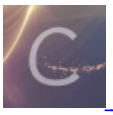
MOOS-ivp 实验三 MOOS简介（1）

原创

铁血豆丁（小高老师） 于 2020-09-26 16:19:14 发布 2350 收藏 5

分类专栏： moos-ivp 文章标签： linux

本文链接：https://blog.csdn.net/weixin_44151170/article/details/108812281



MOOS-ivp 实验三 MOOS简介（1）

实验三主要包含三个实验目标：

1.moos发布-订阅结构体系

2.启动MOOSDB并且进行交互

3.日志记录器的运行与生成

文章目录

前言

本次实验将会生成一个实验文件：pXRelayTest
然后对这个实验文件进行实验验证。

一、提前准备

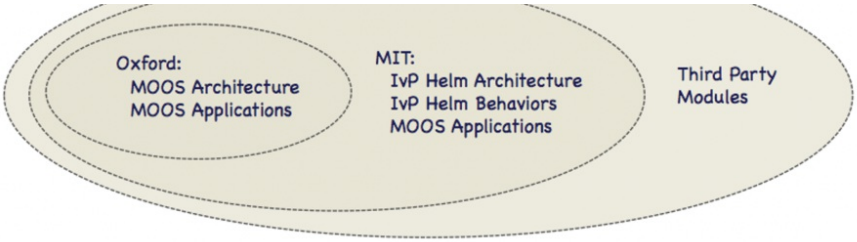
1.确定文件路径已经添加完成

```
$ which MOOSDB
/Users/you/moos-ivp/bin/MOOSDB
$ which pHelmIvP
/Users/you/moos-ivp/bin/pHelmIvP
```

需要提前确定好是否搭建完成moos的路径，输入以上代码进行检查。

2.MOOS和MOOS-ivp的关系

MOOS-ivp是一个基于MOOS的更大模块的集合，它包含了牛津大学研发的moos的基本模块，并且还包含了MIT的其他架构和模块，以下是其关系的示意图：



Oxford MOOS tree

- MOOSDB
- pLogger
- iRemote
- pScheduler
- pShare
- pMOOSBridge
- umm
- uMS
- iMatlab
- pAntler
- uPlayback

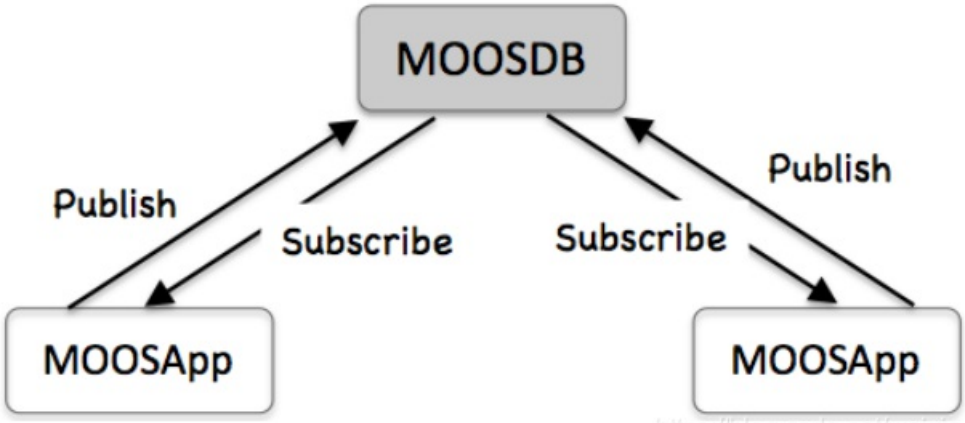
MIT MOOS-IvP tree

- pHelmIvP
- alogcat
- alogcd
- alogcheck
- alogclip
- alogeplot
- aloggrep
- aloghelm
- alogiter
- alogload
- alogpare
- alogrm
- alogscan
- alogsort
- alogsplit
- alogtest
- alogview
- ffview
- gen_hazards
- gen_obstacles
- geoview
- manifest_test
- nsplug
- pickpos
- zaic_hdg
- zaic_peak
- zaic_spd
- zaic_vect
- iSay
- pBasicContactMgr
- pDeadManPost
- pEchoVar
- pEvalLoiter
- pHostInfo
- pMarinePID
- pMarineViewer
- pMovingSurvey
- pNodeReporter
- pObstacleMgr
- uCommand
- uFldBeaconRangeSensor
- uFldCollObDetect
- uFldCollisionDetect
- uFldContactRangeSensor
- uFldMessageHandler
- uFldNodeBroker
- uFldNodeComms
- uFldObstacleSim
- uFldPathCheck
- uFldShoreBroker
- uFldWrapDetect
- uFunctionVis
- uHelmScope
- uMACView
- uFunctionVis
- uLoadWatch
- uMAC
- uMACView
- uMemWatch
- uPlotViewer
- uPokeDB
- uProcessWatch
- uQueryDB
- uSimMarine
- uTermCommand
- uTimerScript
- uXMS

可以看到这是一个层层包含的关系。

3.MOOS的架构

MOOS的架构是一种基于订阅-发布内容的架构。MOOSDB模块通过订阅和发布各个MOOSapp的内容来对其进行服务。具体结构如下图所示：



对于自主机器人来说，每一个机器人上都有一个MOOS community。一个MOOS community由一个MOOSDB和许多MOOSapp所组成。接下来要做的实验内容就是涉及到了MOOS community之间的通讯以及单个MOOS community内部的app的订阅和发布。MOOSDB不同于常见的数据库，它只保存最近发布的MOOS变量。当一个新的app要发布时，必选向MOOSDB注册需要的邮件。在这个app启动时，会收到一封邮件，里面包含了注册变量的最新值。即使这个邮件里有很久之前对MOOSSDB的发布内容。但对于新发布的app来说之前的所有内容都是未知的。（讲道理，第一次看感觉不是很理解这里的讲解，后续搞懂了再详细写一写）

3.启动MOOS

MOOSDB可以直接从命令行里进行启动，启动之后界面如下所示：

```
root@ubuntu:~# MOOSDB
------------------- MOOSDB V10 -------------------
  Hosting  community               "#1"
  Name look up is                  off
  Asynchronous support is          on
  Connect to this server on port   9000
--------------------------------------------------
network performance data published on localhost:9090
listen with "nc -u -lk 9090"
```

正常MOOSDB需要两种参数，一种是运行机器的IP地址，另一种是服务客户端的端口参数。默认的话是主机的IP地址以及端口9000上运行。此时MOOSDB本身已经发布了一些变量，可以打开另一个终端用uXMS进行查看。输入第一行的命令，即可查看消息。

```
root@ubuntu:~#
uXMS DB_CLIENTS DB_TIME DB_UPTIME --serverhost=localhost --serverport=9000
================================================================
uXMS_581  0/0(154)
================================================================
VarName  (S)ource  (T)ime (C) VarValue (SCOPING:EVENTS)
---------- --------- ------ --- -------------------------
DB_CLIENTS MOOSDB_#1 38.14  "uXMS_581,"
DB_TIME  MOOSDB_#1 38.14  1387380731.414707
DB_UPTIME  MOOSDB_#1 38.14  39.00859
```

如果想查看uXMS的具体内容 可以通过在命令行输入 s、t、c来分别查看源、时间、变量。如下图所示:

```
_301                                              1/0(469)
============================================================
Configuration Warnings: 1
 [1 of 1]: Community/Vehicle name not found in mission file

VarName      (S)ource   (T)ime  (C)ommunity  VarValue (SCOPING:PAUSED)
-----------  ---------  ------  -----------  -------------------------
APPCAST_REQ   n/a        n/a                 n/a
DB_CLIENTS   MOOSDB_#1  80.49   #1           "uXMS_301,"
DB_TIME      MOOSDB_#1  80.49   #1           1601170133.257782
DB_UPTIME    MOOSDB_#1  80.49   #1           81.366143
```

4.最小配置启动MOOS

通过下载alpha.moos可以通过最小的配置来启动moos。需要先对其进行下载，原本命令没有忽略验证证书，但是会因为验证无法通过而报错，故做此改进：

```
wget --no-check-certificate http://oceanai.mit.edu/2.680/examples/moosdb_alpha.moos
```

启动之后可以得到以下配置：

```
root@ubuntu:~# MOOSDB moosdb_alpha.moos
------------------- MOOSDB V10 -------------------
  Hosting  community               "alpha"
  Name look up is                  off
  Asynchronous support is          on
  Connect to this server on port   9000
--------------------------------------------------
network performance data published on localhost:9090
listen with "nc -u -lk 9090"
```

二、MOOS scope

1.MOOS scope

MOOSDB不保存之前历史中的变量，而只保存当前变量，一般用来检查MOOS状态的工具最常用的是以下两种模块：
uXMS
uMS

2.uXMS查看

1.启动MOOSDB
MOOSDB通常是通过指定ServeHost和ServePort来启动的。没有参数启动时，这两个参数默认为localhost和9000

```
$ MOOSDB
------------------- MOOSDB V10 -------------------
Hosting community  "#1"
Name look up is  off
Asynchronous support is  on
Connect to this server on port  9000
--------------------------------------------------
network performance data published on localhost:9020
listen with "nc -u -lk 9020"
```

2.打开第二个终端界面，启动uXMS

```
$ uXMS --all
Enter IP address: [localhost]
Enter Port number: [9000]
************************************************
* uXMS_632 starting ...
************************************************
uXMS_632 is Running:
|-Baseline AppTick  @ 5.0 Hz
|--Comms is Full Duplex and Asynchronous
-Iterate Mode 0 :
|-Regular iterate and message delivery at 5 Hz
```

初始化结束之后，可以看到如下的界面

```
================================================================
uXMS_443  0/0(31)
================================================================
VarName  (S) (T) (C) VarValue (SCOPING:EVENTS)
---------------- --- --- --- ------------------------
DB_CLIENTS  "uXMS_443,"
DB_EVENT  "connected=uXMS_443"
DB_QOS  "uXMS_443=0.448942:0.573874:0.364065:0.490189,"
DB_TIME  1424028179.877422
DB_UPTIME  16.741557
UXMS_443_ITER_GAP  1.019045
UXMS_443_ITER_LEN  0.00024
-- displaying all variables --
```

看一下第二行0/0(31)括号内的数字是递增状态，这说明已经刷新到当前终端。这种情况下，基本上每秒刷新一次DB_TIME 和DB_UPTIME。三个以DB开头的变量都是由MOOSDB来进行发布的。

注意事项：

（1）可以通过输入h，来获取帮助，再按一次h返回

（2）点击空格暂停数据，按e返回之前模式

（3）按s、t、c分别可以展开内容：源、时间、社区

（4）启动时输入以下代码

```
uXMS --all --colormap=DB_UPTIME,blue
```

可以蓝色高亮显示并查找到DB_UPTIME变量

（5）查找单个变量

```
uXMS DB_UPTIME
```

（6）要查看某一个变量是如何变化的

```
uXMS --history=DB_UPTIME
```

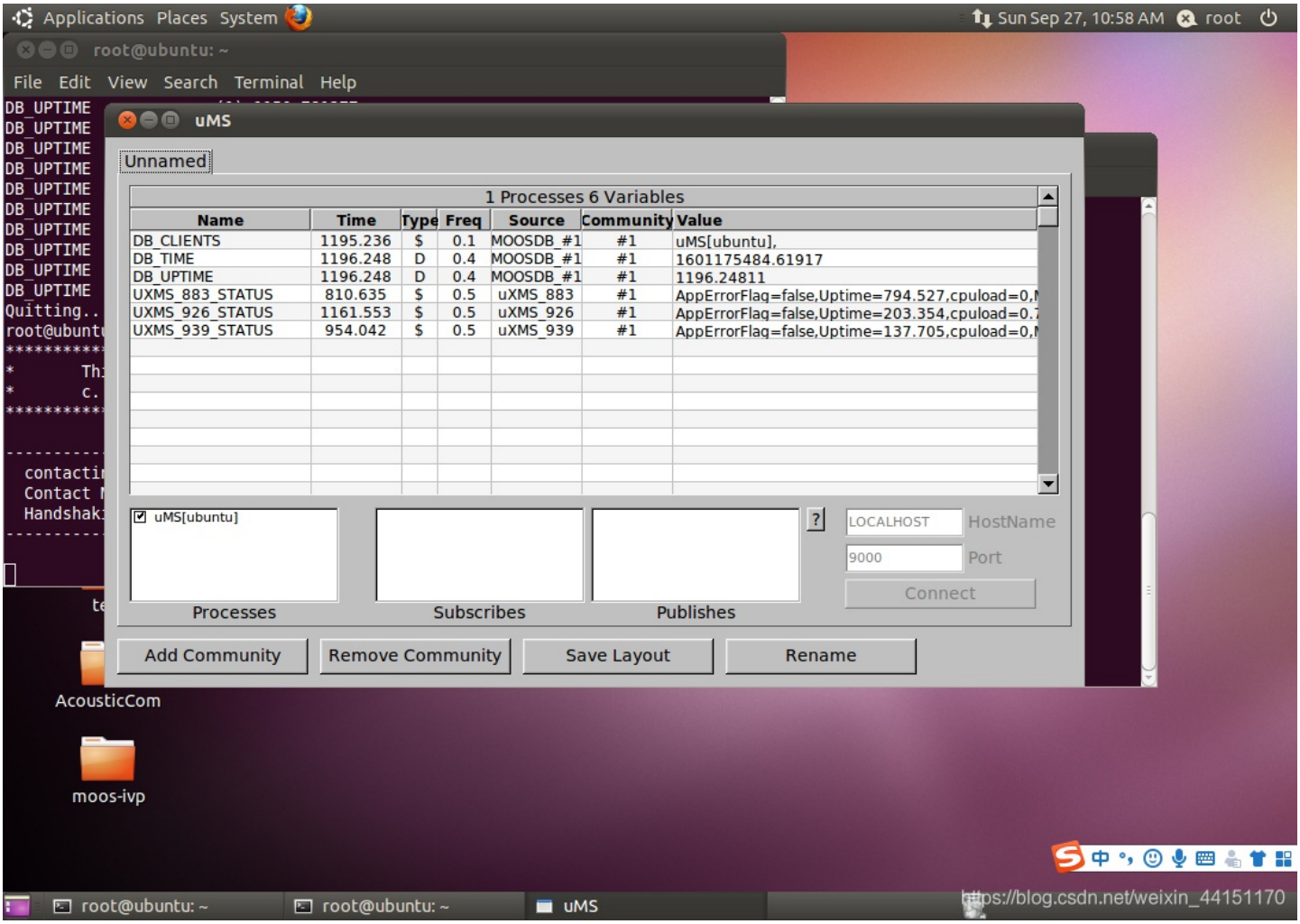
3.uMS查看

uMS是一个图形化的查询界面。

1.打开MOOSDB

2.打开新的终端，输入uMS来运行。

3.点击connec按钮即可看到以下画面：



三、Poking the MOOSDB

正常app在运行过程中都会向MOOSDB发布变量。而Poking代表着计划之外的变量发布，这个通常对debug非常有用，（个人意见就是debug的时候查看相关变量的值或者给某些变量赋值，当然看到后面如果不对我再修改）可以通过以下链接来进行详情查看：

uPokeDB

1.Poking the MOOSDB with uPokeDB

1.先打开两个终端界面，分别打开MOOSDB和uXMS

```
$ MOOSDB alpha.moos
$ uXMS alpha.moos --all
```

再打开一个新的终端输入以下命令

```
$ uPokeDB DEPLOY=true SPEED=2 alpha.moos
```

可以看到有一个新的变量SPEED被发布了而且值为2

```
================================================================
uXMS_655  0/0(204)
================================================================
VarName  (S) (T) (C) VarValue (SCOPING:EVENTS)
---------- --- --- --- ------------------------
DB_CLIENTS  "uXMS_655,"
DB_TIME  1386249435.276804
DB_UPTIME  46.213629
DEPLOY  "true"
SPEED  2
```

2.注意上述表中变量值的类型，可以看出来DEPLOY的值是一个字符串，而SPEED的值没有引号，应该是一个双精度类型。如果你想发布一个变量，内容是数字的字符串可以通过以下命令来进行发布：

```
$ uPokeDB HEIGHT:=192 alpha.moos
```

可以看到uXMS界面上的显示

```
================================================================
uXMS_655  0/0(347)
================================================================
VarName  (S)ource  (T) (C) VarValue (SCOPING:EVENTS)
---------- ---------  --- --- ------------------------
DB_CLIENTS MOOSDB_alpha  "uXMS_655,"
DB_TIME  MOOSDB_alpha  1386250092.847527
DB_UPTIME  MOOSDB_alpha  703.784353
DEPLOY  uPokeDB  "true"
HEIGHT  uPokeDB  "192"
SPEED  uPokeDB  2
```

2.进一步使用uPokeDB

1.试一下以下命令，看看DEPLOY的值是否会改变

```
uPokeDB DEPLOY=100 alpha.moos
```

个人尝试了一下，没有改变。然后改成字符串格式，里面是数字仍旧没有改变。但是改为字符串格式的文字发生了改变。说明发布不符合变量类型的内容时，不能成功发布。

2.一次运行多行uPokeDB命令

```
uPokeDB APPLES=1 alpha.moos; sleep 5; uPokeDB APPLES=2 alpha.moos;
```

3.用vim创建一个脚本，写入以下命令

```
uPokeDB APPLES=1 alpha.moos
sleep 5
uPokeDB APPLES=2 alpha.moos
```

将脚本命名为MyScript
运行命令

```
source myscript
```

总结

以上是对实验三的前半部分内容的叙述，所占篇幅已经够长，我把接下来的内容用一篇新的文章来继续进行记录。本实验记录的目的为：在学习中需要对所学知识有输入和输出，从而才能更好的构建知识结构和更好的掌握知识。如果该篇博客内容对大家有帮助，希望各位多多给我小高老师点个赞！