上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

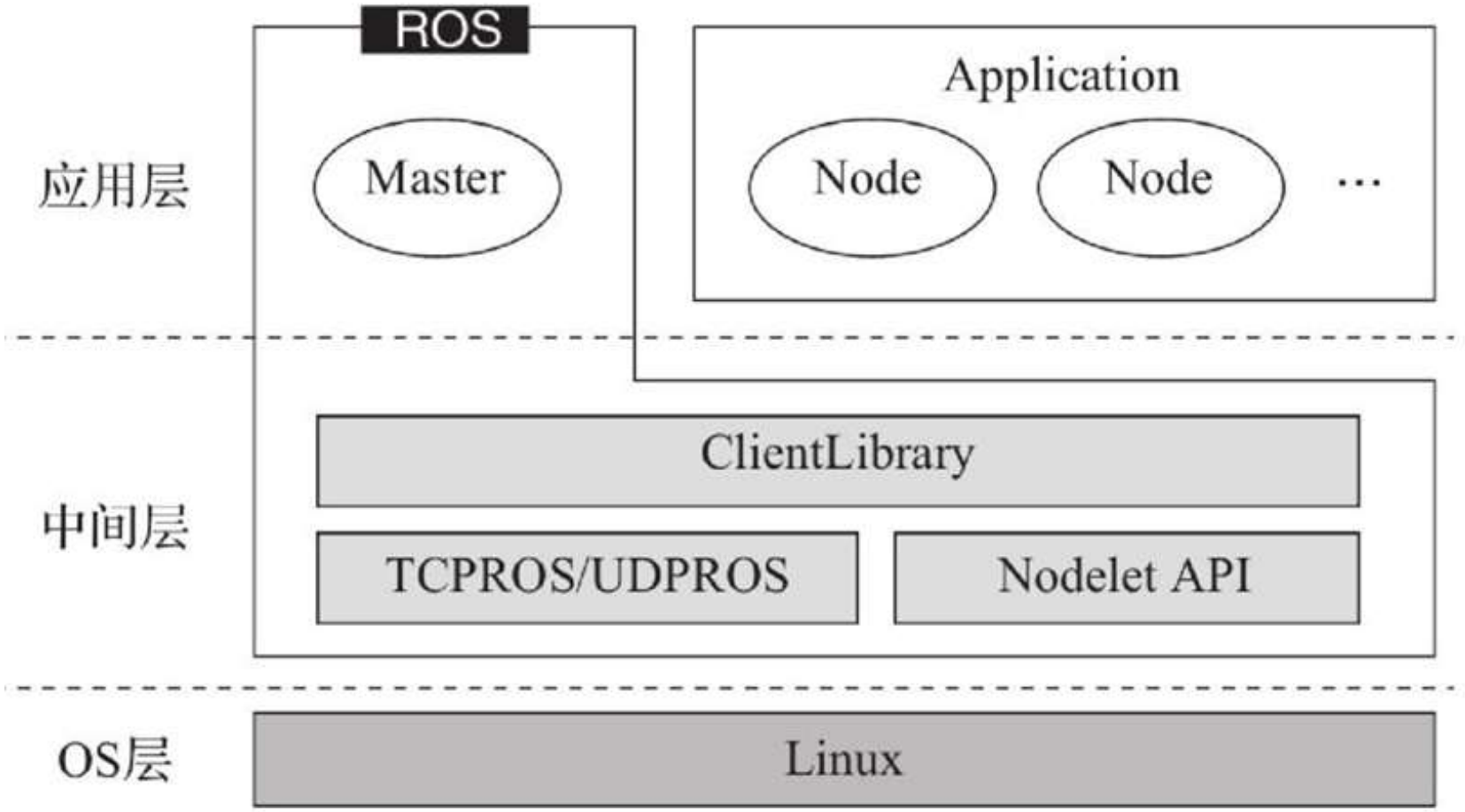
ROS操作系统

1. ROS系统

ROS：Robot Operating System，机器人操作系统

- 通信平台：ROS 提供了一种发布/订阅式的通信框架，用以简单、快速地构建分布式计算系统[3]。
- 工具：ROS 提供了大量可视化、调试等工具组合，用以配置、启动、自检、调试、可视化、仿真、登录、测试、终止分布式计算系统。
- 能力：具有控制、规划、预测、定位操纵等功能。
- 平台支持：ROS 的发展依托于一个强大的社区。wiki. ros. org 尤其关注 ROS 的兼容性和支持文档，它提供了一套“一站式”方案使得用户能够快速搜索并学习来自全球开发者数以千计的 ROS 程序包。

1. ROS系统



ROS主要为机器人开发提供硬件抽象、底层驱动、消息传递、程序管理、应用原型等功能和机制，同时整合了很多第三方工具和库文件，帮助用户快速完成机器人应用的建立、编写和多机整合。

1. ROS系统

- 特点

1.点对点：两个Node之间进行消息通讯是一个点对点的行为。

2.分布式：在部署多机之间的消息通讯时，ROS提供了一个天然的支持。

3.跨语言，并不关注每个节点之间是用什么语言来写的。只需要按照ROS提供的一些接口完成消息的订阅和分发即可以完成一个消息之间的通信。

4.轻量级，用户只需要关注自己核心模块的算法逻辑，不需要关注底层是如何通信、如何断开通信、如何进行Service 和Param之间的一些交互的。

5.开源框架，大家都可以往ROS里面贡献自己的一些想法和代码。

1. ROS系统

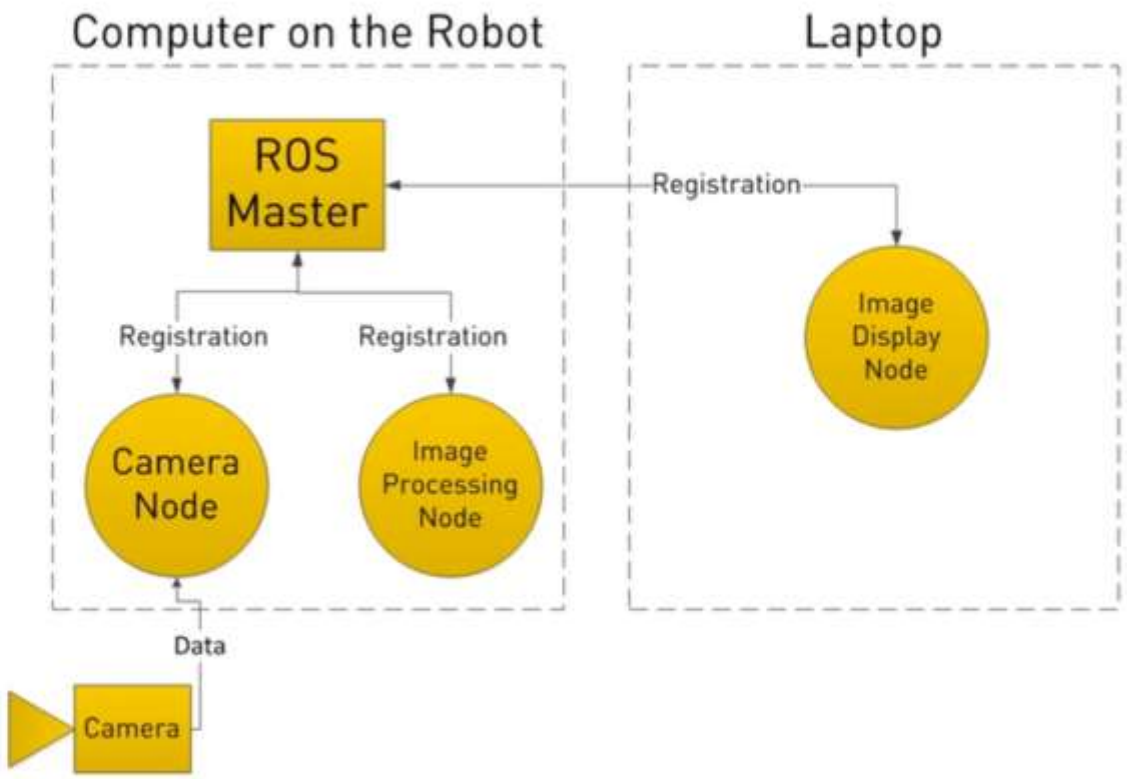
• 节点和节点管理器

■ **节点（Node）** —— 执行单元

- 执行具体任务的进程、独立运行的可执行文件；
- 不同节点可使用不同的编程语言，可分布式运行在不同的主机；
- 节点在系统中的名称必须是唯一的。

■ **节点管理器 （ROS Master）** —— 控制中心

- 为节点提供命名和注册服务；
- 跟踪和记录话题/服务通信，辅助节点相互查找、建立连接；
- 提供参数服务器，节点使用此服务器存储和检索运行时的参数。



1. ROS系统

- ROS数据传输

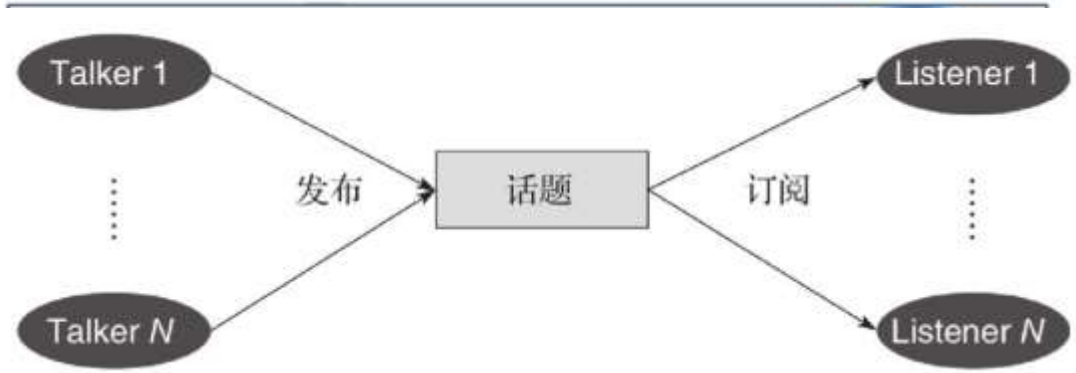
1）话题（Topic）

2）服务（Server）

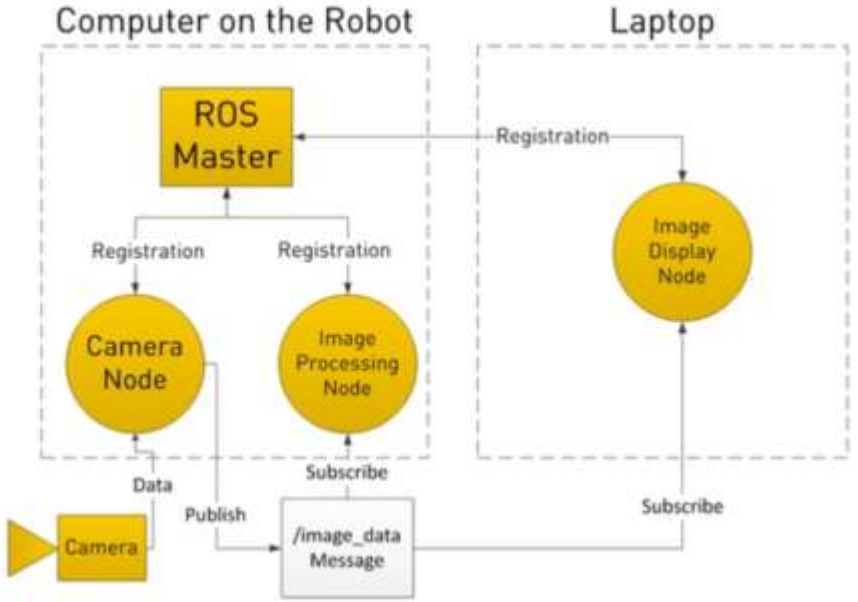
3）参数（Parameter）

1. ROS系统

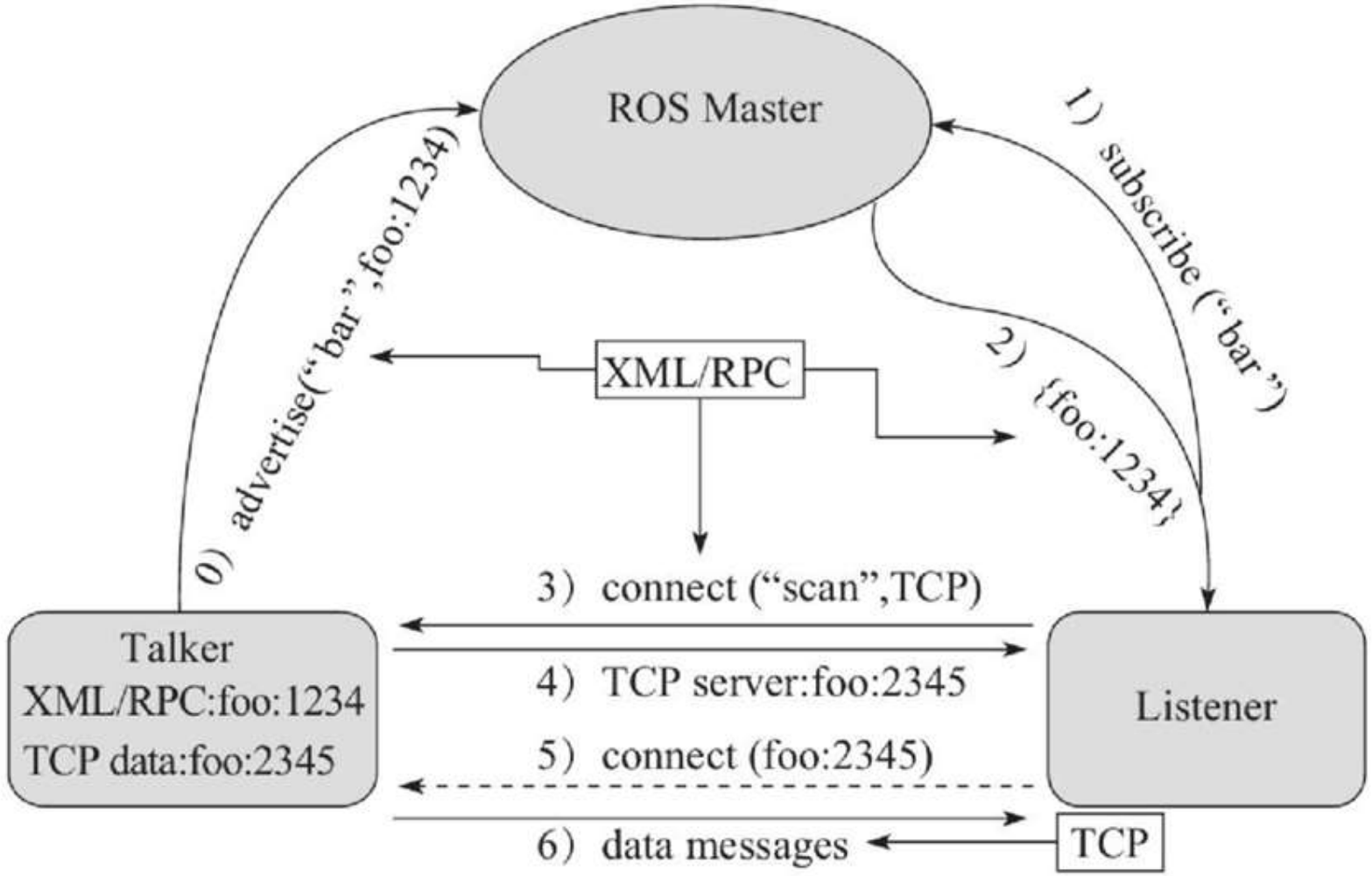
- **话题（Topic）** —— 异步通信机制
  - 节点间用来传输数据的重要总线；
  - 使用发布/订阅模型，数据由发布者传输到订阅者，同一个话题的订阅者或发布者可以不唯一。
- **消息（Message）** —— 话题数据
  - 具有一定的类型和数据结构，包括ROS提供的标准类型和用户自定义类型；
  - 使用编程语言无关的.msg文件定义，编译过程中生成对应的代码文件。



话题模型（发布/订阅）



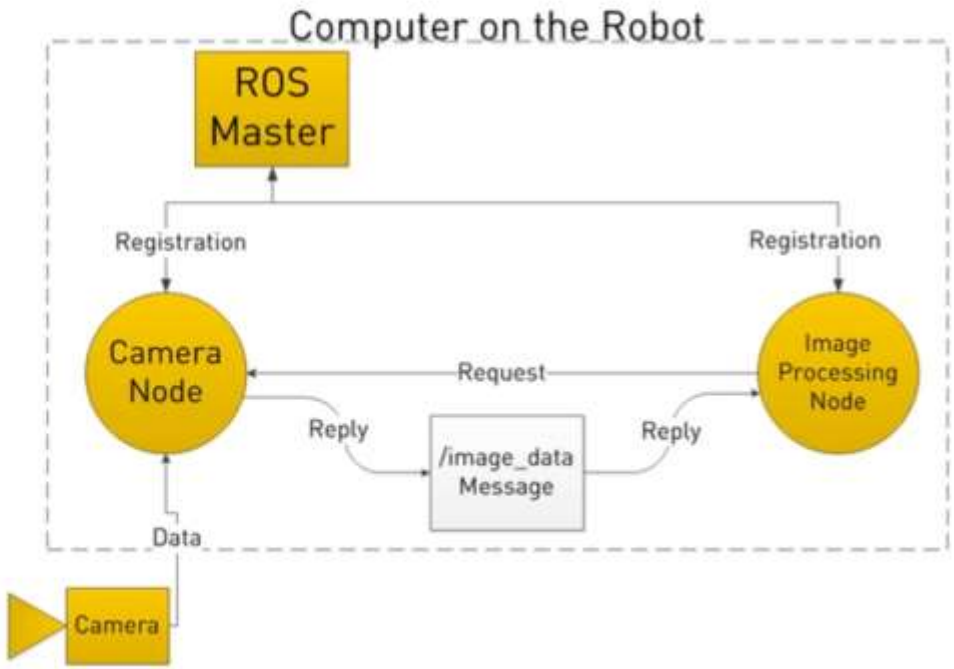
1. ROS系统

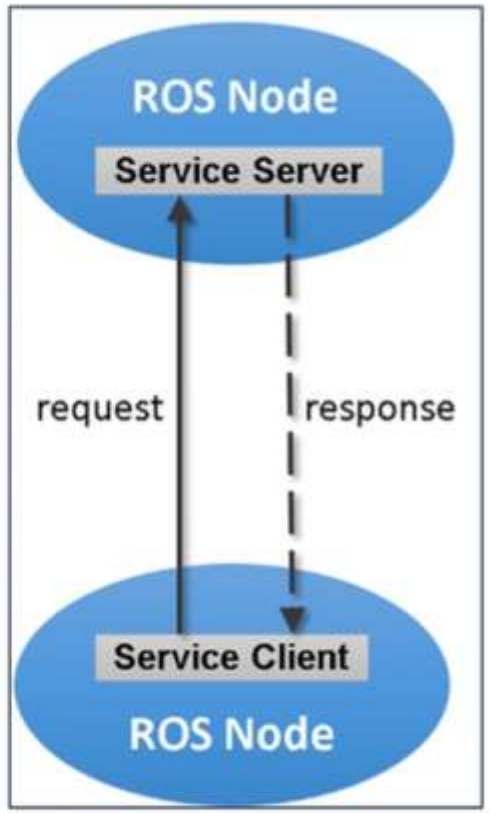

ROS Master
0) advertise("bar",foo:1234)
1) subscribe ("bar")
2) {foo:1234}
XML/RPC
3) connect ("scan",TCP)
4) TCP server:foo:2345
5) connect (foo:2345)
6) data messages
Talker
XML/RPC:foo:1234
TCP data:foo:2345
Listener
TCP

1. ROS系统

■ **服务（Service）—— 同步通信机制**

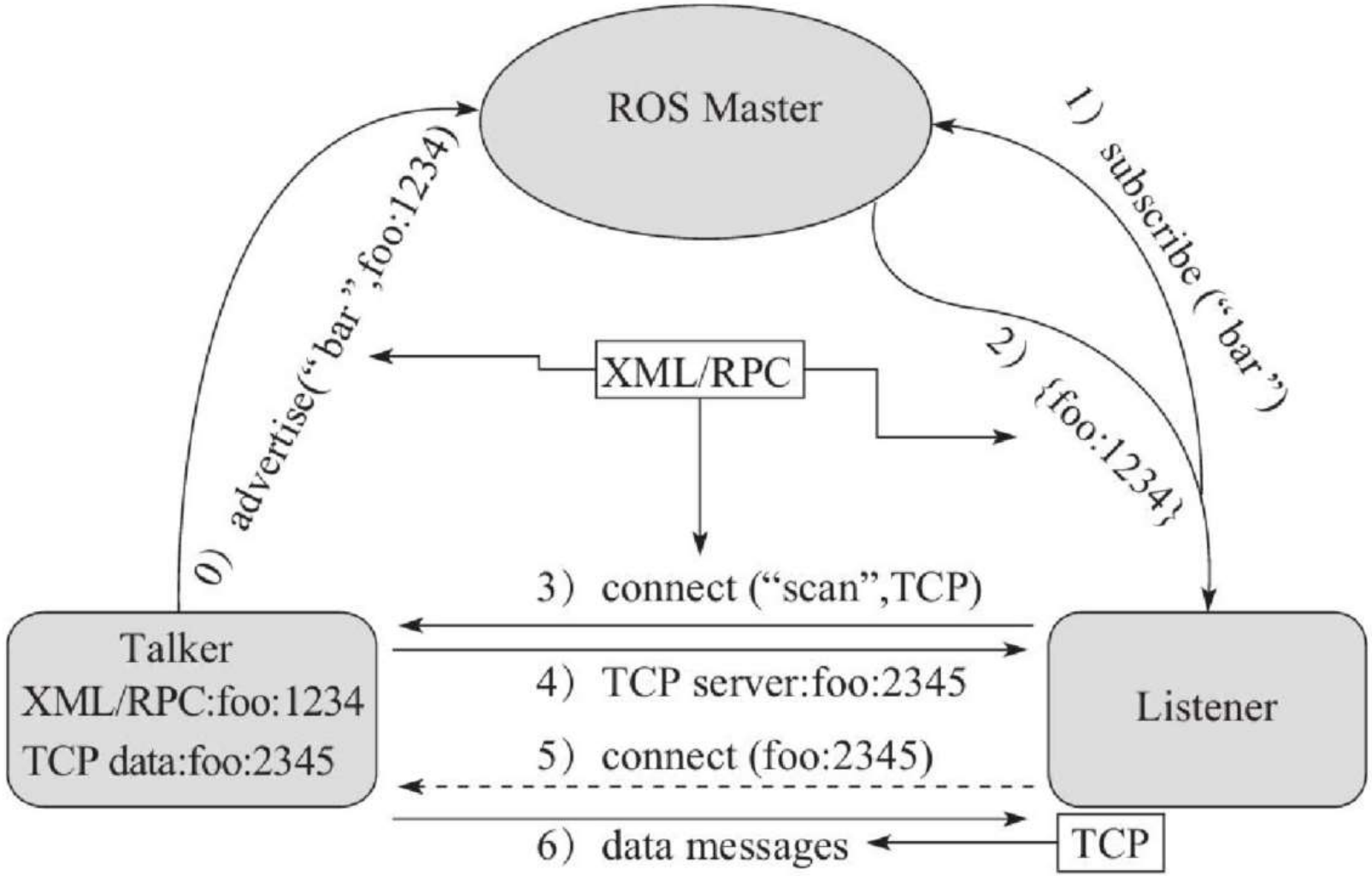
- 使用客户端/服务器（C/S）模型，客户端发送请求数据，服务器完成处理后返回应答数据；
- 使用编程语言无关的.srv文件定义请求和应答数据结构，编译过程中生成对应的代码文件。





服务模型（请求/应答）

1. ROS系统


ROS Master
0) advertise("bar","foo:1234)
1) subscribe ("bar")
2) {foo:1234}
XML/RPC
3) connect ("scan",TCP)
4) TCP server:foo:2345
5) connect (foo:2345)
6) data messages
Talker
XML/RPC:foo:1234
TCP data:foo:2345
Listener
TCP

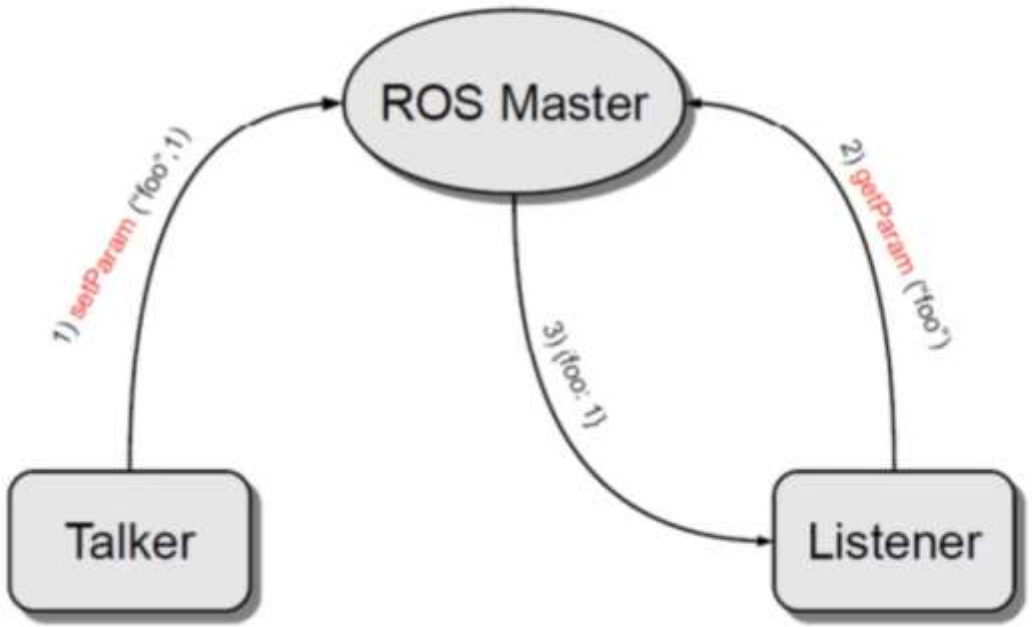
1. ROS系统

- 话题和服务区别

	话题	服务
同步性	异步	同步
通信模型	发布/订阅	服务器/客户端
底层协议	ROSTCP/ROSUDP	ROSTCP/ROSUDP
反馈机制	无	有
缓冲区	有	无
实时性	弱	强
节点关系	多对多	一对多（一个server）
适用场景	数据传输	逻辑处理

1. ROS系统

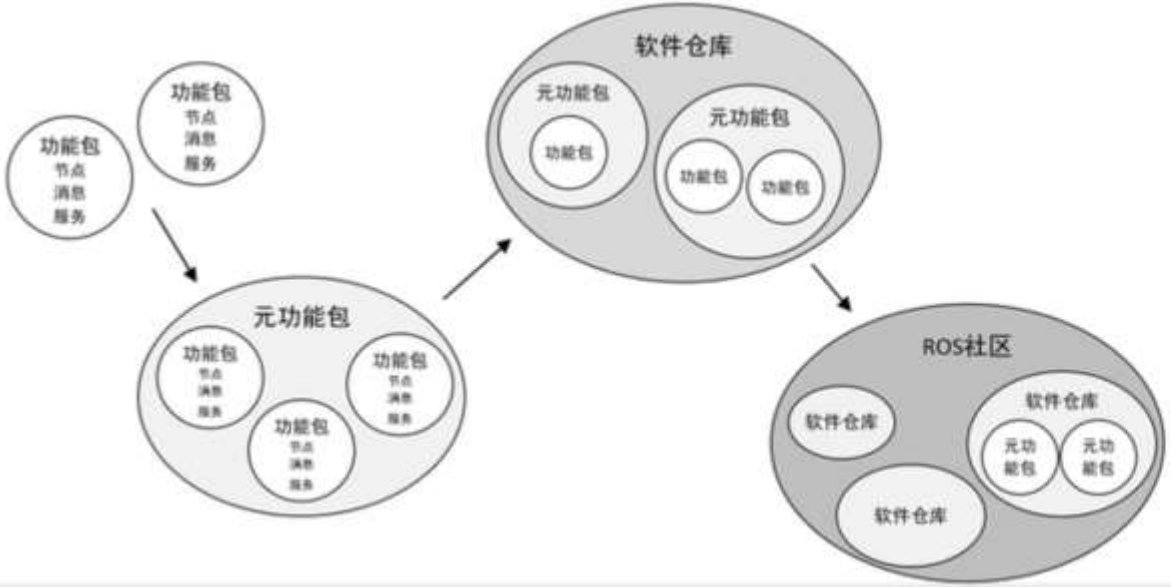
- **参数（Parameter）—— 全局共享字典**
  - 可通过网络访问的共享、多变量字典；
  - 节点使用此服务器来存储和检索运行时的参数；
  - 适合存储静态、非二进制的配置参数，不适合存储动态配置的数据。

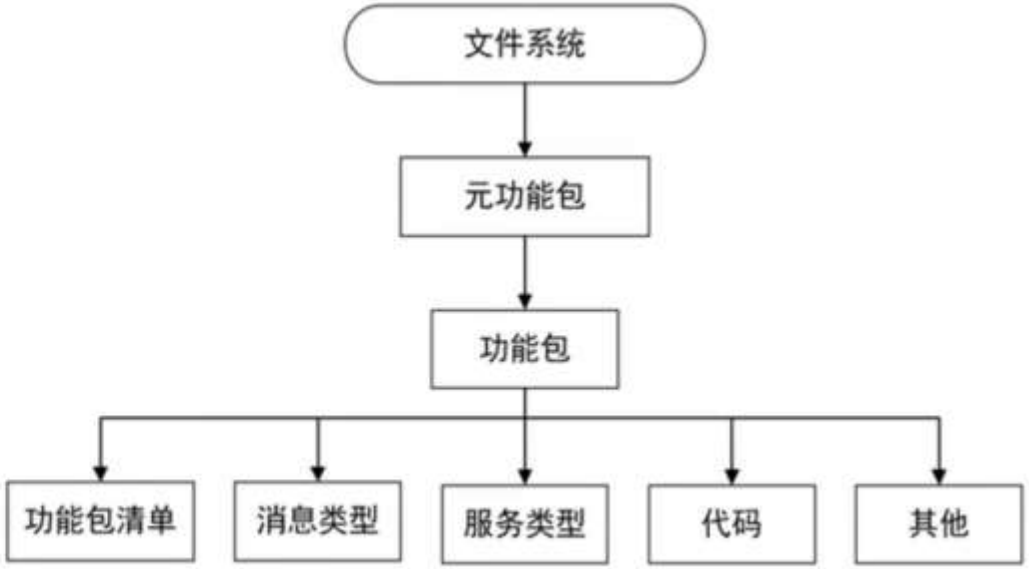


参数模型（全局字典）

1. ROS系统

- **功能包（Package）**
  - ROS软件中的基本单元，包含节点源码、配置文件、数据定义等
- **功能包清单（Package manifest）**
  - 记录功能包的基本信息，包含作者信息、许可信息、依赖选项、编译标志等
- **元功能包（Meta Packages）**
  - 组织多个用于同一目的的功能包





1. ROS系统

1）config：放置功能包中的配置文件，由用户创建，文件名可以不同。

2）include：放置功能包中需要用到的头文件。

3）scripts：放置可以直接运行的Python脚本。

4）src：放置需要编译的C++代码。

5）launch：放置功能包中的所有启动文件。

6）msg：放置功能包自定义的消息类型。

7）srv：放置功能包自定义的服务类型。

8）action：放置功能包自定义的动作指令。

9）CMakeLists.txt：编译器编译功能包的规则。

10）package.xml：功能包清单，图2-7是一个典型的功能包清单示例。

2. ROS实践

• Linux基本命令

➢ cd命令

- 语法：cd <目录路径>
- 功能：改变工作目录，若没有指定“目录路径”，则回到用户的主目录。

➢ pwd命令

- 语法：pwd
- 功能：此命令显示出当前工作目录的绝对路径。

➢ mkdir

- 语法：mkdir [选项] <目录名称>
- 功能：创建一个目录。

➢ ls

- 语法：ls [选项] [目录名称…]
- 功能：列出目录的内容。

➢ touch

- 语法：rouch [选项] [文件名称…]
- 功能：改变文件或目录时间。
- 更多的时候它会被用来快速创建一个空文件。

➢ sudo是linux系统管理指令，是允许系统管理员让普通用户执行一些或者全部的root命令的一个工具

➢ mv

- 语法：mv [选项] <源文件或目录> <目的文件或目录>
- 功能：为文件或目录改名或将文件由一个目录移入另一个目录中。

➢ cp

- 语法：cp [选项] <源文件名称或目录名称> <目的文件名称或目录名称>
- 功能：把给出的一个文件或目录拷贝到另一文件或目录中，或者把多个源文件复制到目标目录中。

➢ rm

- 语法：rm [选项] <文件名称或目录名称…>
- 功能：该命令的功能为删除一个目录中的一个文件或多个文件或目录，它也可以将某个目录及其下的所有文件及子目录均删除。对于链接文件，原有文件均保持不变。
- 递归删除 -r

➢ apt-get

- 语法：apt-get install
- **功能：主要用于自动从互联网的软件仓库中搜索、安装、升级、卸载软件或操作系统。**

➢ --help

- 语法：help
- 功能：查看指令的使用方法

2. ROS实践

ROS安装

1.添加 sources.list

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

2.添加公钥

sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

3.安装

sudo apt update

sudo apt install ros-melodic-desktop-full

4.初始化 rosdep

sudo rosdep init如果出错，运行：sudo apt install python-rosdep

rosdep update

5.设置环境

echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc

source ~/.bashrc

6.安装rosinstall

sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential

2. ROS实践

- 验证（运行小海龟）

1.roscore

2.rosrun turtlesim turtlesim_node

3.rosrun turtlesim turtle_teleop_key

4.rqt_graph

5.rosnode list

6.rostopic rostopic pub -r

7.rosmsg show

8.rosservic

9.rosbag

安装terminator： sudo apt-get install terminator

2. ROS实践

查看话题列表

```
$ rosnode list
```

发布话题消息

```
$ rostopic pub -r 10 /turtle1/cmd_vel geometry_msgs/Twist "linear:
  x: 1.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0"
```

发布服务请求

```
$ rosservice call /spawn "x: 5.0
y: 5.0
theta: 0.0
name: 'turtle2'"
```

2. ROS实践

- 话题记录与重复

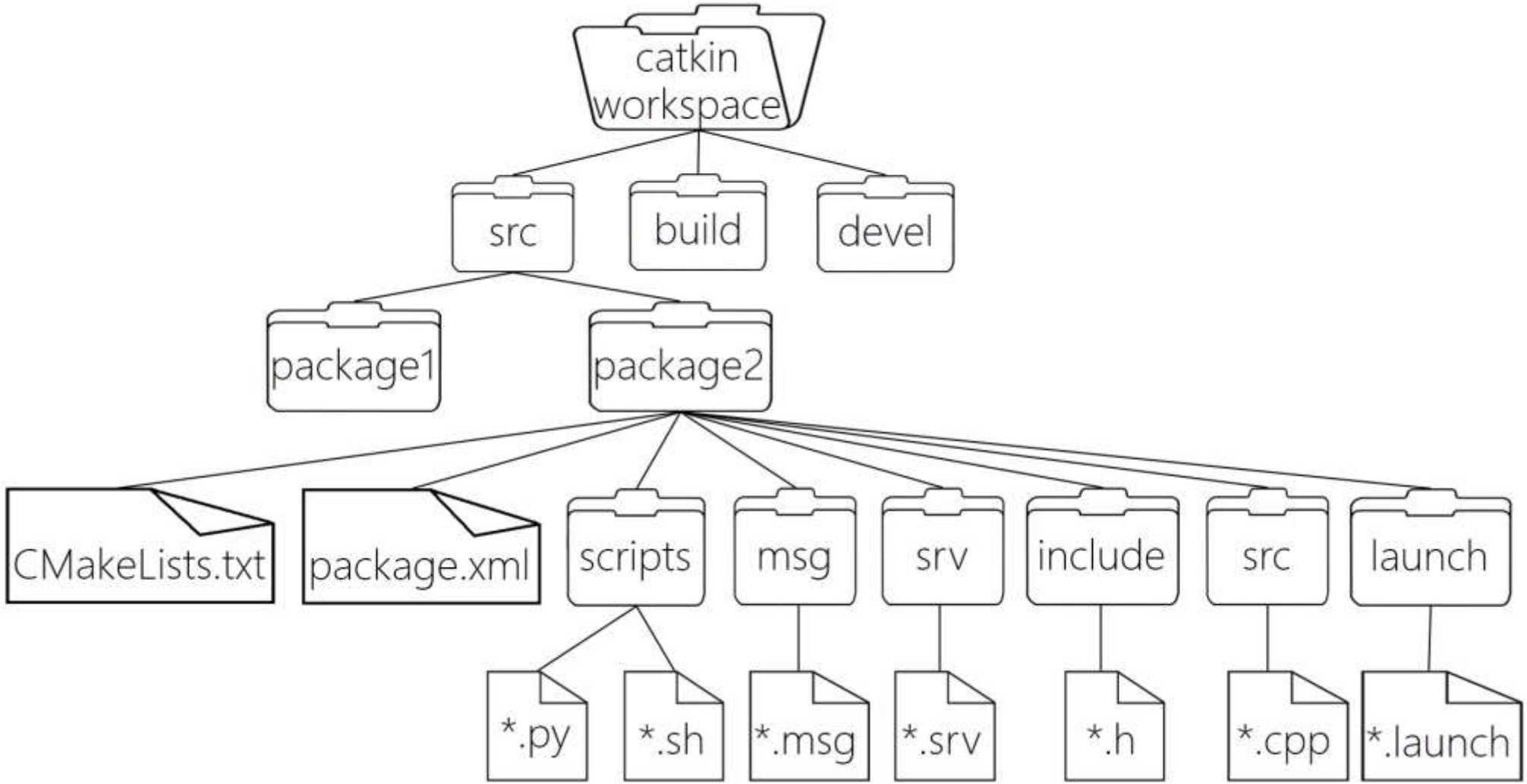
话题记录

```
$ rosbag record -a -O cmd_record
```

话题复现

```
$ rosbag play cmd_record.bag
```

1. ROS系统


catkin
workspace
src
build
devel
package1
package2
CMakeLists.txt
package.xml
scripts
msg
srv
include
src
launch
*.py
*.sh
*.msg
*.srv
*.h
*.cpp
*.launch

2. ROS实践

- 创建ROS工作空间

创建工作空间

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

编译工作空间

```
$ cd ~/catkin_ws/
$ catkin_make
```

设置环境变量

```
$ source devel/setup.bash
```

检查环境变量

```
$ echo $ROS_PACKAGE_PATH
```

2. ROS实践

- 创建功能包

echo "source **/home/hhch/catkin_ws/**devel/setup.bash" >> ~/.bashrc
source ~/.bashrc

$ catkin_create_pkg <package_name> [depend1] [depend2] [depend3]

创建功能包

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg test_pkg std_msgs rospy roscpp
```

编译功能包

```
$ cd ~/catkin_ws
$ catkin_make
$ source ~/catkin_ws/devel/setup.bash
```

同一个工作空间下，不允许存在同名功能包
不同工作空间下，允许存在同名功能包

2. ROS实践

- 话题

```
#!/usr/bin/env python

import rospy

from std_msgs.msg import Int32


rospy.init_node('topic_publisher')

pub = rospy.Publisher('counter', Int32)

rate = rospy.Rate(2)

count = 0
while not rospy.is_shutdown():
    pub.publish(count)
    count += 1
    rate.sleep()
```

2. ROS实践

```
#!/usr/bin/env python


import rospy
from std_msgs.msg import Int32


def callback(msg):
    print msg.data

rospy.init_node('topic_subscriber')

sub = rospy.Subscriber('counter', Int32, callback)

rospy.spin()
```

2. ROS实践

1. roscore
2. rosrun test_pkg topic_publisher.py
3. rosrun test_pkg topic_subscriber.py

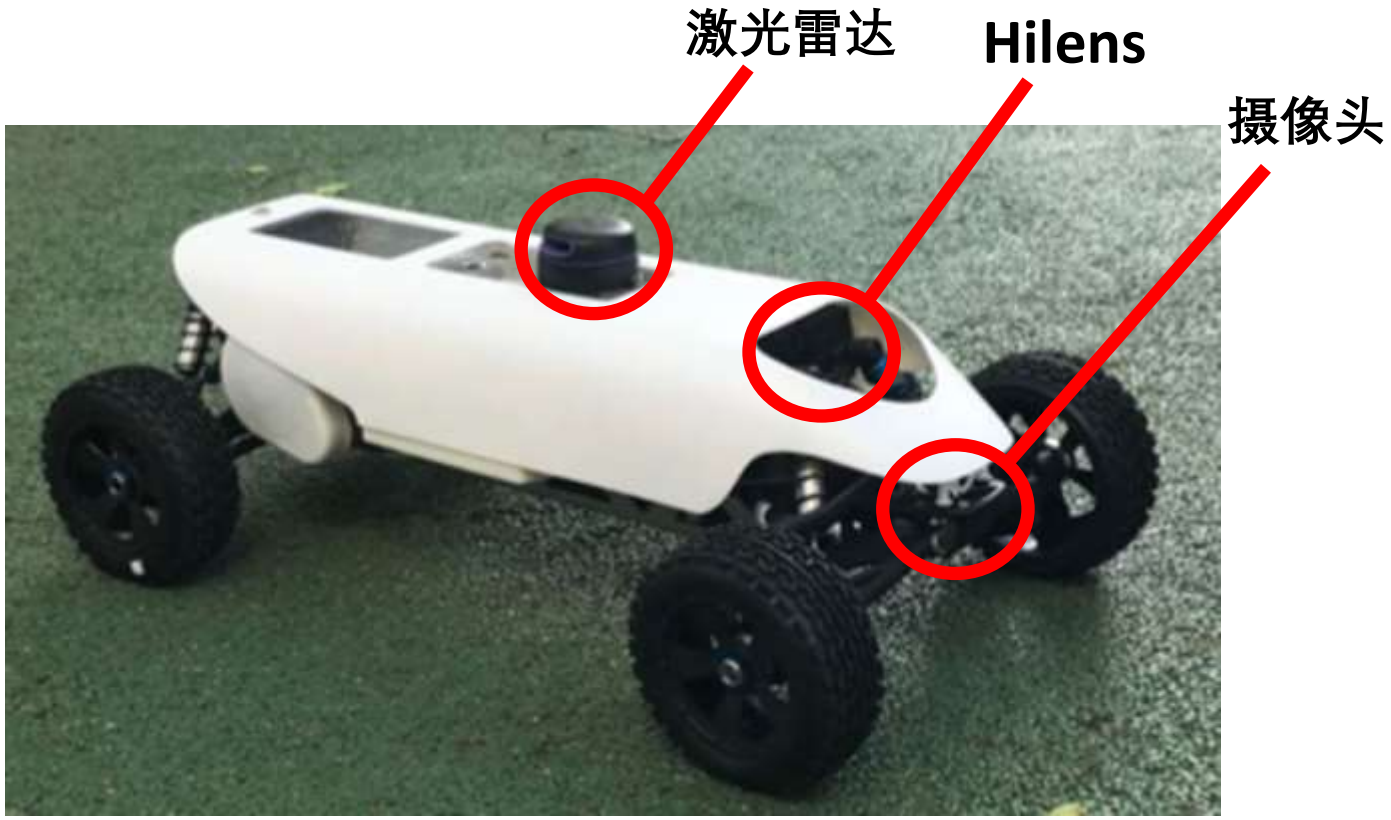
2. ROS实践

- roslaunch
- roslaunch package launch_file

```
<launch>
  <node name="talker" pkg="rospy_tutorials"
        type="talker.py" output="screen" />
  <node name="listener" pkg="rospy_tutorials"
        type="listener.py" output="screen" />
</launch>
```

```
roslaunch rospy_tutorials talker_listener.launch
```

无人小车



- **Hilens（人工智能）**

红黄绿灯、人行横道、限速标志识别

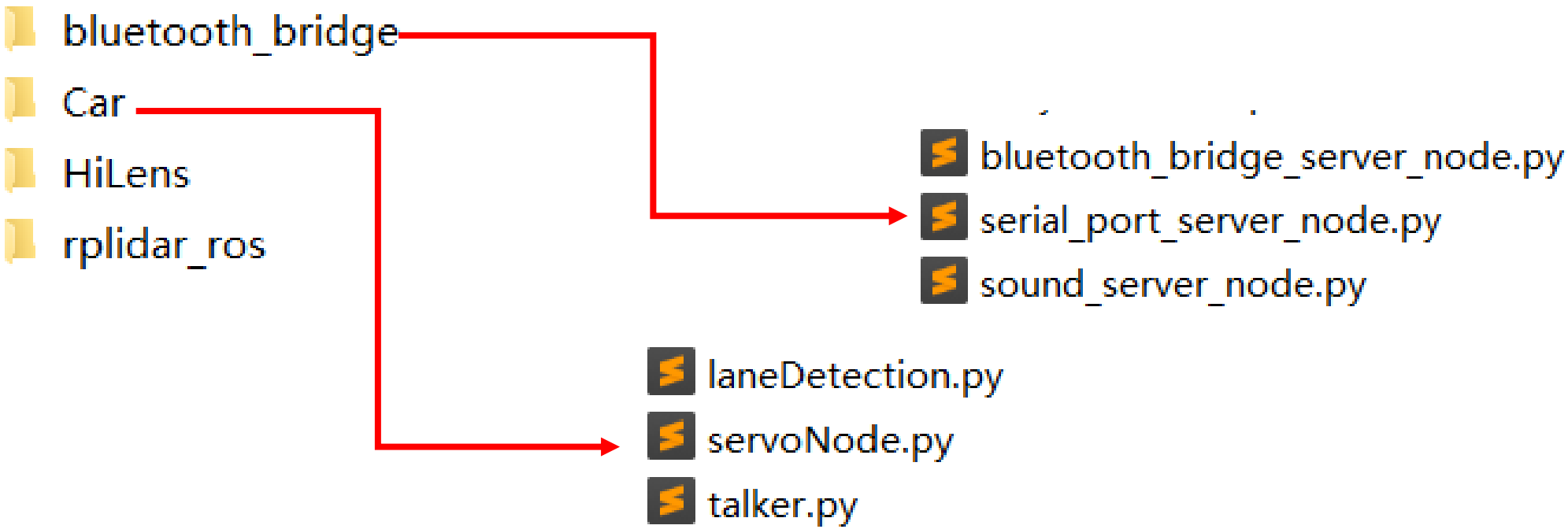
- **摄像头（OpenCV）**

车道线识别

- **激光雷达**

动态障碍物、标志物、SLAM建图

文件包


bluetooth_bridge
Car
HiLens
rplidar_ros
bluetooth_bridge_server_node.py
serial_port_server_node.py
sound_server_node.py
laneDetection.py
servoNode.py
talker.py

1. 底盘控制

- 底盘控制通信功能包（bluetooth_bridge）
  - serial_port_server_node.py（底盘主控进行串口通信）
  - bluetooth_bridge_server_node.py (RK3399开发板通过蓝牙与手机APP通信程序)
  - sound_server_node.py (RK3399开机提示程序)
  - bluetooth_bridge.launch

roslaunch bluetooth_bridge bluetooth_bridge.launch

1. 底盘控制

- **模式话题**:“/bluetooth/received/manual”（自动-0，手动-1）
- **方向话题**：”/auto_driver/send/direction”，取值范围0-100，其中50为直行方向（初始标定）
- **速度话题**：”/auto_driver/send/speed”，取值范围0-100
- **档位话题**：”/auto_driver/send/gear”，取值范围1-4，分别为D、N、P、R四个挡位

1. 底盘控制

- serial_port_server_node.py（订阅）

```
rospy.Subscriber(topic_from_bluetooth, String, callback_bluetooth)
rospy.Subscriber(topic_from_auto_driver_direction, Int32, callback_direction)
rospy.Subscriber(topic_from_auto_driver_speed, Int32, callback_speed)
rospy.Subscriber(topic_from_auto_driver_gear, Int32, callback_gear)
```

- servoNode.py（发布）

```
#Publisher 函数第一个参数是话题名称，第二个参数 数据类型，现在就是我们定义的msg 最后一个是缓冲区的大小
#queue_size: None（不建议）  #这将设置为阻塞式同步收发模式！
#queue_size: 0（不建议）#这将设置为无限缓冲区模式，很危险！
#queue_size: 10 or more  #一般情况下，设为10 。queue_size太大了会导致数据延迟不同步。

pub1 = rospy.Publisher('/bluetooth/received/manul', Int32 , queue_size=10)
pub2 = rospy.Publisher('/auto_driver/send/direction', Int32 , queue_size=10)
pub3 = rospy.Publisher('/auto_driver/send/speed', Int32 , queue_size=10)
pub4 = rospy.Publisher('/auto_driver/send/gear', Int32 , queue_size=10)

manul=0        # 0 - Automatic(自动); 1 - Manual (手动操控)
speed=20       # SPEED (0~100之间的值)
direction=50   # 0-LEFT-50-RIGHT-100 (0-49:左转, 50:直行, 51~100:右转)
gear=1         # 1 - DRIVE, 2 - NEUTRAL, 3 - PARK, 4 - REVERSE
               # 1:前进挡 2:空挡 3:停车挡 4:倒挡
```

1. 底盘控制

- **直行角度标定**

直行右偏：

把右边这个蓝色的杆转动一下，使其变长。或者让左边的杆变短。

直行左偏：

把右边这个蓝色的杆转动一下，使其变短。或者让左边的杆变长。

1. 底盘控制

- 转角控制

```
_servoCmdMsg = msg.angular.z * angularScale + 90
global servodata
servodata = min(max(0, _servoCmdMsg), 180)
servodata=100-servodata*100/180
```

- angular.z 通过车道线识别计算出的转角
- angularScale =6 （可以修改）

1. 底盘控制

- 惯性导航IMU（JY901）IIC接口

aX（IMU加速度X）："/vcu/aX"
aY（IMU加速度Y）："/vcu/aY"
aZ（IMU加速度Z）："/vcu/aZ"
alphaX（IMU角速度X）："/vcu/alphaX"
alphaY（IMU角速度Y）："/vcu/alphaY"
alphaZ（IMU角速度Z）："/vcu/alphaZ"
BX（IMU磁场X）："/vcu/BX"
BY（IMU磁场Y）："/vcu/BY"
BZ（IMU磁场Z）："/vcu/BZ"
thetaX（IMU角度X）："/vcu/thetaX"
thetaY（IMU角度Y）："/vcu/thetaY"
thetaZ（IMU角度Z）："/vcu/thetaZ"

```
vcu_aX_pub.publish(unpack_data1[3])
vcu_aY_pub.publish(unpack_data1[4])
vcu_aZ_pub.publish(unpack_data1[5])
vcu_alphaX_pub.publish(unpack_data1[6])
vcu_alphaY_pub.publish(unpack_data1[7])
vcu_alphaZ_pub.publish(unpack_data1[8])
vcu_BX_pub.publish(unpack_data1[9])
vcu_BY_pub.publish(unpack_data1[10])
vcu_BZ_pub.publish(unpack_data1[11])
vcu_thetaX_pub.publish(unpack_data1[12])
vcu_thetaY_pub.publish(unpack_data1[13])
vcu_thetaZ_pub.publish(unpack_data1[14])
```

serial_port_server_node.py

1. 底盘控制

- 实际参数

ActualMotorSpeed（实际电机转速）："/vcu/ActualMotorSpeed"

ActualVehicleMode（实际挡位信号）："/vcu/ActualVehicleMode"

ActualVehicleDirection（实际方向）："/vcu/ActualVehicleDirection"

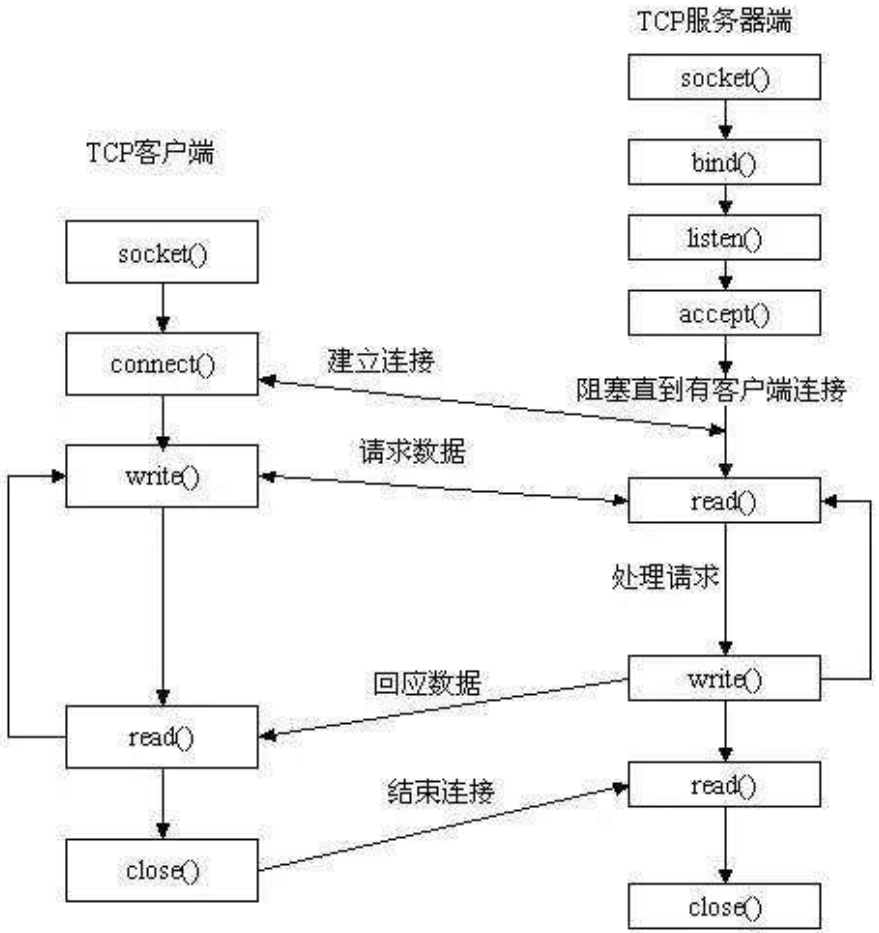
SuperSonicDistance（超声波距离）："/vcu/SupersonicDistance"

```
vcu_ActualMotorSpeed_pub.publish(unpack_data3[0])
vcu_ActualVehicleMode_pub.publish(unpack_data1[0])
vcu_ActualVehicleDirection_pub.publish(unpack_data1[1])
vcu_SupersonicDistance_pub.publish(unpack_data1[2])
```

serial_port_server_node.py

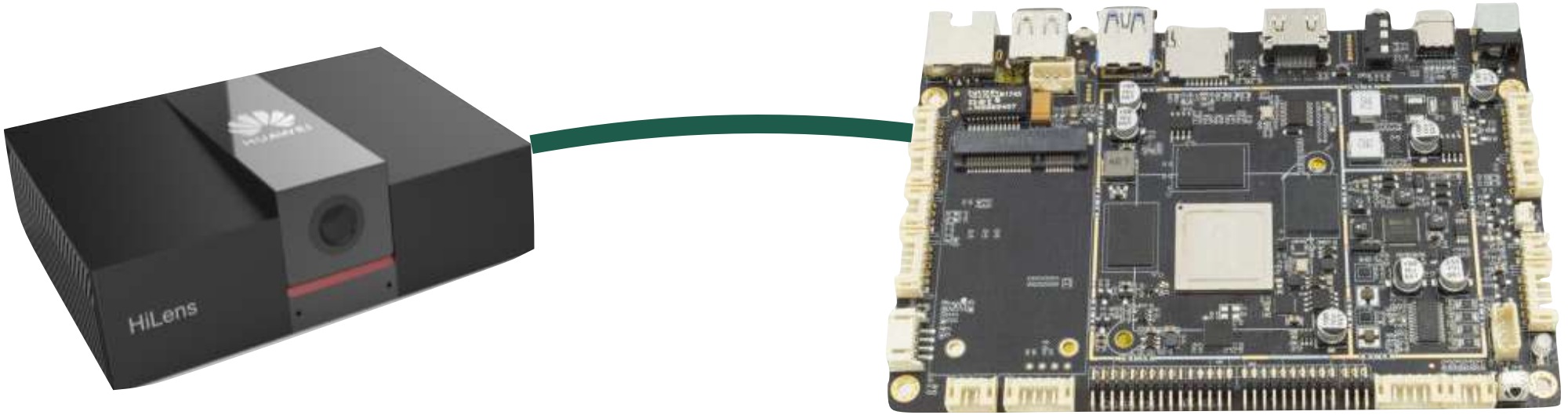
Socket 通信

网络上不同的计算机，可以使用使用网络套接字（socket）进行通信。



Socket 通信

网络上不同的计算机，可以使用使用网络套接字（socket）进行通信。



在无人车上，我们使用网线对 HiLens **和 主控板**进行连接。

HiLens 与主控板通信

HiLens 端

index.py utils.py

主控板端

talker.cpp

无人车运动控制

- KinematicCtrl.py 讲解

谢谢！