基于前置层和消息总线的数据传输①

王东阳[1], 杨春生[1], 江 源[1], 杨晴雅[1], 化晨冰[1], 陈 璐[2]

[1](国网临沂供电公司, 临沂 276003)
[2](安徽南瑞继远电网技术有限公司, 合肥 230088)

摘 要: 大量模块或者第三方数据接入以及上层用户定制应用需求的繁杂性必定要求系统在数据传输上具备灵活、通用、可扩展等性能, 因此设计和实现一种驿站中间层数据传输方案成为开发人员的一项关键任务. 数据接入方面增加统一处理数据的前置层, 设计中采用链路连接层、规约处理层、前置管理层的架构并构建了前置实时数据处理库; 考虑到跨平台和数据类型区分处理清晰原则, 构建了以 ICE 中间件和 JSON 串为核心的消息总线数据传输, 设计与上层应用程序和前置模块的交互接口. 总线传递机制将应用软件层和前置层区分来, 按照协定好的消息类型通过总线实现信息交互, 屏蔽了系统的多种不同的上层应用需求和前置开发通信协议特性, 提高了开发人员的工作效率以及系统本身的健壮性、扩展性, 同时也降低了后续维护人员的维护难度.

关键词: 消息总线; JSON 串; ICE 中间件; 前置开发

Date Transmission Based on Hierarchical Front-Frame and Message Bus

WANG Dong-Yang[1], YANG Chun-Sheng[1], JIANG Yuan[1], YANG Qing-Ya[1], HUA Chen-Bing[1], CHEN Lu[2]

[1](Linyi Power Supply Company, Linyi 276003, China)
[2](Anhui NARI Jiyuan Electric Power System Tech. Co. Ltd., Hefei 230088, China)

Abstract: Numerous modules or third-party data access and multifarious upper application require system data transmission with flexible, universal and extensible performance. Therefore, a middle layer data transmission design becomes a key task of developers. Add front layer in data access, which uses the structure of communication link layer, protocol processing layer, management layer, and front real-time data processing library. Considering the clear principle of cross-platform and data type distinction, the message bus data transmission with ICE middleware and JSON string as the core is constructed and the interaction interface between upper application and front module is designed. The bus transfer mechanism separates the application software layer from the pre-layer, and implements the information exchange through the bus according to the agreed message types. It shields the system's various upper-layer application requirements and the pre-developed communication protocol characteristics, and improves the developer's work efficiency and the system's robustness, scalability, and also reduces the follow-up maintenance personnel to maintain the difficulty.

Key words: message bus; JSON; ICE; front development

1 引言

随着一体化设计要求的提高, 变电站智能辅助监控系统[1]一体化设计涉及到多子系统(安防监控、消防火灾、门禁管理、环境监测与控制等)数据接入问题, 之前各厂家按照独立建设方案造成了一个个堆积式的信息孤岛. 为解决上述这些问题, 前期的辅助监控系统采用了点到点紧耦合方法, 这种方法需要对系统做出较大改动, 进行大规模的代码重写, 这一过程的工程任务其实相当于新的系统工作, 并且这种解决方案缺乏柔性, 更不具备特性化定制需求. 另外, 电力变电站辅助系统的各个功能, 用户往往是分步实施的, 后期建设的功能如果使用第三方软件, 接口的标准化显得

① 收稿时间:2016-04-09;收到修改稿时间:2016-05-30 [doi:10.15888/j.cnki.csa.005550]

© 中国科学院软件研究所 http://www.c-s-a.org.cn

至关重要, 传统的方法是约定私有的接口方式进行数据转化和数据交换, 应用接入非常困难、投运周期长且影响系统的整体性能.

近年来, 计算机接口技术的迅速发展, 网络、串行通信等接口技术及应用已相对成熟, 并且中间件技术[2-4]、总线技术[5-7]也得以业界的广泛关注, 这些都为本文的工作开展提供了良好的理论基础. 为了实现一体化辅助系统接入的各类信息的采集、控制任务, 本文设计前置模块这一重要环节, 达到接入扩展性好、稳定性好以及通信设备接入简单便捷等目的. 前置模块开发层设计了链路层、应用规约层、前置管理层构架思想, 通过各种线缆或无线网络连接到一起, 实现了串口设备、网络设备的信息接入和输出, 并且设计了前置实时库缓存各类接入采集输出信息, 该实时库内数据信息按照接入预定编号和信息类型(数字量输入、数字量输出、模拟量数据)存储. 基于 ICE 中间件技术的总线型集成方法, 它提供了一个灵活、标准的消息通信构架, 降低了各应用层和前置设备层之间依赖性和耦合性以及更加容易和有效的被连接, 实现了相互之间的无缝通信, 完成消息总线设计后, 用户只需向该消息总线注册即可发送和接收数据. 这种良好的扩展性, 对于新的上层应用开发和前置设备的接入实现了“即插即用性”, 这种整体性数据传输架构设计思路将会很好的服务于变电站辅助系统运行和维护, 为电力辅助系统高级应用功能的挖掘和系统可扩展性提供可靠保障.

2　前置模块

信息采集包括对各终端设备的数据采集和第三方软件接入数据处理等, 因此, 系统在设计时必须考虑多种硬件接口(网口、串口等)、多种通信协议(Modbus 协议、电力 104 等)的实现问题, 并且还要考虑信息采集接入接口和协议的扩展性、灵活性, 也就是说系统接入只要确定接口和通信协议就能够实现信息数据的采集. 在实际前置接入规约开发中, 我们需要考虑的问题是设备有效快速的接入, 对于特定设备其通信协议是个性的, 通信方式(串口、UDP 或者 TCP)是可以枚举的, 数据通过消息总线上送方式是由前置开发者设计的. 因此, 在整个流程中, 出现因通信协议内容不可控外, 需要根据现场设备开发的情况. 本文思路是将规约层的通信协议开放出来, 工程开发人员根据不同的设备再进行开发, 这样就有利于应用于不同场合的设备增加, 最后优势表现在前置模块的稳定性、通用性和接入能力强等方面. 基于上述考虑, 本文设计了前置模块开发层, 其整体设计思路如图 1 所示, 前置模块分为链路层、规约层、管理层三层, 并以前置实时库为数据处理核心区域, 实现信息上传下送.

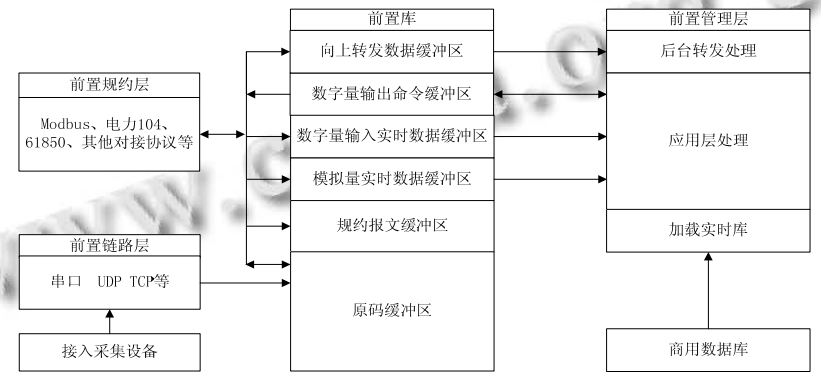

图 1　前置层架构

系统采用数据库实现配置的管理工作, 整个数据信息流从设备到链路层再到前置规约层, 根据需求对应存入缓冲区后, 然后通过前置管理层送往消息总线、实现信息传递. 现按照开发步骤和数据传递过程现将前置层说明如下:

(1) 前置链路层: 负责与接入设备建立通信链路等, 通信方式主要包括串口、UDP、TCP/IP 等, 作为接入设备的共性的部分, 系统对上述通信方式进行封

© 中国科学院软件研究所 http://www.c-s-a.org.cn

装, 通过配置实现不同接入设备调取不同通信方式.

(2) 前置规约层: 该层实现各种通信协议开发, 针对某接入设备协议(如 Modbus 协议)开发相应规约, 该规约一方面实现通过链路层与设备进行信息交互, 另一方面根据需求实现对前置库中各类缓冲区数据的写入. 该层预留出统一封装接口, 方便前置开发人员开发.

(3) 前置库: 前置库作为接入采集数据处理的核心部分, 为了保证数据信息传输可靠性, 系统根据在前置模块中设计了原码, 规约报文, 模拟量实时数据、数字量输入输出和对上转发数据缓冲区. 其中, 原码缓冲区为包括通信链路头数据在内的完整消息, 前置报文缓冲区为设备通信协议的规约报文, 而模拟量实时数据、数字量输入输出缓冲区数据为前置规约解析出的用户数据, 对上转发数据缓冲区数据为系统需要上送后台数据.

(4) 前置管理层: 该层一方面通过数据库下装设备信息完成配置后, 系统将商用数据库信息通过加载实时库程序下装到实时库中, 以备前置层数据流处理所用, 另一方面将前置库中不同缓冲区数据根据需求送往消息总线, 实现对上层应用或者后台系统的数据传输.

3 消息总线

信息数据的有效可靠的传递是系统集成的关键所在, 消息总线传输方法为这一关键提供了良好的方案. 首先, 消息总线能够屏蔽前置开发层各种模块差异性, 简化上层应用程序与模块之间数据传输, 利用其高效的消息传输机制为大型应用系统集成开发提供透明的数据转发服务. 转发服务模式有点对点、消息队列和发布/订阅三种模式, 三种模式各有特点, 点对点具有很强的时间和空间耦合性, 使其数据传输灵活性受限; 消息队列传递消息, 解决了时间和空间耦合性问题, 但是相关的队列服务器需要单独配置, 存在瓶颈和单点失效问题, 可靠性不强, 如队列发生丢失, 可能会影响整个系统, 消息延时也会相应增加; 发布/订阅模式中发布者和订阅者之间通过主题相关联, 可实现数据传递双方空间、时间和消息通信多方面松耦合性, 这种模式在大型系统的集成和开发中优势明显, 因此, 本文采用基于发布/订阅的消息总线模式. 其次, 考虑到系统跨平台需求和总线传输消息类型较多, 选择合适的通讯中间件和数据包传输格式对于整个系统数据传输至关重要.

3.1 中间件

中间件能够解决跨平台和异构网络应用问题, 在具体应用上它是一个 API 定义的分布式软件管理架构, 在分布式的客户和服务之间扮演承上启下的角色, 具有强大的通信能力和良好的扩展性. 以 CORBA 为代表的中间件为分布式系统带来了巨大变革, 但是由于中间件复杂性和学习困难性等原因都没有真正占领计算市场, 而作为原 CORBA 核心成员在 GPL 协议下开发的轻量级面向对象的分布式 ICE 中间件“提供了一个和 CORBA 同样强大却摒除了 CORBA 的各种缺陷的中间件”, 其在概念上与 CORBA 基本一致, 但是 ICE 解决了过去长期困扰中间件的低效问题, 并提供了数据包协议(UDP)支持、异步方法分派、对象保存和接口聚集等技术支持, 从而建立了一个更简单高效的基础架构.

ICE 结构如图 2 所示, 包括客户端和服务端、ICE 核心、ICE API、对象适配器、ICE 代理和 ICE 骨架等部分, 其中客户 ICE 核心、服务器核心和对象适配器来自于库文件, 代理和骨架由 Slice 语言生产, 客户应用和服务器应用由上层应用程序员编写.

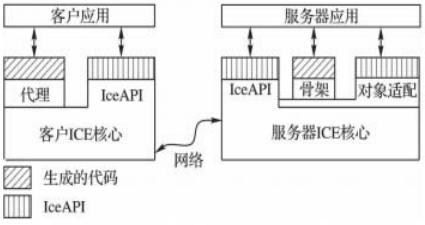

图 2 ICE 结构

本文对 ICE 的 IceStorm 在此重点介绍. IceStorm 是 ICE 服务消息发布/订阅服务, 能够消除消息发布者和订阅者之间的耦合关系, 其充当发布者和订阅者的中介. 当发布者准备分发消息时, 它只需要简单的向 IceStorm 服务器发出一个请求, 而不需要对订阅者有任何了解, 由 IceStorm 服务器负责把消息递送给订阅者, IceStorm 还负责处理由于订阅者的行为有问题或订阅者不存在所造成的异常. 发布者不再需要关注它的订阅, 甚至不需要知道此时是否有订阅者, 与此类似, 订阅者仅需要与 IceStorm 服务器进行简单的交互, 完成像订阅和取消订阅任务, 就可以获得感兴趣的消

© 中国科学院软件研究所 http://www.c-s-a.org.cn

息. 发布者可以专注于其应用程序特有的功能, 而不是管理订阅者这样的琐事, 订阅者也可以集中精力解决对消息的处理, 而不用关心其它琐事. 要把 IceStorm 整合到应用程序中, 消息的发布者和使用者仅需要做出很少的改动. 发布消息时按照主题进行分类, 订阅者订阅感兴趣主题, 只有那些被订阅的主题才会发布给订阅者, 订阅和取消订阅都很简单. IceStorm 支持任意多个主题, 主题还可以根据其 cost 属性结成联盟, IceStorm 作为联盟服务运行时, 服务的多个琐实例可以在不同的机器上运行, 使处理负载分摊到许多 CPU 上. IceStorm 服务还允许指定服务标准质量, 从而在可靠性和性能之间做适当的折衷, 发布者把消息发布给 IceStorm 服务, 由它发布给订阅者. 这样, 发布者发布的单个事件就可以发送给多个订阅者, 如果需要把消息分发给大量的应用程序组件, IceStorm 是十分合适的选择. 关于 ICE 其他知识介绍, 本文就不逐一说明, 详细了解请参考 ZeroC 官网(https://zeroc.com/)ICE 资料.

3.2 JSON 串

JSON 串技术是一种轻量级的数据交换格式, 它是基于JavaScript的一个子集, 采用完全独立与语言的文本格式, 兼容包括 C、 C++、 JAVA、 JavaScript、 Python 等语言, 并且其易于阅读和编写, 同时也易于机器解析和生产, 这些特征使得 JSON 串成为理想的数据交换语言. 以某子系统实现的数据传输过程为例, 说明 JSON 串的构建和解析过程.

(1) 定义数据对应的结构体格式:

```
typedef struct tagDoorRemoteOpenBuf
{
    int len;
    unsigned char type;
    int sn;
    unsigned char door_no;
    char door_tagname[64];
    char belong_controlname[64];
    }DoorRemoteOpenBuf;
```

(2) 调用 JSON 串相关库文件并定义和实现类:

```
CJsonCppUtil:
Class CJsonCppUtil
{
    Public:
    CJsonCppUtil(void);
    CJsonCppUtil(Json::Value value);
    ~CJsonCppUtil(void);
        bool
        getDoorRemoteOpenBufStructFromString(strin
        g str,DoorRemoteOpenBuf &remoteOpen); //
        解析 JSON 串函数
    string
    parseDoorRemoteOpenBufStruct(DoorRemoteOpen
    Buf &remoteOpen);//构建 JSON 串函数
    private:
    Json::Value m_value;}
```

其中解析 JSON 串函数具体实现过程如下:

```
bool
CJsonCppUtil::getDoorRemoteOpenBufStructFromStrin
g(string str,DoorRemoteOpenBuf &remoteOpen)
{
    Json::Reader reader;
    if (!reader.parse(str, m_value, false))
    return false;
    remoteOpen.len = m_value["len"].asInt();
    remoteOpen.door_no = (unsigned
    char)m_value["door_no"].asInt();
    remoteOpen.sn = m_value["sn"].asInt();
    remoteOpen.type = (unsigned
    char)m_value["type"].asInt();
    sprintf(remoteOpen.door_tagname, "%s",
    m_value["door_tagname"].asString().c_str());
    sprintf(remoteOpen.belong_controlname,"%s",
    m_value["belong_controlname"].asString().c_str());
    sprintf(remoteOpen.fes_tagname, "%s",
    m_value["fes_tagname"].asString().c_str());
    return true;}
```

构建 JSON 串函数具体实现过程如下:

```
string
CJsonCppUtil::parseDoorRemoteOpenBufStruct(DoorRe
moteOpenBuf &remoteOpen)
{
    Json::FastWriter writer;
    Json::Value obj;
    obj["len"] = remoteOpen.len;
```


© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
obj["door_no"]= remoteOpen.door_no;
obj["sn"] = remoteOpen.sn;
obj["type"] = (Json::Int)remoteOpen.type;
return writer.write(obj);}
```

该类实现了 DoorRemoteOpenBuf 结构体的 JSON 串构建和解析功能, 在传递消息过程中, 首先相关应用程序按照该结构体格式通过构建 JSON 串方法生产消息总线所需发布的字符串 buf 的内容, 随即调用相关消息总线接口, 消息发布者按照既定格式将消息发出, 而后, 消息订阅者接收到该消息, 同理调用解析 JSON 串函数获得相关结构体数据, 实现了消息的传递. 通过 JSON 串封装的数据清晰明了, 方便开发人员协调工作, 有效的提高了开发效率, 避免了数据传输过程中构建和解析数据包错误导致的 bug 查找调试困难问题.

3.3 总线接口

在 ICE 中间件的基础之上封装的消息总线, 提供一对多的消息广播功能, 实现在复杂网络环境下数据的可靠传送. 消息总线保证同一个程序向同一个主题发布的消息, 接收者接收顺序与发送顺序相同. 消息总线程序由两部分组成: 向应用程序提供调用接口的软件模块, 通过动态库方式实现; 完成实际数据传输、接收的软件模块, 通过单独进程完成. 消息总线结构如图 3 所示.

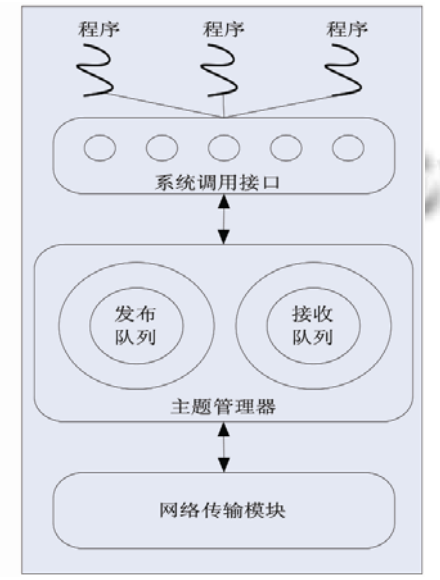

图 3 消息总线结构

应用程序使用消息总线的方式包括:

(1) 订阅主题: 应用程序通知消息总线自己关心的主题, 当此主题有消息到达时, 应用程序可以收取.

(2) 退订主题: 应用程序通知消息总线自己不再关心某个主题, 以后此主题的消息对应用程序不可见.

(3) 发布消息: 应用程序向某个主题发送一条消息, 所有订阅此主题的应用程序, 无论位于何处, 都可以收到此消息.

(4) 接收消息: 应用程序收取自己关心主题的消息.

(5) 主题管理模块: 用于管理本节点及域内其他节点主题订阅信息, 这些信息通过节点间交换的配置信息进行全域同步, 确保每条消息可以准确送至所需的节点.

以 ICE 中间件和 JSON 串为设计核心构建的消息总线, 消息传递思路清晰、层次分明, 以总线为联络通道, 设计与上层应用程序和前置开发模块的交互接口, 上层应用和前置开发按照规定接口通过消息总线实现信息的交互. 消息总线将前置模块和上层应用进行了有效的解耦合, 前置模块与上层应用信息交换时只需与消息总线进行交互, 通过这一方式屏蔽了前置多类设备与上层多应用个性化需求问题, 特定类型的消息通过消息总线由上层应用传递到前置开发, 前置根据消息类型将消息分发给对应模块, 从而实现消息的可靠传输.

消息总线采用 2 个环形队列管理本节点消息:发布队列与接收队列. 发送队列用于存放本节点应用程序发布的消息, 这些消息需要向其他节点进行传送; 接收队列用于存放从网络接收到的消息, 本节点应用程序需要接收这些消息. 如果本节点一个应用程序发布的消息有其他应用程序需要接收, 直接从发布队列复制到接收队列, 为方便开发人员使用, 本文将消息总线内封装, 定义了对外接口供使用者调用, 具体实现接口定义如下:

(1) 注册通道: int register_channel(unsigned int channelID);

(2) 取消注册通道: int unregister_channel(unsigned int channelID);

(3) 向指定通道发送消息: int send_to_channel (unsigned int channelID, const char *buf, unsigned int buf_size,unsigned int msg_type = 0, int dst_domainID = -1, bool send_to_myself = true);

(4) 从消息中线读取消息: int read_from_bus(char

© 中国科学院软件研究所 http://www.c-s-a.org.cn

*buf, unsigned int &buf_size, MSG_TYPE_INFO &stru_msgType);

4 应用案例

图 4 为某应用系统设计前置客户端界面, 现以具体实例对本文所论述系统加以说明, 现有编号为 1616 设备 2 个, 通过串口 1 接入本机软件系统, 其中次编号为 161601 设备为 1616 设备地址为 1 设备, 次编号为 161602 设备为 1616 设备地址为 2 设备. 本文背景为 161601 设备电源未开, 161602 设备正常工作, 从图中可以看出 161601 设备左边按钮显示为红色, 表示工况退出, 而 161602 设备左边按钮显示为绿色, 表示工况正常. 图示中右边部分为 161602 设备信息交互组帧报文情况, 该报文即为前置规约层中通讯协议内容. 通过这种前置模块设计, 本文实现了多类型设备的接入, 至此数据信息通过终端设备采集到了前置模块.

图 4 前置展示界面

信息到达前置缓冲区后, 前置管理层将该数据组帧丢给消息总线, 相应的上层模块通过注册消息总线实现数据的接收, 并按照用户业务需求进行相应处理.

图 5 为上层应用某变电站实际场景呈现, 将变电站现场辅助信息按照业主一次接线图定制要求, 将环境信息、视频监控、报警监视等集中展现.

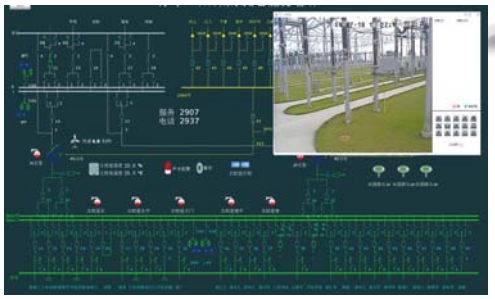

图 5 上层应用展示界面

5 结论

本文完成了基于 ICE 中间件和 JSON 串技术的消息总线传输设计, 为需要进行复杂通信的大型应用系统提供了层次清晰、传递可靠的消息传输解决方法. 该方法能够实现在应用层面系统开发、消息传输交互、前置设备响应这种三级构架的系统设计中发挥出重要作用, 并且应用层将消息结构体以 JSON 串格式下发和前置层, 按照对应格式进行解析保证了该消息的准确传输. 前置层采用链路层、规约层和管理层三层构架, 有利于各类前置终端设备的接入. 文章中所阐述的一体化平台监控设计思路在大型系统设计中优势明显.

参考文献

1 葛晖,王成进,杨建旭,等.基于 RT21 的智能综合监控管理系统.计算机系统应用,2014,23(12):77–81.
2 翟明玉,雷宝龙.电网调度自动化系统消息中间件的特性和关键技术.电力系统自动化,2012,36(14):56–59.
3 丁云亮,谷利泽,杨榆.基于分布式中间件 ICE 的应用架构研究.计算机应用,2009,29(12):27–31.
4 王重楠,王宗陶,鲍忠贵,等.发布/订阅模式测控消息中间件系统设计.计算机应用,2015,35(3):878–881.
5 莫峻,谭建成.智能变电站过程总线通信模型.中国电机工程学报,2014,7:1072–1078.
6 范菁,熊丽荣,徐聪.分布式企业服务总线平台数据集成研究及应用.计算机科学,2014,41(2):206–214.
7 王芳芳,廉东本,高天.企业服务总线的协议转换器的研究与设计.计算机系统应用,2013,22(3):132–135.

© 中国科学院软件研究所 http://www.c-s-a.org.cn