第 3 章 第三方工具库

ROS2 是开源的，是介于操作系统和应用软件之间的次级操作系统，它提供类似操作系统所提供的功能，包含硬件抽象描述、底层驱动程序管理、共用功能的执行、程序间消息传递、程序发行包管理；ROS2 也提供对一些第三方工具程序和库的支持。与 ROS 同样，ROS2 的首要设计目标是在机器人研发领域提高代码复用率。本章将介绍 ROS2 计划支持的第三方库，包括 orocos_kinematics_dynamics、POCO、urdfdom、vision_opencv、PCL、MoveIt 等，这些库也是 ROS 常用的工具库。

3.1 orocos_kinematics_dynamics 库

orocos_kinematics_dynamics 依赖于由 OROCOS 项目分发的运动学和动力学库(KDL)。它是一种依赖 C++ 和 pykdl 的元语言包，其中 pykdl 包含生成的 Python 绑定。KDL 具有广泛支持几何图元的特点，如支持点、框架。KDL 支持动力学参数(如惯性)，支持运动学和动力学求解器、瞬时运动、运动轨迹。KDL 具有实时安全操作功能，可保证在确定的时间内完成任务。除此之外，KDL 还支持 Python 包，支持 OROCOS/RTT 的 Typekits 和 transport-kits，可集成到 ROS。orocos_kinematics_dynamics 目前处于维护状态，遵守 LGPL 的许可证可以从 Git 下载。

1. orocos_kinematics_dynamics 编译环境

1) 软件需求：Eigen2、Sip 4.7.9、Python、Cmake 2.6

2) 支持平台：Linux、Windows、Mac

3) 安装过程

(1) 使用 ament 在 ROS 构建源代码。

首先安装 ament，创建目录，执行如下命令：

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws
```

(2) 下载代码。

需要用到版本控制工具——vcs，可以参考 https://github.com/dirk-thomas/vcstool 进行安装，执行如下命令：

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

```
sudo apt-key adv --keyserver hkp://pool.sks-keyservers.net --recv-key 0xB01FA116
sudo apt-get update
sudo apt-get install python-vcstool
```

(3) 安装依赖包,执行如下命令:

```
sudo apt-get update
sudo apt-get install git wget
sudo apt-get install build-essential cppcheck cmake libopencv-dev libpoco-dev libpocofoundation9v5 libpocofoundation9v5-dbg python-empy python3-dev python3-empy python3-nose python3-pip python3-setuptools python3-vcstool
sudo apt-get install clang-format pydocstyle pyflakes python3-coverage python3-mock python3-pep8 uncrustify
sudo apt-get install libboost-chrono-dev libboost-date-time-dev libboost-program-options-dev libboost-regex-dev libboost-system-dev libboost-thread-dev
sudo apt-get install libboost-all-dev libpcre3-dev zlib1g-dev  python-empy python-pkg-resources
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws
wget https://raw.githubusercontent.com/ros2/ros2/release-latest/ros2.repos
vcs import src < ros2.repos
```

(4) 下载 ament 的代码,执行如下命令:

```
vcs import ~/ros2_ws/src < ros2.repos
```

(5) 开始编译,执行如下命令:

```
$ src/ament/ament_tools/scripts/ament.py build --build-tests --symlink-install
```

编译完成使用下面命令测试:

```
src/ament/ament_tools/scripts/ament.py test
```

(6) 设置环境变量。ament 编译完成后,所有声明的文件都放到 ros2_ws 工作目录下的 install 文件夹里生成的命令在 bin 文件夹下,如果需要在终端调用这些命令,需要设置环境变量,执行如下命令:

```
install/local_setup.bash
```

(7) 编译自己的功能包。将需要编译的源代码复制到 ros2_overlay_ws 目录,执行如下命令:

```
cd ~/ros2_overlay_ws
ament build --cmake-args -DCMAKE_BUILD_TYPE=Debug
```

编译完成后也会在工作区产生一个 install 文件,里边的目录结构和之前编译 ament 所生成的一样,运行的方法一样,先设置环境变量,然后运行,运行的程序在 overlay 工作区。

2. 安装 orocos_kinematics_dynamics

进入 orocos_kdl 目录,执行命令: mkdir <kdl-dir>/build; cd <kdl-dir>/build。启动 ccmake,编译并安装,执行命令: make; make check; make install。

3.2 POCO 库

便携式组件(Portable Components,POCO)库是一个 C++库集合,在概念上类似于 Java 类库或微软的.NET 框架。便携式组件主要解决经常遇到的实际问题。便携式组件库 100%兼容 ANSI/ISO C++标准,基于并且完善了 C++标准库/ STL,高度可移植,可在嵌入式、服务器等不同的平台上使用。

POCO(Version POCO C++Libraries 1.9.0-all)完全兼容 C++标准库,并且填补 C++标准库的许多空白功能。POCO 构建以网络为中心的、跨平台的 C++软件开发,其模块化、高效的设计,使 POCO C++库能极大的提高开发效率。POCO 由 4 个核心库和多个附加库组成。核心库包括基础库、XML、UTL 和 NET 库。两个附加库是 NetSSL 和 Data,NetSSL 为网络库中的网络类提供 SSL 支持,而 Data 库则是一个统一访问不同 SQL 数据库的库。POCO 的目的是构建以网络为中心的、跨平台的 C++软件开发,使构建应用程序的过程变得简单、有趣。

1. 基础库

基础库(The Foundation Library)是 POCO 的核心。它包含平台抽象层,以及常用的实用工具类和函数。基础库包含基本类型、常用工具函数、错误处理及调试的实用工具,提供了许多用于内存管理的类,包括基于引用计数的智能指针,用于缓冲区管理和内存池的类。POCO 包含许多字符串处理功能,其中包括修剪字符串、执行不区分大小写的比较和实例转换。POCO 支持 Unicode 文本也可以以不同的字符编码(包括 UTF-8 和 UTF-16)。支持格式化和解析数字。还提供了基于 PCRE 库(http://www.pcre.org)的正则表达式。POCO 还提供处理各种日期和时间的类、访问文件的类,如 Poco::File、Poco::Path、Poco::DirectoryIterator。

在许多应用中,应用程序内部需要互相通知状态的变化,即内部通信。POCO 提供 Poco::NotificationCenter、Poco::NotificationQueue 和事件使得应用程序内部互相通知状态变化变得非常容易。下面的示例演示如何使用 POCO 事件完成应用程序内部通信。在这个例子中,产生事件源的类有一个 public 的 theEvent 事件,它具有 int 类型的参数。订阅方可以通过调用“操作符+=”来订阅,并通过调用“操作符-=”取消订阅,取消订阅过程传递指向对象的指针和指向成员函数的指针。事件可以通过调用 operator()来触发,这种调用方式与 Source::fireEvent()中所做的相似。POCO 中 Poco::BinaryReader 和 PoCO::BinaryWriter 将二进制数据写入流,自动透明地处理字节顺序问题。

```
#include "Poco/BasicEvent.h"
#include "Poco/Delegate.h"
#include <iostream>
using Poco::BasicEvent;
using Poco::Delegate;
class Source
{
public:
```

```
    BasicEvent < int > theEvent;

    void fireEvent(int n)
    {
        theEvent(this, n);
    }
};
class Target
{
public:
    void onEvent(const void * pSender, int& arg)
    {
        std::cout << "onEvent: " << arg << std::endl;
    }
};
int main(int argc, char * * argv)
{
    Source source;
    Target target;
    source.theEvent += Delegate < Target, int >(
        &target, &Target::onEvent);
    source.fireEvent(42);
    source.theEvent -= Delegate < Target, int >(
        &target, &Target::onEvent);
    return 0;
}
```

针对在复杂的多线程应用程序中发现错误非常困难的情况，POCO 提供了详细的日志信息。POCO 的日志框架很强大且可扩展，它支持对不同通道的过滤、路由和日志消息的格式化。日志消息可以写入控制台、文件、Windows 事件日志、UNIX 系统日志守护进程或网络。如果现有 POCO 提供的信道不足够，则很容易用新类扩展日志框架。为了在运行时加载和卸载共享库，POCO 提供基础 POCO::SysDeLab 类库类，它是 POCO::class 加载器类模板支撑框架，允许动态加载和卸载的 C++类。

POCO 包含多层次的多线程抽象。包括线程类和通常的同步原语（POCO::Mutex，POCO::ScopedLock，POCO::事件，POCO::信号量，POCO::RWLOK）。支持 POCO::线程池类和对线程的本地存储，支持高级抽象对象。活动对象在自己的线程中执行方法，这使得异步成员函数调用成为可能。下面的示例演示了如何在 POCO 中实现异步成员函数调用。ActiveAdder 类定义了一个 add( )，其具体功能由 addImpl( )实现，在主函数中调用 add( )，其返回为 Poco::ActiveResult。

```
#include "Poco/ActiveMethod.h"
#include "Poco/ActiveResult.h"
#include < utility >
#include < iostream >
using Poco::ActiveMethod;
using Poco::ActiveResult;
class ActiveAdder
{
```

```
public:
    ActiveAdder(): add(this, &ActiveAdder::addImpl)
    {
    }
    ActiveMethod<int, std::pair<int, int>, ActiveAdder> add;
private:
    int addImpl(const std::pair<int, int>& args)
    {
        return args.first + args.second;
    }
};
int main(int argc, char** argv)
{
    ActiveAdder adder;
    ActiveResult<int> sum = adder.add(std::make_pair(1, 2));
    // do other things
    sum.wait();
    std::cout << sum.data() << std::endl;
    return 0;
}
```

2. XML 库

POCO 的 XML 库为 XML 的读取、处理和编写提供支持。XML 库基于开源 XML 解析器库 Expat(http://www. LibExab. org)。XML 库使用 STD::String 处理字符串,其中字符采用 UTF-8 编码。这使得 XML 库与应用程序的其他部分变得容易通信。POCO 的 XML 库支持行业标准 SAX(版本 2)和 DOM 接口。SAX 是 XML 的简单 API(http://www. xxPosij. org),定义了一个基于事件的接口读取 XML。SAX 接口构建在 ExpAt 之上,DOM 实现构建在 SAX 接口之上。基于 SAX 的 XML 解析器读取 XML 文档,并在遇到元素、字符数据时通知应用程序。SAX 解析器不需要将完整的 XML 文档加载到内存中,因此可以有效地解析大型 XML 文件。相比之下,DOM(文档对象模型,http://www. W3. OR/DOM/)使用树形对象层次结构为应用程序遍历 XML 文档。POCO 提供的 DOM 解析器必须将整个文档加载到内存中。为了减少 DOM 文档的内存占用,POCO DOM 实现使用字符串池,只存储元素和属性名称等频繁出现的字符串。

3. Util 库

Util 库包含一个用于创建命令行和服务器应用程序的框架,包含处理命令行参数(验证、绑定、配置属性等)和管理配置信息。支持不同的配置文件格式,包括 Windows 风格的 ini 文件和注册表,Java 风格的属性文件、XML 文件。对于服务器应用程序,该框架为 Windows 服务和 UNIX 守护进程提供透明支持。

4. 网络库

POCO 的网络库(Net library)使得编写基于网络的应用程序变得容易。网络库的最底层包含套接字类、支持 TCP 流和服务器套接字、UDP 套接字、多播套接字、ICMP、原始套接字。如果应用程序需要安全套接字,可以调用 NETSSL 库,NETSSL 库使用 OpenSSL 实现。HTTP 服务器实现基于多线程 POCO::NET::TCPServer 类及其支持框架。在客户端,网络库提供用于与 HTTP 服务器交互的类,用 FTP 协议发送和接收文件,使用 SMTP

发送邮件消息，用 POP3 服务器接收邮件。

5. 基于 POCO 库实现简单 HTTP 服务器

下面的示例演示使用 POCO 库实现简单 HTTP 服务器。服务器返回显示当前日期和时间的 HTML 文档。HTTP 服务器框架定义 TimeRequestHandler 用于处理服务请求，返回包含当前日期和时间的 HTML 文档。对于每个请求，日志框架记录请求的内容。服务器框架采用了工厂设计模式，由 TimeRequestHandlerFactory 产生 TimeRequestHandler 作为工厂的一个实例。通过覆盖 defineOptions()，HTTPTimeServer 定义了命令行参数。HTTPTimeServer 可以读取默认配置文件并且可以在 HTTP 服务器启动前获取 main()中的配置属性。

```
#include "Poco/Net/HTTPServer.h"
#include "Poco/Net/HTTPRequestHandler.h"
#include "Poco/Net/HTTPRequestHandlerFactory.h"
#include "Poco/Net/HTTPServerParams.h"
#include "Poco/Net/HTTPServerRequest.h"
#include "Poco/Net/HTTPServerResponse.h"
#include "Poco/Net/HTTPServerParams.h"
#include "Poco/Net/ServerSocket.h"
#include "Poco/Timestamp.h"
#include "Poco/DateTimeFormatter.h"
#include "Poco/DateTimeFormat.h"
#include "Poco/Exception.h"
#include "Poco/ThreadPool.h"
#include "Poco/Util/ServerApplication.h"
#include "Poco/Util/Option.h"
#include "Poco/Util/OptionSet.h"
#include "Poco/Util/HelpFormatter.h"
#include <iostream>
using Poco::Net::ServerSocket;
using Poco::Net::HTTPRequestHandler;
using Poco::Net::HTTPRequestHandlerFactory;
using Poco::Net::HTTPServer;
using Poco::Net::HTTPServerRequest;
using Poco::Net::HTTPServerResponse;
using Poco::Net::HTTPServerParams;
using Poco::Timestamp;
using Poco::DateTimeFormatter;
using Poco::DateTimeFormat;
using Poco::ThreadPool;
using Poco::Util::ServerApplication;
using Poco::Util::Application;
using Poco::Util::Option;
using Poco::Util::OptionSet;
using Poco::Util::OptionCallback;
using Poco::Util::HelpFormatter;
class TimeRequestHandler: public HTTPRequestHandler
{
public:
```

```
    TimeRequestHandler(const std::string& format): _format(format)
    {
    }
    void handleRequest(HTTPServerRequest& request,
                       HTTPServerResponse& response)
    {
        Application& app = Application::instance();
        app.logger().information("Request from "
            + request.clientAddress().toString());
        Timestamp now;
        std::string dt(DateTimeFormatter::format(now, _format));

        response.setChunkedTransferEncoding(true);
        response.setContentType("text/html");
        std::ostream& ostr = response.send();
        ostr << "<html><head><title>HTTPTimeServer powered by "
                    "POCO C++Libraries</title>";
        ostr << "<meta http-equiv=\"refresh\" content=\"1\"></head>";
        ostr << "<body><p style=\"text-align: center; "
                "font-size: 48px;\">";
        ostr << dt;
        ostr << "</p></body></html>";
    }
private:
    std::string _format;
};
class TimeRequestHandlerFactory: public HTTPRequestHandlerFactory
{
public:
    TimeRequestHandlerFactory(const std::string& format):
        _format(format)
    {
    }
    HTTPRequestHandler* createRequestHandler(
        const HTTPServerRequest& request)
    {
        if (request.getURI() == "/")
            return new TimeRequestHandler(_format);
        else
            return 0;
    }
private:
    std::string _format;
};

class HTTPTimeServer: public Poco::Util::ServerApplication
{
public:
    HTTPTimeServer(): _helpRequested(false)
    {
    }
```

```
    ~HTTPTimeServer()
    {
    }
protected:
    void initialize(Application& self)
    {
        loadConfiguration();
        ServerApplication::initialize(self);
    }
    void uninitialize()
    {
        ServerApplication::uninitialize();
    }
    void defineOptions(OptionSet& options)
    {
        ServerApplication::defineOptions(options);
        options.addOption(
        Option("help", "h", "display argument help information")
            .required(false)
            .repeatable(false)
            .callback(OptionCallback<HTTPTimeServer>(
                this, &HTTPTimeServer::handleHelp)));
    }
    void handleHelp(const std::string& name,
                    const std::string& value)
    {
        HelpFormatter helpFormatter(options());
        helpFormatter.setCommand(commandName());
        helpFormatter.setUsage("OPTIONS");
        helpFormatter.setHeader(
            "A web server that serves the current date and time.");
        helpFormatter.format(std::cout);
        stopOptionsProcessing();
        _helpRequested = true;
    }
    int main(const std::vector<std::string>& args)
    {
        if (!_helpRequested)
        {
            unsigned short port = (unsigned short)
                config().getInt("HTTPTimeServer.port", 9980);
            std::string format(
                config().getString("HTTPTimeServer.format",
                                   DateTimeFormat::SORTABLE_FORMAT));
            ServerSocket svs(port);
            HTTPServer srv(new TimeRequestHandlerFactory(format),
                svs, new HTTPServerParams);
            srv.start();
            waitForTerminationRequest();
            srv.stop();
        }
```

```
        return Application::EXIT_OK;
    }
private:
    bool _helpRequested;
};
int main(int argc, char * * argv)
{
    HTTPTimeServer app;
    return app.run(argc, argv);
}
```

3.3　URDF

URDF 使用 XML 在 ROS 中描述机器人模型。ROS 中的 URDF 功能包包含一个 URDF 的 C++解析器。URDF 支持 BSD 许可证。

3.3.1　URDF 语法规范

URDF 使用 XML 在 ROS 中描述机器人模型，URDF 使用的主要 XML 元素包括 link、transmission、joint、gazebo、sensor、model_state、model 等。其语法规范如下：

1. link

link 元素描述了连杆的运动和动力特性。link 示意图如图 3-1 所示。

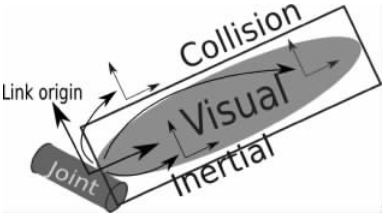

图 3-1　link 示意图

link 示例如下所示。

```
<link name = "my_link">
   <inertial>
     <origin xyz = "0 0 0.5" rpy = "0 0 0"/>
     <mass value = "1"/>
     <inertia ixx = "100"  ixy = "0"  ixz = "0" iyy = "100" iyz = "0" izz = "100" />
   </inertial>
   <visual>
     <origin xyz = "0 0 0" rpy = "0 0 0" />
     <geometry>
       <box size = "1 1 1" />
     </geometry>
     <material name = "Cyan">
```

```
        <color rgba = "0 1.0 1.0 1.0"/>
      </material>
    </visual>
    <collision>
      <origin xyz = "0 0 0" rpy = "0 0 0"/>
      <geometry>
        <cylinder radius = "1" length = "0.5"/>
      </geometry>
    </collision>
  </link>
```

2. transmission

transmission 是用于描述制动器和关节之间关系的 URDF 扩展。transmission 支持齿轮比和平行连杆的概念。transmission 变换工作/流量，使得它们的功率保持不变。多个致动器可以通过复杂传输连接到多个关节。

transmission 元素示例如下所示。

```
<transmission name = "simple_trans">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name = "foo_joint">
    <hardwareInterface>EffortJointInterface</hardwareInterface>
  </joint>
  <actuator name = "foo_motor">
    <mechanicalReduction>50</mechanicalReduction>
    <hardwareInterface>EffortJointInterface</hardwareInterface>
  </actuator>
</transmission>
```

3. joint

joint 描述关节的运动学和动力学特性，并且还指定关节的安全极限。

joint 元素的示例如下所示。

```
<joint name = "my_joint" type = "floating">
    <origin xyz = "0 0 1" rpy = "0 0 3.1416"/>
    <parent link = "link1"/>
    <child link = "link2"/>
    <calibration rising = "0.0"/>
    <dynamics damping = "0.0" friction = "0.0"/>
    <limit effort = "30" velocity = "1.0" lower = " - 2.2" upper = "0.7" />
    <safety_controller k_velocity = "10" k_position = "15" soft_lower_limit = " - 2.0" soft_
upper_limit = "0.5" />
 </joint>
```

4. gazebo

gazebo 元素描述模拟特性，如阻尼、摩擦等。

5. sensor

sensor 元素可描述传感器的基本特性，如描述视觉传感器(照相机/光线传感器)的基本特性。

sensor 元素描述照相机的示例如下所示。