起源信息

ACI Data Distribution Service

ACI 数据分发服务

起源交互数据发布-订阅中间件

用户使用手册

版本 1.0

The China Leader in DDS

目录

第 1 章 概述

本章描述 ACI 数据分发服务的基本概念。ACI 是面向分布式实时应用的网络中间件。ACI 数据分发服务简化了应用程序开发、部署和维护，并在各种传输网络上提供快速、可预测的时间关键型数据分发。使用 ACI 数据分发服务中间件，您可以:

- 执行复杂的一对多和多对多网络通信。
- 应用程序使用 OMG 标准接口 DDS 发布和订阅数据。
- 定制应用程序操作，满足各种实时性、可靠性和服务质量要求。
- 提供应用透明的容错能力和应用的鲁棒性。

本章介绍了中间件的基本概念和常见的通信模型，并描述了 ACI 数据分发服务的特性集如何满足实时系统的需求。

1. ACI 数据分发服务

ACI 数据分发服务是面向实时分布式应用的网络中间件。 它提供了通信服务，程序员需要在嵌入式 和/或企业设备或节点之间分发时间关键数据。ACI 数据分发服务使用发布订阅通信模型，使数据分发更高效、更健壮。

ACI 数据分发服务在 OMG 的数据分发服务 (DDS) 中为实时系统实现了以数据为中心的发布-订阅 (DCPS) API。DDS 是第一个为满足实时系统需求而开发的标准。DCPS 提供了一种在分布式系统中传输数 据的有效方法。

使用 ACI 数据分发服务，系统设计人员和程序员从一个容错和灵活的通信基础设施开始，该基础设施将在各种计算机硬件、操作系统、语言和网络传输

协议上工作。ACI 数据分发服务是高度可配置的，因此程序员可以调整它来满足应用程序的特定通信需求。

2. 消息中间件介绍

中间件是介于应用程序和操作系统之间的软件层。网络中间件将应用程序与底层计算机体系结构、操作系统和网络堆栈的细节隔离开来。网络中间件允许应用程序发送和接收信息，而无需使用底层协议(如套接字和 TCP 或 UDP/IP)进行编程，从而简化了分布式系统的开发。图 1.1 网络中间件

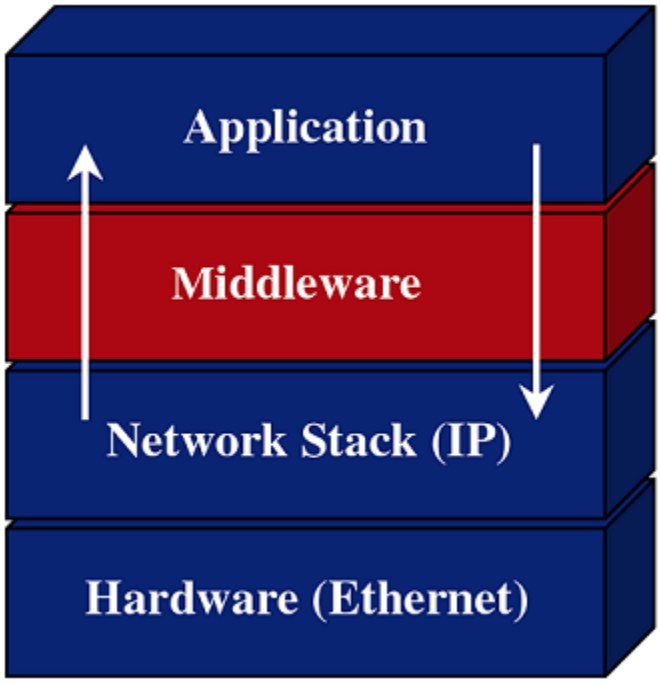

图 1.1

ACI 数据分发服务是将应用程序与原始操作系统网络堆栈隔离的中间件。发布订阅(PS)中间件提供了一种简单直观的数据分发方式。它将创建和发送的软件解耦。

3. 网络通信模型介绍

数据—数据发布者—来自接收和使用数据的软件—数据订阅者。发布者只需声明他们发送数据的意图，然后发布数据。订阅者声明他们的接收意图，然后由中间件自动交付数据。图 3.1 通信模型

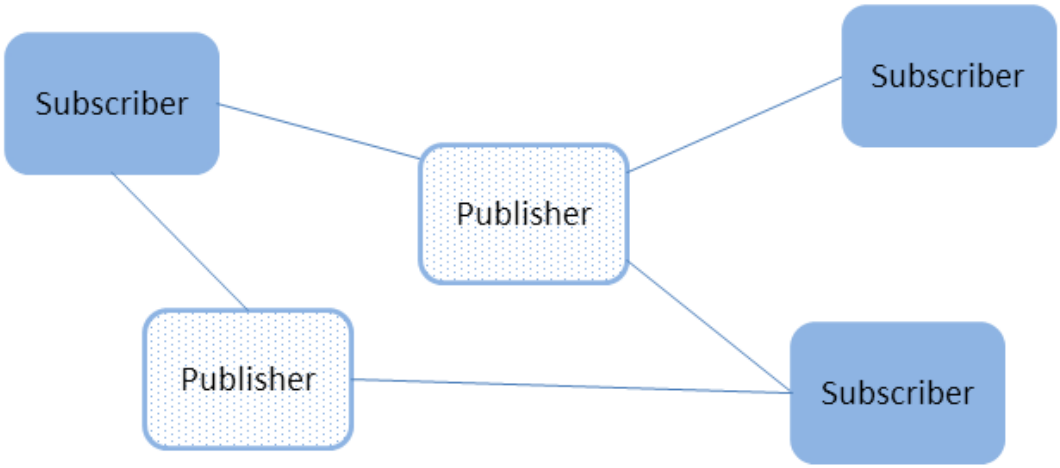

图 3.1

尽管模型简单，但是 PS（发布订阅）中间件可以处理复杂的信息流模式。PS 中间件的使用带来了更简单、更模块化的分布式应用程序。也许最重要的是，PS 中间件可以自动处理所有网络杂务，包括连接、故 障和网络更改，从而消除了用户应用程序为所有这些特殊情况编写程序的需要。有经验的网络中间 件开发人员所知道的是，处理特殊情况占到工作和代码的 80%以上。

4. ACI 数据分发服务特点

ACI 数据分发服务支持超越基本发布-订阅模型的机制。关键的好处是，使

用 ACI 数据分发服务进行通信的应用程序是完全解耦的。他们的设计时间很少花在如何处理他们的相互交互上。特别是，应用程序从来不需要关于其他参与应用程序的信息，包括它们的 1）确定谁应该收到这些信息 2）收件人的位置 3）如果信息无法发送该怎么办。

这是因为 ACI 数据分发服务允许用户指定服务质量(QoS)参数，从而配置自动发现机制，并指定发送和接收消息时使用的行为。这些机制是预先配置的，不需要用户做进一步的工作。通过以完全匿名的方式交换消息，ACI 数据分发服务大大简化了分布式应用程序的设计，并鼓励模块化、结构良好的程序。此外，ACI 数据分发服务包括以下功能，旨在满足分布式实时应用的需求:

- 以数据为中心的发布-订阅通信简化了分布式应用程序编程。
- 为管理相同数据的多个源提供清晰的语义。
- 高效的数据传输、可定制的服务质量和错误通知。
- 通过回调例程通知数据到达，以最小化延迟。
- 能够有效地向多台计算机发送相同的消息。
- 用户定义的数据类型使你能够定制发送到每个应用程序的信息的格式。
- 可靠的消息传递使订阅应用程序能够指定可靠的样品交付。
- 多个独立的通信网络(域)，应用程序只能参与它们所属的域。可以将单个应用程序配置为参与多个域。

第 2 章 DCPS

本章描述了 ACI 数据分发服务(以数据为中心的发布订阅(DCPS)标准)使用

的正式通信模型。DCPS 是 (通过标准化 API) 的形式化，是 1.3 节中提出的发布-订阅通信模型的扩展。

1. DCPS 服务介绍

DCPS 是 OMG 标准接口 DDS(以数据为中心的发布-订阅服务)的一部分，它处理以数据为中心的发布订阅通信。DDS 标准定义了一个独立于语言的发布订阅通信模型，分布式通信的发布订阅方法是一种通用机制，可以被许多不同类型的应用程序使用。本章描述的 DCPS 模型扩展了发布订-阅模型，以满足实时、数据关键型应用程序的特定需求。您将看到，它提供了几种机制，允许应用程序开发人员控制通信的工作方式，以及中间件如何处理资源限制和错误条件。

DCPS 的“以数据为中心”部分描述了 API 设计支持的基本概念。以数据为中心的系统由数据发布者和数据订阅者组成。通信基于将命名流中已知类型的数据从发布者传递给订阅者。在分布式系统中，实时通信通常更自然地适合以数据为中心的模型。

2. DCPS 应用特点

DCPS，特别是 ACI 数据分发服务实现，非常适合于实时应用程序。例如，实时应用程序通常需要 以下特性:

- 高效的实时系统需要高效的数据收集和传输。
- 关键的数据传输路径应该只引入最小的延迟。
- 对于周期性的数据交换，在延迟和带宽方面都比客户机-服务器更有

效。

- 与客户机-服务器体系结构相比，大大减少了通过网络发送数据所需的开销。
- 偶尔的订阅请求(在低带宽下)会替代大量的高带宽客户机请求。
- 一旦新的发布数据样本可用，它就被发送到相应的订阅。

实时应用程序通常关心交付周期数据的确定性以及交付事件数据的延迟。一旦缓冲区被引入数据流 以支持可靠的连接，新的数据就可能被引入在等待确认接收到旧数据时，不可预测地保持未交付状态一段时间。由于发布订阅本质上不需要可靠的连接，所以实现(如 ACI 数据分发服务)可以在新数据的确定性交付和所有数据的可靠交付之间提供可配置的权衡。

3. DCPS 数据结构

在使用 DCPS 的一组应用程序中，不同的应用程序不会自动知道正在发送的数据的结构，也不一定以相同的方式解释数据(例如，如果它们使用不同的操作系统，使用不同的语言编写，或者使用不同的编译器编译)。必须有一种方法不仅共享数据，而且共享有关数据结构的信息。在 DCPS 中，使用 OMG IDL(一种描述数据的独立于语言的方法)在应用程序之间共享数据定义。(待完成)

第 3 章 QOS 服务策略

第 4 章 域

本章描述了域与域参与者，您可能有几个独立的 DCPS 应用程序都运行在同一组计算机上。您可能希望隔离这些应用程序中的一个(或多个)，这样它就不会受到其他应用程序的影响。为了解决这个问题，DCPS 有一个称为域的概念。

1. 域参与者

域表示逻辑的、孤立的通信网络。在不同域中的同一组主机上运行的多个应用程序彼此完全隔离 (即使它们在同一台机器上)。属于不同域的数据写入器和数据阅读器永远不会交换数据。希望使用 DCPS 交换数据的应用程序必须属于同一个域。要属于域，DCPS 接口用于配置和创建具有特定域索引的域参与者。域由域索引(一个整数值)区分。创建具有相同域索引的域参与者的应用程序属于相同的域。域参与者拥有主题、发布者和订阅者，而订阅者又拥有数据编写者和数据阅读器。因此，所有 DCPS 实体都属于一个特定的域。

通过创建具有不同域索引的多个域参与者，应用程序可以同时属于多个域。然而，发布者/数据编写器和订阅者/数据阅读器只属于它们创建时所在的域。

2. 使用场景

如前所述，多个域可用于应用程序隔离，这在用户使用同一网络甚至同一台计算机测试其应用程序时非常有用。通过为每个用户分配不同的域，可以确保一个用户的应用程序生成的数据不会意外地被另一个用户接收。此外，域可能是一种扩展和构建由多节点子系统组成的更大系统的方法。每个子系统将使用一个内

部域来进行系统内部通信，并使用一个外部域来连接到其他子系统。

3. 接口示例

```
/* 资源初始化 */
DDS_ParticipantFactory_init();

/* 创建参与者 */
DDS_DomainParticipant* pstParticipant =
DDS_DomainParticipantFactory_create_participant(0, NULL);
if (NULL == pstParticipant)
{
    printf("ERROR! Failed to create participant!");
}
```

第 5 章 主题

1. 数据主题

数据类型的共享知识是不同应用程序与 DCPS 通信的必要条件。应用程序还必须共享一种方法来标识要共享哪些数据。数据(任何数据类型的数据)通过使用名为 a 的名称进行惟一区分的话题。根据定义，主题对应于单个数据类型。但是，有几个主题可能涉及相同的数据类型。主题将数据写入器和数据阅读器连接起来。DataWriter 是应用程序中的一个对象，它告诉 ACI 数据分发服务(以及其他应用程序)它具有某个主题的一些值。对应的 DataReader 是应用程序中的一个对象，它告诉 RTI 数据分发服务它希望接收相同主题的值。从 DataWriter 传递到 DataReader 的数据属于与主题关联的数据类型。如图 4.1

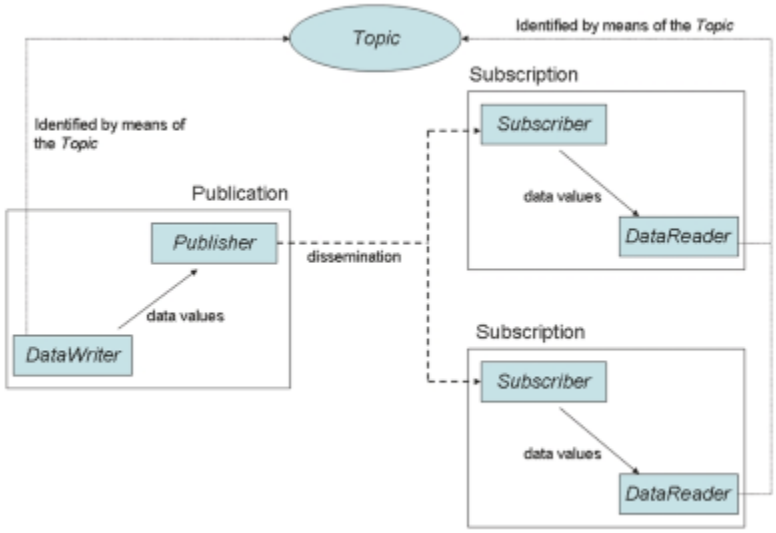

图 4.1

2. 使用场景

例如，考虑一个在应用程序之间分配股票报价的系统。应用程序可以使用名为 StockPrice 的数据类型。股票价格数据类型可以有多个主题，每个公司的股票都有一个主题，如 IBM、MSFT、GE 等。数据类型:上涨空间结构股票价格{浮动价格; 时间的时间戳;}; 主题: “ IBM ” 主题: “ MSFT ” 主题: “ GE ”

现在，一个跟踪客户投资组合当前价值的应用程序将订阅客户所拥有股票的所有主题。当每只股票的价值发生变化时，相应主题的新价格将被发布并发送给应用程序。

3. 接口示例

```
/* 创建主题 */
DDS_Topic* pstTopic = DDS_DomainParticipant_create_topic(pstParticipant,
"Example HelloWorld", "HelloWorld", NULL);
if (NULL == pstTopic)
{
```

```
        printf("ERROR! Failed to create topic!");
    }
```

第 6 章 发布

1. 数据写入器

在 DCPS 中，应用程序必须使用 api 创建实体，以便在彼此之间建立发布订阅通信。与数据本身相关的实体和术语已经讨论过—主题、实例和示例。本节将介绍用户代码必须创建的发送和接收数据的 DCPS 实体。注意，实体实际上是一个基本的 DCPS 概念。在面向对象术语中，实体是派生其 DCPS(主题、数据写入器、数据阅读器、发布者、订阅者、参与者)的基类。

数据编写者和发布者之间的关联通常被称为发布，尽管从来没有创建过一个称为发布的 DCPS 对象。应用程序使用数据阅读器访问通过 DCPS 接收到的数据。数据阅读器与单个主题关联。在一个应用程序中可以有多个数据阅读器和主题。但每个主题最多只有一个数据写入器。

2. 使用场景

应用程序使用数据写入器发送数据。数据写入器与单个主题关联。在一个应用程序中可以有多个数据写入器和主题。发布者是负责实际数据发送的 DCPS 对象。主题拥有并管理数据编写者。一个 Data Write 只能由一个主题拥有。当用户代码调用 DataWriter 上的 write()方法，数据样本被传递给发布者对象，发布者对象在网络上执行数据的实际传播。。

3. 接口示例

```
/* 创建写入器 */
DDS_DataWriter* pstDataWriter = DDS_Topic_create_datawriter(pstTopic,
NULL, NULL);
if (NULL == pstDataWriter)
{
    printf("ERROR! Failed to create dataWriter!");
}

/* 写数据发送 */
DDS_ReturnCode_t retureCode = DDS_StringDataWriter_write(pstDataWriter,
" hello world! " , 64);
if (DDS_RETCODE_ERROR == retureCode)
{
    printf("ERROR! Failed to send message, Insufficient memory space!\n");
}
```

第 7 章 订阅

1. 数据阅读器

订阅者是负责实际接收已发布数据的 DCPS 对象。订阅者拥有并管理数据阅读器。一个数据阅读器只能由一个主题拥有，当数据发送到应用程序时，首先由订阅者处理；然后，数据样本存储在 DataReader 中。用户代码既可以注册一个侦听器，以便在新数据到达时调用，也可以使主动轮询 DataReader 以获取新数据。

2. 使用场景

数据阅读器和主题之间的关联通常被称为订阅，尽管从来没有创建过称为订阅的 DCPS 对象。发布订阅通信模型类似于杂志出版物和订阅。可以将出版物想

象为周刊，例如：《人民日报》，主题是期刊的名称(在本例中是字符串“人民日报”)。类型指定信息的格式，例如：印刷杂志。用户数据是每个样本(每周一期)的内容(文本和图形)。中间件是分发服务(通常是中国邮政服务)，它将杂志从创建它的地方(印刷厂)发送到单个订阅者(人们的家)。请注意，通过订阅出版物，订阅者正在请求该出版物的当前和将来的样本(例如《新闻周刊》每周一次)，这样，当新样本发布时，就无需再提交另一个数据请求。

3. 接口示例

```
/* 监听回调函数 */
VOID STACALL ReceiveData(DDS_DataReader* pstDataReader)
{
    UINT32 dataNum = 0;
    DataSeq stDataSeq;
    DataLen dataLen;
    /* 收数据 */
    DDS_ReturnCode_t retCode = DDS_StringDataReader_take(
    pstDataReader, &stDataSeq);
    if (DDS_RETCODE_NO_DATA != retCode)
    {
        for (dataNum = 0; dataNum < stDataSeq.length; dataNum++)
        {
            printf("LEN:%d, MSG: %s \n", dataLen, DDS_StringSeq_get(
            &stDataSeq, dataNum, &dataLen));
        }
        /* 释放缓存 */
        DDS_StringDataReader_return_loan(pstDataReader, &stDataSeq);
    }
}

/* 创建阅读器 */
DDS_DataReader* pstDataReader = DDS_Topic_create_datareader(pstTopic, NULL,
ReceiveData);
if (NULL == pstDataReader)
{
    printf("ERROR! Failed to create dataReader!");
}
```

第 8 章 应用示例

本章主要演示几种常见的使用场景，发布应用（只发送数据）、订阅应用（只接收数据）、发布-订阅应用（同时能发送和接收数据，如果为同一主题，可自发自收）。

1. 发布应用

1) 创建域为 0 的参与者。
2) 在该参与者下创建名称为 Example HelloWorld，类型为 HelloWorld 的主题。
3) 设置写入器 qos。
4) 在该主题下创建写入器。
5) 循环发送数据。

```
#include "../Include/GlobalDefine.h"

void main_pub()
{
	UINT32 i = 0;
	CHAR* buff = NULL;
	UINT32 uiLen = 32;
	DDS_ReturnCode_t retureCode;
	DDS_DataWriterQos stWriterQos;

	/* 资源初始化 */
	DDS_ParticipantFactory_init();

	/* 创建参与者 */
	DDS_DomainParticipant* pstParticipant =
	DDS_DomainParticipantFactory_create_participant(0, NULL);
	if (NULL == pstParticipant)
	{
		printf("ERROR! Failed to create participant!");
		return;
	}
```

```
/* 创建主题 */
DDS_Topic* pstTopic = DDS_DomainParticipant_create_topic(pstParticipant,
"Example HelloWorld", "HelloWorld", NULL);
if (NULL == pstTopic)
{
    printf("ERROR! Failed to create topic!");
    return;
}

/* 获取写入器 qos */
retureCode = DDS_Topic_get_default_datawriter_qos(pstTopic, &stWriterQos);
if (DDS_RETCODE_ERROR == retureCode)
{
    printf("ERROR! Data writer setup error !");
    return;
}

/* 设置写入器可靠性 */
stWriterQos.reliability.kind = RELIABLE_RELIABILITY_QOS;

/* 创建写入器 */
DDS_DataWriter* pstDataWriter = DDS_Topic_create_datawriter(pstTopic,
&stWriterQos, GetWinTime);
if (NULL == pstDataWriter)
{
    printf("ERROR! Failed to create dataWriter!");
    return;
}

/* 待发送数据内存分配 */
buff = (CHAR*)DDS_STATIC_MALLOC(uiLen);
if (NULL == buff)
{
    printf("ERROR! Insufficient memory space.\n");
    return;
}

while (TRUE)
{
    sprintf_s(buff, uiLen, "Hello World from QY ! (%d)", ++i);
    /* 发数据 */
    retureCode = DDS_StringDataWriter_write(pstDataWriter, buff, uiLen);
    if (DDS_RETCODE_ERROR == retureCode)
```

```
        {
            printf("ERROR! Failed to send message, Insufficient memory space!\n");
        }
        Sleep(1000);
    }
}
```

2. 订阅应用

1) 创建域为 0 的参与者。
2) 在该参与者下创建名称为 Example HelloWorld，类型为 HelloWorld 的主题。
3) 设置阅读器 qos。
4) 在该主题下创建阅读器。
5) 循环接收数据。

```
#include "../Include/GlobalDefine.h"

void main_sub()
{
    UINT32 dataNum = 0;
    DataSeq stDataSeq;
    DataLen dataLen;
    DDS_ReturnCode_t retureCode;
    DDS_DataReaderQos stReaderQos;

    /* 资源初始化 */
    DDS_ParticipantFactory_init();

    /* 创建参与者 */
    DDS_DomainParticipant* pstParticipant =
    DDS_DomainParticipantFactory_create_participant(0, NULL);
    if (NULL == pstParticipant)
    {
        printf("ERROR! Failed to create participant!");
        return;
    }

    /* 创建主题 */
    DDS_Topic* pstTopic = DDS_DomainParticipant_create_topic(pstParticipant,
    "Example HelloWorld", "HelloWorld", NULL);
    if (NULL == pstTopic)
    {
```

```
        printf("ERROR! Failed to create topic!");
        return;
    }

    /* 获取阅读器 qos */
    retureCode = DDS_Topic_get_default_datareader_qos(pstTopic, &stReaderQos);
    if (DDS_RETCODE_ERROR == retureCode)
    {
        printf("ERROR! Data reader setup error !");
        return;
    }

    /* 设置阅读器可靠性 */
    stReaderQos.reliability.kind = RELIABLE_RELIABILITY_QOS;

    /* 创建阅读器 */
    DDS_DataReader* pstDataReader = DDS_Topic_create_datareader(
    pstTopic, &stReaderQos, NULL);
    if (NULL == pstDataReader)
    {
        printf("ERROR! Failed to create dataReader!");
        return;
    }

    while (TRUE)
    {
        /* 循环主动收数据 */
        DDS_ReturnCode_t retCode = DDS_StringDataReader_take(pstDataReader,
        &stDataSeq);
        if (DDS_RETCODE_NO_DATA != retCode)
        {
            for (dataNum = 0; dataNum < stDataSeq.length; dataNum++)
            {
                printf("LEN:%d, MSG: %s \n", dataLen,
                DDS_StringSeq_get(&stDataSeq, dataNum, &dataLen));
            }
            /* 释放缓存 */
            DDS_StringDataReader_return_loan(pstDataReader, &stDataSeq);
        }
    }
}
```

3. 发布-订阅应用

1) 创建域为 0 的参与者。
2) 在该参与者下创建名称为 Example HelloWorld，类型为 HelloWorld 的主题。
3) 设置写入器 qos。
4) 在该主题下创建写入器，并注册消息发送时间戳回调函数。
5) 设置阅读器 qos。
6) 在该主题下创建阅读器，并注册消息监听回调函数。
7) 循环接收数据。

此例子由于写入器和阅读器属于统一主题，所有可自发自收。

```
/* 发布回调函数，创建时间戳 */
VOID STACALL GetWinTime(Time_t* pstTimeInfo)
{
    pstTimeInfo->sec = time(NULL);
}

/* 订阅回调函数 */
VOID STACALL ReceiveData(DDS_DataReader* pstDataReader)
{
    UINT32 dataNum = 0;
    DataSeq stDataSeq;
    DataLen dataLen;
    Time_t timeInfo;
    char timeStr[64];
    /* 收数据 */
    DDS_ReturnCode_t retCode = DDS_StringDataReader_take(pstDataReader, &stDataSeq);
    if (DDS_RETCODE_NO_DATA != retCode)
    {
        for (dataNum = 0; dataNum < stDataSeq.length; dataNum++)
        {
            printf("TIME:%s, LEN:%d, MSG: %s \n", timeStr, dataLen,
            DDS_StringSeq_get(&stDataSeq, dataNum, &dataLen));
        }
        /* 释放缓存 */
        DDS_StringDataReader_return_loan(pstDataReader, &stDataSeq);
    }
}

void main_sub_pub()
{
    UINT32 i = 0;
    CHAR* buff = NULL;
```

```
UINT32 uiLen = 32;
DDS_ReturnCode_t retureCode;
DDS_DataWriterQos stWriterQos;
DDS_DataReaderQos stReaderQos;

/* 资源初始化 */
DDS_ParticipantFactory_init();

/* 创建参与者 */
DDS_DomainParticipant* pstParticipant =
DDS_DomainParticipantFactory_create_participant(0, NULL);
if (NULL == pstParticipant)
{
    printf("ERROR! Failed to create participant!");
    return;
}

/* 创建主题 */
DDS_Topic* pstTopic = DDS_DomainParticipant_create_topic(pstParticipant,
"Example HelloWorld", "HelloWorld", NULL);
if (NULL == pstTopic)
{
    printf("ERROR! Failed to create topic!");
    return;
}

/* 获取写入器 qos */
retureCode = DDS_Topic_get_default_datawriter_qos(pstTopic, &stWriterQos);
if (DDS_RETCODE_ERROR == retureCode)
{
    printf("ERROR! Data writer setup error !");
    return;
}

/* 设置写入器可靠性 */
stWriterQos.reliability.kind = RELIABLE_RELIABILITY_QOS;

/* 创建写入器 */
DDS_DataWriter* pstDataWriter = DDS_Topic_create_datawriter(pstTopic,
&stWriterQos, GetWinTime);
if (NULL == pstDataWriter)
{
    printf("ERROR! Failed to create dataWriter!");
```

```
        return;
    }

    /* 获取阅读器 qos */
    retureCode = DDS_Topic_get_default_datareader_qos(pstTopic, &stReaderQos);
    if (DDS_RETCODE_ERROR == retureCode)
    {
        printf("ERROR! Data reader setup error !");
        return;
    }

    /* 设置阅读器可靠性 */
    stReaderQos.reliability.kind = RELIABLE_RELIABILITY_QOS;

    /* 创建阅读器 */
    DDS_DataReader* pstDataReader = DDS_Topic_create_datareader(pstTopic,
    &stReaderQos, ReceiveData);
    if (NULL == pstDataReader)
    {
        printf("ERROR! Failed to create dataReader!");
        return;
    }

    /* 待发送数据内存分配 */
    buff = (CHAR*)DDS_STATIC_MALLOC(uiLen);
    if (NULL == buff)
    {
        printf("ERROR! Insufficient memory space.\n");
        return;
    }

    while (TRUE)
    {
        sprintf_s(buff, uiLen, "Hello World from QY ! (%d)", ++i);
        /* 发数据 */
        retureCode = DDS_StringDataWriter_write(pstDataWriter, buff, uiLen);
        if (DDS_RETCODE_ERROR == retureCode)
        {
            printf("ERROR! Failed to send message, Insufficient memory space!\n");
        }
        Sleep(1000);
    }
}
```

第 9 章 结束