPE=Debug  
> catkin config --cmake-args  
-DCMAKE_BUILD_TYPE=Release
```

- 使用调试器断点，例如在Eclipse中。
- 编写单元测试和集成测试以发现回归问题

More info

wiki.ros.org/UnitTesting

wiki.ros.org/gtest

wiki.ros.org/roctest

wiki.ros.org/roslaunch/Tutorials/Roslaunch%20Nodes%20in%20Valgrind%20or%20GDB



更多资料

- **ROS Wiki**
 - <http://wiki.ros.org/>
- **Installation**
 - <http://wiki.ros.org/ROS/Installation>
- **Tutorials**
 - <http://wiki.ros.org/ROS/Tutorials>
- **Available packages**
 - <http://www.ros.org/browse/>
- ★ **ROS Cheat Sheet**
 - <https://www.clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/>
 - https://kapeli.com/cheat_sheets/ROS.docset/Contents/Resources/Documents/index
- ★ **ROS Best Practices**
 - https://github.com/leggedrobotics/ros_best_practices/wiki
- ★ **ROS Package Template**
 - https://github.com/leggedrobotics/ros_best_practices/tree/master/ros_package_template