《ROS机器人开发技术》

课程讲稿

|  |  |  |
| --- | --- | --- |
| 课程名称 | ： | ROS机器人开发技术 |
| 教师姓名 | ： | XXX |
| 提交时间 | ： | 2018年7月X日 |

中国大学MOOC制

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 课程  名称 | 《ROS机器人开发技术》 | | 章 | | 第一章 | | | 课程类型 |
| 节 | | 第一节 | | | 授课(√ )  实训( ) |
| 名称 | Rospy高级用法 | | | | | | |
| 教师 |  | 课时 | | | | 1课时 | | |
| 参考  资料 | 1. Roswiki | | | | | | | |
| 教学  目的  要求 | 掌握：工作空间及架构，git使用操作，用python编写ros节点程序 | | | | | | | |
| 教学  重点  难点 |  | | | PPT页面 | | | 时间分配 | |
| 教学重点 | | | | | | | |
| 1. Catkin工作空间 | | | 5~9页 | | | 15分钟 | |
| 1. Github上使用git操作 | | | 17~26页 | | | 15分钟 | |
| 1. 用python编写service节点 | | | 63~79页 | | | 15分钟 | |

|  |  |  |
| --- | --- | --- |
| 教学方法 | | |
| 本授课以课堂讲授为主，与课堂演示方式相结合 | | |
| 教学内容 | 操作演示 | 知识点 |
| **PPT第1页：**  本课程接下来所要讲述的内容分为八个部分  **1caktin文件空间的架构和布局**  2 **git常见操作**  3. **github上面git常见操作**  4. **vim高级操作**  5. **makefile和make的详细介绍**  6. **cmake和cmakelists.txt的详细介绍**  7.**用python编写ros service节点**  8. **Vscode高级操作总结** |  |  |
| **PPT第2页:**  **接下来我们先将第一部分**  **caktin文件空间的架构和布局** |  |  |
| **PPT第3页:**  **接下来我们讲述系统架构的分类，分为三个级别：** |  |  |
| **PPT第4页**  **它们分别是**  **文件系统级**  **计算图级**  **开源社区级** |  |  |
| **PPT第5页:**  **我们接下来介绍catkin工作空间的文件结构。** |  |  |
| **PPT第6页:**  **Catkin工作空间包括三块内容他们分别是：**  **src（pkg源代码包）、build（cmake和catkin缓存和中间文件）以及devel（目标文件）** |  |  |
| **PPT第7页:**  **我们对src目录所包含内容再进行细分如图所示** |  |  |
| **PPT第8-9页:**  **这两页介绍src目录下具体内容的格式** |  |  |
| **PPT第10页:**  接下来我们介绍git常见操作。 |  |  |
| **PPT第11页：**  接下来我们介绍什么是git |  |  |
| **PPT第12页**  Git是一个版本控制系统，用来追踪计算机文件的变化的工具，也是一个供多人使用的协同工具。它是一个分布式的版本控制系统。简单来说，就是你要和你的伙伴一起完成一项任务，但是你们要互相交换修改，查看自己的历史版本等。 |  |  |
| **PPT第13页**  **版本控制系统就是帮助我们做这个的。由于Git是一个分布式文件控制系统，所以一般采用一个服务器方便大家交换修改用的。每个人本地都有一个版本库，保存自己的历史版本，然后每个人可以把自己的修改提交到服务器上，被人就可以获取你的修改了。因此，Git的版本库（Repository）对于每个人来说有两个，一个是远程的，一个是本地的。** |  |  |
| **PPT第14页**  接下来介绍那么我们为什么学习git？ |  |  |
| **PPT第15页**  Git 是一个软件，或者说是一种服务，可以把它集成到其他软件中，于是就有了享誉全世界的 GitHub 社区，还有中国的 码云社区。这里是程序员的天堂，每个人可以把自己的代码公开，让更多的人看到并参与项目中来，当然也可以不公开，将代码托管到上面来作为项目的中心仓库使用。  GitHub 是一片代码世界的蓝海，里面有无穷无尽的宝藏，也有很多的大牛 |  |  |
| **PPT第16页:**  我们接下来介绍git常见指令 |  |  |
| **PPT第17页:**  git clone  将远程仓库中的项目下载到本地仓库  在浏览器中打开github或公司的gitlab，进入项目，复制项目的git地址（如果公司的gitlab不支持SSH，因此请注意选择HTTPS的地址）进入到工作目录（通常我们会把各个项目放在同一个文件夹，例如workingDir啥的，当然，你可以在任意目录做git操作），敲下命令git clone+git地址 |  |  |
| **PPT第18页:**  **git checkout**  切换分支或将修改的文件恢复  clone下代码后，代码默认会在master分支，我们需要把代码切换到一个自己的分支，这样的话，我们随便怎样修改，怎样提交，都不会影响到master或其它分支。  git checkout -b feature-test  这个时候，feature-test分支和master分支的代码是完全一样的，但是此后，你做的所有修改都不会影响master分支，只会在feature-test分支上往下进行。 |  |  |
| PPT第19页: git status  显示当前工作目录的文件状态  用法如图所示 |  |  |
| PPT第20页: git add  **将一个普通文件交给git管理或者将修改过的文件添加到git的索引中。当我们使用git commit时，git将依据索引库中的内容来进行文件的提交**  **用法：**  git add index.html  如果我们一次修改比较多的文件，需要把所有修改的文件都添加到索引库，可以使用：  git add –all  如果我们误把一个文件添加进了索引库，可以使用以下命令将其“拉回”暂存区：  git reset HEAD index.html  如果我们想把一个文件恢复成修改前的状态，可以使用以下命令，就撤销了我们所有的修改：  git checkout index.html |  |  |
| PPT第21页: git commit  将所有添加到索引库的文件提交到本地仓库  用法：  最简单的commit操作如下：  git commit -m "upd" |  |  |
| PPT第22页: git push  将本地仓库的修改推送到远程仓库  用法：  git push origin feature-test  当我们做完功能后，要记得使用git push将自己的改动推送到远程代码仓库。 |  |  |
| PPT第23页 git fetch  将远程代码仓库的代码下载到本地  用法：  本地仓库维护这一堆“remote-tracking branches”，翻译过来叫“远程跟踪分支”，对应着本地仓库里remote/master/remote/develop这些分支。  fetch操作默认只会fetchorigin仓库的分支，如果要fetch其它仓库的分支，需要在后面加上仓库名：  git fetch repo-caiyou  通常，我们的feature分支在完成使命后会被管理员删除掉，而我们各自的本地却不知道，硬盘空间吃紧。所以，如果我们想在管理员删除了远端的某些分支的时候，将自己本地对应的“远程跟踪分支”删除掉，需要在fetch操作时加一个-p参数：  git fetch -p |  |  |
| **PPT第24页：**  git merge  将一个或多个分支合并  用法：  比如我们当前在develop分支上，git fetch之后，我们把远程代码仓库中的develop分支下载到了本地仓库，但是并没有merge到我们当前工作的develop分支里，看log的话，我们会发现我们本地的develop分支和remote/develop分支还差了老远~因此需要进行如下操作：  git merge origin/develop  这样，develop分支就和remote/develop分支一样了。 |  |  |
| PPT第25页： **git pull**  将远程代码仓库里的代码下载下来并自动合并到当前工作的分支  用法：  git pull基本上等于git fetch + git merge FETCH\_HEAD，但是，git pull操作无法看清中间的代码差异和合并的逻辑。因此不提倡此用法 |  |  |
| PPT第26页: **git log**  查看代码提交日志  用法：  会显示最基本的log信息，但是不直观，不能看清分支之间的分叉、合并、前因后果，所以建议加上—all`--decorate--graph这3个参数：  git log --all --decorate --graph |  |  |
| PPT第27页: 接下来我们介绍在github中常见git操作 |  |  |
| PPT第28页: 我们先来介绍安装git： |  |  |
| PPT第29页: Linux下：打开终端，在命令端口输入sudo apt-get install gitcore回车输入密码即可。 |  |  |
| PPT第30页: **接下来介绍在github中使用git** |  |  |
| PPT第31页: 首先 配置身份  git config --global user.name "mingbai" 配置用户mingbai  git config --global user.email "251098199@qq.com"  将最后的名字和邮箱去掉可以查看配置结果 |  |  |

|  |  |  |
| --- | --- | --- |
| PPT第32页:  接下来我们介绍初始化，创建仓库，提交等简单命令。  项目根目录初始化： 　　cd 跳转到项目所在路径 　　eg: cd f: 　　cd codes/project/one 　　git init 即可创建代码仓库。 　　git add name.后缀 添加单个文件 　　git add 文件夹名 添加多个文件 　　git add . 添加所有文件 　　git commit -m “信息” 提交操作时加上信息。 |  |  |
| PPT第33页:  忽略文件： 　　在仓库根目录下创建一个.gitignore文件，这个文件里边指定的文件或目录会被自动忽略 　　eg：（文件内容） 　　　　bin/ 　　　　mingbai\_\* 　　　　这样，bin目录下和以mingbai\_开头的文件就会被忽略掉。 　查看撤销修改： 　　git status 查看修改 会提示所在分支等信息 　　git diff 查看修改的内容 　　git diff 路径/文件名 可以参看单个文件的修改状况 　　git checkout 路径/文件名 撤销修改（其实就是重新导出一个）   　对于已经添加（add）到Git上的的修改则需要reset再撤销 　　git reset HEAD 路径名/文件名 撤销添加 　　然后再git checkout 路径/文件名 即可 |  |  |
| PPT第34页：  接下来我们介绍分支的用法，首先分支的意义  比如说，推出个1.0版本，就可以给1.0建立一个分支；继而开发2.0版本。1.0出bug的话，在分支上更改内容后，推出新的1.0版本而不用影响新的2.0版本；而且还可以把　　1.0版本的bug修改合并到2.0的相同位置上，从而使2.0不出现1.0出现过的bug。  那么有哪些命令操作呢：  git branch -a 查看已经存在的所有分支  　　git branch 分支名 创建一个分支  　　git checkout 分支名 即可切换到对应的分支进行编码  　　git merge 分支名 将分支名所在分支上的修改合并过来  　　git branch -D 分支名 删除相应的分支 |  |  |
| PPT第35页：  关于远程版本库有下面一些命令可能与前面有部分重合:  git clone 远程地址 将远程版本库代码下载到本地  git push origin 分支名 将本地修改同步到远程版本库，orgin 指定远程版本库的Git地址  git fetch origin 分支名 会将远程代码同步到本地，并存放到 origin/分支名 的分支上，这时可以通过git diff origin/分支名 查看修改了哪些东西；之后再用merge合并即可。　　git merge origin/分支名 即可。  pull 命令相当于将fetch 和merge一块执行：git pull origin master |  |  |
| PPT第36页：  接下来我们对makefile和make进行详细介绍 |  |  |
| PPT第37页：  要介绍make和makefile，那么我们先从他们的关系说起 |  |  |
| PPT第38页：  make:是一个非常重要的编译命令,本质上它是一个程序。利用make工具，可以将大型的开发项目分解成为多个更易于管理的模块，对于一个包括几百个源文件的应用程序，使用make和makefile工具就可以简洁明快地理顺各个源文件之间纷繁复杂的相互关系。Make工具最主要也是最基本的功能就是通过makefile文件来描述源程序之间的相互关系并自动维护编译工作，本质上makefile文件是个文本文件，用于配置编译过程。makefile定义了一系列的规则，来规定哪些部分先编译，哪些部分后编译，写好makefile以后，只需一个make命令就可以让整个工程完全自动编译，所以简单的说，make&makefile实现了大工程的自动化编译。 |  |  |
| PPT第39页：  那么makefile文件中装的是什么： |  |  |
| PPT第40页  看个makefile的简单例子：  1 target：main.o test.o  2 cc -o target main.o teat.o  3 main.o:main.c x.h  4 cc -c main.c  5 test.o:test.c y.h  6 cc -c test.c  7 clean:  8 rm target main.o test.o |  |  |
| PPT第41页  可以看出，makefile中主要由两个部分组成：#1文件依赖关系和#2生成所需命令。其中clean是一个伪目标文件，可用".PHONY:clean"在其前面注明。执行make命令即可编译，执行make clean即可删除所有目标文件。 |  |  |
| PPT第42页  那么make是如何工作的呢一共分为四个步骤 |  |  |
| PPT第43页  步骤如下：  1）make会在当前目录下找名字为"makefile "或"Makefile"的文件；  　　2）若找到makefile文件，接着找第一个目标文件（target），并把它作为最终的目标文件；  　　3）若target文件依赖后面的.o文件更新，则执行后面定义的命令会生成target文件；  　　4）某个.o文件也会依赖若干个源文件和头文件，它们根据命令生成.o文件。 |  |  |
| PPT第44页：  根据make的依赖性，make会一层层地去找文件的依赖性，直到最终第一个target文件被编译成功。整个过程类似于数据结构中栈的操作。 |  |  |
| PPT第45页：  需要注意的是  "="是最基本的赋值，makefile会将整个makefile展开后，再决定变量的值。  ":="是覆盖当前的值，其取决于当前的赋值，而不是最终的赋值（比较好判断）。  "?="是若没有被赋值过就赋等号后的值。  "+="是添加等号后的值。 |  |  |
| PPT第46页：  接下来我们简单讲一下简化makefile文件 |  |  |
| PPT第47页：  那么为什么可以简化呢？  看下简化原理：可以利用变量减少重复，及利用makefile的自动推导 例如：  1 #Makefile  2 objects：main.o test.o  3 target:$(objects)  4 cc -o target $(objects)  5 main.o:x.h #这就用到了自动推导。  6 test.o:y.h  7 .PHONY:clean #说明clean是一个伪目标文件。  8 clean:  9 rm target $(objects) |  |  |
| PPT第48页：  我们需要知道的是：因为这个编译较简单，优势可能不太明显，若要是大工程，变量和自动推导的使用会让makefile有很大的简化。这个部分其实还很丰富，先暂记到这。 |  |  |
| PPT第49页：  最后我们还需要注意以下几点 |  |  |
| PPT第50页：  1）makefile中命令必须以[tab]键开始。  2）若指定特定的makefile，可用"-f"和"--file"参数。  3）include，被包含的文件会被原封不动的放在当前文件包含的位置，如：include foo.make |  |  |
| PPT第51页：  **接下来我们介绍cmake和cmakelists.txt** |  |  |
| PPT第52页  在详细介绍之前我们先来介绍cmake |  |  |
| PPT第53页  CMake是一个跨平台的安装(编译)工具,可以用简单的语句来描述所有平台的安装(编译过程)。他能够输出各种各样的makefile或者project文件,能测试编译器所支持的C++特性,类似UNIX下的automake。 |  |  |
| PPT第54页  知道了cmake是一个编译工具接下来我们了解一下其使用方法 |  |  |
| PPT第55页  cmake的所有语句都写在一个CMakeLists.txt的文件中,CMakeLists.txt文件确定后，直接使用cmake命令进行运行，但是这个命令要指向CMakeLists.txt所在的目录，cmake之后就会产生我们想要的makefile文件。 |  |  |
| PPT第56页  我们接下来看一下其执行流程 |  |  |
| PPT第57页  1.$> ccmake directory  2.$> cmake directory  3.$> make  其中directory为CMakeList.txt所在目录；  第一条语句用于配置编译选项，如VTK\_DIR目录 ，一般这一步不需要配置，直接执行第二条语句即可，但当出现错误时，这里就需要认为配置了，这一步才真正派上用场；  第二条命令用于根据CMakeLists.txt生成Makefile文件；  第三条命令用于执行Makefile文件，编译程序，生成可执行文件； |  |  |
| PPT第58页  介绍了cmake接下来我们编写一个CMakeLists.txt文件 |  |  |
| PPT第59页  结合例子简单介绍CMakeLists.txt的编写。  1.#project name  2.PROJECT(test\_math)  3.#head file path  4.INCLUDE\_DIRECTORIES(  5.include  6.)  7.#source directory  8.AUX\_SOURCE\_DIRECTORY(src DIR\_SRCS)  9.#set environmentvariable  10.SET(TEST\_MATH |  |  |
| PPT第60页  11.${DIR\_SRCS} 12.) 13.#set externlibraries 14.SET(LIBRARIES 15.limb.so 16.) 17.#add executablefile 18.ADD\_EXECUTABLE(../bin/bin ${TEST\_MATH}) 19#add link library 20.TARGET\_LINK\_LIBRARIES(../bin/bin ${LIBRARIEs}) |  |  |
| PPT第61页  我们对此cmakelists.txt进行解释  这是一个测试数学函数的程序的CMakeLists.txt，"#"后面为注释的内容，CMake的命令全部为大写,我们可以看出一般的CMakeLists.txt文件包含七个部分，以示例代码为例。 第2行指定生成的工程名为test\_math 第4行指定头文件目录为include 第8行指定源文件目录为src，并将其赋值给环境变量DIR\_SRCS 第10行设定环境变量TEST\_MATH的值为环境变量DIR\_SRCS的值，此处用于显示如何用环境变量对环境变量进行赋值 第14行将数学函数库赋值给环境变量LIBRARIES，当然，可以不用这个环境变量，而在后面直接使用该库名 第18行用于指定生成文件，将环境变量TEST\_MATH目录下的所有文件编译生成../bin目录下的可执行文件bin 第20行指定../bin/bin执行时的链接库为环境变量LIBRARIES的值－libm.so |  |  |
| PPT第62页  **接下来我们讲述用python编写ros service节点** |  |  |
| PPT第63页  在详细介绍ros service之前我们先比较service与topic通信 |  |  |
| PPT第64页  Topic采用的是广播/订阅的模式，没有反馈值，也就是发送的节点不知道谁会接收或者接收到没有。Service则采用request/response的模式，即客户端发送request，服务端处理request并给出response，客户端继而response，这种方式可以实现客户端的阻塞，也能保证消息通信的反馈。 |  |  |
| PPT第65页  我们通过下面表格进行比较 |  |  |
| PPT第66页  接下来我们详细介绍service通信方式 |  |  |
| PPT第67页  Service是ROS中的同步通信方式  Node间可以通过request-reply方式通信可以看到 |  |  |
| PPT第68页  我们需要知道一些简单的rosservice命令  $rosservice list #列出当前所有活跃的service  $rosservice info service\_name #显示某个service的属性信息  $rosservice call service\_name args #调用某个service  $rosservice type service\_name #打印服务类型  $rosservice uri #打印服务的uri  $rosservice find #按服务类型查找服务  $rosservice args #打印服务参数 |  |  |
| PPT第69页  接下来我们介绍Service数据格式 .srv文件以及.msg文件 |  |  |
| PPT第70页  我们先来介绍Service数据格式.srv文件  .srv文件：包括请求（request）和响应（reply）两部分可见下图 |  |  |
| PPT第71页  再来举例看一下.msg文件 |  |  |
| PPT第72页  我们需要说明的是  上述以DetectHUman.srv文件为例，该服务例子取自OpenNI的人体检测ROS软件包。它是用来查询当前深度摄像头中的人体姿态和关节数的。  需要注意的是：.srv文件只能嵌套.msg文件，不能再嵌套.srv文件 |  |  |
| PPT第73页  .srv或.msg运行时需要的依赖：  修改package.xml，添加依赖  修改CMakeList.txt，添加 |  |  |
| PPT第74页  我们看一下Rossrv有哪些命令  $rossrv list #列出系统上所有srv  $rossrv show srv\_name #显示某个srv内容  $rossrv md5 srv\_name #显示服务md5sum  $rossrv package package\_name #列出包中的服务  $rossrv packages #列出包含服务的包 |  |  |
| PPT第75页  接下来我们就来编写ros service节点 |  |  |
| PPT第76页  需要说明的是  我们会创建服务端节点 ("add\_two\_ints\_server") ，节点接收两个数字，并返回和。  前提:我们已经创建好包beginner\_tutorials。已经在之前创建好AddTwoInts.srv |  |  |
| PPT第77页  第一步  1.进入beginner\_tutorials包  $ roscd beginner\_tutorials |  |  |
| PPT第78页  第二步  2.在beginner\_tutorials包创建scripts/add\_two\_ints\_server.py文件 代码如下： #!/usr/bin/env python from beginner\_tutorials.srv import \* import rospy def handle\_add\_two\_ints(req):  print "Returning [%s + %s = %s]"%(req.a, req.b, (req.a + req.b))  return AddTwoIntsResponse(req.a + req.b) def add\_two\_ints\_server():  rospy.init\_node('add\_two\_ints\_server')  s = rospy.Service('add\_two\_ints', AddTwoInts, handle\_add\_two\_ints)  print "Ready to add two ints."  rospy.spin() if \_\_name\_\_ == "\_\_main\_\_":  add\_two\_ints\_server() |  |  |
| PPT第79页  第三步  3.文件更改为可执行权限: chmod +x scripts/add\_two\_ints\_server.py 其中：init\_node()初始化节点，并声明服务： s = rospy.Service('add\_two\_ints', AddTwoInts, handle\_add\_two\_ints) 声明一个名为add\_two\_ints新服务，使用AddTwoInts服务类型。  所有请求传递到handle\_add\_two\_ints函数处理，并传递实例AddTwoIntsRequest和返回AddTwoIntsResponse实例。  类似订阅实例，rospy.spin()会保持代码不退出，直到服务关闭 |  |  |
| PPT第80页  接下来介绍vscode的一些操作 |  |  |
| PPT第81页  我们先来介绍什么是vscode |  |  |
| PPT第82页  vscode是由 Microsoft(微软) 发布的一个免费的，开源的跨平台文本编辑器。他们基于在线编辑 Visual Studio Online (代号为 “Monaco”)，并结合 GitHub 的 Electron 实现的一个跨平台编辑器。 |  |  |
| PPT第83页  那么我们为什么选择vscode |  |  |
| PPT第84页  在vscode中找到的每个功能都完成一项出色的工作，构建了一些简单的功能集，包括语法高亮、智能补全、集成 git 和编辑器内置调试工具等，将使你开发更高效。所有你所找到的 包(packages) 都是用 JavaScript 构建的，因此任何人都可以轻松地编写自己的扩展包。 |  |  |
| PPT第85页  接下来介绍vscode基本功能 |  |  |
| PPT第86页  VS Code 最重要的功能是它的侧边栏，它集成了在编码和重构时会用到的核心功能，你需要的其他任何功能都可以通过安装扩展来满足。  智能感知功能：一个非常有用的语法高亮和自动完成功能，提供了基于变量类型、函数定义和导入模块的自动补全功能。 |  |  |
| PPT第87页  调试功能： 内置调试器可以通过添加断点和观察器进行调试，以帮助你加快编辑，编译等。 |  |  |
| PPT第88页  内置git： VS Code 内置了一个 Git GUI，支持最常用 Git 命令，这使得您可以很容易地看到您在项目中所做的更改。 主题及配色方案实时预览： 当你改变 VS Code 的主题及配色方案时，你可以在选择一个，以实时预览它们。 |  |  |
| PPT第89页  终端命令行工具 ： VS Code 提供了一个功能齐全的集成终端，可以让你选择终端，并且运行常用命令。 |  |  |
| PPT第90页  接下来我们介绍vscode |  |  |
| PPT第91页  将自动高亮的变量、字符一次性替换  操作是双击变量，右键‘更改所有匹配项’。或者，双击变量，Ctrl+F2 |  |  |
| PPT第92页  删除当前字符串中当前光标之后的内容：  ctrl +delete |  |  |
| PPT第93页  列出所有定义  shift+F12 |  |  |
| PPT第94页  格式调整  代码行缩进Ctrl+[， Ctrl+] 折叠打开代码块 Ctrl+Shift+[， Ctrl+Shift+] Ctrl+C Ctrl+V如果不选中，默认复制或剪切一整行 代码格式化：Shift+Alt+F，或Ctrl+Shift+P后输入format code 修剪空格Ctrl+Shift+X 上下移动一行： Alt+Up 或 Alt+Down 向上向下复制一行： Shift+Alt+Up或Shift+Alt+Down 在当前行下边插入一行Ctrl+Enter 在当前行上方插入一行Ctrl+Shift+Enter |  |  |
| PPT第95页  接下来介绍光标相关的操作：包括移动到行首，移动到行尾，移动到文件结尾和开头等 |  |  |
| PPT第96页  接下来介绍重构代码相关的操作，包括跳转到定义处，定义处缩略图，重命名等 |  |  |
| PPT第97页  查找替换操作  查找 Ctrl+F 查找替换 Ctrl+H 整个文件夹中查找 Ctrl+Shift+F |  |  |
| PPT第98页显示相关  全屏：F11 zoomIn/zoomOut：Ctrl + =/Ctrl + - 侧边栏显/隐：Ctrl+B 预览markdown Ctrl+Shift+V |  |  |
| PPT第99页  **接下来介绍用python编写ros action节点** |  |  |
| PPT第100页  我们先来介绍action通信方式 |  |  |
| PPT第101页  类似service，带有状态反馈的通信方式通常用在长时间、可抢占的任务中  我们先来看一下其工作模式结构示意图 |  |  |
| PPT第102页  client和server之间通过actionlib定义的”action protocol”进行通讯。这种通讯协议是基于ROS的消息机制实现的，为用户提供了client和server的接口，接口如下图所示： |  |  |
| PPT第103页  在上边的action的接口框图上，我们可以看到，client向server端发布任务目标以及在必要的时候取消任务，server会向client发布当前的状态，实时的反馈和最终的任务结果.其中：  goal:任务目标  cancel:请求取消任务  status:通知client当前的状态  feedback:周期反馈任务运行的监控数据  result:向client发送任务的执行结果，这个topic只会发布一次。 |  |  |
| PPT第104页  接下来介绍action文件格式 |  |  |
| PPT第105页  介绍文件格式之前我们先来介绍Action规范：  利用动作库进行请求响应，动作的内容格式应包含三个部分，目标、反馈、结果。  目标  机器人执行一个动作，应该有明确的移动目标信息，包括一些参数的设定，方向、角度、速度等等。  从而使机器人完成动作任务。  反馈  在动作进行的过程中，应该有实时的状态信息反馈给服务器的实施者，告诉实施者动作完成的状态，  可以使实施者作出准确的判断去修正命令。  结果  当运动完成时，动作服务器把本次运动的结果数据发送给客户端，使客户端得到本次动作的全部信息，  例如可能包含机器人的运动时长，最终姿势等等。 |  |  |
| PPT第106页  接下来我们看action文件格式示例： |  |  |
| PPT第107页  下面进行实例讲解  首先写DoDishes.action文件，包括三个部分，目标，结果，反馈： |  |  |
| PPT第108页  首先是修改文件夹里CmakeLists.txt如下内容:  1. find\_package(catkin REQUIRED genmsg actionlib\_msgs actionlib)  2. add\_action\_files(DIRECTORY action FILES DoDishes.action)  generate\_messages(DEPENDENCIES actionlib\_msgs) |  |  |
| PPT第109页  其次修改package.xml,添加所需要的依赖如下:  1. <build\_depend>actionlib </build\_depend>  2. <build\_depend>actionlib\_msgs</build\_depend>  3. <run\_depend>actionlib</run\_depend>  4. <run\_depend>actionlib\_msgs</run\_depend> |  |  |
| PPT第110页  进行编译  现在就可以进行编译了，编译完成后会产生一系列的.msg文件 |  |  |
| PPT第111页  使用rosmsg list命令查看：    从这里我们也可以看到，action确实是基于message实现的 |  |  |