MYCP 开发指南
Guide for Development

Version: 1.0.7 Build: 480

作者：Akee Yang akee.yang@gmail.com

目录

前言

MYCP 是用于开发、集成、部署和管理大型分布式应用、网络应用和数据库应用的 C++web 应用服务器。将 C++的高效稳定和各种通讯标准 (HTTP/UDP/TCP/RTP 等)引入大型网络应用的开发、集成、部署和管理之中。

MYCP 拥有处理关键分布式应用系统问题所需的性能、可扩展性和高可用性。

MYCP 所需的多种特色和优势，包括：

- **领先的标准**

对业内多种通讯标准的全面支持，包括 HTTP、TCP、UDP 和 RTP 等，使分布式应用系统的实施更为简单，并且保护了投资，同时也使基于标准的解决方案的开发更加简便。

- **无限可扩展性**

MYCP 以其高扩展的架构体系闻名，包括内置 SOTP 通讯协议、访问用户管理和后台多种群集功能。

- **快速开发**

MYCP 独特的设计体系模式，可简化开发，加速投放市场速度。并可利用已有技能，迅速部署应用系统。

- **部署更趋灵活**

服务端支持“一次开发，到处运行”，支持跨平台部署，如 Windows 和 Linux 系统。

MYCP 支持所有标准浏览器客户端，另外还支持 C/S 结构应用程序客户端。

- **体系结构**

MYCP 是专门为企业分布式 web 应用服务系统开发。企业分布式网络应用系统需要快速开发，并要求服务器端组件具有良好的灵活性和安全性，同时还要支持关键任务所必需的扩展、性能、和高可用性。

MYCP 简化了可移植及可扩展的应用系统的开发，并为其它应用系统提供了丰富的互操作性。

MYCP 拥有最高水平的可扩展性和可用性。轻松实现访问验证功能、负载平衡和群集功能，而且不需要任何专门的硬件或操作系统支持。

第1章 获得 MYCP

1.1 下载 MYCP

在下面链接可以下载到最新版本的 MYCP；

http://code.google.com/p/mycp/downloads/list

http://sourceforge.net/projects/mycp/

截至本书发布时，最新版本是 1.0.7 版本，使用下列命令解压；

```
# tar xzvf mycp_1_0_7.tar.gz
```

自动解压到 mycp 目录。

SVN：http://code.google.com/p/mycp/source/checkout

Wiki：http://code.google.com/p/mycp/w/list

1.2 MYCP 发布说明

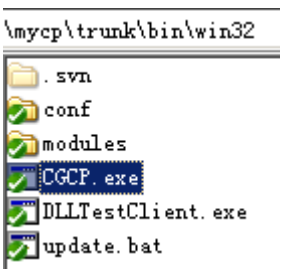
下载的 MYCP 开发包，不单包含有 MYCP 所有源码，还包含一个可部署运行环境，请看$(MYCP_ROOT)/bin 目录，包括有 linux 和 win32 二个部署目录；

可以复制整个部署运行环境，然后将开发的组件部署到该目录下，详细内容可看“MYCP 系统管理”；

1.3 测试 MYCP 服务

1.3.1 启动 MYCP

win32 环境，双击执行 CGCP.exe 文件：

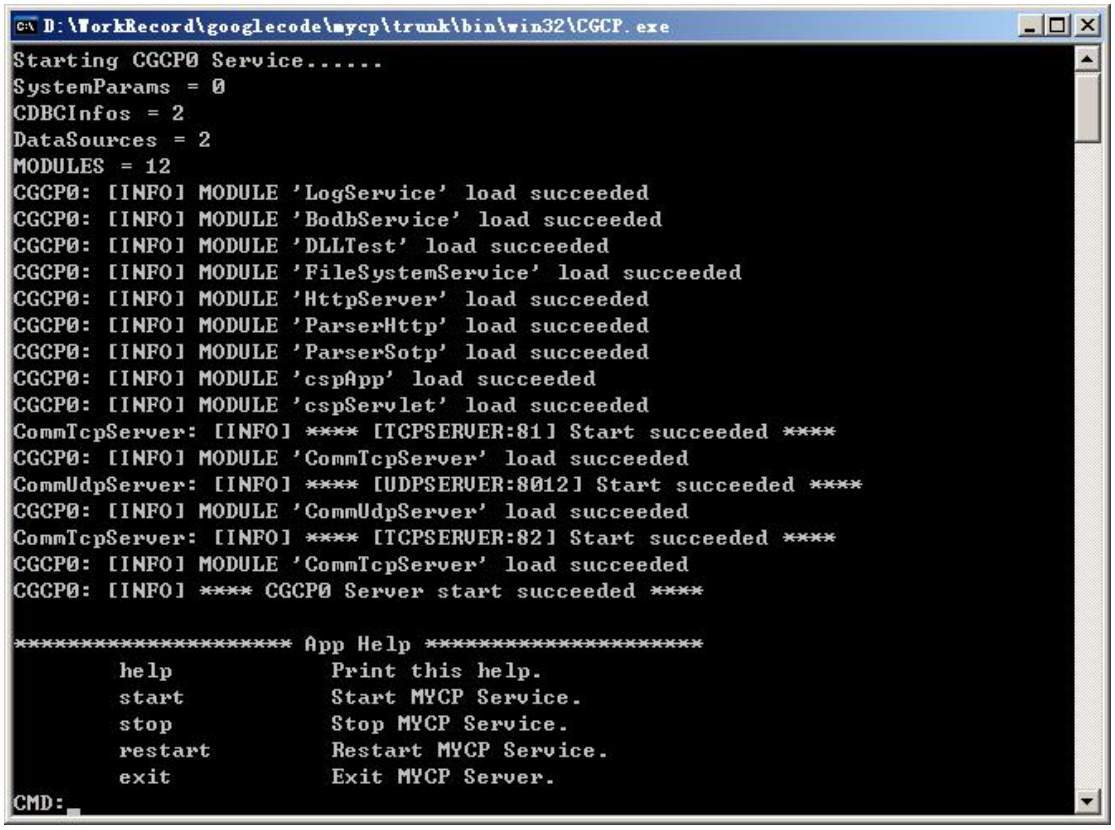


linux 环境，直接执行 CGCP 程序：


```
# sudo cp lib/* /usr/lib
# ./CGCP
```

Linux 环境下有些第三方库需要安装，请执行上面命令。

启动 MYCP；可以看到下图显示信息启动成功：



1.3.2 远程调用 RCA 组件

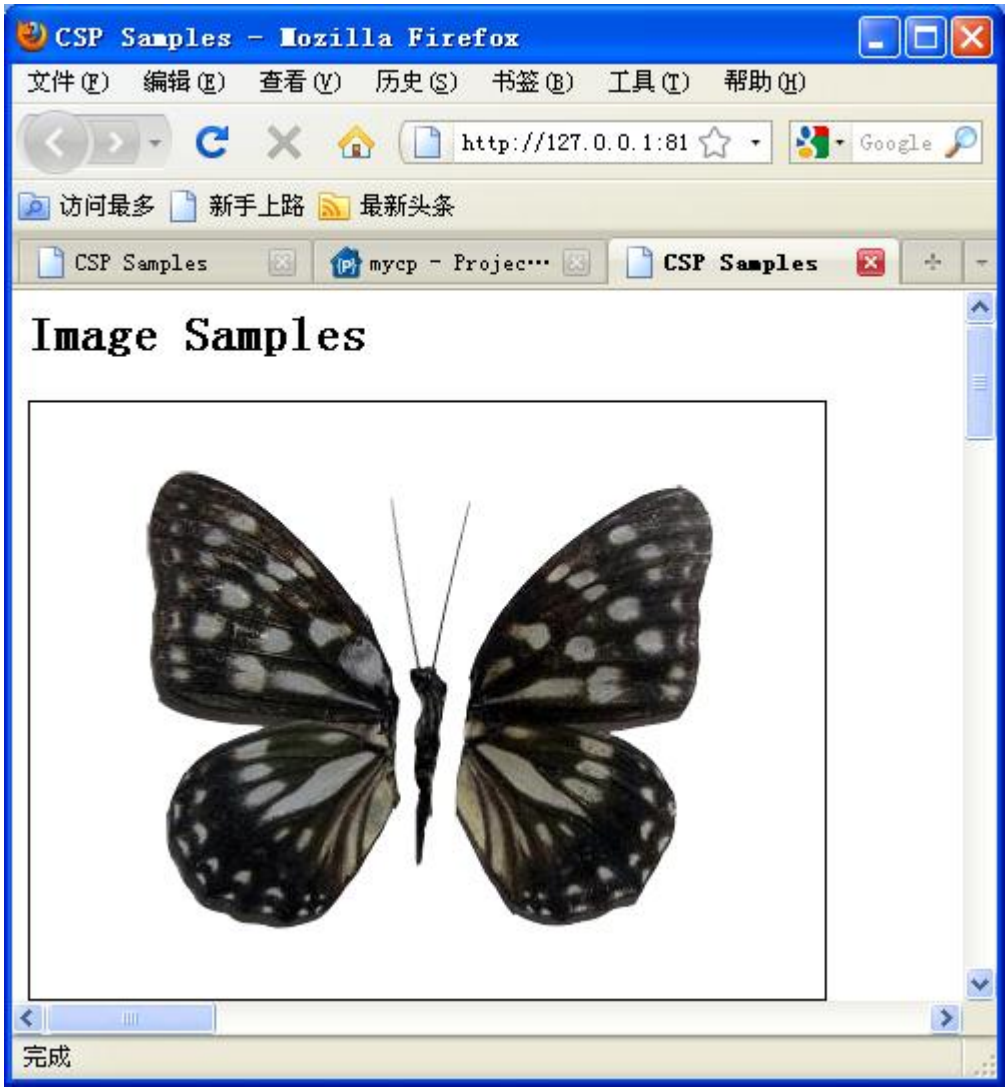
DLLTestClient 是一个远程调用 RCA 组件的客户端例子程序，具远程调用、输入参数，打印返回信息等功能。

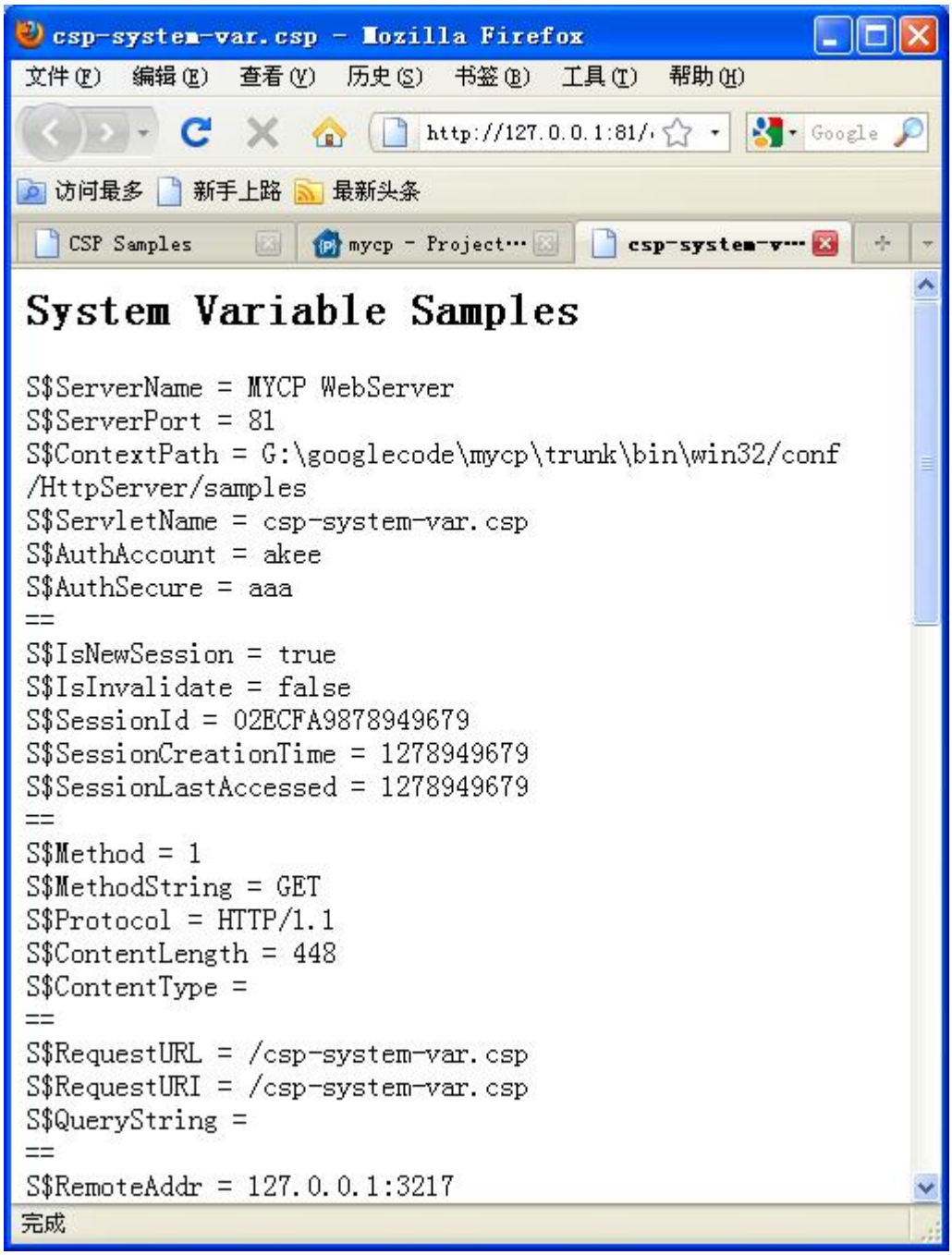
执行 DLLTestClient 程序，根据相关提示进行输入，显示下列信息为成功调用：

1.3.3 浏览器访问 CSP 页面

打开标准浏览器，输入访问 http://127.0.0.1:81，访问CSP页面：


CSP Samples - Mozilla Firefox
文件(F) 编辑(E) 查看(V) 历史(S) 书签(B) 工具(T) 帮助(H)
http://127.0.0.1:81
Google
访问最多 新手上路 最新头条
CSP Samples
mycp - Project Hosting on ...
Welcome to CSP!!!
MYCP Home Page CSP 1.0
CSP Samples
Base Samples Sample File
CSP output samples. csp-output.csp
Define variable samples. csp-define.csp
vector Variable type Samples csp-vector.csp
Get HTTP header parameter samples. csp-header.csp
Get request parameter samples. csp-param.csp
CSP System Variants. csp-system-var.csp
Logic Samples Sample File
operator Samples(+ - * / ...) csp-operator.csp
if Compare Samples csp-if.csp
while Samples csp-while.csp
if & while Samples csp-if-while.csp
foreach Samples csp-foreach.csp
foreach & if Samples csp-foreach-if.csp
continue Samples csp-continue.csp
break Samples csp-break.csp
Page Samples Sample File
page forward Samples csp-forward.csp
完成


CSP Samples - Mozilla Firefox
文件(F) 编辑(E) 查看(V) 历史(S) 书签(B) 工具(T) 帮助(H)
http://127.0.0.1:81
Google
访问最多 新手上路 最新头条
CSP Samples
mycp - Projec…
CSP Samples
Image Samples
完成



1.4 下个发布版本计划功能

下面列出当前版本未包含，但计划在未来发布版本会提供的功能：

- CDBC for ODBC
- CMS：C++ Message Serveice
- C++ Mail
- MVC 开发框架
- 其他功能及更新

第2章 开发入门

2.1 开发概述

MYCP除了支持标准浏览器访问外,还支持C/S结构应用程序客户端进行访问;利用 CSP 或者 C++ Servlet 开发的 web 应用将使用标准浏览器进行访问，不需要单独开发客客户端。

2.1.1 客户端运行环境

目前 MYCP 支持 HTTP/TCP/UDP/RTP 通信方式，客户端只要具备其中任何一种的通信能力，即可访问 MYCP 后台的功能服务组件;

另外，利用 MYCP 开放的通信接口，用户可以开发自己的通信协议能力，支持更多运行环境;

Windows、Linux、PDA 以及其他所有支持标准通信能力的各种软硬件平台都可以做为 MYCP 客户端，访问 MYCP 系统。

2.1.2 客户端开发工具（语言）

MYCP 内置标准 SOTP 通信协议，MYCP 专注于组件功能开发，提供标准的通讯能力（TCP/UDP/RTP 等）进行访问、交互，客户端只要按照 SOTP 协议格式进行封装传输，即可访问 MYSP。

所以，目前市场上所有的开发语言和工具都可以拿来开发 MYCP 客户端应用;

如 VC, VB, Delphi, CB, JAVA, Python，ASP, PHP，Perl.....

2.1.3 CGCLib 组件

随包提供 CGCLib 模块，封装实现了 SOTP 协议栈，利用 CGCLib 可以快速开发客户端。CGCLib 需要 Boost 和 jrtp(修改 src/CGCBase/cgcusers.h 文件禁用）库支持，同时支持 TCP、UDP、RTP 三种访问连接方式，具多线程，事件响应处理机制；支持跨平台使用。

用户可以参照 CGCLib 的实现，开发自己其他语言（如 JAVA）的 SOTP 协议栈库。

2.2 开发第一个 C++ Servlet

C++ Servlet 跟 CSP 一样，是 MYCP 的 web 层组件，用于输出动态 HTML 页面，实现 web 应用的界面显示。关于 CSP 的开发规范请看《CSP_1_1.pdf》文档，本节主要简单描述 Servlet 的开发流程。

2.2.1 cspServlet.cpp 文件

新建一个 cspServlet.cpp 文件，或者利用 VC 新建一个普通 DLL 类型工程项目。

2.2.2 添加头文件

```
#include <CGCBase/httpapp.h>
using namespace cgc;
```

2.2.3 实现标准 doGET 函数

```
extern "C" HTTP_STATUSCODE CGC_API doGET(const cgcHttpRequest::pointer & request,
cgcHttpResponse::pointer response)
{
    cgcSession::pointer session = request->getSession();
    response->println("<HTML>");
    response->println("<TITLE>MYCP Web Server</TITLE>");
    response->println("<h1>Get Sample</h1>");
    response->println("<h2>Session Info:</h2>");
    if (session->isNewSession())
    {
        response->println("This is a new session.");
    }else
    {
        response->println("This is not a new session.");
    }
    response->println("<h2>Headers:</h2>");
    std::vector<cgcKeyValue::pointer> headers;
    if (request->getHeaders(headers))
    {
```

```
        for (size_t i=0; i<headers.size(); i++)
        {
            cgcKeyValue::pointer keyValue = headers[i];
            response->println("%s: %s", keyValue->getKey().c_str(),
keyValue->getValue()->getStr().c_str());
        }
    }else
    {
        response->println("Not header info.");
    }
    response->println("<h2>Propertys:</h2>");
    std::vector<cgcKeyValue::pointer> parameters;
    if (request->getParameters(parameters))
    {
        for (size_t i=0; i<parameters.size(); i++)
        {
            cgcKeyValue::pointer keyValue = parameters[i];
            response->println("%s = %s", keyValue->getKey().c_str(),
keyValue->getValue()->getStr().c_str());
        }
    }else
    {
        response->println("Not property.");
    }
    response->println("</HTML>");
    return STATUS_CODE_200;
}
```

2.2.4 编译部署

编译 cspServlet 工程，生成 cspServlet.dll 或者 libcspServlet.so 文件，复制文件到$(MYCP_BINPATH)/modules 目录。

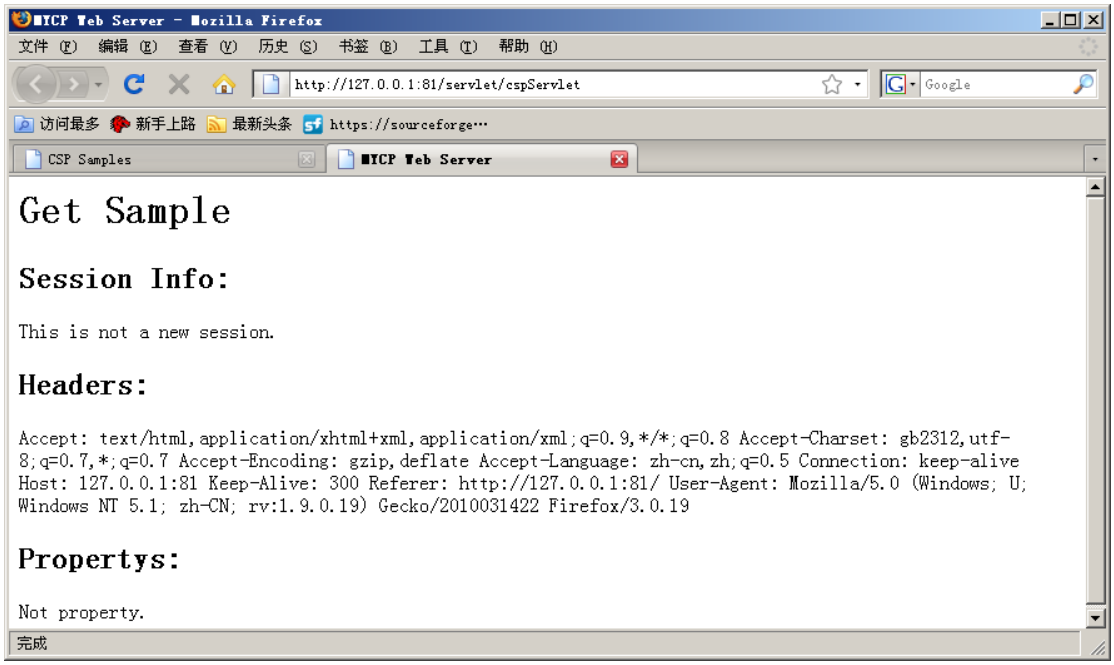
修改$(MYCP_BINPATH)/conf/modules.xml 文件，在 app 配置项增加一个组件配置项；

```
<app>
    …
    <module>
            <file>cspServlet</file>
            <allowall>1</allowall>
    </module>
    …
</app>
```

设置组件文件，允许开放所有函数。

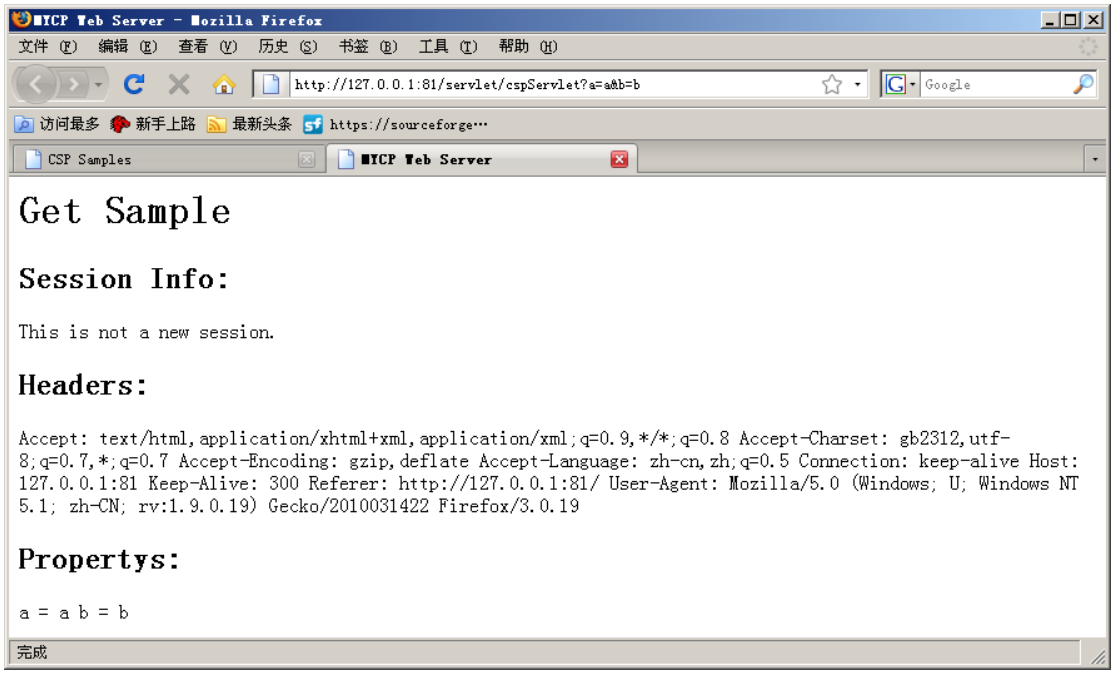
2.2.5 测试 cspServlet 组件

打开浏览器，输入 http://127.0.0.1:81/servlet/cspServlet，访问cspServlet组件的doGET函数，如下图：



http://127.0.0.0:81 是服务器域名或IP地址，后面带servlet是固定格式，用于访问所有的C++ Servlet组件，默认访问doGET函数。

下图演示带参数访问 cspServlet 组件的 doGET 函数：



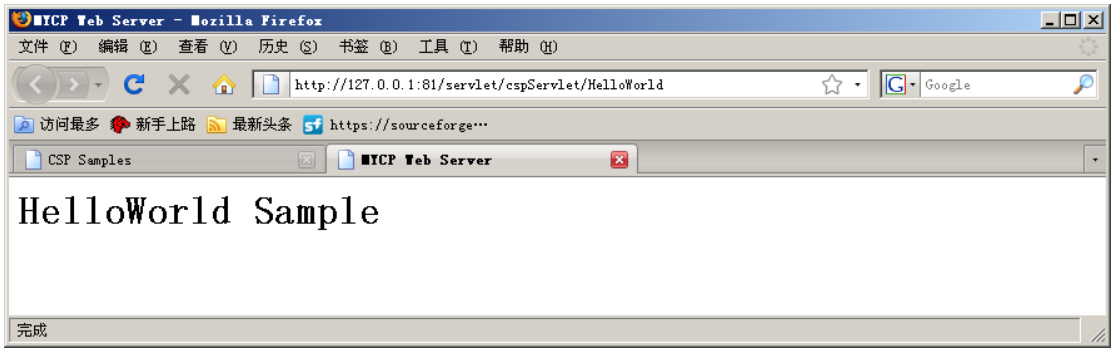
2.2.6 开发自定义函数

如下代码实现一下自定义函数 HelloWorld：

```
extern "C" HTTP_STATUSCODE CGC_API doHelloWorld(const cgcHttpRequest::pointer & request,
cgcHttpResponse::pointer response)
{
    response->println("<HTML>");
    response->println("<TITLE>MYCP Web Server</TITLE>");
    response->println("<h1>HelloWorld Sample</h1>");
    response->println("</HTML>");
    return STATUS_CODE_200;
}
```

2.2.7 测试自定义函数

打开浏览器，输入 http://127.0.0.1:81/servlet/cspServlet/HelloWorld，访问cspServlet组件的doHelloWorld函数，如下图：



2.2.8 支持 Servlet 函数

MYCP 除了支持以下标准 HTTP 函数，还支持自定义函数：

- doGET
- doPOST
- doPUT
- doDELETE
- doHEAD
- doOPTIONS
- doTRACE
- doCONNECT

2.2.9 总结

C++ Servlet 支持实现 CSP 相同功能，输出动态 HTML 页面，CSP 不用编译，而 Servlet 需要，所以 UI 层面层建议使用 CSP 做开发，而 Servlet 做为辅助开发，帮助实现业务处理文件的功能。

可以利用 Servlet 支持的其他标准函数，如 doPUT、doDELETE 等，配合输出 XML 数据，实现 REST 风格开发框架。

2.3 开发第一个 C++ APP

C++ APP 是 MYCP 业务层组件，用于处理业务逻辑。

2.3.1 cspApp.cpp 文件

新建一个 cspApp.cpp 文件，或者利用 VC 新建一个普通 DLL 类型工程项目。

2.3.2 添加头文件

```
#include <CGCBase/httpapp.h>
#include <CGCBase/cgcServices.h>
using namespace cgc;
```

2.3.3 实现应用服务接口

```
class CAppService
    : public cgcServiceInterface
{
public:
    typedef boost::shared_ptr<CAppService> pointer;
    static CAppService::pointer create(void)
    {
        return CAppService::pointer(new CAppService());
    }
    virtual tstring serviceName(void) const {return _T("AppService");}
protected:
    virtual bool callService(int function, const cgcValueInfo::pointer& inParam,
cgcValueInfo::pointer outParam)
    {
```

```
        theApplication->log(LOG_INFO, "AppService callService = %d\n", function);
        return true;
    }
    virtual bool callService(const tstring& function, const cgcValueInfo::pointer& inParam,
cgcValueInfo::pointer outParam)
    {
        theApplication->log(LOG_INFO, "AppService callService = %s\n", function.c_str());
        return true;
    }
    virtual cgcAttributes::pointer getAttributes(void) const {return theAppAttributes;}
};
```

定义 CappService 服务接口，继承于 cgcServiceInterface，实现 callService 函数，getAttributes 函数用于实现组件的参数管理，详细代码请看 samples/cspApp/cspApp.cpp 文件。

2.3.4 导出应用服务接口

```
extern "C" void CGC_API CGC_GetService(cgcServiceInterface::pointer & outService, const
cgcValueInfo::pointer& parameter)
{
    CAppService::pointer appService = CAppService::create();
    appService->initService();
    outService = appService;
    cgcAttributes::pointer attributes = theApplication->getAttributes();
    assert (attributes.get() != NULL);
    theAppAttributes->setAttribute(ATTRIBUTE_NAME, outService.get(), appService);
}

extern "C" void CGC_API CGC_ResetService(cgcServiceInterface::pointer inService)
{
    if (inService.get() == NULL) return;
    theAppAttributes->removeAttribute(ATTRIBUTE_NAME, inService.get());
    inService->finalService();
}
```

2.3.5 编译部署

编译 cspApp 工程，生成 cspApp.dll 或者 libcspApp.so 文件，复制文件到 $(MYCP_BINPATH)/modules 目录。

修改$(MYCP_BINPATH)/conf/modules.xml 文件，在 app 配置项增加一个组件

配置项；

```
<app>
    …
    <module>
            <file>cspApp</file>
            <allowall>1</allowall>
    </module>
    …
</app>
```

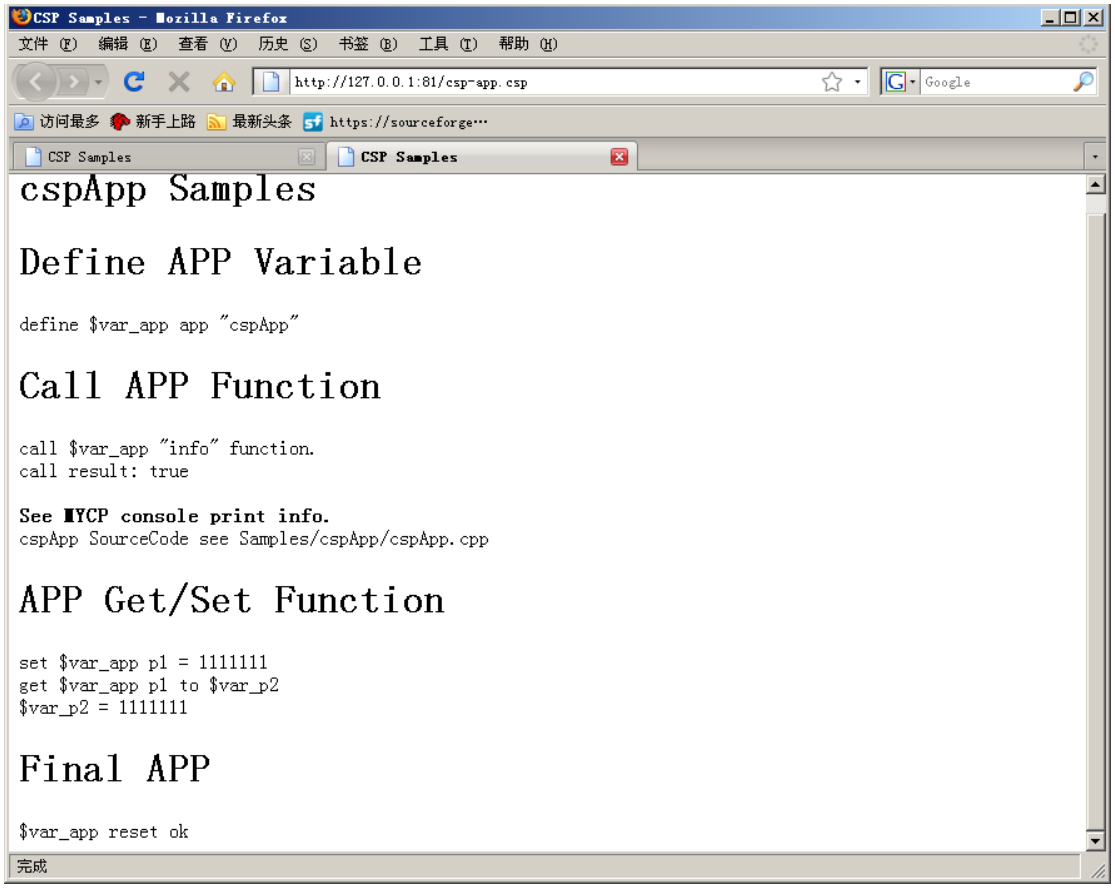
设置组件文件，允许开放所有函数。

2.3.6 CSP 调用 cspApp 组件

以下内容为调用 cspApp 组件的 CSP 代码，详细请看 bin/web/samples/csp-app.csp 文件：

```
    <csp:define type="app" id="$var_app" name="cspApp" />
<h1>Call APP Function</h1>
    call $var_app "info" function.<br>
    <csp:app:call id="$var_app" name="info" />
    call result: <%=_$result%><br>
    <br>
    <b>See MYCP console print info.</b><br>
    cspApp SourceCode see Samples/cspApp/cspApp.cpp<br>
<h1>APP Get/Set Function</h1>
    set $var_app p1 = 1111111<br>
    <csp:app:set id="$var_app" name="p1" in="1111111" />
    get $var_app p1 to $var_p2<br>
    <csp:app:get id="$var_app" name="p1" out="$var_p2" />
    $var_p2 = <%=$var_p2%><br>
<h1>Final APP</h1>
    <csp:reset id="$var_app" />
    $var_app reset ok<br>
```

浏览器访问如下图：



2.3.7 配置访问 C++ APP 组件

以上例子演示如何定义一个 C++ APP 组件，在页面退出的时候需要清空该组件变量；为了方便开发者使用 C++ APP 应用组件；MYCP 支持通过配置 apps.xml 文件，部署好所有的 C++ APP 应用组件，配置好的应用组件，直接在 CSP 页面用 A$变量直接使用，不需要定义，和清空操作。

配置例子请看 conf/HttpServer/apps.xml 文件；

使用例子请看 bin/web/samples/csp-app-fs.csp 文件等例子；

2.3.8 总结

本节学习如何开发一个 C++ APP 应用组件，包括定义实现服务接口、实现接口函数、接口参数管理和导出服务接口等。

2.4 开发第一个 RCA 应用组件

RCA（Remote C++ APP）是 MYCP 的远程 C++ APP 应用组件，在第一章测试 MYCP 服务那里，远程调用的就是 DLLTest 应用组件。

本节学习如何开发 DLLTest 应用组件；所有的 MYCP 应用组件开发模式和流程，完全一样，通过学习开发 DLLTest 应用组件，就基本掌握了全部应用组件的开发方法。

2.4.1 DLLTest.cpp 文件

新建一个 DLLTest.cpp 文件，或者利用 VC 新建一个普通 DLL 类型工程项目。

2.4.2 添加头文件

```
#include <CGCBase/app.h>
using namespace cgc;
```

2.4.3 添加 HelloUser 函数

```
extern "C" int CGC_API HelloUser(const cgcSotpRequest::pointer & request,
cgcSotpResponse::pointer response)
{
    // Get request input parameters.
    cgcParameter::pointer pUserName = CGC_REQ_PARAMETER(_T("UserName"));
    if (pUserName.get() == 0) return -1;

    // Set response output parameters.
    tstring sResponse(_T("Hello, "));
    sResponse.append(pUserName->getStr());
    sResponse.append(_T(", How are you!"));

    CGC_RES_LOCK();
    CGC_RES_PARAMETER(CGC_PARAMETER("Hi", sResponse));

    // Send response.
    return 1;
}
```

extern "C" int CGC_API HelloUser(...):

MYCP 函数为固定格式，修改函数名称（如 HelloUser）满足你自己的业务功能需求；

request 对象用于获取远程客户端输入参数；服务组件通过 respoinse 对象返回参数给远程客户端应用程序； session 对象用于管理远程客户端打开 DLLTest 组件的整个会话（SESSION）的状态。

2.4.4 编译部署

编译 DLLTest 工程，生成 DLLTest.dll 或者 libDLLTest.so 文件，复制文件到$(MYCP_BINPATH)/modules 目录。

修改$(MYCP_BINPATH)/conf/modules.xml 文件，在 app 配置项增加一个组件配置项；

```
<app>
    ...
      <module>
            <file>DLLTest</file>
            <allowall>1</allowall>
            <authaccount>1</authaccount>
            <lockstate>LS_WAIT</lockstate>
            <disable>0</disable>
      </module>
    ...
</app>
```

设置组件文件名，允许开发所有函数等等；

2.4.5 总结

RCA 组件同样以动态链接库文件（dll/so）存在，开发步骤、引用头文件、固定函数格式等基本一致；

通过本节学习，你知道如何开发并部署自己的 RCA 应用组件，可以尝试开发功能更加丰富的应用组件，MYCP 提供了丰富的 SERVICE（服务）组件，方便组件开发；

2.5 开发第一个客户端程序

本节学习如何开发 DLLTestClient 客户端应用程序，访问远程 RCA 应用组件。

2.5.1 DLLTestClient.cpp 文件

新建一个 DLLTestClient.cpp 文件，或者利用 VC 新建一个普通控制台应用程序工程项目：

2.5.2 添加头文件

```
#include <CGCLib/CGCLib.h>
using namespace cgc;
```

2.5.3 链接库文件

利用 CGCLib 开发客户端应用程序，需要 CGCClass 和 CGCLib 库的支持，如果是 win32 编译环境，在<CGCLib/CGCLib.h>文件里面，已经默认添加库的编译链接；

```
#ifdef WIN32
#ifdef _DEBUG
#pragma comment(lib, "CGCLibd.lib")
#else
#pragma comment(lib, "CGCLib.lib")
#endif // _DEBUG
#endif // WIN32
```

如果是 Linux 环境，请链接 libCGCClass.a 和 libCGCLib.a 库文件。

2.5.4 定义一个事件处理器

CGCLib 采用异步通信机制，需要定义事件处理器，用于接收所有 MYCP 事件，包括 SESSION 会话事件、调用函数（如 HelloUser）事件，等等，全部源码如下：

```
class MyCgcClientHandler
    : public CgcClientHandler
{
private:
    virtual void OnCgcResponse(const cgcParserSotp & response)
    {
        // Open DLLTest module session return.
        if (response.isResulted() && response.isOpenType())
        {
            std::cout << "SESSION: ";
            std::cout << response.getSid().c_str() << std::endl;
```

```
        }

        // Call HelloUser() function return.
        if (response.getSign() == const_CallId_HelloUser)
        {
            cgcParameter::pointer pHi = response.getRecvParameter(_T("Hi"));
            if (pHi.get() != NULL)
                std::cout << _T("[RETURN]:") << pHi->getStr().c_str() << std::endl;
            // response.getResultString() == response.getResultValue()
            std::cout << _T("[ResultCode]:") << response.getResultValue() << std::endl;
            return;
        }
    }
    virtual void OnCgcResponse(const unsigned char * recvData, size_t dataSize)
    {
        std::cout << _T("[OnCgcResponse]:") << recvData << std::endl;
    }
    virtual void OnCidTimeout(unsigned long callid, unsigned long sign, bool
canResendAgain){}
};
```

2.5.5 初始化

使用 CSotpClient 类初始化通信环境；

CSotpClient 是一个 SOTP 协议客户端封装类，使用 CSotpClient 访问远程 MYCP RCA 组件；连接 MYCP 服务，设置事件处理器等，源码如下：

```
CSotpClient gCgcClient;
DoSotpClientHandler::pointer gSotpClientHandler;

MyCgcClientHandler gMyCgcClientHandler;

void cgc_start(void)
{
    std::cout << _T("ADDR(ip:port):");
    CCgcAddress::SocketType st(CCgcAddress::ST_UDP);
    tstring sIp;
    std::getline(std::cin, sIp);
    if (sIp.empty())
    {
        sIp = _T("127.0.0.1:8012");
    }

    cgc_stop();
```

```
    gSotpClientHandler = gCgcClient.startClient(CCgcAddress(sIp, st));
    BOOST_ASSERT(gSotpClientHandler.get() != NULL);
    // Specifies an event handler.
    gSotpClientHandler->doSetResponseHandler(&gMyCgcClientHandler);
    std::cout << _T("APP Name:");
    tstring appname;
    std::getline(std::cin, appname);
    if (appname.empty())
        appname = _T("DLLTest");
    std::cout << _T("StartClient ") << sIp.c_str() << "..." << std::endl;
    // Specify the connection module name.
    gSotpClientHandler->doSetAppName(appname);
}
```

主要源码介绍：

```
CCgcAddress::SocketType st(CCgcAddress::ST_UDP);
```

设置 UDP 连接地址类型；

```
CSotpClient gCgcClient;
DoSotpClientHandler::pointer gSotpClientHandler;
gSotpClientHandler = gCgcClient.startClient(CCgcAddress(sIp, st));
```

连接 MYCP 系统，并返回 DoSotpClientHandler 对象，所有跟 MYCP 的交互都通过返回的 DoSotpClientHandler 对象进行操作。

```
gSotpClientHandler->doSetAppName(appname);
```

连设置要连接的 MYCP 组件名称。

2.5.6 调用 HelloUser 函数

调用 HelloUser 函数，源码如下：

```
void cgc_call_hellouser(void)
{
    BOOST_ASSERT(gSotpClientHandler.get() != NULL);
    std::cout << "UserName:";
    tstring userName;
    std::getline(std::cin, userName);
    if (userName.empty())
        userName = _T("Akeeyang");
    // Set UserName parameter.
    gSotpClientHandler->doAddParameter(CGC_PARAMETER("UserName", userName));
    // Call HelloUser() function.
    gSotpClientHandler->doSendAppCall(const_CallId_HelloUser, _T("HelloUser"));
}
```

主要源码介绍：

```
gSotpClientHandler->doAddParameter(CGC_PARAMETER("UserName", userName));
```

调用输入参数“UserName”的值；

```
gSotpClientHandler->doSendAppCall(const_CallId_HelloUser, _T("HelloUser"));
```

执行调用 HelloUser 函数；const_CallId_HelloUser 为事件标识，后台返回时带回该事件标识，用于标识调用某个函数返回，详细看处理返回事件源码。

2.5.7 处理返回事件

返回事件处理源码如下：

```
virtual void OnCgcResponse(const cgcParserSotp & response)
{
    // Open DLLTest module session return.
    if (response.isResulted() && response.isOpenType())
    {
        std::cout << "SESSION: ";
        std::cout << response.getSid().c_str() << std::endl;
    }
    // Call HelloUser() function return.
    if (response.getSign() == const_CallId_HelloUser)
    {
        cgcParameter::pointer pHi = response.getRecvParameter(_T("Hi"));
        if (pHi.get() != NULL)
            std::cout << _T("[RETURN]:") << pHi->getStr().c_str() << std::endl;
        // response.getResultString() == response.getResultValue()
        std::cout << _T("[ResultCode]:") << response.getResultValue() << std::endl;
        return;
    }
}
```

主要源码介绍：

```
if (response.isResulted() && response.isOpenType())
```

第一次执行 MYCP 后台组件，会默认打开一个 SESSION 连接，返回 SESSION 事件。

```
if (response.getSign() == const_CallId_HelloUser)
{
    cgcParameter::pointer pHi = response.getRecvParameter(_T("Hi"));
    if (pHi.get() != NULL)
        std::cout << _T("[RETURN]:") << pHi->getValue().c_str() << std::endl;
}
```

判断是否是调用 HelloUser 函数（const_CallId_HelloUser 事件标识）返回，

如果是取出 Hi 参数，并打印到系统屏幕上。

2.5.8 关闭 SESSION 连接

成功打开某个组件 SESSION 后，可以访问该组件的所有开放函数，在最后退出的时候，关闭 SESSION 会话，源码如下：

```
void cgc_stop(void)
{
    if (gSotpClientHandler.get() != NULL)
    {
        gCgcClient.stopClient(gSotpClientHandler);
        gSotpClientHandler.reset();
    }
}
```

2.5.9 总结

本节学习如何开发一个 MYCP 客户端应用程序，包括定义事件处理器，指定要连接的组件名称，调用组件函数，标识事件，处理返回事件等；

至此，你已经学会基本的 MYCP 开发流程，包括后台应用组件和客户端应用程序的开发，你可以利用目前学到的知识，开发大部分的通信服务；

通过后面的一些高级课程，你可以学习到更多高级的开发技艺，比如系统及组件管理、数据对象存储管理、配置管理、日志服务、多线程编程，等等。

第3章 MYCP 基础知识

在学习更多高级开发技巧之前，先来了解 MYCP 的基础知识。

3.1 基本概念

3.1.1 基本数据类型

MYCP 通过 cgcParameter 参数类，封装所有数据类型，下表列出 SOTP 协议、C++数据类型和 cgcParameter 类对应关系：

SOTP 协议数据类型	C++数据类型	cgcParameter 数据类型
int	int	cgcParameter::TYPE_INT
bigint	__int64/long long	cgcParameter::TYPE_BIGINT
time	std::time_t	cgcParameter::TYPE_TIME
string	std::string	cgcParameter::TYPE_STRING
boolean	bool	cgcParameter::TYPE_BOOLEAN
float	double	cgcParameter::TYPE_FLOAT

以下方法，用于新建一个字符串参数变量 pUserName，参数名称为“UserName”，参数的值为“H.D”：

```
cgcParameter::pointer pUserName = CGC_PARAMETER(“UserName”, “H.D”);
```

新建一个整数类型参数：

```
cgcParameter::pointer pUserAge = CGC_PARAMETER(“UserAge”, 30);
```

新建一个布尔类型参数：

```
cgcParameter::pointer pLogined = CGC_PARAMETER(“Logined”, true);
```

客户端和服务端组件通过 cgcParameter 类传递数据，利用参数名称、类型以及参数值实现各种业务需求。

3.1.2 二进制流数据

MYCP 支持传输二进制流数据，通过 cgcAttachment 类封装附件；实现文件流、音视频通信等传输。

3.1.3 MYCP 组件类型

MYCP 所有类型组件，都是通过相同标准接口对外提供功能，包括本地服务接口，远程访问调用。功能细分不同包括下列几种：

组件类型	描述	MYCP 加载顺序
LOG	日志组件；提供 MYCP 和其他所有组件的日志打印功能。	最先加载，最后卸载。
PARSER	协议解析组件；提供业务系统通信协议的解析能力；开发者可以方便自定义通信协议用于 MYCP 平台。	第二加载/卸载
SERVER	服务组件；提供其他组件功能接口；具体接口及实现功能，由开发者决定。	第二加载/卸载
APP	应用组件；用户自己开发的业务功能组件。	第二加载/卸载
COMM	通讯组件；提供 MYCP 网络接入通信能力；如 TCP、UDP。	最后加载/最先卸载。

组件类型区分没有明确界定；比如：

LOG 组件同时也可以对外提供 SERVICE 功能，用于打印日志；

SERVICE 组件也可以开放 API 接口远程访问，实现 APP 应用功能；

开发的 APP 应用组件，同样可以开放 SERVICE 接口，用于组件化开放；等等。

MYCP 的组件灵活性，可扩展性，最大限度的让用户方便快速实现分布式、组件化、可管理的大型通信应用平台。

所有开发组件以 DLL 动态库部署，文件格式如下：

Widnows: [ModuleName).dll

Linux: lib[ModuleName].so

3.2 RCA 组件接口函数说明

RCA 组件接口函数格式定义如下：

```
typedef int (FAR *FPCGCApi)(const cgcRequest::pointer & request, cgcResponse::pointer response);
```

用户开发的业务功能组件，对远程客户端提供调用接口函数，使用固定格式函数接口。

以 DLLTest 组件 HelloUser 函数为例，函数定义如下：

```
extern "C" int CGC_API HelloUser(const cgcRequest::pointer & request, cgcResponse::pointer response, cgcSession::pointer session) {}
```

一般开发步骤是，复制整个函数行，修改函数名称，实际功能，比如 UserLogin 如下：

```
extern "C" int CGC_API UserLogin(const cgcRequest::pointer & request, cgcResponse::pointer response, cgcSession::pointer session) {}
```

3.2.1 请求管理类对象 request

请求管理类对象 request，用于获取远程客户端输入参数数据。

使用例子：

```
cgcParameter::pointer pUserName = request->getParameter(_T("UserName"));
cgcParameter::pointer pUserName = CGC_REQ_PARAMETER(_T("UserName"));

const tstring & sAccountId = request->getParameterValue(_T("AccountId"));
const tstring & sAccountId = CGC_REQ_VALUE(_T("AccountId"), "");

long nUserId = CGC_REQ_VALUE(_T("UserId"), 0);
bool bFlag = CGC_REQ_VALUE(_T("Flag"), false);
```

3.2.2 响应管理类对象 response

应用组件通过响应管理类对象 response，返回参数数据给远程客户端。

使用例子：

```
response->lockResponse();
response->setParameter(CGC_PARAMETER("Account", sAccount));
response->sendResponse();
```

或者：

```
CGC_RES_LOCK();
CGC_RES_PARAMETER(CGC_PARAMETER("Account", sAccount));
```

3.2.3 会话管理类对象 session

会话管理类对象 session，用于管理一个远程客户端，打开组件时整个 SESSION 会话周期的状态。更多关于 SESSION 介绍，请看后面介绍，《附录 SOTP 协议》。

使用例子：

```
cgcSession::pointer session = request->getSession();
const tstring & sSessionId = session->getId();
const tstring & sAccount = session->getAccount();
const tstring & sPassword = session->getPasswd();
session->setAttribute(BMT_ACCOUNTIDS, sAccountId, accountInfo);
CAccountInfo::pointer accountInfo =
CGC_POINTER_CAST<CAccountInfo>(session->getAttribute(BMT_ACCOUNTIDS,
sAccountId));
```

会话管理类对象 session，还具有数据对象存储管理功能，详细看后面内容。

3.3 MYCP 内置对象

类似 JSP 或者 PHP 等脚本语言开发一样，MYCP 同样提供有各种内置对象，方便用户开发应用组件，满足各种需求。

3.3.1 内置系统对象 theSystem

```
cgcSystem::pointer theSystem;
```

内置系统对象 theSystem，对 MYCP 系统内所有组件唯一有效；唯一有效，表示所有组件通过 theSystem 获取的系统信息、配置参数等都是一致的。

使用例子：

```
cgcParameterMap::pointer initParameters = theSystem->getInitParameters();
tstring sDbHost = initParameters->getParameterValue("DB_HOST");
const tstring & sServerPath = theSystem->getServerPath();
```

theSystem 内置对象还具有数据对象存储管理功能，详细看后面内容。

3.3.2 内置组件对象 theApplication

```
gcApplication::pointer theApplication;
```

内置组件对象 theApplication，是组件自身的一个内置对象，在组件自身生命周期内有效；主要用于管理组件配置参数，打印日志，定时器等。

使用例子：

```
cgcParameterMap::pointer initParameters =
theApplication->getInitParameters();
cgcParameter::pointer pDbHost = initParameters->getParameter("DB_HOST");
string sDbHost = initParameters->getParameterValue("DB_HOST");
```

theApplication 内置对象还具有数据对象存储管理功能，详细看后面内容。

3.3.3 内置服务管理器对象 theServiceManager

```
extern CGC_CLASS cgcServiceManager::pointer theServiceManager;
```

MYCP 最大的亮点之一，就是可扩展性，通过内置服务管理器对象（theServiceManager）可以访问其他组件接口，轻松实现组件化应用系统。

theServiceManager 属于系统级内置对象，对系统内所有组件唯一有效。

使用例子：

```
cgcServiceInterface::pointer theFileSystemService;
theFileSystemService = theServiceManager->getService("FileSystemService");
```

该源码演示了通过服务名称，获得 FileSystemService 服务接口，用于实现文件基本操作功能，如 delete、rename、copyto 等功能；

MYCP 包含有丰富的服务接口，方便用户开发应用组件；而且还在不断完善丰富中，更多关于服务接口，请看《附录服务组件汇总》

3.4 MYCP 可选函数

本处列出的可选函数，用户可以自由选择，根据自身功能需求决定是否实现该函数。

可选函数为固定格式，包括函数名称、参数定义等，不可修改。

3.4.1 CGC_Module_Init 函数

```
extern "C" bool CGC_API CGC_Module_Init(void) {}
```

MYCP 加载组件时，会自动调用 CGC_Module_Init 函数。

返回 true 组件加载成功；

返回 false 组件加载失败，组件不能提供对外接口。

3.4.2 CGC_Module_Free 函数

```
extern "C" void CGC_API CGC_Module_Free(void) {}
```

MYCP 系统退出，卸载组件时，会自动调用 CGC_Module_Free 函数。

3.4.3 CGC_Session_Open 函数

```
extern "C" bool CGC_API CGC_Session_Open(cgcSession::pointer pHandler) {}
```

远程客户端调用打开 SESSION，或者第一次访问组件函数时，自动打开 SESSION，会自动调用 CGC_Session_Open 函数。

cgcSession::pointer pHandler 包括远程客户端用户信息，可以实现动态客户验证功能。

返回 true 远程客户端连接组件，打开 SESSION 成功，可以开始访问组件接口；

返回 false 远程客户端连接组件，打开 SESSION 失败，不可以访问组件接口。

3.4.4 CGC_Session_Close 函数

```
extern "C" void CGC_API CGC_Session_Close(cgcSession::pointer pHandler) {}
```

远程客户端调用关闭 SESSION 时，会自动调用 CGC_Session_Close 函数。

3.4.5 CGC_GetService 函数

```
extern "C" void CGC_API CGC_GetService(cgcServiceInterface::pointer&
outService, const cgcServiceParameter::pointer& parameter) {}
```

组件通过 CGC_GetService 对外提供标准服务（SERVICE）接口。更多信息请看后面 SERVICE 开发。

3.4.6 CGC_ResetService 函数

```
extern "C" void CGC_API CGC_ResetService(cgcServiceInterface::pointer
inService) {}
```

外部组件使用完服务（SERVICE）接口，通过 CGC_ResetService 函数重置接口，用于交回服务接口；

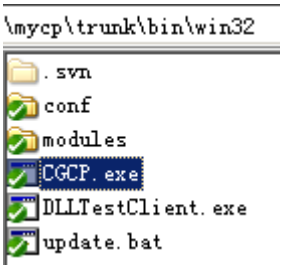
MYCP 不强制外部组件必须重置服务接口，所以对外提供 SERVICE 服务接口组件必须清空内部所有接口；更多信息请看后面 SERVICE 开发。

第4章 MYCP 系统管理

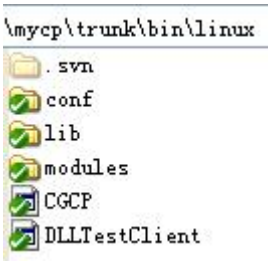
4.1 部署环境组织结构

4.1.1 MYCP 主目录结构

win32：



Linux：



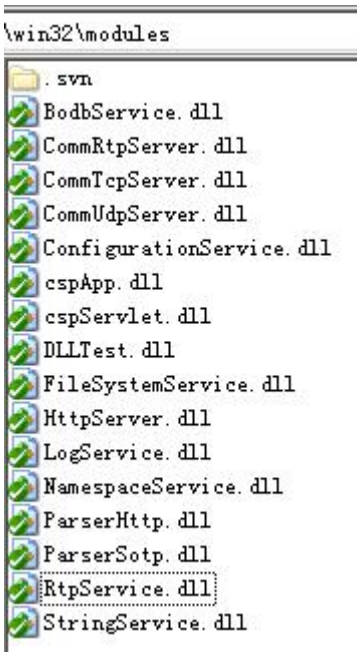
CGCP: MYCP 核心服务程序；
conf 目录: MYCP 配置目录；
modules 目录: MYCP 后台应用功能组件；
DLLTestClient: 可执行 MYCP 客户端例子；
lib 目录: （linux）Linux 环境第三方支持运行库；

4.1.2 MYCP/conf 目录结构

DLLTest 目录: DLLTest 后台模块配置文件；
auths.xml: MYCP 验证用户配置文件；
cdbcs.xml CDBC 数据源配置文件；
default.xml: MYCP 默认参数默认文件；
modules.xml: MYCP 组件配置文件；
params.xml: MYCP 系统参数配置文件；

4.1.3 MYCP/modules 目录结构

win32：



linux：

CommTcpServer: （核心组件）MYCP TCP 通信服务器组件；
CommUdpServer: （核心组件）MYCP UDP 通信服务器组件；
LogService: （核心组件）日志打印组件；
ParserSotp: （核心组件）SOTP 协议解析组件；
ParserHttp: （核心组件）HTTP 协议解析组件；
HttpServer: （核心组件）HTTP 服务器，C++ web 容器组件；
DLLTest: 一个后台功能组件例子；
…

4.1.4 MYCP 系统配置文件

MYCP 配置文件如下：

配置文件	配置功能描述
conf/default.xml	配置 MYCP 系统基本信息。
conf/modules.xml	配置 MYCP 系统加载组件。
Conf/params.xml	配置 MYCP 系统初始化参数。

4.2 配置 MYCP 基本信息

conf/default.xml

MYCP 的基本信息，可以通过内置系统对象 theSystem 进行访问，主要用于标识系统，及显示系统基本信息等作用。

4.2.1 配置说明

配置例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <cgcp>
            <name>CGCP0</name>
            <address>192.168.19.77</address>
            <code>c0</code>
            <rank>0</rank>
    </cgcp>
    <time>
            <waitsleep>3</waitsleep>
    </time>
  <!-- linux may be 'UTF-8' -->
    <encoding>GBK</encoding>
</root>
```

配置说明：

cgcp:

配置项	描述	默认
name	MYCP 名称	CGCP
address	MYCP 地址	127.0.0.1
code	MYCP 代码	cgcp0
rank	MYCP 级别	0

time:

配置项	描述	默认
waitsleep	（保留）启动等待时间，单位秒	0

encoding: （保留）编码；

4.2.2 编程访问

通过内置系统对象 theSystem 访问配置基本参数，函数接口如下：

```
virtual const tstring & getServerName(void) const = 0;
```

```
virtual const tstring & getServerAddr(void) const = 0;
virtual const tstring & getServerCode(void) const = 0;
virtual int getServerRank(void) const = 0;
```

例如：

```
const tstring& sServerName = theSystem->getServerName();
const tstring& sServerCode = theSystem->getServerCode();
```

4.3 配置 MYCP 加载组件

conf/modules.xml

配置 MYCP 系统的加载组件，当 MYCP 系统启动时，会自动加载组件，加载成功的组件，会对外提供相应功能接口。

4.3.1 配置说明

配置例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <supportdebug>1</supportdebug>
    <app>
      <module>
            <file>DLLTest</file>
            <allowall>1</allowall>
            <authaccount>1</authaccount>
            <lockstate>LS_WAIT</lockstate>
            <disable>0</disable>
      </module>
      <module>
            <file>cspServlet</file>
            <allowall>1</allowall>
      </module>
      <module>
            <file>cspApp</file>
            <allowall>1</allowall>
      </module>
      <module>
            <file>StreamModule</file>
            <allowall>1</allowall>
            <authaccount>0</authaccount>
            <lockstate>LS_WAIT</lockstate>
            <disable>1</disable>
```

```
        </module>
        <module>
                <file>StringService</file>
                <disable>1</disable>
        </module>
        <module>
                <file>FileSystemService</file>
        </module>
        <module>
                <file>BodbService</file>
                <disable>0</disable>
        </module>
    </app>
  <parser>
        <module>
                <file>ParserSotp</file>
        </module>
        <module>
                <file>ParserHttp</file>
                <protocol>1</protocol>
        </module>
    </parser>
  <communication>
        <!-- param: 1 - 100, [DEFAULT] 1 -->
        <module>
                <file>CommTcpServer</file>
                <protocol>1</protocol>
                <commport>81</commport>
                <param>3</param>
                <disable>0</disable>
        </module>
        <module>
                <file>CommTcpServer</file>
                <name>uploadfile</name>
                <protocol>1</protocol>
                <commport>82</commport>
                <param>1</param>
                <disable>0</disable>
        </module>
        <module>
                <file>CommUdpServer</file>
                <commport>8012</commport>
                <param>3</param>
        </module>
```

```
        <module>
                        <file>CommRtpServer</file>
                        <commport>8020</commport>
                        <param>3</param>
                        <disable>1</disable>
        </module>

      </communication>
    <server>
        <module>
                        <file>HttpServer</file>
                        <protocol>1</protocol>
                        <disable>0</disable>
        </module>
      </server>
    <log>
        <module>
              <file>LogService</file>
        </module>
    </log>
</root>
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <!-- ******** PARSER -->
    <module>
      <name>SotpParser</name>
      <file>ParserSotp</file>
      <type>PARSER</type>
    </module>
      <!-- ******** COMM -->
    <!-- capacity: 1 - 100, [DEFAULT] 1 -->
    <module>
      <name>CommTcpServer</name>
      <file>CommTcpServer</file>
      <type>COMM</type>
      <commport>8010</commport>
      <capacity>3</capacity>
    </module>
    <module>
      <name>CommUdpServer</name>
      <file>CommUdpServer</file>
      <type>COMM</type>
      <commport>8012</commport>
      <capacity>3</capacity>
```

```
  </module>
  <!-- ******** APP -->
  <module>
    <name>DLLTest</name>
    <file>DLLTest</file>
    <type>APP</type>
    <allowall>1</allowall>
    <authaccount>1</authaccount>
    <lockstate>LS_WAIT</lockstate>
    <disable>0</disable>
  </module>
</root>
```

通用配置项：

配置项	描述	默认
name	组件名称	
file	组件文件；DLLTest.dll 或者 libDLLTest.so 统一配置为“DLLTest”；	
type	组件类型	
protpcol	使用协议：0 SOTP；1 HTTP	0 SOTP
disable	是否禁用组件；0/1；	0 不禁用

COMM 通信组件专用配置项：

配置项	描述	默认
commport	通信组件监听端口；	0
param	组件通信能力；有效：1 – 100；	1

APP 应用组件专用配置项：

配置项	描述	默认
allowall	是否允许开放所有接口，0/1	0 不允许
authaccount	是否自动验证客户端帐号信息，0/1	0 不验证
lockstate	配置组件加锁状态	LS_NONE，不加锁

lockstate 配置：

LS_NONE	无锁控制，可以并行处理，组件内部自己处理并发控制；
LS_WAIT	等待锁控制，临界区状态，单线程处理模式，等待前一个调用返回后，继续当次调用；
LS_RETURN	返回锁控制，如果前一个调用未完成，不等待，立即返回，客户端会收到'-107'错误；

4.3.2 修改 HTTP 端口

默认的 TCP 端口是 HTTP 端口，通过以下配置项修改：

```
<module>
    <file>CommTcpServer</file>
    <protocol>1</protocol>
    <commport>81</commport>
    <param>3</param>
</module>
```

protocol 设置为 1，表示处理 HTTP 协议接入，修改 commport 可以修改 HTTP 监听端口，param 用于设置系统多线程处理能力，可以根据系统规模设置 1-100。（以下同）

4.3.3 增加 HTTP 文件上传端口

通过增加以下配置项，实现增加 HTTP 文件上传端口：

```
<module>
    <file>CommTcpServer</file>
    <name>uploadfile</name>
    <protocol>1</protocol>
    <commport>82</commport>
    <param>1</param>
</module>
```

使用相同的 file 参数，不同的 name、commport 增加配置不同的 HTTP 文件上传端口，建议 param 设为 1，保证文件能够正确处理。

可以根据系统需要，增加多个 HTTP 文件上传端口。

4.3.4 修改 UDP 端口

默认的 UDP 端口是远程访问 RCA 应用组件的端口，通过以下配置项修改：

```
<module>
    <file>CommUdpServer</file>
    <commport>8012</commport>
    <param>3</param>
</module>
```

4.3.5 编程访问

头文件：

```
#include <CGCBase/cgcServices.h>
```

通过内置服务管理器对象 gCgcSeviceManager，可以访问配置组件对外提供的功能接口，服务管理器接口定义如下：

```
virtual cgcServiceInterface::pointer getService(const tstring & serviceName,
const cgcServiceParameter::pointer& parameter = cgcNullServiceParameter) = 0;
virtual void resetService(cgcServiceInterface::pointer service) = 0;
```

例如：

```
cgcConfiguration::pointer configurationService =
CGC_CONFIGURATIONSERVICE_DEF(theServiceManager->getService("ConfigurationSe
rvice"));
```

4.4 配置 MYCP 初始化参数

```
conf/params.xml
```

有一些配置信息，比如 TCP 监听端口，系统数据库名称，帐号信息等，可以通过配置 MYCP 系统初始化参数，让系统所有组件都可以访问。

4.4.1 配置说明

配置例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <parameter>
            <name>DBNAME</name>
            <type>string</type>
            <value>ips</value>
    </parameter>
    <parameter>
            <name>DBACCOUNT</name>
            <type>string</type>
```

```
            <value>ips</value>
        </parameter>
    </root>
```

配置说明：

配置项	描述	默认
name	参数名称	
type	参数类型	string
value	参数配置值	

4.4.2 编程访问

MYCP 系统初始化参数，通过内置系统对象 theSystem 访问，函数接口如下：

```
cgcParameterMap::pointer getInitParameters(void) const = 0;
```

例如：

```
cgcParameterMap::pointer initParameters = theSystem->getInitParameters();
tstring sDbName = initParameters->getParameterValue("DBNAME");
tstring sDbAccount = initParameters->getParameterValue("DBACCOUNT");
```

4.5 配置虚拟主机

MYCP 支持虚拟主机技术，也即一台服务器，配置部署多个不同 web 应用。

4.5.1 配置说明

```
conf/HttpServer/hosts.xml
```

配置例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<hosts>
    <virtualhost>
            <host>*</host>
            <servername></servername>
            <documentroot>../web/samples</documentroot>
            <index>index.csp</index>
    </virtualhost>

    <virtualhost>
```

```
            <host>*:81</host>
            <servername></servername>
            <documentroot>../web/samples</documentroot>
            <index>index.csp</index>
        </virtualhost>
    </hosts>
```

配置项	配置描述
virtualhost	虚拟主机配置项
host	虚拟主机匹配规则；支持类似以下格式： *：匹配所有 *:port：匹配端口所有 IP，如*:81 Ip:*：匹配 IP 所有端口，如 127.0.0.1:*
servername	虚拟主要名称，默认为空
documentroot	虚拟主机 web 目录，支持绝对路径和相对路径二种配置
index	默认访问首页地址文件

4.6 配置 HTTP 文件上传

默认 MYCP 限制 HTTP 文件上传功能，需要通过配置开放 HTTP 文件上传功能，而且可以配置限制文件的大小、类型和文件个数等等。

4.6.1 配置说明

conf/ParserHttp/upload.xml

配置例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<upload>
    <!-- enable-upload: 0/1; Default 0 disable -->
    <enable-upload>1</enable-upload>

    <!-- temp-path: Default "uploads", Must exist dir. -->
    <temp-path>uploads</temp-path>

    <!-- max-upload-count: 0 not limit; Default 1 -->
    <max-file-count>2</max-file-count>
```

```
        <!-- max-upload-size: KB; 0 not limit; Default 10240KB = 10MB -->
        <max-file-size>1024</max-file-size>

        <!-- max-upload-size: MB; 0 not limit; Default 0 -->
        <max-upload-size>0</max-upload-size>

        <!-- enable-all-content-type: 0/1; Default 0 -->
        <enable-all-content-type>1</enable-all-content-type>

        <enable-content-type>
                <content-type>application/msword</content-type>
                <content-type>application/pdf</content-type>
        </enable-content-type>

        <disable-content-type>
                <!--
                <content-type>application/octet-stream</content-type>
                -->
        </disable-content-type>
    </upload>
```

配置项	**配置描述**
enable-upload	0/1，是否开放 HTTP 文件上传功能，默认为 0；
temp-path	HTTP 上传文件临时保存目录，默认为 uploads； **必须手工新建该目录，程序不会主动新建该目录；**
max-file-count	最大上传文件个数，0 不限制，默认为 1
max-file-size	最大单个上传文件大小，单位 KB，默认 10240KB=10MB，0 不限制；
max-upload-size	最大全部上传文件大小，单位 MK，默认 0 不限制；
enable-all-content-type	0/1，是否接受上传所有文件类型，默认 0；
enable-content-type/ content-type	接受上传文件类型
disable-content-type/ content-type	限制上传文件类型

第5章 应用组件配置管理

5.1 组件配置组织结构

5.1.1 组件配置目录

组件配置默认目录：

```
$(MYCP_ROOT)/conf/[ModuleName]
```

[ModuleName] 是 conf/modules.xml 配置文件里面的组件名称（module/name），比如：

```
<log>
  <module>
    <name>LogService</name>
    <file>LogService</file>
  </module>
</log>
```

conf/modules.xml 配置有一个名称为 LogService 的日志服务，LogService 日志服务组件对应的配置目录就是：

```
$(MYCP_ROOT)/conf/LogService
```

其他组件配置目录依此类推。

5.1.2 组件配置文件

组件配置文件如下：

配置文件	配置功能描述
conf/[ModuleName]/params.xml	配置组件初始化参数。
conf/[ModuleName]/auths.xml	配置远程客户端帐号验证信息。
conf/[ModuleName]/methods.xml	配置对外开放接口函数。

以 DLLTest 组件为例，配置初始化参数文件为：

```
$(MYCP_ROOT)/conf/DLLTest/param.xml
```

其他配置文件依此类推。

5.2 配置组件初始化参数

conf/[ModuleName]/params.xml

有一些配置信息，属于组件内部配置信息，可以通过配置组件初始化参数，让组件可以访问。

组件初始化参数只属于配置组件内部使用，其他组件不能访问。

5.2.1 配置说明

组件初始化参数格式，跟 MYCP 初始化参数格式一样；

配置例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <parameter>
            <name>DB_HOST</name>
            <type>string</type>
            <value>ES</value>
    </parameter>
</root>
```

5.2.2 编程访问

组件初始化参数，通过 theApplication 内置对象访问，函数接口如下：

```
cgcParameterMap::pointer getInitParameters(void) const = 0;
```

例如：

```
cgcParameterMap::pointer initParameters = theSystem->getInitParameters();
tstring sDbName = initParameters->getParameterValue("DBNAME");
```

5.3 实现验证客户端帐号功能

可以通过配置验证客户端帐号功能，轻松实现对远程客户端的帐号验证功能，而所有这一切功能的实现，已经由 MYCP 系统和 SOTP 协议内部支持并实现，你所要做的就是填写配置文件，增加相应的用户账号信息。

设置验证客户端帐号功能，只有通过验证的远程客户可以访问，其他用户都不用访问，保护系统组件的安全。

5.3.1 配置不验证客户端帐号

conf/modules.xml

来看一个配置例子：

```
<module>
  <file>DLLTest</file>
  <allowall>1</allowall>
  <authaccount>0</authaccount>
  <lockstate>LS_WAIT</lockstate>
</module>
```

authaccount 设置为 0，表示不验证客户端帐号。

5.3.2 配置验证客户端帐号

conf/modules.xml

来看一个配置例子：

```
<module>
  <file>DLLTest</file>
  <allowall>1</allowall>
  <authaccount>1</authaccount>
  <lockstate>LS_WAIT</lockstate>
</module>
```

authaccount 设置为 1，表示需要验证客户端帐号。

增加或修改配置文件：

conf/[ModuleName]/auths.xml
conf/DLLTest/auths.xml

配置例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <auth>
            <account>admin</account>
            <password>manager</password>
    </auth>
</root>
```

配置说明：

account: 验证帐号
password: 帐号密码

5.3.3 动态验证客户端帐号

虽然可以通过配置验证文件，方便实现决定是否对客户端帐号进行验证。

本节介绍一种动态验证客户端帐号信息功能。

实现原理：

远程客户端第一次访问组件，或者打开连接组件的 SESSION 的时候，系统会自动调用 CGC_Session_Open (cgcSession::pointer)函数，实现原理就是，通过 cgcSession::pointer 获得远程客户端帐号信息，自行内部验证后，返回验证结果。

返回 true 验证成功，客户端连接 SESSION 成功;

返回 false 验证失败，客户端连接 SESSION 失败。

5.4 配置组件开放接口函数

可以利用配置组件开放接口函数，实现不同的开放接口函数功能；可以配置开放所有组件接口函数，或者有选择的开放一部分接口函数。

5.4.1 配置开放所有接口函数

conf/modules.xml

配置开放组件所有接口函数，远程客户端能够调用组件的所有接口函数。

来看一个配置例子：

```
<module>
  <file>DLLTest</file>
  <allowall>1</allowall>
  <authaccount>1</authaccount>
  <lockstate>LS_WAIT</lockstate>
</module>
```

设置 allowall 项为 1，表示允许访问所有接口函数。

5.4.2 配置开放部分接口函数

conf/modules.xml

配置开放组件部分接口函数，远程客户端只能调用配置开放的部分接口函数。

来看一个配置例子：

```
<module>
  <file>DLLTest</file>
  <allowall>0</allowall>
  <authaccount>1</authaccount>
  <lockstate>LS_WAIT</lockstate>
</module>
```

设置 allowall 项为 0，表示不允许开放所有接口函数。

增加或修改配置文件：

conf/[ModuleName]/methods.xml
conf/DLLTest/methods.xml

配置例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <!-- Account Manager -->
    <allow><name>1001</name><method>AccCreate</method></allow>
    <allow><name>1002</name><method>AccDestroy</method></allow>
    <allow><name>1011</name><method>AccRegister</method></allow>
    <allow><name>1012</name><method>AccRegConfirm</method></allow>
    <allow><name>1013</name><method>AccUnRegister</method></allow>
    <allow><name>1014</name><method>AccLoad</method></allow>
    <allow><name>1021</name><method>AccSetPwd</method></allow>
    <allow><name>1022</name><method>AccSetNick</method></allow>
    <allow><name>1023</name><method>AccSetInfo</method></allow>
</root>
```

配置说明：

name: 远程客户端调用时，method 名称。请保持 name 项值的唯一性。
method: DLL 对应实现业务功能的名称。

第6章 数据对象存储管理

开发应用服务组件，离不开数据存储，数据存储需要不同的存储容器，比如保存用户信息的 MAP 表；数据存储有内存，数据库，XML/INI 配置文件等方式。

本章讨论的是属于内存数据对象存储的管理，数据对象存储管理功能提供丰富开发接口，免去用户自定义 std::list、std::map 等容器定义，利用 MYCP 提供的标准接口，实现功能强大的数据对象存储功能。

MYCP 支持以下不同区域的数据对象存储管理功能：

用户可以根据功能的需要，选择一个或者混合多种对象使用。

存储区域	存储对象	描述
系统	theSystem	系统级数据对象存储管理
应用组件	theApplication	组件内部数据对象存储管理
会话	session	远程客户端 SESSION 会话周期，数据对象存储管理
请求	requst	用户请求周期

6.1 数据对象存储简介

6.1.1 存储原理

MYCP 存储容器是一种标准的键/值对，索引表，利用 C++模板和 std::map（或者 std::multimap）容器技术，实现丰富的数据对象存储管理功能。

MYCP 存储数据对象机制，具有智能安全计数指针，和多线程安全功能，可以应付各种复杂的业务应用环境。

6.1.2 存储容器类型

MYCP 数据对象存储管理类型，包括 2 大类*3 小类，共有 6 个存储对象容器类型。2 大类表示二种命名空间存储窗口类型，分别是，基于整数和字符串二种命名空间 MAP 表存储对象窗口类型。

第 1 小类：**StringPointerMapPointer**

实现一个**字符串**指向 cgcPointer 键值对存储容器类型。

第 2 小类：**LongPointerMapPointer**

实现一个**长整数**指向 cgcPointer 键值对存储容器类型。

第 3 小类：**VoidPointerMapPointer**

实现一个**无类型指针**指向 cgcPointer 键值对存储容器类型。

6.2 基本存储方法

6.2.1 定义数据对象

```
#include <CGCBase/cgcobject.h>
```

要存储的数据对象，必须继承于 cgc::cgcObject 类，例子：

```
class CUserInfo
	: public cgc::cgcObject
{
public:
	typedef boost::shared_ptr<CUserInfo> pointer;
	static CUserInfo::pointer create(const tstring & account, const tstring
& password)
	{
		return CUserInfo::pointer(new CUserInfo(account, password));
	}
	CUserInfo(const tstring & account, const tstring & password)
		: m_sAccount(account), m_sPassword(password)
	{}
	~CUserInfo(void)
	{}
public:
	void setAccount(const tstring & newValue) {m_sAccount = newValue;}
	const tstring & getAccount(void) const {return m_sAccount;}
	void setPassword(const tstring & newValue) {m_sPassword = newValue;}
	const tstring & getPassword(void) const {return m_sPassword;}
private:
	tstring m_sAccount;
	tstring m_sPassword;
};
```

定义一个 CUserInfo 类，继承于 cgcObject。

6.2.2 存入数据对象

选择内置对象进行存储管理，假设该用户类属于 APP 应用的数据，存储数据对象到 theApplication 内置对象容器。

```
CUserInfo::pointer userInfo = CUserInfo::create("H.D", "");
theApplication->setAttribute(1, userInfo->getAccount(), userInfo);
```

按照默认 1，把 userInfo 数据对象存储到 userInfo->getAccount()的位置。

6.2.3 判断是否存在数据对象

使用下列方法，判断在指定位置，是否存在数据对象：

```
tstring sUserAccount = “H.D”;
bool isExist = theApplication->existAttribute(1, sUserAccount);
```

6.2.4 访问数据对象

使用下列方法可以访问到指定位置的数据对象：

```
tstring sUserAccount = “H.D”;
CUserInfo::pointer userInfo =
CGC_OBJECT_CAST<CUserInfo>(theApplication->getAttribute(1, sUserAccount));
```

通过判断 userInfo.get() == NULL，是否为空可以知道是否存在数据对象。

6.2.5 删除数据对象

可不需要数据对象的时候，可以通过下列方法删除数据对象：

```
tstring sUserAccount = “H.D”;
theApplication->removeAttribute(1, sUserAccount));
```

MYCP 数据对象采用智能安全计数指针技术，数据对象会管理自己的内存空间，不用强制使用 delete 语句删除对象。

6.2.6 总结

利用 MYCP 提供的标准数据对象存储管理接口，可以应付大部分业务的需求，提供的智能指针和线程安全等技术，充分提高系统的安全性，稳定性。

通过本节的学习，你知道如何定义、存储、访问和管理一个数据对象。

第7章 多线程编程

本节不是介绍多线程编程的基础知识，本章的主要目的是介绍如何利用 MYCP 的接口函数，实现应用组件的多线程功能。

涉及到事件回调，互斥加锁等基本概念。

7.1 基本概念

7.1.1 多线程接口

启动一个独立工作线程：

```
virtual bool SetTimer(unsigned int nIDEvent, unsigned int nElapse,
cgcOnTimerHandler::pointer handler, bool bOneShot = false, const void * pvParam
= NULL) = 0;
```

参数说明：

nIDEvent：唯一标识线程 ID
nElapse：线程回调时间间隔，单位毫秒，1 秒=1000 毫秒
handler：事件回调处理器
bOneShot：是否执行一次，默认不是
pvParam：用户自定义参数，默认为空

停止线程：

```
virtual void KillTimer(unsigned int nIDEvent) = 0;
```

参数说明：

nIDEvent:，要停止线程 ID

停止所有线程：

```
virtual void KillAllTimer(void) = 0;
```

7.1.2 线程事件处理器

事件处理器定义如下：

```
class cgcOnTimerHandler
     : public cgcLock
{
public:
```

```
    typedef boost::shared_ptr<cgcOnTimerHandler> pointer;
    virtual bool IsThreadSafe(void) const {return false;}
    virtual void OnTimeout(unsigned int nIDEvent, const void * pvParam) = 0;
    virtual void OnTimerExit(unsigned int nIDEvent, const void * pvParam) {}
};
```

接口说明：

IsThreadSafe: 是否线程安全标识，默认不实现线程安全
OnTimeout: 线程时间到，执行函数
OnTimerExit: 线程退出执行函数

7.1.3 实现原理

通过 SetTimer 函数启动多线程，类似定时器（timer）一样概念，当线程运行时间到达 nElapse 时间，执行回调事件处理器 OnTimeout 函数。

通过 KillTimer 函数停止线程，执行 OnTimerExit 函数。

7.2 基本编程方法

7.2.1 实现事件处理器

先看一个以下代码：

```
class CMyThreadServer
    , public cgcOnTimerHandler
{
    typedef boost::shared_ptr<CUdpServer> pointer;
    static CMyThreadServer::pointer create(void)
    {
            return CMyThreadServer::pointer(new CMyThreadServer());
    }
    virtual void OnTimeout(unsigned int nIDEvent, const void * pvParam)
    {
    }
};
```

定义 CMyThreadServer 事件处理器，继承 cgcOnTimerHandler 类，OnTimeout 函数就是线程执行接口函数。

7.2.2 启动线程

```
CMyThreadServer::pointer myThread = CMyThreadServer::create();
```

```
cgc::theApplication->SetTimer(1, 100, myThread);
cgc::theApplication->SetTimer(2, 100, myThread);
```

启动 2 个独立线程，每 100 毫秒执行 myThread 对象 OnTimeout 函数一次。

myThread 对象 OnTimeout 函数每隔 100 毫秒执行一次，直到 KillTimer 函数停止线程为止。

如果只想启动执行一次的线程，使用以下地址：

```
CMyThreadServer::pointer myThread = CMyThreadServer::create();
cgc::theApplication->SetTimer(1, 100, myThread, true);
```

7.2.3 停止线程

停止线程很简单，调用以下代码即可：

```
cgc::theApplication->KillTimer(1);
cgc::theApplication->KillTimer(2);
```

或者使用以下函数，停止所有线程：

```
cgc::theApplication->KillAllTimer();
```

7.2.4 线程安全

如果 CMyThreadServer 的 OnTimeout 函数有共享资源，需要进行互斥保护，增加实现 IsThreadSafe 函数即可实现线程安全功能，全部代码如下：

```
class CMyThreadServer
	, public cgcOnTimerHandler
{
	typedef boost::shared_ptr<CUdpServer> pointer;
	static CMyThreadServer::pointer create(void)
	{
		return CMyThreadServer::pointer(new CMyThreadServer());
	}
	virtual bool IsThreadSafe(void) const {return true;}
	virtual void OnTimeout(unsigned int nIDEvent, const void * pvParam)
	{
	}
};
```

IsThreadSafe 函数返回 true，MYCP 系统会自动实现互斥控制，实现线程安全功能。

第8章 使用日志服务

每个程序都需要显示错误消息、调试信息，等等。在传统上，我们会使用 printf 或者调用 cerr 语句。MYCP 的提供的日志服务组件为我们提供了多种日志打印输出的解决方案，同时给予开发人员极大的控制能力，可以决定打印多少日志信息，以及把日志信息定向到哪里，等等。

在这一章，将主要涉及以下内容：

- 使用基本的日志记录
- 启用和禁用各种日志级别的显示
- 把输出日志定向到各种目标

8.1 基本的日志记录

用来输出日志宏是：CGC_LOG，该输出日志宏跟以下语句操作完全相同：

```
theApplication->log(cgcLogService::LogLevel level, const char* format,...)
theApplication->log(cgcLogService::LogLevel level, const wchar_t* format,...)
```

level 参数指定你的日志信息严重级别，比较常用的是 cgcLogService:: LOG_DEBUG，cgcLogService:: LOG_ERROR。

format 参数是一组格式转换操作符， 与 printf 语句使用相同格式参数。下面给出一个日志输出例子：

```
CGC_LOG((cgcLogService::LOG_DEBUG, “My Account is %s”, sAccount.c_str()));
```

该行语句把日志信息输出到日志服务器中，默认情况下，会把日志信息输出到 cerr，同时会在系统 log 目录下，新建一个对应组件名称目录，在该日志下会生成一个[组件名称.log]的日志文件，该日志文件会打印一定格式的日志信息内容。

8.1.1 日志严重级别

下面列出所有的日志严重级别：

日志级别	描述	配置值
LOG_TRACE	跟踪日志信息	0x1 = 1

LOG_DEBUG	测试日志信息	0x2 = 2
LOG_INFO	一般日志信息	0x4 = 4
LOG_WARNING	警告日志信息	0x8 = 8
LOG_ERROR	错误日志信息	0x10 = 16
LOG_ALERT	警报日志信息	0x20 = 32

程序开发的时候，就已经确定输出所有日志的不同级别，然后由日志配置文件指定在不同的运行时期或环境中，启用输出不同的某些日志级别的日志信息。

8.2 日志服务组件

MYCP 日志服务由 LogService 组件提供日志记录功能，LogService 日志服务器配置在 conf/modules.xml 文件，如下例子：

```
<log>
  <module>
    <file>LogService</file>
  </module>
</log>
```

配置 LOG 类型表示为 MYCP 日志服务器组件类型。

所有组件的日志配置文件为 XML 格式,统一保存于 conf/LogService 目录下，文件格式如下：

```
conf/LogService/[ModuleName].xml
```

[ModuleName]表示组件名称，举例子，DLLTest 组件的日志服务配置文件为：

```
conf/LogService/DLLTest.xml
```

8.2.1 一个配置格式例子

下面是一个日志服务配置例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <loglevel>20</loglevel>
    <!-- log to -->
    <tostderr>1</tostderr>
    <tofile>1</tofile>
    <tosystem>0</tosystem>
```

```
        <tocallback>0</tocallback>
        <tologger>0</tologger>
        <!-- tofile setting-->
        <path></path>
        <file></file>
        <maxsize>1</maxsize>
        <backupnumber>1</backup>
        <locale>chs</locale>
        <!-- Format -->
        <dateformat>1</dateformat>
        <timeformat>1</timeformat>
        <onelineformat>1</onelineformat>
        <levelstring>1</levelstring>
    </root>
```

8.2.2 启用日志输出级别

日志配置文件的 loglevel，是最主要的配置项，通过 loglevel 可以配置启用输出某个日志级别的日志信息，例如默认配置值是 20，20=4+16=LOG_INFO+LOG_ERROR，表示只有 LOG_INFO 和 LOG_ERROR 二个严重级别的日志信息，日志服务器才会处理，其他级别的日志信息不会输出，

建议在编码开发过程中，通过 CGC_LOG 日志宏，输出所有各种不同级别的日志信息，然后在项目开发的不同阶段，选择启用不同的日志级别，输出不同的日志信息。方便后期维护、跟踪调试。

例如在项目的初期，可以配置 255，输出所有的日志信息，在项目中后期，只配置启用 48=16+32=LOG_ERROR| LOG_ALERT，等等。

8.2.3 选择不同输出定向

日志服务器可以通过配置，选择日志的不同输出定向，日志服务器可以选择重定向输出到以下 5 个输出目标：

配置项	描述	默认值
tostderr	0/1，输出到标准错误流 std::cerr	1 是
tofile	0/1，输出到文件	1 是
tosystem	0/1，输出到系统日志记录器，Linux syslog 或	0 否

	NT Lvent Log，保留	
tocallback	0/1，输出到 CallBack 函数，保留	0 否
tologger	0/1，输出到网络某个日志服务器中，保留	0/否

8.2.4 格式化日志信息内容

可以选择设置日志信息的输出格式，目前支持以下四个格式输出：

配置项	描述	默认值
dateformat	0/1，是否在日志信息头添加日期信息，例如：2010-01-20	1 是
timeformat	0/1，是否在日志信息头添加时间信息，例如：14:03:25.798	1 是
onelineformat	0/1，是否格式化日志信息为一行信息，所有换行符替换为空格。	1 是
levelstring	0/1，是否在日志信息头添加日志级别名称，如[INFO]等。	1 是

8.2.5 动态配置日志服务

LogService 支持动态配置日志服务，可以在不需要重新启动应用组件时，修改组件的日志配置文件，动态加载新的日志配置信息。

动态服务，每 60 秒（即 1 分钟），自动加载一次。

8.3 输出日志信息到文件

默认情况下，或者设置配置文件的 tofile 为 1，日志服务器将会把日志信息输出保存到日志文件中。

8.3.1 输出文件配置

日志文件名称、保存目录，文件大小及备份文件等，配置如下：

配置项	描述	默认值
path	输出日志保存目录	$(MYCP_ROOT)/log/[ModuleName]
file	输出日志文件名称	[MOduleName].log

maxsize	最大日志文件大小	1MB
backupnumber	备份日志文件数量	1
locale	日志文件区域	chs

举个例子，下面为 DLLTest 组件的日志配置内容：

```
<!-- tofile setting-->
<path></path>
<file></file>
<maxsize>2</maxsize>
<backupnumber>2</backup>
<locale>chs</locale>
```

以上组件的日志信息保存目录为 MYCP/log/DLLTest，日志文件为 DLLTest.log，日志文件最大为 2MB，当 DLLTest.log 文件大于 2MB 时，备份文件，最多备份文件数为 2 个，备份文件名为 DLLTest.log0、DLLTest.log1。

可以修改日志记录文件名和保存到其他目录，如：

```
<path>/usr/mylog</path>
<file>mylogfile.l</file>
```

将把日志文件名改为 mylogfile.l，保存到/usr/mylog 目录，备份文件不变，只是在日志文件名后加上备份序列号，例如 mylogfile.l0、mylogfile.l1。

第9章 附录 A、组件服务接口汇总

目前 MYCP 提供的各种 C++ APP 应用服务接口，配置加载后可以直接使用，方便开发应用组件。用户可以自行扩展功能，或者开发自己的应用组件服务接口。

细分不同服务类型包括有：

- LOG：日志服务组件
- COMM：通信服务器组件
- APP：普通功能服务组件；或者用户自己开发的业务组件

9.1 LOG 组件

LOG 是日志服务器组件，目前只提供一个核心解析组件，用于系统的日志记录：

服务组件	功能描述	开发者
LogService	日志服务组件	H.D

9.2 COMM 组件

COMM 通信服务器，负责 MYCP 网络接入，有以下几个网络通信服务器：

服务组件	功能描述	开发者
CommRtpServer	RTP 传输服务器	H.D
CommTcpServer	TCP 传输服务器	H.D
CommUdpServer	UDP 传输服务器	H.D

9.3 APP 组件

APP 组件可以提供普通服务接口功能，目前提供有下列组件：

服务组件	功能描述	开发者
ConfigurationService	配置服务组件，提供 XML 和 INI 文件配置能力	H.D
NamespaceService	命名空间服务组件，提供键/值对访问管理服务	H.D

FileSystemService	基本文件操作服务组件	H.D
BodbService	Bodb 数据库服务组件，提供简单标准数据库存储功能	H.D
RtpService	RTP 传输能力服务组件	H.D
SipService	SIP 协议通信能力服务组件	H.D
StringService	基本字符串转换操作功能组件	H.D
…		

第10章 附录 B、编译第三方库

10.1 Boost_1_41_0

MYCP 最主要使用的第三方库是 Boost；目前使用 Boost_1_41_0 版本；

Boost主页：http://www.boost.org/

10.1.1 Visual Studio 2005 编译：

选择进入 Visual Studio 2005 Command Prompt，如下例子：
All Programs > Microsoft Visual Studio 2005 > Visual Studio Tools > Visual Studio 2005 Command Prompt

输入以下命令编译 Boost：

```
cd $(BOOST_PATH)
bootstrap.bat
bjam --build-type=complete --with-filesystem
bjam --build-type=complete --with-thread
```

10.1.2 Linux 编译：

```
# tar xzvf boost_1_41_0.tar.gz
# cd boost_1_41_0
# ./bootstrap.sh
# sudo ./bjam install --with-filesystem
# sudo ./bjam install --with-thread
```

Boost 库编译时间较久，请耐心等候！！！

10.2 JRTP

CGCLib 和 CommRtpServer 项目使用了 jrtp 和 jthread 库，libRTP 封装了对 jrtp 库的使用；可以通过修改 CGCBase/cgcuses.h 文件，默认去掉对 RTP 的支持；

jrtp和jthread主页：http://www.edm.uhasselt.be/ http://www.uhasselt.be/

10.2.1 jthread-1.2.1 编译：

Windows 编译：

打开 $(JTHREAD_PATH)\jthread.sln 或者 jthread.dsw 或者 jthread_vc9.sln 文件，然后编译整个 solution。

Linux 编译：

```
# unzip jthread-1.2.1.zip
# cd jthread-1.2.1
# ./configure
# make
# sudo make install
```

10.2.2 jrtplib-3.7.1 编译

Windows 编译：

打开$(JRTPLIB_PATH)\jrtplib.sln 或者 jrtplib.dsw 或者 jrtplib_vc9.sln 文件，然后编译整个 solution。

Linux 编译：

```
# unzip jrtplib-3.7.1.zip
# cd jrtplib-3.7.1
# ./configure
# make
# sudo make install
```

jrtplib-3.7.1 库在有些 linux 编译不通过，需要做以下几点修改：

- 在 src/rtperrors.cpp 源文件添加头文件，#include "stdio.h"
- 在 src/rtcpcompoundpacketbuilder.cpp 和 src/rtppacket.cpp 源文件添加头文件，#include "string.h"

10.3 Bodb 数据库

Bodb 数据库是一个简单易用、支持标准 SQL 语句的内存数据库系统；

Bodb 原本是独立的一个开源项目，为了方便 MYCP 相关服务开发，将 Bodb 项目合并到 MYCP 项目中；有需要可以将 Bodb 分开，单独使用。

Bodb 数据库的编译安装请看《编译 MYCP 基础库》。

第11章　附录 C、编译 MYCP 基础库

MYCP 使用 C++语言开发，同时使用了部分第三库协助开发，开始普通 CSP、C++ Servlet 和 C++ APP 应用组件不需要编译 MYCP；开发 RCA 客户端程序，必须先编译 MYCP 及所有第三方库，本章主要涉及以下内容：

- Bodb 数据库的安装编译
- MYCP 基础库的安装编译

11.1 Linux 用户

在 Linux 环境下，如果部分 configure 文件不可执行，可以输入下列命令，将 configure 修改成可执行属性：

```
# chmod a+x configure
```

假设 boost、jrtp 和 jthread 库安装到/usr/local/include，请使用下列 configure 命令：

```
# ./configure CPPFLAGS="-I/usr/local/include/boost-1_41
-I/usr/local/include/jrtplib3 -I/usr/local/include/jthread"
```

11.1.1 编译 MYCP 核心

第一步先编译 MYCP 核心，执行下面命令：

```
# cd mycp/src
# ./configure
# make
# sudo make install
```

输出文件	描述	第三方库
libClass.a	MYCP 基础库	
libCGCLib.a	MYCP 客户端，SOTP 协议栈	jrtp

11.1.2 编译 Bodb 数据库

```
# cd mycp/src/ThirdParty/Bodb
# ./configure
# make
```

```
# sudo make install
```

输出文件	描述	第三方库
libsqlparser.a	Bodb 数据库 SQL 解析库	
libbodb.a	Bodb 数据库核心实现库	
bodbclient	Bodb 数据库客户端测试程序	

11.1.3 编译 APP（服务接口）组件

```
# cd mycp/src/CGCServices
# ./configure
# make
# sudo make install
```

输出文件	描述	第三方库
libBodbService.so	Bodb 数据库服务组件	
libConfigurationService.so	XML、INI 文件配置服务组件	
libFileSystemService.so	基本文件操作服务组件	
libNamespaceService.so	命名空间服务组件，提供 Key/Value 存储管理功能	
libRtpService.so	RTP 协议传输服务组件	jrtp
libSipService.so	SIP 协议通讯服务组件	osip
libStringService.so	基本字符串操作服务组件	
…		

11.1.4 编译 COMM（通讯）组件

```
# cd mycp/src/CGCComms
# ./configure
# make
# sudo make install
```

输出文件	描述	第三方库
libCommRtpServer.so	RTP 传输服务器	jrtp
libCommTcpServer.so	TCP 传输服务器	
libCommUdpServer.so	UDP 传输服务器	

11.1.5 编译 Samples

```
# cd mycp/samples
# ./configure
# make
# sudo make install
```

输出文件	描述	第三方库
cspApp.so	C++ APP 组件例子	
cspServlet.so	C++ Servlet 组件例子	
libDLLTest.so	MYCP APP（RCA）组件例子	
DLLTestClient	RCA 客户端应用程序例子	
…		

11.2 Windows 用户

Windows 必须先手工编译 Bodb 数据库，和 ThirdParty 目录的一些第三方库。

11.2.1 编译 Bodb 数据库

项目名称	sqlparser bodb bodbclient
MSVC8.0	$(MYCP_ROOT)\src\ThirdParty\Bodb\proj\MSVC8.0\bodb.sln
输出目录	$(MYCP_ROOT)\src\ThirdParty\Bodb\build\
第三方库	Boost

11.2.2 编译 ThirdParty

项目名称	libRTP
MSVC8.0	$(MYCP_ROOT)\src\ThirdParty\proj\MSVC8.0\libThirdParty.sln
输出目录	$(MYCP_ROOT)\src\ThirdParty\build\
第三方库	Boost jrtp jthread

11.2.3 编译 MYCP 核心

项目名称	CGCBas CGCClass
MSVC8.0	$(MYCP_ROOT)\src\proj\MSVC8.0\MYCP.sln
输出目录	$(MYCP_ROOT)\src\build\
第三方库	Boost jrtp jthread

11.2.4 编译 SERVICE（服务）组件

项目名称	BodbService ConfigurationService FileSystemService NamespaceService RtpService SipService StringService
MSVC8.0	$(MYCP_ROOT)\src\CGCServices\proj\MSVC8.0\CGCServices.sln
输出目录	$(MYCP_ROOT)\src\CGCServices\build\
第三方库	Boost Bodb libRTP

11.2.5 编译 COMM（通讯）组件

项目名称	CommRtpServer CommTcpServer CommUdpServer
MSVC8.0	$(MYCP_ROOT)\src\CGCComms\proj\MSVC8.0\CGCCommunications.sln
输出目录	$(MYCP_ROOT)\src\CGCComms\build\
第三方库	libRTP

第12章　附录 D、SOTP 协议

SOTP：Sample Object Transmission Procotol，简单对象传输协议。

12.1 协议概述

12.1.1 二个协议项

SOTP 包括有 SESSION 会话协议和 APPLICATION 应用协议，二个协议项。

12.1.2 协议动作

SOTP 包括有下列四协议动作：

协议动作	描述
OPEN	打开会话
CLOSE	关闭会话
ACTIVE	激活会话
CALL	调用组件接口函数

12.1.3 系统返回错误代码

>=-100 返回代码给应用组件自己定义；

<=-101 返回代码为系统保留错误代码，MYCP 后台错误代码：

错误代码	错误描述
-101	不允许访问应用组件
-102	错误模块句柄
-103	错误会话句柄
-104	验证帐号失败

-105	打开会话，初始化组件失败。
-106	不允许调用该接口函数
-107	调用组件函数加锁返回
-111	发生系统异常
-112	协议格式错误
-113	错误应用组件句柄
-114	错误应用接口函数
-115	错误 SOTP 协议
-116	错误 SOTP 协议类型

12.2 SESSION 协议

12.2.1 打开 SESSION 会话

请求打开 SESSION 会话协议：

```
OPEN SOTP/2.0\n
Cid: [CALL_ID]\n
App: [APP_MODULE_NAME]\n
Ua: [ACCOUNT];pwd=[PASSWD];enc=[ENCODE]\n
```

OPEN 表示请求打开一个 SESSION 会话，SOTP/2.0 是固定格式 SOTP 协议头；[CALL_ID]呼叫编号，长整数，用于标识每一个协议请求，协议返回时带回该值；[APP_MODULE_NAME]是后台应用组件名称。

Ua 协议项为可选协议字段，后台配置需要验证帐号时有效；[ACCOUNT]为帐号名称，[PASSWD]为帐号密码，[ENCODE]为帐号或者密码的加密方式，由前后端配合实现。

例子 1：

```
OPEN SOTP/2.0
Cid: 0
App: DLLTest
```

以上例子，请求打开 DLLTest 组件的 SESSION 会话。

例子 2：

```
OPEN SOTP/2.0
Cid: 0
App: DLLTest
Ua: akee;pwd=akee;enc=
```

以上例子，请求打开 DLLTest 组件的 SESSION 会话，帐号和密码为 akee。

返回协议：

```
OPEN SOTP/2.0 [RETURN_CODE]\n
Cid: [CALL_ID]\n
Sid: [RETURN_SESSIONID]\n
```

[RETURN_CODE]是系统调用返回代码，[CALL_ID]是客户端调用时呼叫编号的值，如果打开组件 SESSION 会话成功，返回[RETURN_SESSIONID]值，成功返回 16 位唯一数字字符串，唯一标识该会话。

例子：

```
OPEN SOTP/2.0 0
Cid: 1
Sid: 9879431313133879
```

返回代码 0，呼叫编号 1，成功打开组件的 SESSION 会话，编号为 9879431313133879。

12.2.2 关闭 SESSION 会话

请求关闭 SESSION 会话协议：

```
CLOSE SOTP/2.0\n
Cid: [CALL_ID]\n
Sid: [SESSIONID]\n
```

CLOSE 请求关闭 SESSION 会话，[SESSIONID]请求关闭的会话编号。

例子：

```
CLOSE SOTP/2.0
Cid: 1
Sid: 9879431313133879
```

请求关闭编号为 9879431313133879 的 SESSION 会话。

返回协议：

```
CLOSE SOTP/2.0 [RETURN_CODE]\n
Cid: [CALL_ID]\n
```

```
Sid: [CLOSE_SESSIONID]\n
```

例子：

```
CLOSE SOTP/2.0 0
Cid: 1
Sid: 9879431313133879
```

关闭 SESSION 会话成功。

12.2.3 激活 SESSION 会话

请求激活 SESSION 会话协议：

```
ACTIVE SOTP/2.0\n
Cid: [CALL_ID]\n
Sid: [SESSIONID]\n
```

ACTIVE 请求激活 SESSION 会话，[SESSIONID]请求激活的会话编号。

例子：

```
ACTIVE SOTP/2.0
Cid: 1
Sid: 9879431313133879
```

请求激活编号为 9879431313133879 的 SESSION 会话。

返回协议：

```
ACTIVE SOTP/2.0 [RETURN_CODE]\n
Cid: [CALL_ID]\n
Sid: [CLOSE_SESSIONID]\n
```

例子：

```
ACTIVE SOTP/2.0 0
Cid: 1
Sid: 9879431313133879
```

激活 SESSION 会话成功。

12.3 APP 应用协议

APP 应用组件协议支持以下各种不同调用协议格式：

- 通过已经打开 SESSION 会话调用应用组件接口函数；
- 未打开 SESSION 会话，支持第一次调用接口函数，自动打开 SESSION 会话；
- 支持不带、带一个或多个的输入参数，调用接口函数
- 支持传输附件调用接口函数

应用组件调用协议格式如下：

```
CALL SOTP/2.0\n
App: [APP_MODULE_NAME]\n
Ua: [ACCOUNT];pwd=[PASSWD];enc=[ENCODE]\n
Sid: [SESSION_ID]\n
Cid: [CALL_ID]\n
Sign: [SIGN_ID]\n
Api: [FUNCTION_NAME]\n
Pv: [PARAMETER_NAME];pt=[PARAMETER_TYPE];pl=[PARAMETER_VALUE_LEN]\n
[PARAMETER_VALUE]\n
Pv: …
At: [ATTACH_NAME];at=[TOTAL_LEN];ai=[ATTACH_INDEX];al=[ATTACH_LEN]\n
[ATTACH_DATA]\n
```

应用组件调用返回协议格式如下：

```
CALL SOTP/2.0 [RETURN_CODE]\n
```

[RETURN_CODE]表示为返回协议，其他协议项完全一样。

协议配置说明：

CALL SOTP/2.0\n	**调用组件 API 接口函数，协议头**
[APP_MODULE_NAME]	（可选）应用组件名称，用于未打开 SESSION 会话时使用
[ACCOUNT] [PASSWD] [ENCODE]	（可选）帐号名称、密码和加密方式；用于未打开 SESSION 会话时使用。 可选
[SESSION_ID]	SESSION 会话编号，如果不填使用 App:项
[CALL_ID]	呼叫编号
[SIGN_ID]	标识编号
[FUNCTION_NAME]	组件接口函数名称

Pv: [PARAMETER_NAME] [PARAMETER_TYPE] [PARAMETER_VALUE_LEN] [PARAMETER_VALUE]	（可选）输入参数；可不填，也可重复多个 参数名称 参数类型 参数值长度 参数的值
At: [ATTACH_NAME] [TOTAL_LEN] [ATTACH_INDEX] [ATTACH_LEN] [ATTACH_DATA]	（可选）传输附件 附件名称 附件总长度 当前附件序列位置 当前传输附件长度 当前传输附件数据

12.3.1 调用已经打开 SESSION 会话接口函数

之前已经通过 OPEN 协议成功打开某个应用组件的 SESSION 会话，利用 API 协议，实现调用该组件的接口函数；调用例子如下：

```
CALL SOTP/2.0
Sid: 9879431313133879
Cid: 1
Sign: 1
Api: MyFunction
```

请求调用 9879431313133879 会话对应组件的接口 MyFunction 函数。

12.3.2 调用接口函数，自动打开 SESSION 会话

SOTP 协议支持第一次请求调用接口函数，自动打开 SESSION 会话，调用例子如下：

```
CALL SOTP/2.0
App: MyDLL
Cid: 1
Sign: 1
Api: MyFunction
```

请求调用 MyDLL 组件，自动打开 SESSION 会话，并调用 MyFunction 接口函数。

12.3.3 输入参数调用组件接口函数

输入参数，调用例子如下：

```
CALL SOTP/2.0
Sid: 9879431313133879
Cid: 1
Sign: 1
Api: Login
Pv: UserName;pt=sotp.string;pl=6
system
Pv: Password;pt=sotp.string;pl=7
manager
```

请求调用 9879431313133879 会话对应组件的 Login 接口函数，有 UserName 和 Password 二个参数，分别参数值为 system 和 manager。

12.3.4 传输附件调用组件接口函数

SOTP 协议支持传输二进制数据，用于文件，或者音视频流的传输等，调用例子如下：

```
CALL SOTP/2.0
Sid: 9879431313133879
Cid: 1
Sign: 1
Api: UpFile
At: file;at=2048;ai=0;al=1024
.....
```

请求 UpFile 接口函数，上传附件名称 file，文件总长度 2048，当前序列 0，当前附件长度为 1024 的附件。