opensplice dds 应用开发说明书

刘加权

电话: 15651950821　邮箱: vimcpp@163.com

南京元几科技有限公司

www.yuanji.tech

版本: v1.0.0

2021 年 11 月 18 日

目录

1 opensplice 源代码的使用

1.1 关于 opensplice

数据分发服务 DDS(DataDistributionService) 是对象管理组织 (OMG) 在 HLA 及 CORBA 等标准的基础上制定的新一代分布式实时通信中间件技术规范，DDS 采用发布/订阅体系架构，强调以数据为中心，提供丰富的 QoS 服务质量策略，能保障数据进行实时、高效、灵活地分发，可满足各种分布式实时通信应用需求。DDS 信息分发中间件是一种轻便的、能够提供实时信息传送的中间件技术。

DDS 有多种实现，比较著名的有 Opensplice DDS、OpenDDS、RTI DDS 等。其中 opensplice dds 便是其中比较常用的一种。

OpenSplice DDS 采用多播和广播（默认方式）传输模型，采用第三层网络接口（IP 组播和广播，这里要特别注意一下，它不是我们常用的 TCP/IP 通信，在开发调试中要注意区分）来处理不同通信模型的网络传输。

1. 优点：不需要通过专门的配置节点，时延和抖动会减少，不存在节点失效的问题

2. 不足：指定的配置细节（如：多播地址、端口号、可靠性等）必须在应用层完成，存在潜在的不可移植性问题；该结构存在同一节点上多个应用程序发送的数据缓存困难的问题

技术参考:opesplice 官方开源代码库 [1]。opesplice 官方帮助文档 [2]。

1.2 opensplice 的编译

下载 opensplice 源代码放到 ~/git/目录下, 如下图 1所示。

[1]代码库: https://github.com/ADLINK-IST/opensplice.git
[2]文档: http://download.prismtech.com/docs/Vortex/html/ospl/GettingStartedGuide/index.html

```
vi@vi-OptiPlex-7080:~/git$ pwd
/home/vi/git
vi@vi-OptiPlex-7080:~/git$ ls
core.light  log                        run             tool_sh  xilinx
demo        monitor                    sd_card         v2.0.1   zynq_release
env_config  opensplice-master          sd_card.tar.gz  v2.0.2
fip         opensplice-master.tar.gz   tmp             vm
vi@vi-OptiPlex-7080:~/git$ ls opensplice-master
LICENSE              demos                      exec         release_info
MPC                  docs                       external     scripts
README.md            envs---help.sh             index.html   setup
azure-pipelines.yml  envs--h.sh                 install      src
bin                  envs-x86_64.linux-dev.sh   lib          submods
build                etc                        makefile     testsuite
configure            examples                   ospl_docs    vbase
```

图 **1:** opensplice 代码放置目录

用户可以将 opensplice 代码放置于任何目录如果不放置在 ~/git 目录下需要用户修改 opensplice_dir 环境变量配置，目录命名时注意不要有空格以方便脚本执行。为了简单建议用户将 opensplice 放置 ~/git 目录下。

编译 opensplice 源代码涉及很多个步骤，我们已经把编译步骤做成脚本，支持一键编译，方便用户使用。

将 zynq_release 目录也放置到 ~/git/目录下,zynq_release 目录也已存在于 sd_card 系统包中。执行命令 cd ~/git/zynq_release 切换当前目录到 zynq_release 目录下。如下图 2所示:

```
vi@vi-OptiPlex-7080:~/git/zynq_release$ cd ~/git/zynq_release/
vi@vi-OptiPlex-7080:~/git/zynq_release$ ls
demo        opensplice-master    release.sh.x86                    shell
demo.bak    opensplice-master.1  release_opensplice_on_zynq         tmp
opensplice  release.sh           release_opensplice_on_zynq.tar.gz  version.txt
vi@vi-OptiPlex-7080:~/git/zynq_release$ pwd
/home/vi/git/zynq_release
vi@vi-OptiPlex-7080:~/git/zynq_release$
```

图 **2:** zynq_release 放置目录

直接执行命令./release.sh 查看该命令帮助说明，如下图 3所示:

```
vi@vi-OptiPlex-7080:~/git/zynq_release$ ./release.sh
    useage:
      build opensplice lib: ./release.sh --build
    release opensplice lib: ./release.sh --dev
    install opensplice lib: ./release.sh --install

    --build                    - compile opensplice srource and example
    --dev                      - release develop binary version without srource
    --install                  - install opensplice library, please first execut: ./release.sh --dev


    --log_level=<debug> - control pint log on streen,trace|debug|info|warn|error|critical

    --help  - prints help screen, script will exit after printing
```

图 3: 脚本 release.sh 帮助说明

release.sh 脚本参数主要有三个：build,dev,install。注意命令执行顺序，编译-> 发布-> 安装。后者依赖前者，后者可以多次执行，代码修改后发布重新执行一遍流程。才能将新的库更新到 /run/opensplice 目录

如果是 ubuntu 系统，请先执行下面命令安装依赖项 sudo apt install -y gawk bison flex doxygen git

执行命令进行编译:

```
./release.sh --build
```

编译开始，如下图 4所示:

```
vi@vi-OptiPlex-7080:~/git/zynq_release$ ./release.sh --build
[11-18 16:21:33] [info] [release.sh:235 main()] platform_info:x86_64.linux-dev
[11-18 16:21:33] [info] [release.sh:236 main()] opensplice_dir:/home/vi/git/opensplice-master

Setup at 16:21:33 for Vortex - Version 6.9.210323OSS - Date 2021-11-18


GCC: OK - Using version 7
  NOTE - Enable link-time optimizations (for release build)
GLIBC: version 2.27
MAKE: OK - using GNU Make 4.1
Perl: OK - using perl version='5.26.1';
Qt: Cannot find Qt libraries. Standalone demo package will not be built. Please specifiy QTLIBDIR.
Off. Ok, but you're missing out on bouncing shapes...
GAWK: OK - using GNU Awk 4.1.4, API: 1.1 (GNU MPFR 4.0.1, GNU MP 6.1.2)
BISON: OK - using 3.0.4
FLEX: OK - using 2.6.4
JAVAC: Warning - Java compiler environment not set, building of all Java related features is disabled.
GMCS: Warning - No gmcs compiler found
   gmcs C# compiler not found, disabling SACS api build.
TAO: Warning - No TAO found
   TAO environment not set, disabling TAO related features.
JACORB: Warning - JACORB_HOME not set
   JACORB environment not set, disabling JACORB related features.
GSOAP: Warning - Not found, cmsoap will not be built
DOXYGEN: OK
GOOGLE PROTOCOL BUFFERS: PROTOBUF_HOME has not been set
Warning - Protobuf compiler environment not set, building of all protobuf related features is disabled.
C99: OK - supported
Python3: OK - Using /usr/bin/python3 v3.6.9
    Warning: Cython module not found. Cannot build Python DCPS API
    Warning: wheel module not found. Cannot build Python DCPS API
NODEJS: Python not found - skipping build of NodeJS DCPS API and unit tests.
MAVEN: maven is not installed (properly):
   mvn not found in path
   M2_HOME variable not set
Warning - Java compiler environment not set, building of all Java related features is disabled.
Key-value store implementations - SQLITE: Warning - Not found, LEVELDB: Warning - Not found
Configuration OK

Variable Setup
SPLICE_TARGET = x86_64.linux-dev
SPLICE_HOST = x86_64.linux-dev
OSPL_HOME = /home/vi/git/opensplice-master
SPLICE_ORB =
make[1]: Entering directory '/home/vi/git/opensplice-master/setup'
make[2]: Entering directory '/home/vi/git/opensplice-master/setup/wrappers'
make[2]: Nothing to be done for 'link'.
make[2]: Leaving directory '/home/vi/git/opensplice-master/setup/wrappers'
```

图 4: 编译开始

如果不出错，可以看到如下 5编译成功输出。

```
magic_make.pl: Generating MPC build file(s): mwc.pl --ospl-libmod --ospl-home /home/vi/git/opensplice-master/install/HDE/x86_64.linux-dev --features ndebug=1 --type make --features no_c99=0 --features isocpp2_cxx11=1 HDE/x86_64.linux-dev/custom_lib
OSPL_HOME is /home/vi/git/opensplice-master/install/HDE/x86_64.linux-dev
MPC_ROOT was set to /home/vi/git/opensplice-master/submods/MPC_ROOT.
Run mpc with: --features ospl_os_host_linux=1 --features ospl_os_linux=1 --features ospl_64_bit=1 --include /bin/MakeProjectCreator/config --include /bin/MakeProjectCreator/config/TAO --features x86_64_linux=1 --features ospl_arch_x86_64=1 --features ndebug=1 --type make --features no_c99=0 --features isocpp2_cxx11=1 HDE/x86_64.linux-dev/custom_lib
Using .../opensplice-master/bin/../MPC/config/MPC.cfg
make[2]: Leaving directory '/home/vi/git/opensplice-master/install'
################### Compile (Building installer for inner Host_Development_Environment) [2021年 11月 18日 星期四 16:31:11 CST]
Making linux builder for Host_Development_Environment, platform x86_64.linux, environment -dev
make_builder params are HDE Host_Development_Environment x86_64.linux linux release.com -dev getlic_info normal
Making builder for Host_Development_Environment (HDE, x86_64.linux-dev, linux)
Release is 6.9.210323OSS
Date is 2021-11-18
/bin/tar cf /home/vi/git/opensplice-master/install/VC/PXXX-VortexOpenSplice-6.9.210323OSS-HDE-x86_64.linux-gcc7-glibc2.27-dev-installer.tar HDE
Backup original with tar
################### Compile (Populating inner Runtime_System) [2021年 11月 18日 星期四 16:31:13 CST]
Making mirror for Runtime_System, platform x86_64.linux, environment -dev
* Build SPLICE RTS
*   Remove old RTS/x86_64.linux directory ... Ready
*   Prepare RTS master directory ... Ready
*   Prepare RTS for target x86_64.linux-dev
*     Copy OpenSplice default configuration file
*                 copy /home/vi/git/opensplice-master/etc/ospl_sp_no_network.xml ... Ready
*                 copy /home/vi/git/opensplice-master/etc/ospl_sp_ddsi.xml ... Ready
*                 copy /home/vi/git/opensplice-master/etc/ospl_sp_ddsi_statistics.xml ... Ready
*                 copy /home/vi/git/opensplice-master/etc/ospl_sp_ddsi_1gbps.xml ... Ready
*                 copy /home/vi/git/opensplice-master/etc/ospl_sp_ddsi_10gbps.xml ... Ready
*                 copy /home/vi/git/opensplice-master/etc/ospl_sp_ddsi.xml ... Ready
*                 copy /home/vi/git/opensplice-master/src/tools/cm/config/code/splice_metaconfig_6.1.xml ... Ready
*     Ready
*     Copy library files ... Ready
*     Copy script files
*                 copy /home/vi/git/opensplice-master/install/bin/osplconf ... Ready
*     Script files Ready
*     Copy OpenSplice executables
*       copy shmdump ... Ready
*       copy mmstat ... Ready
*       copy dbd ... Ready
*     Ready
*     Prepare default release.com ... Ready
*   RTS prepared for target x86_64.linux-dev Ready
* SPLICE RTS Ready
################### Compile (Building installer for inner Runtime_System) [2021年 11月 18日 星期四 16:31:13 CST]
Making linux builder for Runtime_System, platform x86_64.linux, environment -dev
make_builder params are RTS Runtime_System x86_64.linux linux release.com -dev getlic_info normal
Making builder for Runtime_System (RTS, x86_64.linux-dev, linux)
Release is 6.9.210323OSS
Date is 2021-11-18
/bin/tar cf /home/vi/git/opensplice-master/install/VC/PXXX-VortexOpenSplice-6.9.210323OSS-RTS-x86_64.linux-gcc7-glibc2.27-dev-installer.tar RTS
Backup original with tar
make[1]: Leaving directory '/home/vi/git/opensplice-master/install'
vi@vi-OptiPlex-7080:~/git/zynq_release$ _
```

图 5: 编译成功

1.3 opensplice 的发行

编译成功后，执行下列命令生成发行二进制包

```
./release.sh --dev
```

该命令会在 zynq_release 目录下生成目录 release_opensplice_on_zynq, 以及该目录的二进制包 release_opensplice_on_zynq.tar.gz 文件。二进制包中包含了 opensplice 编译后生成的库和执行文件等配置文件。可以直接发布给开发用户。

1.4 opensplice 的安装部署

发行成功后，执行下列命令将发行的 opensplice 二进制文件安装到 ~/run/opensplice 目录下

```
./release.sh --install
```

安装成功后，目录如下图 6所示:

```
vi@vi-OptiPlex-7080:~/git/zynq_release$ ./release.sh --install
[11-18 16:37:43] [info] [release.sh:235 main()] platform_info:x86_64.linux-dev
[11-18 16:37:43] [info] [release.sh:236 main()] opensplice_dir:/home/vi/git/opensplice-master
[11-18 16:37:45] [info] [release.sh:157 install_opensplice()] succeed, install opensplice into /home/vi/run/opensplice
vi@vi-OptiPlex-7080:~/git/zynq_release$
vi@vi-OptiPlex-7080:~/git/zynq_release$ ls ~/run/opensplice/
bin  doc  etc  idl  include  lib  readme.txt  shell
vi@vi-OptiPlex-7080:~/git/zynq_release$
```

图 6: 安装到 run 目录

2 demo 目录模板

2.1 关于 demo 目录模板

demo 目录模板使用 cmake 来管理 c/c++ 开发的应用代码工程。使用 cmake 相比于直接使用 makefile 有著多好处，使得工程管理干净、准确、快速构建。

用户新建自己的应用只需在 demo 中简单的修改个配置，并向工程中添加自己的代码和相关目录即可。

demo 目录结构如下图 7所示:

```
vi@vi-OptiPlex-7080:~/git/demo$ tree -L 3  | grep -vE "xml|cfg"
.
├── LICENSE
├── project
│   ├── doc
│   │   ├── host_info.txt
│   │   └── readme.txt
│   ├── etc
│   │   ├── DDS_QoSProfile.xsd
│   │   ├── docs_git_tag
│   │   ├── idl
│   │   ├── idlpp
│   │   ├── java
│   │   └── valgrind-suppressions.txt
│   ├── idl
│   │   ├── HelloWorldData.idl
│   │   ├── Topic1Data.idl
│   │   ├── Topic2Data.idl
│   │   └── Topic3Data.idl
│   ├── readme.txt
│   ├── shell
│   │   ├── assist
│   │   ├── demo.sh
│   │   ├── ospl-error.log
│   │   ├── ospl-info.log
│   │   └── source.sh
│   └── src
│       ├── CMakeLists.txt
│       ├── build
│       ├── discover_topic
│       ├── helloworld_pub
│       ├── helloworld_sub
│       ├── include
│       ├── multiple_topic_pub1
│       ├── multiple_topic_pub2
│       └── multiple_topic_sub
└── shell
    ├── auto_commit.sh
    └── remote.sh

19 directories, 41 files
vi@vi-OptiPlex-7080:~/git/demo$
```

图 **7:** demo 目录结构

demo/project 目录下主要有：

1. etc: opensplice 工程相关的配置文件
2. shell: demo 编译脚本，环境变量配置文件
3. idl: IDL opensplice 消息协议配置文件

4. src: 进行源代码目录,src 目录里面可以包含多个子进程, 用户可以在这个目录下添加自己的进程代码目录

2.2 demo 模板里涉及的 cmake 编译方式

2.2.1 编译 demo 模板

将 demo 目录放置于 ~/git/目录下。切换当前目录到 ~/git/demo/project/shell, 执行下列命令:

```
./demo.sh --build
```

demo 编译输出，如下图 8所示:

```
vi@vi-OptiPlex-7080:~/git/demo/project/shell$
vi@vi-OptiPlex-7080:~/git/demo/project/shell$ pwd
/home/vi/git/demo/project/shell
vi@vi-OptiPlex-7080:~/git/demo/project/shell$ ./demo.sh --build
[11-18 17:03:04] [info] [demo.sh:53 generate_idl()] execute cmd: rm -rf /home/vi/git/demo/project/src/build/idl_auto_gen/
[11-18 17:03:04] [info] [demo.sh:58 generate_idl()] generate src by idl file.
[11-18 17:03:04] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/HelloWorldData.idl
[11-18 17:03:04] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/Topic1Data.idl
[11-18 17:03:04] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/Topic2Data.idl
[11-18 17:03:04] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/Topic3Data.idl
[11-18 17:03:04] [info] [demo.sh:64 generate_idl()] generate all c files into directory: /home/vi/git/demo/project/src/build/idl_auto_gen
[11-18 17:03:04] [info] [demo.sh:73 compile()] start to cmake and make ...
-- opensplice_home: /home/vi/run/opensplice
running chmod +x /home/vi/git/demo/project/src/../shell/demo.sh 2>&1
running /home/vi/git/demo/project/src/../shell/demo.sh --gen_idl  2>&1
mesg: ttyname failed: Inappropriate ioctl for device
[11-18 17:03:04] [info] [demo.sh:53 generate_idl()] execute cmd: rm -rf /home/vi/git/demo/project/src/build/idl_auto_gen/
[11-18 17:03:04] [info] [demo.sh:58 generate_idl()] generate src by idl file.
[11-18 17:03:04] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/HelloWorldData.idl
[11-18 17:03:04] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/Topic1Data.idl
[11-18 17:03:04] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/Topic2Data.idl
[11-18 17:03:04] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/Topic3Data.idl
[11-18 17:03:04] [info] [demo.sh:64 generate_idl()] generate all c files into directory: /home/vi/git/demo/project/src/build/idl_auto_gen
-- build date: 2021-11-18 17:03:04
-- Configuring done
-- Generating done
-- Build files have been written to: /home/vi/git/demo/project/src/build
Scanning dependencies of target helloworld_pub
[  1%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/DDSEntitiesManager.c.o
[  2%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/HelloWorldDataPublisher.c.o
[  4%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/HelloWorldDataSacDcps.c.o
[  5%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/HelloWorldDataSplDcps.c.o
[  6%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic1DataSacDcps.c.o
[  8%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic1DataSplDcps.c.o
[  9%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic2DataSacDcps.c.o
[ 11%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic2DataSplDcps.c.o
[ 12%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic3DataSacDcps.c.o
[ 13%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic3DataSplDcps.c.o
[ 15%] Linking C executable ../bin/helloworld_pub
[ 16%] Built target helloworld_pub
Scanning dependencies of target helloworld_sub
[ 18%] Building C object helloworld_sub/CMakeFiles/helloworld_sub.dir/DDSEntitiesManager.c.o
[ 19%] Building C object helloworld_sub/CMakeFiles/helloworld_sub.dir/HelloWorldDataSubscriber.c.o
[ 20%] Building C object helloworld_sub/CMakeFiles/helloworld_sub.dir/__/idl_auto_gen/HelloWorldDataSacDcps.c.o
[ 22%] Building C object helloworld_sub/CMakeFiles/helloworld_sub.dir/__/idl_auto_gen/HelloWorldDataSplDcps.c.o
[ 23%] Building C object helloworld_sub/CMakeFiles/helloworld_sub.dir/__/idl_auto_gen/Topic1DataSacDcps.c.o
[ 25%] Building C object helloworld_sub/CMakeFiles/helloworld_sub.dir/__/idl_auto_gen/Topic1DataSplDcps.c.o
[ 26%] Building C object helloworld_sub/CMakeFiles/helloworld_sub.dir/__/idl_auto_gen/Topic2DataSacDcps.c.o
[ 27%] Building C object helloworld_sub/CMakeFiles/helloworld_sub.dir/__/idl_auto_gen/Topic2DataSplDcps.c.o
[ 29%] Building C object helloworld_sub/CMakeFiles/helloworld_sub.dir/__/idl_auto_gen/Topic3DataSacDcps.c.o
[ 30%] Building C object helloworld_sub/CMakeFiles/helloworld_sub.dir/__/idl_auto_gen/Topic3DataSplDcps.c.o
[ 31%] Linking C executable ../bin/helloworld_sub
[ 33%] Built target helloworld_sub
```

图 8: demo 编译打印输出

demo 编译成功后，所生成的文件位于 demo/project/src/build/bin 目录, 如下图 9，其中 build 目录是生成目录，不属于源代码可以删除。

```
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$ ls
CMakeCache.txt  Makefile  cmake_install.cmake  helloworld_pub  idl_auto_gen         multiple_topic_pub2
CMakeFiles      bin       discover_topic       helloworld_sub  multiple_topic_pub1  multiple_topic_sub
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$ ls bin
discover_topic  helloworld_pub  helloworld_sub  multiple_topic_pub1  multiple_topic_pub2  multiple_topic_sub
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$
```

图 **9:** demo 编译生成的 build

除了使用 demo.sh –build 来编译, 用户还可以直接使用 cmake，make 来编译, 如图 10。

1. 切换到 demo/project/src/build/目录下。
2. 执行 cmake .. 生成编译所需要的 makefile 文件，如果是因为环境变量不对失败按提示执行 source your_path/source.sh, 以设置环境变量
3. 在 build 目录下执行 make 进行编译。

```
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$ cmake ..                                    <-- step:1
-- opensplice_home:
CMake Error at CMakeLists.txt:57 (message):
  'opensplice_home' system environment variable is not set!

  you can fix it by executing:                                                                <-- step:2 如果cmake成功，跳过此步骤

  source /home/vi/git/demo/project/src/../shell/source.sh


-- Configuring incomplete, errors occurred!
See also "/home/vi/git/demo/project/src/build/CMakeFiles/CMakeOutput.log".
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$ source /home/vi/git/demo/project/src/../shell/source.sh
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$ cmake ..
-- opensplice_home: /home/vi/run/opensplice
running chmod +x /home/vi/git/demo/project/src/../shell/demo.sh 2>&1
running /home/vi/git/demo/project/src/../shell/demo.sh --gen_idl  2>&1
mesg: ttyname failed: Inappropriate ioctl for device
[11-18 17:16:36] [info] [demo.sh:53 generate_idl()] execute cmd: rm -rf /home/vi/git/demo/project/src/build/idl_auto_gen/
[11-18 17:16:36] [info] [demo.sh:58 generate_idl()] generate src by idl file.
[11-18 17:16:36] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/HelloWorldData.idl
[11-18 17:16:36] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/Topic1Data.idl
[11-18 17:16:36] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/Topic2Data.idl
[11-18 17:16:36] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/Topic3Data.idl
[11-18 17:16:36] [info] [demo.sh:64 generate_idl()] generate all c files into directory: /home/vi/git/demo/project/src/build/idl_auto_gen
-- build date: 2021-11-18 17:16:36
-- Configuring done
-- Generating done
-- Build files have been written to: /home/vi/git/demo/project/src/build
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$ make                                        <-- step:3
Scanning dependencies of target helloworld_pub
[  1%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/DDSEntitiesManager.c.o
[  2%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/HelloWorldDataPublisher.c.o
[  4%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/HelloWorldDataSacDcps.c.o
[  5%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/HelloWorldDataSplDcps.c.o
[  6%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic1DataSacDcps.c.o
[  8%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic1DataSplDcps.c.o
[  9%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic2DataSacDcps.c.o
[ 11%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic2DataSplDcps.c.o
[ 12%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic3DataSacDcps.c.o
[ 13%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic3DataSplDcps.c.o
[ 15%] Linking C executable ../bin/helloworld_pub
[ 16%] Built target helloworld_pub
Scanning dependencies of target helloworld_sub
```

图 **10:** demo 编译直接执行 cmake，make

如果因为路径问题导致 cmake 执行失败，可以直接删除 demo/project/src/build/目录下面的所有文件, 然后再执行 cmake .. 重新生成。

2.3 demo 的编译与执行

2.4 从 IDL 文件创建应用工程

IDL 文件是 opensplice dds 系统中定义的框架平台无关的消息描述文件。看一下示例如下：

```
/**
* @addtogroup examplesdcpsHelloWorldisocpp
*/
/** @{*/
/** @file */

module HelloWorldData
{
    struct Msg
    {
        /** User ID */
        long userID;
        /**  message */
        string message;
    };
    #pragma keylist Msg userID
};

/** @}*/
```

demo 模板中所有的 IDL 文件都存储于 demo/project/idl/目录下，用户可将自己定义的新的消息描述文件放到 idl 目录下，系统在编译的时候会自动调用 idlpp 预处理程序将 idl 文件翻译成对应的 c 语言源文件。翻译后的 c 文件存储于 project/src/build/idl_auto_gen/ 目录下。

用户代码在编译的时候直接 #include "*.h" 就行了。例如：

```
HelloWorldData.idl -->(idlpp预处理翻译)生成如下几
    个文件
HelloWorldData.h
HelloWorldDataDcps.h
HelloWorldDataSacDcps.c
HelloWorldDataSacDcps.h
HelloWorldDataSplDcps.c
HelloWorldDataSplDcps.h

用户源代码中#include "HelloWorldData.h"
```

用户可以只执行预处理操作而有需要编译代码。切抱到 demo/project/shell 目录下，执行下列命令：

```
./demo.sh --gen_idl
```

从 IDL 文件创建应用工程, 在这里就是从 demo 模板创建用户工程。从整体上看，可以分两种方式来创建应用工程，分别是：

1. 直接在 demo 模板中增加进程源代码目录
2. 修改 demo 模板中现有进程配置

下面将分别进行详细说明。

2.4.1 直接在 demo 模板中增加进程源代码目录

在这里以增加一个新的进程 helloworld_pub2 为例。

1. 在 src 目录下复制 helloworld_pub 为 helloworld_pub2, 执行 cp -rp helloworld_pub helloworld_pub2

2. 切换到 helloworld_pub2 目录下，编辑 helloworld_pub2 目录下的 CMakeLists.txt, 修改第一行的 set(tgt helloworld_pub) 为 set(tgt helloworld_pub2) 如下图 11所示:

```
vi@vi-OptiPlex-7080:~/git/demo/project/src/helloworld_pub2$ ls
CMakeLists.txt  CheckStatus.c  CheckStatus.h  DDSEntitiesManager.c  DDSEntitiesManager.h  HelloWorldDataPublisher.c
vi@vi-OptiPlex-7080:~/git/demo/project/src/helloworld_pub2$
vi@vi-OptiPlex-7080:~/git/demo/project/src/helloworld_pub2$ head -n 5 ./CMakeLists.txt

set(tgt helloworld_pub2)
set(common_dir ${CMAKE_CURRENT_SOURCE_DIR}/common)

######################################################################
vi@vi-OptiPlex-7080:~/git/demo/project/src/helloworld_pub2$
```

图 **11:** 编辑后的 CMakeLists.txt

3. 切换到目录 demo/project/src/目录下，编辑 src 目录下的 CMakeLists.txt, 在大约 141 行处 (只要在 add_subdirectory 行附近即可) 增加一行 add_subdirectory(helloworld_pub 增加后的 CMakeLists.txt 如下图所示:

```
140
141 add_subdirectory(helloworld_pub)
142 add_subdirectory(helloworld_pub2)
143 add_subdirectory(helloworld_sub)
144 add_subdirectory(discover_topic)
145 add_subdirectory(multiple_topic_pub1) #TODO, on developing
146 add_subdirectory(multiple_topic_pub2) #TODO, on developing
147 add_subdirectory(multiple_topic_sub)  #TODO, on developing
148
149 ######################################################################
150 # build a CPack driven installer package
151 #include (InstallRequiredSystemLibraries)
152 #set (CPACK_RESOURCE_FILE_LICENSE
153 #    "${CMAKE_CURRENT_SOURCE_DIR}/doc/License.txt")
154 #
155 #set (CPACK_PACKAGE_VERSION_MAJOR "1")
156 #set (CPACK_PACKAGE_VERSION_MINOR "0")
157 #include (CPack)
"CMakeLists.txt" 158L, 4970C written
```

图 **12:** 编辑后的 src/CMakeLists.txt

4. 切换目录到 src/build 目录下，执行命令 cmake .. 重新生成编译管理文件 makefile。生成后执行 make 对工程进行编译。如下图 13所示:

```
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$ cmake ..
-- opensplice_home:
CMake Error at CMakeLists.txt:57 (message):
  'opensplice_home' system environment variable is not set!

  you can fix it by executing:

  source /home/vi/git/demo/project/src/../shell/source.sh


-- Configuring incomplete, errors occurred!
See also "/home/vi/git/demo/project/src/build/CMakeFiles/CMakeOutput.log".
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$   source /home/vi/git/demo/project/src/../shell/source.sh
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$ cmake ..
-- opensplice_home: /home/vi/run/opensplice
running chmod +x /home/vi/git/demo/project/src/../shell/demo.sh 2>&1
running /home/vi/git/demo/project/src/../shell/demo.sh --gen_idl  2>&1
mesg: ttyname failed: Inappropriate ioctl for device
[11-18 19:02:37] [info] [demo.sh:53 generate_idl()] execute cmd: rm -rf /home/vi/git/demo/project/src/build/idl_auto_gen/
[11-18 19:02:37] [info] [demo.sh:58 generate_idl()] generate src by idl file.
[11-18 19:02:37] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/HelloWorldData.idl
[11-18 19:02:37] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/Topic1Data.idl
[11-18 19:02:37] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/Topic2Data.idl
[11-18 19:02:37] [info] [demo.sh:61 generate_idl()] /home/vi/git/demo/project/idl/Topic3Data.idl
[11-18 19:02:37] [info] [demo.sh:64 generate_idl()] generate all c files into directory: /home/vi/git/demo/project/src/bui
ld/idl_auto_gen
-- build date: 2021-11-18 19:02:37
-- Configuring done
-- Generating done
-- Build files have been written to: /home/vi/git/demo/project/src/build
vi@vi-OptiPlex-7080:~/git/demo/project/src/build$ make
Scanning dependencies of target helloworld_pub
[  1%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/DDSEntitiesManager.c.o
[  2%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/HelloWorldDataPublisher.c.o
[  3%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/HelloWorldDataSacDcps.c.o
[  4%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/HelloWorldDataSplDcps.c.o
[  5%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic1DataSacDcps.c.o
[  7%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic1DataSplDcps.c.o
[  8%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic2DataSacDcps.c.o
[  9%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic2DataSplDcps.c.o
[ 10%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic3DataSacDcps.c.o
[ 11%] Building C object helloworld_pub/CMakeFiles/helloworld_pub.dir/__/idl_auto_gen/Topic3DataSplDcps.c.o
[ 13%] Linking C executable ../bin/helloworld_pub
[ 14%] Built target helloworld_pub
Scanning dependencies of target helloworld_pub2
[ 15%] Building C object helloworld_pub2/CMakeFiles/helloworld_pub2.dir/CheckStatus.c.o
[ 16%] Building C object helloworld_pub2/CMakeFiles/helloworld_pub2.dir/DDSEntitiesManager.c.o
[ 17%] Building C object helloworld_pub2/CMakeFiles/helloworld_pub2.dir/HelloWorldDataPublisher.c.o
[ 19%] Building C object helloworld_pub2/CMakeFiles/helloworld_pub2.dir/__/idl_auto_gen/HelloWorldDataSacDcps.c.o
[ 20%] Building C object helloworld_pub2/CMakeFiles/helloworld_pub2.dir/__/idl_auto_gen/HelloWorldDataSplDcps.c.o
[ 21%] Building C object helloworld_pub2/CMakeFiles/helloworld_pub2.dir/__/idl_auto_gen/Topic1DataSacDcps.c.o
[ 22%] Building C object helloworld_pub2/CMakeFiles/helloworld_pub2.dir/__/idl_auto_gen/Topic1DataSplDcps.c.o
```

图 13: 重新执行 cmake .. 和 make，对工程进行编译

编译成功后生成 src/build/bin/helloworld_pub2 可执行程序。

用户也可以直接执行 demo/project/shell 目录下的 demo.sh –build 进行重新编译。

到此已完成新加进程的工作，用户可以在新加的目录 src/helloworld_pub2/下面添加自己的源代码文件，进行后续应用的开发工作。

2.4.2 修改 demo 模板中现有进程配置

直接在 demo 现在进程目录里修改进程名为用户需要的名称即可以。以把 helloworld_pub 程序修改为 user_pub 为例。

1. 编辑 src/helloworld_pub/CMakeLists.txt 文件的第一行。修改 set(tgt helloworld_pub) 为 set(tgt user_pub)

2. 同上一样切换到 src/build 目录下面，执行 cmake .. 然后执行 make, 对新的工程进行编译。

3. 编译成功后在 src/build/bin 目录下面会生成 user_pub 文件

2.4.3 执行用户自己编译生成的进程

因为 opensplice 系统用到了一些配置文件，所以在执行前需要初始化一些环境变量，以告诉 opensplice 库去哪个目录加载相关配置文件。

这里以执行用户自己添加的 helloworld_pub2 程序为例。

1. 执行命令 source /git/demo/project/shell/source.sh , 以初始化 opensplice 相关的环境变量。(同一个 terminal 窗口只需要执行一次即可)

2. 切换目录到 demo/project/src/build/bin/目录下，执行./helloworld_pub2 如下图 14所示

```
vi@vi-OptiPlex-7080:~/git/demo/project/src/build/bin$
vi@vi-OptiPlex-7080:~/git/demo/project/src/build/bin$ ls
discover_topic  helloworld_pub2  multiple_topic_pub1  multiple_topic_sub  ospl-info.log
helloworld_pub  helloworld_sub   multiple_topic_pub2  ospl-error.log
vi@vi-OptiPlex-7080:~/git/demo/project/src/build/bin$
vi@vi-OptiPlex-7080:~/git/demo/project/src/build/bin$ source ~/git/demo/project/shell/source.sh
vi@vi-OptiPlex-7080:~/git/demo/project/src/build/bin$
vi@vi-OptiPlex-7080:~/git/demo/project/src/build/bin$ ./helloworld_pub2
=== powerd by www.yuanji.tech
=== HelloWorldPublisher
=== [Publisher] writing a message containing :
    userID  : 1
    Message : "Hello World"

Allocation validation [   OK   ] #allocated: 60, #remaining: 0, #expected: 0
vi@vi-OptiPlex-7080:~/git/demo/project/src/build/bin$
```

图 **14:** 执行 helloworld_pub2 用户自己的进程

3 技术支持

微信: vimcpp

电话: 15651950821

邮件: vimcpp@163.com

官网: www.yuanji.tech

博客: https://www.cnblogs.com/colin-vio/p

petalinux系统下载地址:

https://yuanji.tech/petalinux/petalinux.html