Research on Distributed Network Communication Based on ICE Middleware

Li Yonggang[1, a], Zhou Jinbiao[2, b], Guo Libing[3, c],Wang Yi[4, d], Yi Ruihai[5, e]

[1, 2, 3, 4, 5]China Satellite Maritime Tracking and Controlling Department

[a]lee@163.com, [b]jinbiao_zhou@163.com, [c]guoguo_leo@163.com,

[d]yi_wang@163.com,[e]ruihai_yi@163.com

Keywords: ICE Middleware; Distributed; Network Communication.

Abstract. In connection with features and requirements of distributed system, this paper presents a solution of network communication based on ICE distributed middleware. This solution not only achieves efficient network communication in heterogeneous environment, but also realizes functions in the distributed network such as central node server migration, redundancy backup, data publishing/subscription and network information security.

Introduction

Achieving network communication is a basic requirement of the distributed system, in present application, socket technology[1] based on LAN (TCP/UDP) is widely used. However, the above network communication technology has obvious drawbacks: The first is poor flexibility, in order to achieve network expansion or changes, there tend to be a large scale changes of software and hardware[2]; The second is that the data contracts are complex, requiring specific provision to bit data interface specification which increases software development, maintenance workload; The third is poor reliability, using of fixed-configuration, once the center node software and hardware failure can lead to paralysis of the entire network; The fourth is unable to support cross-platform information exchange, at present, the center computer system has adopted domestic operating system gradually, the next step will be the localization of the hardware system. Existing software system can not be cross-platform network communications, and this is a problem need to solve urgently[3].

In recent years, with rapid development of network middleware technology, there has been a series of network middleware products. Taken over the basic communication management by middleware, the developers and maintainers can be released from the underlying network communications to focus more on the implementation of high-level applications. ICE[4][5] (Internet Communications Engine) is generated to solve a distributed network communications. ICE provides an easy-to-use middleware platform, with better performance and scalability, and also provides a series of new technical features. At NASA software system architecture and large-scale distributed systems such as finance, online games and other fields, ICE has been successfully applied[6] [7].

ICE Middleware Technology

ICE is a network middleware platform developed by ZeroC company, its main design goals are:

(1)It provides object-oriented middleware platform for a heterogeneous environment;

(2)It provides a complete set of features which support distributed application development in wide range field;

(3)It avoids unnecessary complexity so that the platform is easier to learn and use;

(4)It provides efficient implementation in the aspects of broadband network, memory usage, CPU overhead, etc;

(5)It provides a built-in security implementation which is used to across insecure networks.

ICE makes innovations on many aspects of the middleware technology, and it provides the perfect solution for distributed systems which is suitable for all heterogeneous network environments: the client and the server can use different programming language and different network communication technologies, running on different operating systems and machines of different architectures. ICE provides a complete separation from the client to the service object, and the client does not need to understand the service object's implementation process and the specific location. ICE adopts soft-bus mechanism, in any case, whatever development language it uses, the software that meet the definition of interface specification can be integrated into the distribution environment. ICE uses the object model, and all applications can be seen as a collection of objects and related operations. Among distributed systems which are built on top of ICE, the access to application objects only depends on the network connectivity and the accuracy to get service object characteristics, and it has nothing to do with the position of the object or in which device environment the object is.

In order to build object-oriented distributed client / server applications, ICE provides a range of tools, API, and library. The users only need to concentrate on implementation of the business logic, without concern for specific network communications. Fig1 shows the application model. The ICE core function defines the basic mechanism that objects can send requests and receive responses transparently in heterogeneous environment, and it is the core component which build client / server relationship between objects. Provided by ICE core function, the loosely coupled communication mode between client-side and server-side allows developers to put more focus on the application logic implementations.

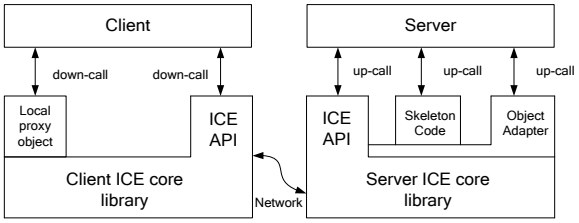

Fig.1 ICE Structure for Client / Server

ICE Network Communication Implementation

In order to achieve the hardware platform and software platform independent, based on ICE middleware platform, the assembly process of ICE components is as follows: First, the ICE specification language Slice is used to write public interface file, and the generated file is named "*.ice"; Independent of particular programming language, ICE defines data, interfaces, operations and other client / service contracts; And then ICE built-in compiler is used to convert the file into specific programming language API. Currently, ICE supports C, Java, C #, VB, Objective-C, Python, Ruby and other common development languages and tools.

The development process of ICE-based middleware platform is described as follows:

Components Definition. Generating ICE file, Slice language is used to define component's constants, data structures, interfaces, operations, etc. The generated file is named "*.ice". Examples are as follows (assuming the file is named "middleware.ice"):

```
module Middleware
{
interface JyDataProc
{
    void    InitData( );
    void    ProcessJYData(int T,short ZT,SequenceByte data);
    void    SetT0(int T);
}
}
```

Compiling ICE File. File “*.ice” is added to Microsoft Visual Studio 2010 integrated development environment, according to user’s choice of programming language, ice can automatically compile the ice file. If user selects the C language, then files “*.cpp” and “*.h” are automatically generated which named “middleware.h” and “middleware.cpp”.

Components Implementation. The component executors implement skeleton through expansion SLICE compiler and add business logic to achieve the ultimate components. An example is as follows:

```
class CHJProc:public JyDataProc
{
    void    InitData(const Ice::Current& c );
    void    ProcessJYData( int T,short ZT,const SequenceByte & data,const Ice::Current& c );
    void    SetT0(int T ,const Ice::Current& c );
}
```

Register Component on Server-side. The server-side implements the main program which calls the subclass of ICE application to achieve component’s registration and loading. First of all, an adapter object is created to run ICE, and then components are added in the adapter object. An example is as follows:

```
class HelloClient : public Ice::Application
{
public:
    HelloClient();
    virtual int run(int, char*[]);
};
int main(int argc, char* argv[])
{
    HelloClient app;
    return app.main(argc, argv, “config.Middleware”);
}
int HelloClient::run(int argc, char* argv[])
{
    if(argc > 1)
    {
        cerr << appName() << “: too many arguments” << endl;
        return EXIT_FAILURE;
    }
    JyDataProcPtr jyPrx=new CHJProc( );
    jyPrx->InitData( );
    Ice::ObjectAdapterPtr adapter = communicator()->createObjectAdapter(“JyDataProc”);
    adapter->add(jyPrx, communicator()->stringToIdentity(“JyDataProcAdapter”));
    adapter->activate();
    communicator()->waitForShutdown();
    return EXIT_SUCCESS;
}
```

Compiling the Project and Generating the EXE File. Set configuration file “config.Middleware” the parameter values of service interface endpoints and service object proxy, and the components can be separately deployed on any server. The interface endpoints include communications agreement and port number.

Call Components on Client-side. In client's main program, ICE is run to search the agent of the server component and communicate with it when succeeded. An example is as follows:

```
JyDataProcPrx jyPrx = JyDataProcPrx::
checkedCast(communicator()->propertyToProxy(“JyDataProc.Proxy”)->ice_twoway());
```

```
if(!jyPrx)
{
    cerr << argv[0] << ": invalid proxy" << endl;
    return EXIT_FAILURE;
}
JyDataProcPrx jyPrxOneWay=jyPrx->ice_oneway( );
```

Through the above steps, you can complete ICE-based network communication.

ICE Service Applications for Distributed Network

ICE not only achieves efficient network communication in heterogeneous environment, but also provides a range of services for the realization of server location, on-demand start, data publishing / subscription, information security and other features, providing a perfect platform for distributed applications.

Depending on the device characteristics and requirements, the network protocols are not the same, you can use the TCP protocol or UDP protocol, and you can also define their own communication protocol. ICE can shield these heterogeneous, making development, maintenance, configuration and usage easy and flexible.By deploying ICEGrid services, as long as the server is running on the network, other control devices can automatically place to the corresponding server, sending and receiving data. When a server fails, the locator can automatically give up access to the original server, thus repositioning server on the network.Through the deployment of ICEStorm service, the center computer system can subscribe center interface data from pooled and distributed servers, and pooled and distributed servers then publish the data to central computer system. This method greatly reducing the coupling between the device servers, and it improves the efficiency and real-time performance of network communications.

Summary

Compared to traditional network programming, based on the ICE middleware, the distributed network has object-oriented semantics, and it is easy to learn and achieve; Due to the independence of machine, language, implementation, operating systems, transport mechanism, it is applicable to different homogeneous environment; Due to the transparency of the location and server, it is able to achieve a server migration and backup, improving system reliability.

The distributed network architecture is feasible, and it can effectively solve the distributed deployment problem. It can adapt to changing network environments, different programming languages, different operating systems, and reduce the coupling between software modules, so that improve system scalability, reusability.

References

[1] Michi HenningMark Spruiell.Distributed Programming with Ice [EB /OL]. [2010-02-10] .http: //www.zeroc.corn/download/Ice/3.4/Ice-3.4.0.pdf.

[2] Guo Libing. Design on the Architecture of Rocket Telemetry Data Processing Based on Distributed Middleware ICE[J].Journal of Telemetry, Tracking and Command 2011,4 (32) :15-18.

[3] Yin Shenggang, Zhai Zhengjun, Distributed control network Based on Ice [J], Control Technology, 2011(10):35-37.

[4] Wang Ning, Wang Zheng, Research on Ice Plug-in technology [J], Computer Technology and Development, 2012(05):18-21.

[5] Zhang Min Jin, Research and Implementation of Distributed Real-time Monitoring and Control Network technology [D], Wuhan University, 2004.

[6] Guo Libing. Design of Spacecraft TT & C Software Based on the COM Component[J]. Journal of Spacecraft TT & C Technology, 2009,10 (28) :60-62.

[7]Paul C, Felix B, Len B, et al.Documenting Sofware Architectures.Beijing:TSINGHUA University Press, 2003.