**实验项目二 ROS节点的创建与通信**

1. **实验目的**

1、 掌握ROS操作系统的基本框架；

2、 掌握ROS操作系统中的节点编程；

3、 掌握ROS操作系统的基本通信原理。

1. **实验设备**

1、 雷达模块一台；

2、 无线键鼠一套；

3、 显示器一套。

1. **实验步骤和内容**

机器人是一种高度复杂的系统性实现，在机器人上可能集成各种传感器（雷达、摄像头、GPS...）以及运动控制实现，为了解耦合，在ROS中每一个功能点都是一个单独的进程，每一个进程都是独立运行的。更确切的讲，ROS是进程（也称为Nodes）的分布式框架。 因为这些进程甚至还可分布于不同主机，不同主机协同工作，从而分散计算压力。不过随之也有一个问题: 不同的进程是如何通信的？也即不同进程间如何实现数据交换的？在此我们就需要介绍一下ROS中的通信机制了。

ROS中的基本通信机制主要有如下三种实现策略：

* 话题通信（发布订阅模式）
* 服务通信（请求响应模式）
* 参数服务器（参数共享模式）

本次项目着重介绍话题通信的实现及应用，由于服务通信及参数服务器这二种通信机制与话题通信相比应用范围较小，因此在这本次项目中只做简要概述，另请自行学习。

1. **话题通信**

话题通信是ROS中使用频率最高的一种通信模式，话题通信是基于发布订阅模式的，也即：一个节点发布消息，另一个节点订阅该消息。话题通信的应用场景也极其广泛，比如下面一个常见场景：

机器人在执行导航功能，使用的传感器是激光雷达，机器人会采集激光雷达感知到的信息并计算，然后生成运动控制信息驱动机器人底盘运动。

在上述场景中，就不止一次使用到了话题通信。

* 以激光雷达信息的采集处理为例，在ROS中有一个节点需要时时的发布当前雷达采集到的数据，导航模块中也有节点会订阅并解析雷达数据。
* 再以运动消息的发布为例，导航模块会根据传感器采集的数据时时的计算出运动控制信息并发布给底盘，底盘也可以有一个节点订阅运动信息并最终转换成控制电机的脉冲信号。

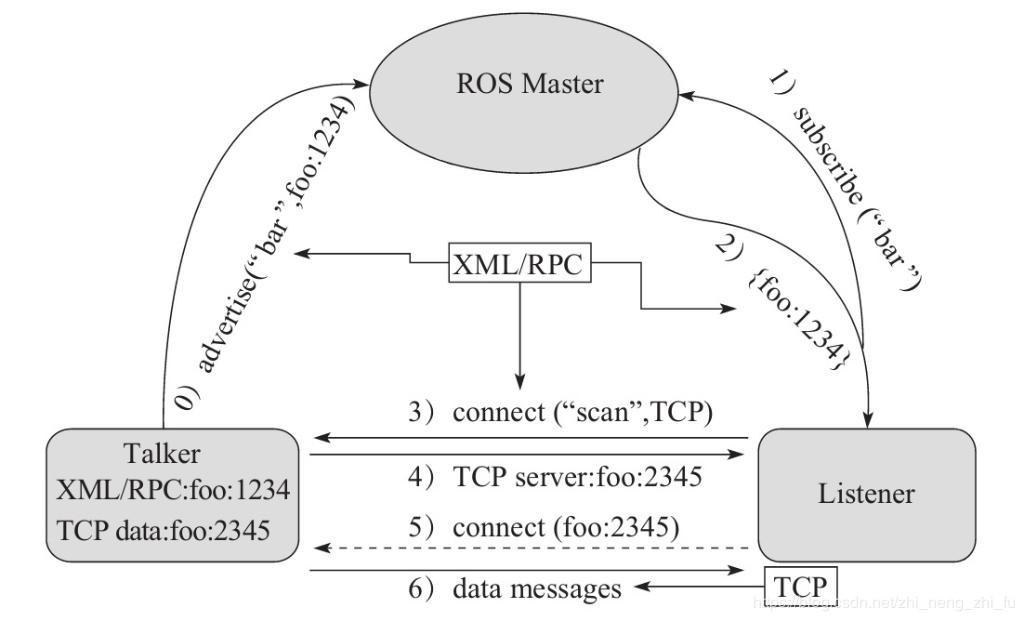
以此类推，像雷达、摄像头、GPS...等等一些传感器数据的采集，也都是使用了话题通信，换言之，话题通信适用于不断更新的数据传输相关的应用场景。

1. **理论模型**

话题通信实现模型是比较复杂的，该模型如下图所示，该模型中涉及到三个角色：

* ROS Master (管理者)
* Talker (发布者)
* Listener (订阅者)

ROS Maste负责保管Talker和Listener注册的信息，并匹配话题相同的Talker与Listener，帮助Talker与Listener建立连接，连接建立后，Talker可以发布消息，且发布的消息会被Listener订阅。



整个流程由以下步骤实现：

1. **Talker注册**

Talker启动后，会通过RPC在ROS Master中注册自身信息，其中包含所发布消息的话题名称。ROS Master会将节点的注册信息加入到注册表中。

1. **Listener注册**

Listener启动后，也会通过RPC在ROS Master中注册自身信息，包含需要订阅消息的话题名。ROS Master会将节点的注册信息加入到注册表中。

1. **ROS Master实现信息匹配**

ROS Master 会根据注册表中的信息匹配Talker和Listener，并通过RPC向Listener发送Talker的RPC地址信息。

1. **Listener向Talker发送请求**

Listener根据接收到的RPC地址，通过RPC向Talker发送连接请求，传输订阅的话题名称、消息类型以及通信协议（TCP/UDP）。

1. **Talker确认请求**

Talker接收到Listener的请求后，也是通过RPC向Listener确认连接信息，并发送自身的TCP地址信息。

1. **Listener与Talker件里连接**

Listener根据步骤4返回的消息使用TCP与Talker建立网络连接。

1. **Talker向Listener发送消息**

连接建立后，Talker开始向Listener发布消息。

**注意1：**上述实现流程中，前五步使用的RPC协议，最后两步使用的是TCP协议。

**注意2：**Talker与Listener的启动无先后顺序要求。

**注意3：**Talker与Listener都可以有多个。

**注意4：**Talker与Listener连接建立后，不再需要ROS Master。也即，即便关闭ROS Master，Talker与Listener照常通信。

1. **话题通信基本操作**

**需求：**

编写发布订阅实现，要求发布方以10HZ（每秒10次）的频率发布文本消息，订阅方订阅消息并将消息内容打印输出。

**分析：**

在模型实现中，ROS master不需要实现，而连接的建立也已经被封装了，需要关注的关键点有三个：

1. 发布方
2. 接收方
3. 数据（此处为普通文本）

**流程：**

1. 编写发布方实现
2. 编写订阅方实现
3. 为Python文件添加可执行权限
4. 执行程序
5. **发布方**

#! /usr/bin/env python

#coding=utf-8

"""

需求: 实现基本的话题通信，一方发布数据，一方接收数据，

实现的关键点:

1.发送方

2.接收方

3.数据(此处为普通文本)

PS: 二者需要设置相同的话题

消息发布方:

循环发布信息:HelloWorld 后缀数字编号

实现流程:

1.导包

2.初始化 ROS 节点:命名(唯一)

3.实例化 发布者 对象

4.组织被发布的数据，并编写逻辑发布数据

"""

# 导包

import rospy

from std\_msgs.msg import String

if \_\_name\_\_ == "\_\_main\_\_":

# 初始化 ROS 节点:命名(唯一)

rospy.init\_node("talker\_p")

# 实例化 发布者 对象

pub = rospy.Publisher("chatter",String,queue\_size=10)

# 组织被发布的数据，并编写逻辑发布数据

msg = String() # 创建 msg 对象

msg\_front = "hello 你好"

count = 0 # 计数器

# 设置循环频率

rate = rospy.Rate(1)

while not rospy.is\_shutdown():

#拼接字符串

msg.data = msg\_front + str(count)

pub.publish(msg)

rate.sleep()

rospy.loginfo("写出的数据:%s",msg.data)

count += 1

1. **订阅方**

#! /usr/bin/env python

#coding=utf-8

"""

需求: 实现基本的话题通信，一方发布数据，一方接收数据，

实现的关键点:

1.发送方

2.接收方

3.数据(此处为普通文本)

消息订阅方:

订阅话题并打印接收到的消息

实现流程:

1.导包

2.初始化 ROS 节点:命名(唯一)

3.实例化 订阅者 对象

4.处理订阅的消息(回调函数)

5.设置循环调用回调函数

"""

# 导包

import rospy

from std\_msgs.msg import String

def doMsg(msg):

rospy.loginfo("I heard:%s",msg.data)

if \_\_name\_\_ == "\_\_main\_\_":

# 初始化 ROS 节点:命名(唯一)

rospy.init\_node("listener\_p")

# 实例化 订阅者 对象

sub = rospy.Subscriber("chatter",String,doMsg,queue\_size=10)

# 处理订阅的消息(回调函数)

# 设置循环调用回调函数

rospy.spin()

1. **添加可执行权限**

终端下进入scripts执行：chmod +x \*.py

1. **执行**

* 启动roscore；
* 启动发布节点；
* 启动订阅节点。

运行结果见后文实验参考结果中的案例1。

1. **话题通信自定义msg**

在ROS通信协议中，数据载体是一个较为重要组成部分，ROS中通过std\_msgs封装了一些原生的数据类型，比如：String、Int32、Int64、Char、Bool、Empty...但是，这些数据一般只包含一个data字段，结构的单一意味着功能上的局限性，当传输一些复杂的数据，比如: 激光雷达的信息...std\_msgs由于描述性较差而显得力不从心，这种场景下可以使用自定义的消息类型。

msgs只是简单的文本文件，每行具有字段类型和字段名称，可以使用的字段类型有：

* int8, int16, int32, int64 (或者无符号类型: uint\*)
* float32, float64
* string
* time, duration
* othermsg files
* variable-length array[] and fixed-length array[C]

ROS中还有一种特殊类型：Header，标头包含时间戳和ROS中常用的坐标帧信息。会经常看到msg文件的第一行具有Header标头。

**需求：**

创建自定义消息，该消息包含人的信息:姓名、身高、年龄等。

**流程：**

1. 按照固定格式创建msg文件；
2. 编辑配置文件；
3. 编译生成可以被Python或C++调用的中间文件。
4. **定义msg文件**

功能包下新建msg目录，添加文件Person.msg

string name

uint16 age

float64 height

1. **编辑配置文件**

package.xml中添加编译依赖与执行依赖：

<build\_depend>message\_generation</build\_depend>

<exec\_depend>message\_runtime</exec\_depend>

<!--

exce\_depend 以前对应的是 run\_depend 现在非法

-->

CMakeLists.txt编辑msg相关配置：

find\_package(catkin REQUIRED COMPONENTS

roscpp

rospy

std\_msgs

message\_generation

)

# 需要加入 message\_generation,必须有 std\_msgs

## 配置 msg 源文件

add\_message\_files(

FILES

Person.msg

)

# 生成消息时依赖于 std\_msgs

generate\_messages(

DEPENDENCIES

std\_msgs

)

#执行时依赖

catkin\_package(

# INCLUDE\_DIRS include

# LIBRARIES demo02\_talker\_listener

CATKIN\_DEPENDS roscpp rospy std\_msgs message\_runtime

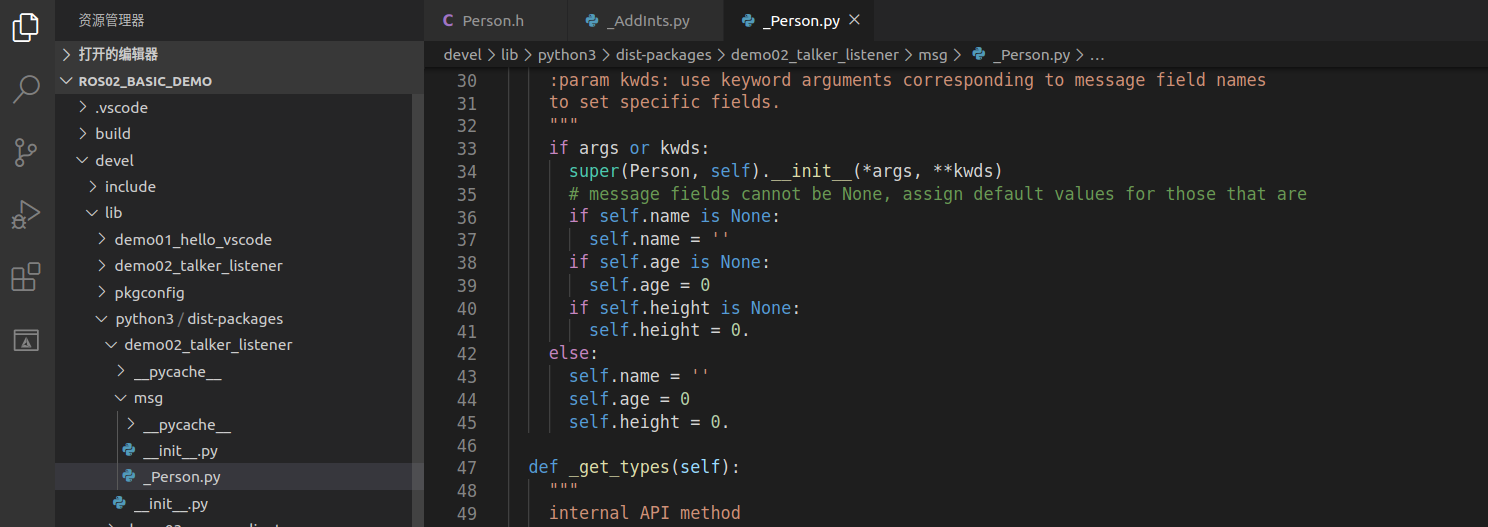
# DEPENDS system\_lib

)

1. **编译**

编译后的中间文件查看：

Python需要调用的中间文件（.../工作空间/devel/lib/python2.7/dist-packages/包名/msg）。



后续调用相关msg时，是从这些中间文件调用的。

1. **话题通信自定义msg调用**

**需求：**

编写发布订阅实现，要求发布方以1HZ（每秒1次）的频率发布自定义消息，订阅方订阅自定义消息并将消息内容打印输出。

**分析：**

在模型实现中，ROS master不需要实现，而连接的建立也已经被封装了，需要关注的关键点有三个：

1. 发布方
2. 接收方
3. 数据（此处为自定义消息）

**流程：**

1. 编写发布方实现；
2. 编写订阅方实现；
3. 为python文件添加可执行权限；
4. 执行程序。
5. **发布方**

#! /usr/bin/env python

#coding=utf-8

"""

发布方:

循环发送消息

"""

import rospy

from test\_01.msg import Person

if \_\_name\_\_ == "\_\_main\_\_":

# 初始化 ROS 节点

rospy.init\_node("talker\_person\_p")

# 创建发布者对象

pub = rospy.Publisher("chatter\_person",Person,queue\_size=10)

# 组织消息

p = Person()

p.name = "张三"

p.age = 18

p.height = 0.75

# 编写消息发布逻辑

rate = rospy.Rate(1)

while not rospy.is\_shutdown():

pub.publish(p) #发布消息

rate.sleep() #休眠

rospy.loginfo("姓名:%s, 年龄:%d, 身高:%.2f",p.name, p.age, p.height)

1. **订阅方**

#! /usr/bin/env python

#coding=utf-8

"""

订阅方:

订阅消息

"""

import rospy

from test\_01.msg import Person

def doPerson(p):

rospy.loginfo("接收到的人的信息:%s, %d, %.2f",p.name, p.age, p.height)

if \_\_name\_\_ == "\_\_main\_\_":

# 初始化节点

rospy.init\_node("listener\_person\_p")

# 创建订阅者对象

sub = rospy.Subscriber("chatter\_person",Person,doPerson,queue\_size=10)

rospy.spin()

1. **权限设置**

终端下进入scripts执行：chmod +x \*.py

1. **执行**

* 启动roscore；
* 启动发布节点；
* 启动订阅节点。

运行结果见后文实验参考结果中的案例2。

1. **服务通信**

服务通信也是ROS中一种极其常用的通信模式，服务通信是基于请求响应模式的，是一种应答机制。也即：一个节点A向另一个节点B发送请求，B接收处理请求并产生响应结果返回给A。比如如下场景：

机器人巡逻过程中，控制系统分析传感器数据发现可疑物体或人,此时需要拍摄照片并留存。在上述场景中，就使用到了服务通信。一个节点需要向相机节点发送拍照请求，相机节点处理请求，并返回处理结果。

与上述应用类似的，服务通信更适用于对时时性有要求、具有一定逻辑处理的应用场景。

**另请参考：**

[http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv](http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv" \t "/home/ha/文档x/_blank)

[http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29](http://wiki.ros.org/ROS/Tutorials/WritingServiceClient(c++)" \t "/home/ha/文档x/_blank)

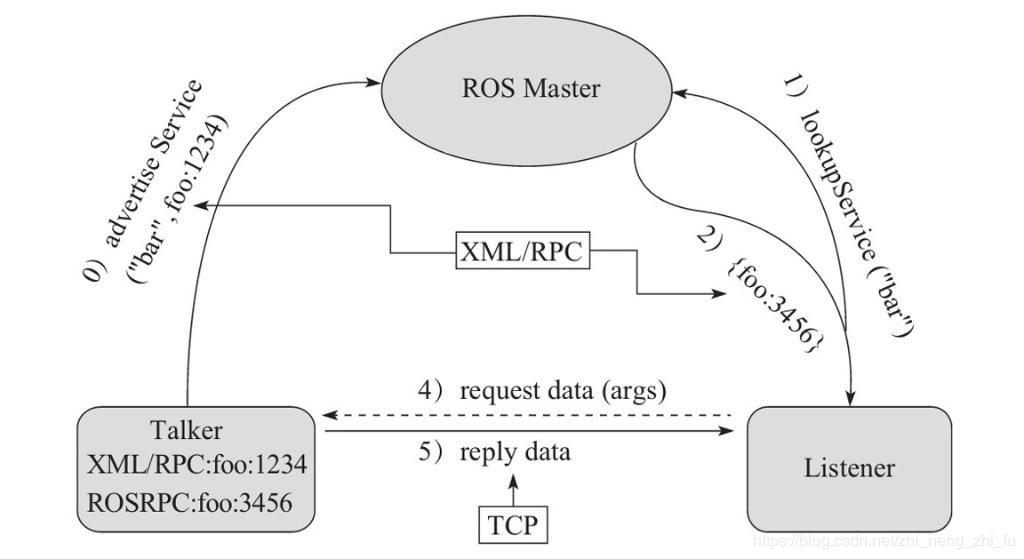
[http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28python%29](http://wiki.ros.org/ROS/Tutorials/WritingServiceClient(python)" \t "/home/ha/文档x/_blank)

**理论模型**

服务通信较之于话题通信更简单些，理论模型如下图所示，该模型中涉及到三个角色：

* ROS master（管理者）
* Server（服务端）
* Client（客户端）

ROS Master负责保管Server和Client注册的信息，并匹配话题相同的Server与Client，帮助Server与Client建立连接，连接建立后，Client发送请求信息，Server返回响应信息。



1. **Server注册**

Serve启动后，会通过RPC在ROS Master中注册自身信息，其中包含提供的服务的名称。ROS Master会将节点的注册信息加入到注册表中。

1. **Client注册**

Client启动后，也会通过RPC在ROS Master中注册自身信息，包含需要请求的服务的名称。ROS Master会将节点的注册信息加入到注册表中。

1. **ROS Master实现信息匹配**

ROS Master会根据注册表中的信息匹配Server和Client，并通过RPC向Client发送Server的TCP地址信息。

1. **Client发送请求**

Client根据步骤2响应的信息，使用TCP与Server建立网络连接，并发送请求数据。

1. **Server发送响应**

Server接收、解析请求的数据，并产生响应结果返回给Client。

**注意：**

* 客户端请求被处理时，需要保证服务器已经启动；
* 服务端和客户端都可以存在多个。

1. **参数服务器**

参数服务器在ROS中主要用于实现不同节点之间的数据共享。参数服务器相当于是独立于所有节点的一个公共容器，可以将数据存储在该容器中，被不同的节点调用，当然不同的节点也可以往其中存储数据，关于参数服务器的典型应用场景如下：

导航实现时，会进行路径规划，比如：全局路径规划，设计一个从出发点到目标点的大致路径。本地路径规划，会根据当前路况生成时时的行进路径。

上述场景中，全局路径规划和本地路径规划时，就会使用到参数服务器：

路径规划时，需要参考小车的尺寸，我们可以将这些尺寸信息存储到参数服务器，全局路径规划节点与本地路径规划节点都可以从参数服务器中调用这些参数。

参数服务器，一般适用于存在数据共享的一些应用场景。

**另请参考：**

[http://wiki.ros.org/Parameter%20Server](http://wiki.ros.org/Parameter%20Server" \t "/home/ha/文档x/_blank)

[http://wiki.ros.org/roscpp/Overview/Parameter%20Server](http://wiki.ros.org/roscpp/Overview/Parameter%20Server" \t "/home/ha/文档x/_blank)

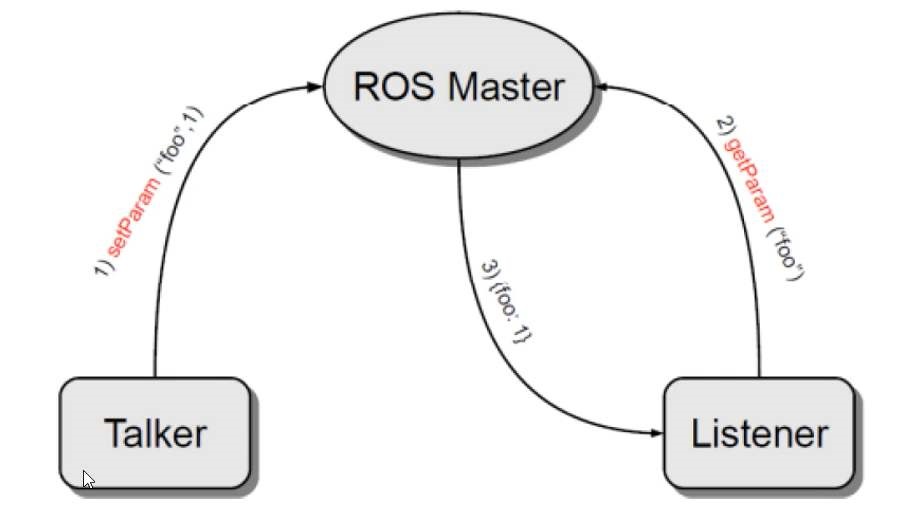
[http://wiki.ros.org/rospy/Overview/Parameter%20Server](http://wiki.ros.org/rospy/Overview/Parameter%20Server" \t "/home/ha/文档x/_blank)

**理论模型**

参数服务器实现是最为简单的，该模型如下图所示，该模型中涉及到三个角色：

* ROS Master （管理者）
* Talker （参数设置者）
* Listener （参数调用者）

ROS Master作为一个公共容器保存参数，Talker可以向容器中设置参数，Listener可以获取参数。



整个流程由以下步骤实现：

1. **Talker设置参数**

Talker通过RPC向参数服务器发送参数(包括参数名与参数值)，ROS Master将参数保存到参数列表中。

1. **Listener获取参数**

Listener通过RPC向参数服务器发送参数查找请求，请求中包含要查找的参数名。

1. **ROS Master向Listener发送参数值**

ROS Master根据步骤2请求提供的参数名查找参数值，并将查询结果通过RPC发送给Listener。

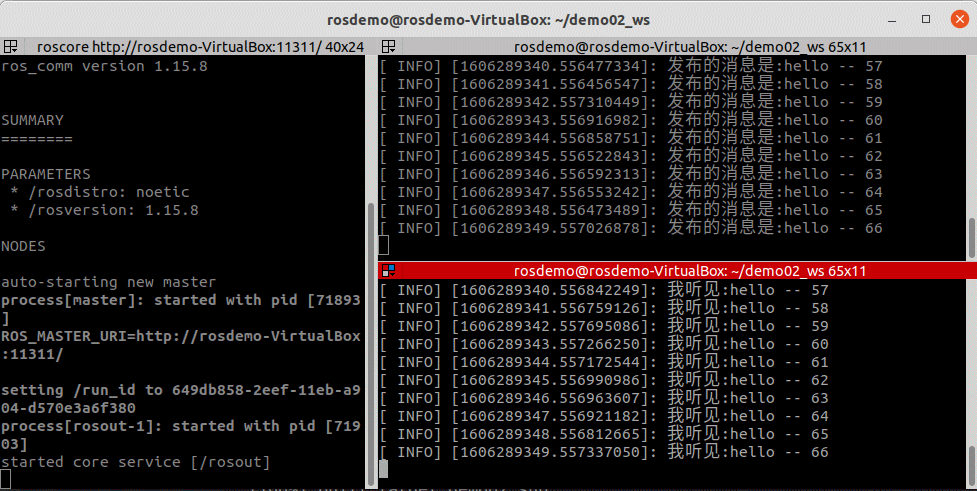
参数可使用数据类型：

* 32-bit integers
* booleans
* strings
* doubles
* iso8601 dates
* lists
* base64-encoded binary data
* 字典

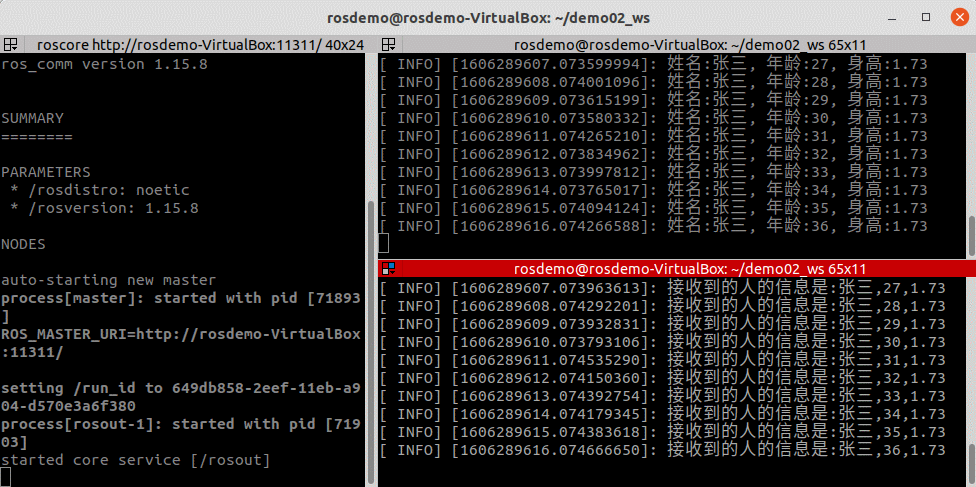
**注意：**参数服务器不是为高性能而设计的，因此最好用于存储静态的非二进制的简单数据。

1. **实验参考结果**

**案例1：**



**案例2：**



1. **相关的测试代码**

见代码包test\_01。