TCE (tiny communication engine)

轻量级的RPC通信引擎

author: scott

url: github.com/adoggie/tce

版本

什么是TCE？

TCE功能特性

使用TCE

1. 安装

Gevent,qpid,ply

1. 运行
2. 代码集成

一个简单的接口定义场景： 学生、班级、学校、课程

Base.idl

module school {

struct Student{

string no;

string name;

short sex;

short age;

int grade;

string address;

};

sequence<Student> StudentList;

struct Teacher{

string name;

string subject;

};

struct Class{

string grade;

string name;

string teacher;

};

sequence<Class> ClassList;

struct Course{

string teacher;

string intro;

};

sequence<Course> CourseList;

}

School.idl

import base

module school {

dictionary<string,Course> CourseListOfTeacher;

interface IServerBase{

string getCurrentDateTime();

void heartbeat(string source);

};

interface ICourseServer extends IServerBase{

CourseListOfTeacher getCoursesOfTeacherInNextWeek(string nameOfTeacher);

};

interface IMainServer extends IServerBase{

StudentList getStudents(string className);

ClassList getClassList(string grade);

bool updateStudent( string no, Student studentinfo);

};

}

# 接口语法介绍(IDL)：

Idl文件结构:

TCE的接口文件采用.IDL作为文件后缀，可以采用任何文档编辑器进行编辑。

在idl文档内由若干的module组成，module内部定义多种数据类型和接口。

Module

IDL

Module

Module

Module

1. module

意思为模型，业务逻辑需求被归类到不同的模型部分。

python和java视为包(package) 定义

c++ 对应的是namespace

module是个容器，其包含多种数据类型和接口的定义

module可以被分开编写： 在同一个idl文件中，相同的名称的module可以定义多次，例如:

module sns{}

module sns{}

…

同一个moudle也可编写在不同的idl文件中。

1. Import:

数据和接口定义可以分隔在不同的idl文件被定义，当某个idl需要引入其他idl定义的数据对象时，需要使用 import关键字来导入其他接口文件。

1. 数据类型

谈到数据类型，从动态语言角度看，可可分为3大类，基础数据类型(primitive types) 、复合类型struct、容器container

Tce的idl基础数据类型包括: byte , short ,int , long ,float ,double, string,bool

数据容器包含: sequence<T>, dictionary<K,V> 分别对应数据和字典

Sequence<string > StringList 定义字符数组

Dictionary<string,int> StringIntList; 定义以string为key,int为value的字典类型

同样来点复杂的

Dictionary<int,StringIntList> IntStringIntList;

Sequence<StringList> StringListList;

复合类型 可以是任意数据类型的组合体，当然可以是不同的复合类型再次组合产生新的复合类型。

Struct Point{

Int x;

Int y;

};

Sequence<Point> PointList;

Struct Color{

Int red;

Int blue;

Int green;

Float opacity;

};

Struct Shape{

String name;

Color color;

Int type;

PointList points;

};

Sequence<Shape> ShapeList;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 不同语言映射 | | | | | | |
| Name | Size | C++ | Python | Java | Actionscript | Javascript |
| Byte | 1 | Uint8 | integer | Byte |  |  |
| Short | 2 | Short | integer | Short |  |  |
| Int | 4 | Int | integer | Int |  |  |
| Long | 8 | Long(64) | integer | Long |  |  |
| Float | 4 | Float | float | Float |  |  |
| Double | 8 | Double | Float | Double |  |  |
| String |  | Std::string | Str/utf-16 | String |  |  |
| bool | 1 | Bool | bool | Bool |  |  |
| sequence |  | Std::vector | [] | Vector |  |  |
| Dictionary |  | Std::map | {} | HashMap |  |  |
| struct |  | struct | class | class | class | Function |

1. 接口 interface

函数不允许重命名，为兼顾到不同语言，所以不能考虑多态

1. 接口派生 extends
2. 映射到编程语言

调用类型：

接口代理对象包含多种调用模型

1. Two-way

此调用方式最接近于模拟本地函数调用习惯，但是在网络调用时，接口调用请求和调用结果被封包并被传递交换。由于网络故障、系统问题等原因，调用可能导致失败。

函数原型: [result] function\_name ( parameter,..,timeout=0 )

Two-way调用时，调用者的执行线程将被挂起，直到函数返回，线程解锁继续运行。 timeout 参数为0时TCE引擎将采用默认的等待超时时间(默认：30秒);

用户可以指定timeout等待超时时间

1. One-way单向调用

此调用方式仅存在于 void函数定义,TCE 将自动生成后缀\_oneway()的代理调用函数。

例如：IServerBase::heartbeat，TCE将在IServerBaseProxy接口中增加 heartbeat\_oneway()函数。

\_oneway()的好处在于调用者无需等待返回，对于单向传递的需求非常有用。

1. Async 异步调用

函数原型: [void] function\_name\_async ( parameter,.., async )

代理对象中TCE自动生成后缀\_async的异步调用函数

异步模式下，async指定返回数据的接收函数, 调用函数返回类型为void

Python : async为函数对象，参数必须为idl定义的返回值

C++: async 为接收数据接口的派生子类指针

*Python调用：*

def getClassList\_Resut( ClassList result):

…

IMainServerPrx. getClassList( getClassList\_Result)

Void函数接口类型

void通常用于oneway方式

常见问题：