4-1 加餐 - 手写序列化与反序列化

完整的代码链接：https://gitee.com/whb-helloworld/linux-plus-meal/tree/master/serialize-and-deserialize

手动实现,这里主要是体会具体序列化和反序列化的过程

- 本质：就是对字符串的处理
- 实际情况肯定会更复杂，但是序列化与反序列化已经有很多的现成解决方案了。我们有一次手写的经历就够了。

```C++
#pragma once

#include <iostream>
#include <memory>
#include <jsoncpp/json/json.h>

// #define SelfDefine 1

namespace Protocol
{
    // 问题
    // 1. 结构化数据的序列和反序列化
    // 2. 还要解决用户区分报文边界 --- 数据包粘报问题

    // 讲法
    // 1. 自定义协议
    // 2. 成熟方案序列和反序列化

    // 总结：
    // 我们今天定义了几组协议呢？？我们可以同时存在多个协议吗？？？可以
    // "protocol_code\r\nlen\r\nx op y\r\n" : \r\n 不属于报文的一部分，约定

    const std::string ProtSep = " ";
    const std::string LineBreakSep = "\r\n";

    // "len\r\nx op y\r\n" : \r\n 不属于报文的一部分，约定
    std::string Encode(const std::string &message)
```

```
    {
        std::string len = std::to_string(message.size());
        std::string package = len + LineBreakSep + message +
LineBreakSep;
        return package;
    }
    // "len\r\nx op y\r\n" : \r\n 不属于报文的一部分，约定
    // 我无法保证 package 就是一个独立的完整的报文
    // "l
    // "len
    // "len\r\n
    // "len\r\nx
    // "len\r\nx op
    // "len\r\nx op y
    // "len\r\nx op y\r\n"
    // "len\r\nx op y\r\n""len
    // "len\r\nx op y\r\n""len\n
    // "len\r\nx op
    // "len\r\nx op y\r\n""len\nx op y\r\n"
    // "len\r\nresult code\r\n""len\nresult code\r\n"

    bool Decode(std::string &package, std::string *message)
    {
        // 除了解包，我还想判断报文的完整性，能否正确处理具有"边界"的报
文
        auto pos = package.find(LineBreakSep);
        if (pos == std::string::npos)
            return false;
        std::string lens = package.substr(0, pos);
        int messagelen = std::stoi(lens);
        int total = lens.size() + messagelen + 2 *
LineBreakSep.size();
        if (package.size() < total)
            return false;
        // 至少 package 内部一定有一个完整的报文了！
        *message = package.substr(pos + LineBreakSep.size(),
messagelen);
        package.erase(0, total);
        return true;
    }

    class Request
    {
    public:
```

```
        Request() : _data_x(0), _data_y(0), _oper(0)
        {
        }
        Request(int x, int y, char op) : _data_x(x), _data_y(y),
_oper(op)
        {
        }
        void Debug()
        {
            std::cout << "_data_x: " << _data_x << std::endl;
            std::cout << "_data_y: " << _data_y << std::endl;
            std::cout << "_oper: " << _oper << std::endl;
        }
        void Inc()
        {
            _data_x++;
            _data_y++;
        }
        // 结构化数据->字符串
        bool Serialize(std::string *out)
        {
#ifdef SelfDefine // 条件编译
            *out = std::to_string(_data_x) + ProtSep + _oper +
ProtSep + std::to_string(_data_y);
            return true;
#else
            Json::Value root;
            root["datax"] = _data_x;
            root["datay"] = _data_y;
            root["oper"] = _oper;
            Json::FastWriter writer;
            *out = writer.write(root);
            return true;
#endif
        }
        bool Deserialize(std::string &in) // "x op y" []
        {
#ifdef SelfDefine
            auto left = in.find(ProtSep);
            if (left == std::string::npos)
                return false;
            auto right = in.rfind(ProtSep);
            if (right == std::string::npos)
                return false;
```

```
            _data_x = std::stoi(in.substr(0, left));
            _data_y = std::stoi(in.substr(right +
ProtSep.size()));
            std::string oper = in.substr(left + ProtSep.size(),
right - (left + ProtSep.size()));
            if (oper.size() != 1)
                return false;
            _oper = oper[0];
            return true;
#else
            Json::Value root;
            Json::Reader reader;
            bool res = reader.parse(in, root);
            if(res)
            {
                _data_x = root["datax"].asInt();
                _data_y = root["datay"].asInt();
                _oper = root["oper"].asInt();
            }
            return res;
#endif
        }
        int GetX() { return _data_x; }
        int GetY() { return _data_y; }
        char GetOper() { return _oper; }

    private:
        // _data_x _oper _data_y
        // 报文的自描述字段
        // "len\r\nx op y\r\n" : \r\n 不属于报文的一部分，约定
        // 很多工作都是在做字符串处理!
        int _data_x; // 第一个参数
        int _data_y; // 第二个参数
        char _oper;  // + - * / %
    };

    class Response
    {
    public:
        Response() : _result(0), _code(0)
        {
        }
        Response(int result, int code) : _result(result),
```

```
_code(code)
        {
        }
        bool Serialize(std::string *out)
        {
#ifdef SelfDefine

            *out = std::to_string(_result) + ProtSep +
std::to_string(_code);
            return true;
#else
            Json::Value root;
            root["result"] = _result;
            root["code"] = _code;
            Json::FastWriter writer;
            *out = writer.write(root);
            return true;
#endif
        }
        bool Deserialize(std::string &in) // "_result _code" []
        {
#ifdef SelfDefine
            auto pos = in.find(ProtSep);
            if (pos == std::string::npos)
                return false;
            _result = std::stoi(in.substr(0, pos));
            _code = std::stoi(in.substr(pos + ProtSep.size()));
            return true;
#else
            Json::Value root;
            Json::Reader reader;
            bool res = reader.parse(in, root);
            if(res)
            {
                _result = root["result"].asInt();
                _code = root["code"].asInt();
            }
            return res;
#endif
        }
        void SetResult(int res) { _result = res; }
        void SetCode(int code) { _code = code; }
        int GetResult() { return _result; }
        int GetCode() { return _code; }
```

```
    private:
        // "len\r\n_result _code\r\n"
        int _result; // 运算结果
        int _code;   // 运算状态
    };

    // 简单的工厂模式，建造类设计模式
    class Factory
    {
    public:
        std::shared_ptr<Request> BuildRequest()
        {
            std::shared_ptr<Request> req =
std::make_shared<Request>();
            return req;
        }
        std::shared_ptr<Request> BuildRequest(int x, int y, char
op)
        {
            std::shared_ptr<Request> req =
std::make_shared<Request>(x, y, op);
            return req;
        }
        std::shared_ptr<Response> BuildResponse()
        {
            std::shared_ptr<Response> resp =
std::make_shared<Response>();
            return resp;
        }
        std::shared_ptr<Response> BuildResponse(int result, int
code)
        {
            std::shared_ptr<Response> req =
std::make_shared<Response>(result, code);
            return req;
        }
    };
}
```