XSRV

一、 简介

XSRV 是一个基于 libev 和 jsonrpc 实现的服务器框架，主要提供短链接的 rpc 服务，客户端以 json 字符串提交请求，服务器端接收到请求后分析并执行相应的服务，让后返回数据。

libev 主要提供接入服务，libev 是高性能事件循环/事件模型的网络库，并且包含大量新特性。 它是继 lievent 和 Event perl module 之后的一套全新网络库。它追求的目标：速度更快，bug 更少，特性更多，体积更小。
jsonrpc 是以 json 为数据传输的 rpc 服务，这里借鉴了 cpp 的 jsonrpc 实现。

二、 目标

XSRV 的主要目标是提供一个简单可用的高性能以 json 为数据传输的 rpc 短链接服务器框架程序，特点可以分为一下下几点：

1. 使服务器程序开发简单化
2. 以 json 为数据客户端和服务器端传输格式
3. 主要提供 rpc 的短链接服务
4. 提供可配置的服务器，以提供灵活性
5. 集成 mysql 的数据库封装(sqlite)

三、 简单 demo：

server 端程序：

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "xtcpserver.h"

class TestRpc
{
public:
    TestRpc(json::rpc::Handler& handler)
    {
        handler.AddMethod(new json::rpc::RpcMethod<TestRpc>(*this,
```

```
                    &TestRpc::Print, "Print"));
    }

    bool Print(const json::Value& msg, json::Value& response, void *data)
    {
        response["jsonrpc"] = "2.0";
        response["id"] = msg["id"];
        response["result"] = "success";
        response["time"] = (char)time(NULL);

        return true;
    }
};
int main(int argc,char **argv){
    json::rpc::Handler handler;
    TestRpc rpc1(handler);

    xtcpserver server(std::string("127.0.0.1"), 8086);

    if(server.Start(handler)) {
        printf("init server\n");
    } else {
        printf("error");
    }

    return 1;
}
```

服务端程序抽象出了事务处理程序**json::rpc::Handler**，我们只需要将我们自己所写的方法**TestRpc**注册到这个**handler**中，再将这个**handler**传递给我们的服务器**server**，之后服务器接收到到的所有服务都会交给**handler**去处理。

以上这个服务，在服务器收到**{"id":10,"jsonrpc":"2.0","method":"Print"}**这样的请求后就会调用**TestRpc**中的**Print**方法，并且将这个**json**串以参数**msg**的形式传递给**Print**函数，返回值也是由**Print**函数填充到**response**中，客户端接收到的串也就是**response**这个**json**串。

client 端程序：

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <xtcpclient.h>

int main(int argc, char *argv[])
```

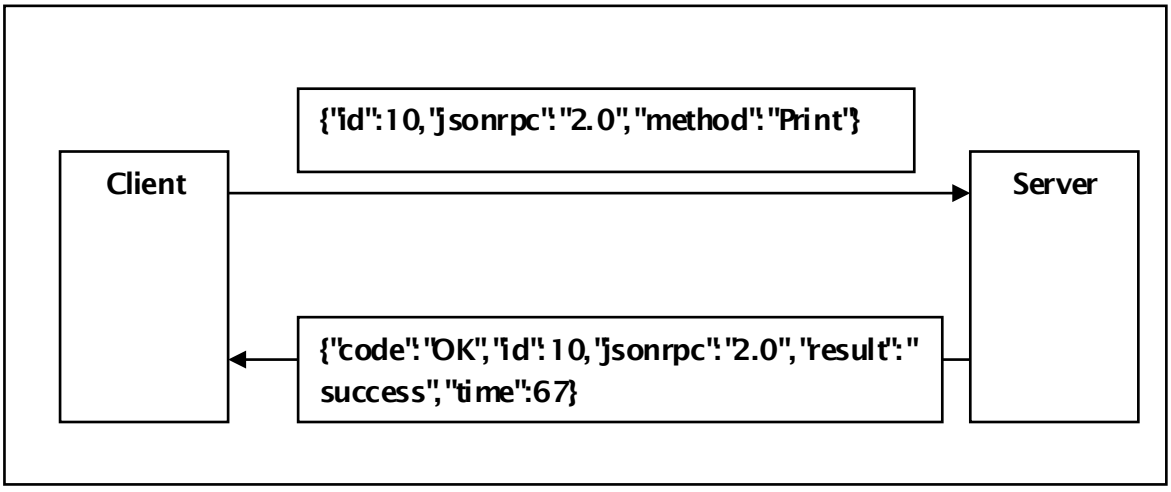
```
{
    int len;
    string recvdata;
    string strreq="{\"id\":10,\"jsonrpc\":\"2.0\",\"method\":\"Print\"}";

    xtcpclient client(std::string("127.0.0.1"), 8086);

    for(int i = 0; i < 10000; i++){
        client.connect();
        client.send(strreq);
        len = client.recv(recvdata);
        printf("%s\r\n", recvdata.c_str());
        client.close();
    }

    return 0;
}
```

客户端的请求串中必须要又 **id**，**jsonrpc** 和 **method** 三个关键字，
id 为任意整数，
jsonrpc 一般固定，都是 **2.0**
method 是我们要指定调用的处理函数，这个字段是和 **handler.AddMethod(new json::rpc::RpcMethod<TestRpc>(*this, &TestRpc::Print, "Print"));** 这里引号中的 **Print** 相关的。
client 和 **server** 之间数据交互图



在客户端的 **json** 数据处理上我们这边除了使用 **c++** 的 **jsoncpp** 外，还提供了较好的 **c** 的 **json** 数据处理库，这里 **c** 的 **json** 处理库使用 **json-c** 库。在文档末会专门介绍 **json-c** 的相关信息。

四、 服务器扩展

XSRV 服务器在原来jsonrpc的基础之上提供了服务器扩展接口，使的服务器在运行时候可以将服务器相关的数据传递到所请求的服务。本次修改是基于OD分离项目中需要获取所请求客户端和服务器相关信息时添加的。

接口定义在 include/xsrv/commnet.h 文件中，

以下是测试服务器中所用的数据接口，该结构和自己扩展添加，以进一步方便服务器请求调用和服务器之间的数据互访。

```
struct client {
        int fd;
        ev_io ev_write;
        ev_io ev_read;
        struct in_addr   ipaddr;          /*client ip*/
        char rbuff[1024];
        json::Value response;
        json::Value recv;
        json::rpc::Handler *handler;
}
```

在服务器调用程序是参数 void *data，如下：

```
bool Print(const json::Value& msg, json::Value& response, void *data)
{
    struct client *cli= (struct client*)data;
    。。。。。。
```

这里将 data 转换为相应的数据结构即可完成对服务器定义数据的相关访问。而该数据是在服务器端的 xsrv/commnet.cc Accep 等相关函数中进行相关数据的赋值和初始化。

五、 相关 API 说明：

这里主要指出服务器、客户端和公共组件中常用的 api 函数说明

服务器端：

主要有 xtcpserver 类、jsonrpchandle 类、日志类，配置文件解析类和网络链接相关方法

xtcpserver 类：主要是提供一个基于 libev 的服务器框架，包括服务器的网络初始化，daemon 化，信号处理等。

在实际使用中主要用到构造方法和 start 方法。

构造方法实现 ip 地址和端口号的初始化，

start 方法是实现网络初始化，daemon 话调用，信号初始化方法调用，开启监听，并且将 jsonrpc 的类传递给新来链接。

相关函数方法：(xsrv/xtcpserver.cc)

```
|-    function
||      sighandler
||      xtcpserver [xtcpserver]
||      ~xtcpserver [xtcpserver]
||      Init [xtcpserver]
||      DaemonInit [xtcpserver]
||      InitSigHandler [xtcpserver]
||      Start [xtcpserver]
||      GetSocket [xtcpserver]
||      GetAddress [xtcpserver]
||      GetPort [xtcpserver]
||      Close [xtcpserver]
```

jsonrpchandler 类主要是实现服务事件添加（**json::rpc::Handler.AddMethod**）,事件删除，和事件调用，一般主要使用 **AddMethod** 方法。

相关函数方法：**(jsonrpc/jsonrpc_handler.cc)**

```
|-    function
||      ~CallbackMethod [json::rpc::CallbackMethod]
||      Handler [json::rpc::Handler]
||      ~Handler [json::rpc::Handler]
||      AddMethod [json::rpc::Handler]
||      DeleteMethod [json::rpc::Handler]
||      SystemDescribe [json::rpc::Handler]
||      GetString [json::rpc::Handler]
||      Check [json::rpc::Handler]
||      Process [json::rpc::Handler]
||      Process [json::rpc::Handler]
||      Process [json::rpc::Handler]
||      Lookup [json::rpc::Handler]
```

日志类是用于服务器记录日志的的类，提供文本日志记录，**hex** 日志记录，还提供日志文件轮换

相关函数方法：**(xsrv/log.cc)**

```
|-    function
||      xsrv_log [xsrv_log]
||      ~xsrv_log [xsrv_log]
||      getlogfilename [xsrv_log]
||      getlogfilename [xsrv_log]
||      hexlog [xsrv_log]
||      log [xsrv_log]
||      shiftfiles [xsrv_log]
||      _InitLock [xsrv_log]
||      _Lock [xsrv_log]
||      _UnLock [xsrv_log]
||      _DestoryLock [xsrv_log]
```

配置文件解析类是用户段式配置文件的读取解析
相关函数方法：**(xsrv/parse_conf.cc)**

```
|-    function
||      CReadIni [CReadIni]
||      ~CReadIni [CReadIni]
||      LoadConfigFile [CReadIni]
||      AddToMap [CReadIni]
||      GetSection [CReadIni]
||      GetKey [CReadIni]
||      GetValue [CReadIni]
||      GetStr [CReadIni]
||      GetConfigStr [CReadIni]
||      GetConfigInt [CReadIni]
||      CharToLower [CReadIni]
||      ToLower [CReadIni]
```

客户端：

主要有客户端类组成 **xtcpclient** 类，这类是对常用的 **tcp** 客户端的一个链接、接收、发送和关闭的封装，避免客户端程序中重复写代码。
相关函数方法：（**xsrv/xtcpclient.cc**）

```
|-    function
||      xtcpclient [xtcpclient]
||      ~xtcpclient [xtcpclient]
||      connect [xtcpclient]
||      close [xtcpclient]
||      recv [xtcpclient]
||      send [xtcpclient]
```

公共组件：

有数据库的封装类，**json** 数据结构的相关类
数据库的封装类：（**comm/dbconn.cc** ）提供 **mysql** 数据的链接 **sql** 语句执行，结果处理等

```
|-    function
||      connect [dbconn]
||      Select [dbconn]
||      getRecordNum [dbconn]
||      output [dbconn]
```

```
||      FreeResult [dbconn]
||      StoreResult [dbconn]
||      FetchRow [dbconn]
||      Insert [dbconn]
||      Update [dbconn]
||      Delete [dbconn]
||      GetFieldsLengthArray [dbconn]
||      DataSeek [dbconn]
||      GetCursor [dbconn]
||      GetFieldsCount [dbconn]
||      GetFieldsName [dbconn]
||      GetAffectedRows [dbconn]
||      getLimitNum [dbconn]
||      GetResultRows [dbconn]
||      close [dbconn]
```

六、 jsoncpp 接口使用

参考网址：http://jsoncpp.sourceforge.net/index.html

在 jsoncpp 中主要使用的接口有 3 种：json_reader，json_value，json_writer

json_reader 主要是进行 json 字串解析为 json 数据格式，主要提供 parse 方法用于 json 字符串数据的解析。

```
Reader ()
Constructs a Reader allowing all features for parsing.
Reader (const Features &features)
Constructs a Reader allowing the specified feature set for parsing.
bool parse (const std::string &document, Value &root, bool collectComments=true)
Read a Value from a JSON document.
bool parse (const char *beginDoc, const char *endDoc, Value &root, bool collectComments=true)
Read a Value from a JSON document.
bool parse (std::istream &is, Value &root, bool collectComments=true)
Parse from input stream.
```

json_value 主要是 json 数据格式的定义，json 数据的组织，json 字段的添加删除等等，主要提供数据赋值，数据提取，数据字段类型判断，比如如下方法：

```
ValueType type () const
bool operator< (const Value &other) const
bool operator<= (const Value &other) const
bool operator>= (const Value &other) const
bool operator> (const Value &other) const
bool operator== (const Value &other) const
bool operator!= (const Value &other) const
```

```
int   compare (const Value &other) const
const char *      asCString () const
std::string   asString () const
Int   asInt () const
UInt      asUInt () const
Int64     asInt64 () const
UInt64    asUInt64 () const
LargestInt    asLargestInt () const
LargestUInt       asLargestUInt () const
float     asFloat () const
double    asDouble () const
bool      asBool () const
bool      isNull () const
bool      isBool () const
bool      isInt () const
bool      isUInt () const
bool      isIntegral () const
bool      isDouble () const
bool      isNumeric () const
bool      isString () const
bool      isArray () const
bool      isObject () const
bool      isConvertibleTo (ValueType other) const
```

json_writer 主要是将json数据格式的数据转换成字符串流等，将json数据格式输出为字符串格式：

```
virtual   ~Writer ()
virtual std::string     write (const Value &root)=0
```

七、 libev 参考资料

参考资料

1. libev 的文档 http://pod.tst.eu/http://cvs.schmorp.de/libev/ev.pod
2. http://software.schmorp.de/pkg/libev.html

八、 jsonrpc 参考资料

参考网址：http://sourceforge.net/projects/jsonrpc-cpp/

九、 json-c 参考资料

http://oss.metaparadigm.com/json-c/
https://github.com/jehiah/json-c
http://groups.google.com/group/json-c