基于 HTTP 协议的计算机远程控制系统设计和实现

SinSoul

摘要：本论文主要介绍了一套基于 HTTP 协议的远程控制软件的控制端与被控制端从设计到实现的过程，以及控制端与被控制端交互的过程及原理。这是一套较为新型的计算机远程控制软件，控制者无需安装任何软件或插件，只需要在常用浏览器中打开控制端网站就可进行远程控制，相比传统远程控制软件具有易配置，跨平台，多用户等优点。被控制端使用 C++语言在 Windows SDK 下进行开发，使用基于 HTTP 协议的 C/S（客户端/服务器）模式。控制端的前端代码为 HTML、Javascript、CSS 的普通网页，并使用部份 HTML5 和 CSS3 代码，后端使用 PHP 语言处理并使用 MySQL 作为数据库。

关键词：C++、HTTP、LAMP、Json、Ajax、远程控制

The Computer Remote Control System Based on HTTP

SinSoul

Computer Science and Technology Instructor:

Abstract: This paper introduces the remote control system design and implementation, and client and server how to communicate. This is a new type computer remote control software suite. Controller only need open a site in normal browser then they can remote control PC without other software or plug-in. It should be simple, cross platform, multi user. Client used the Windows SDK with C + + language development, using the Client/Server mode based on HTTP. The server front end is HTML, Javascript and css code formed normal web page, back end is php and mysql database server.

KeyWords: C++,HTTP,LAMP,Json,Ajax,Remote Control

目录

基于 HTTP 协议的计算机远程控制系统设计和实现 1
目录 2
第一章　引言 5
1.1　本软件开发的背景及必要性 5
1.1.1　黑客远程控制软件的不足 5
1.1.2　正规远程控制软件的不足 5
1.1.3　本系统的设计目标 5
第二章　开发环境、工具及运行平台的选择 6
2.1 被控制端运行的操作系统 6
2.2 控制端运行平台的选择 6
2.3 编程语言及开发工具 6
2.3.1 C++语言 6
2.3.2 Microsoft Visual Studio 2010 10.0.40219.1 SP1Rel 7
2.3.3 Notepad++ 6.1.5 7
2.3.4 Google Chrome 23.0.1271.64 7
2.4 其它辅助工具 7
2.4.1 Visual Assist X 10.7.1903.0 built 2012.04.03 7
2.4.2 VMware Workstation 8.0.1 build-528992 7
2.4.3 SVN 1.6.17 (r1128011) 7
2.4.4 GIT 1.7.5.1 7
第三章　设计 8
3.1 数据库的设计 8
3.1.1 要考虑的问题 8
3.2 整个系统的总体结构 8
3.3 数据库的最终设计 8
3.3.1 实体以及属性分析 9
3.3.2 控制系统数据库 E-R 图 9
3.3.3 将 E-R 图转换为数据库表 10

3.3.4 应用三大范式对多张表进行审核，规范表结构 10
3.3.5 创建数据库的 SQL 语句以及解释 10
3.4 远程控制系统数据流图 13
3.4.1 顶层数据流图: 13
3.4.2 控制系统中层数据流图 13
第四章 基础模块的实现 14
4.1 被控制端 HTTP 通信类的实现 14
4.1.1 HTTP 通信类概述 14
4.1.2 HttpMessenger 类关键成员函数及变量的介绍和使用 14
4.1.3 HttpMessenger 使用示例 15
4.2 控制命令与执行结果的传输过程 16
4.2.1 并发的命令执行 16
4.2.2 命令与命令执行结果的具体获取方式 17
4.2.3 服务器端命令和执行结果的等待方式 17
4.2.4 添加命令的关键 PHP 代码 18
4.2.5 命令获取的 PHP 代码 19
4.3 被控制端与控制端的关联 20
4.3.1 关联凭据的具体传递方式 20
4.3.2 被控制端登录时，服务器处理的 PHP 代码 21
第五章 各个功能模块的实现 24
5.1 被控制端界面的实现 24
5.2 控制端的各种功能模块 24
5.2.1 用户登录及用户注册界面 24
5.2.2 被控制主机选择界面 25
5.2.3 控制主机的虚拟桌面环境 25
5.2.4 命令行及摄像头监控 26
5.2.5 文件管理功能 26
第六章 测试及总结 27
6.1 测试 27
6.1.1 局域网内测试 WAMP 环境 27

6.1.2 广域网环境 SAE 平台测试 27
6.1.3 控制端在移动设备上的测试 27
6.2 关于开源 27
6.2.1 开源的原因 27
6.2.2 开源的方式 27
6.3 总结 28
参考文献： 28
致谢 28

第一章　　引言

1.1 本软件开发的背景及必要性

1.1.1 黑客远程控制软件的不足

（1）在校园网、企业网等内网环境中难于控制外网的主机。

（2）被杀毒软件查杀，影响正常使用。

（3）性能受限于作者对线程池和完成端口的具体实现，某些远控软件在控制较多主机时出现被控制端反复上线，大片掉线等不稳定情况。

（4）控制端和被控制端很多只支持 Windows 平台。

（5）各种流传的修改版本，难于选择，甚至很多被捆绑其它木马程序。

1.1.2 正规远程控制软件的不足

（1）部分控制端虽然可运行于浏览器，却是以 Active X 控件的方式，只支持 IE 浏览器，这与直接提供控制端应用程序并没有多大区别，

（2）提供的控制端应用程序只支持屏幕控制的功能，在网络或流量不允许的情况下，或是只需要执行简单命令的情况下显得不实用。

（3）移动设备的控制端应用程序只支持部分智能手机，对于机能较低的设备则无法使用。

1.1.3 本系统的设计目标

为了改善和解决上述不足和问题，决定设计一个具有如下优势的计算机远程控制系统。

（1）使用免费或廉价的 HTTP 服务器做为中介，连接控制端与被控制端，只要能访问互联网，就能实现控制。

（2）对于不同的操作系统提供不同的被控制端程序，但只提供一个统一的基于 Web 的控制端，也就是只需要一个浏览器，不需要安装任何软件，只要能打开网页，就能进行控制，这当然也包括对手机，平板电脑等移动设备的支持。

（3）提供简单，直观，易于使用的图形操作界面，让用户几乎不需要任何学习过程便可上手使用。

（4）异步，并发的执行控制命令，也就是在执行一条指令后不必等到它执行完成，便可执行下一条指令，执行结果会在执行完成过后显示出来，这样能节省使用者的时间，同时在网络状况不佳时也能有较好的表现。

（5）不依赖于任何的第三方服务支持，任何个人或组织都可以在任何环境下搭建并使用整个系统。

（6）遵循 BSD 协议进行开源。免费。

第二章　开发环境、工具及运行平台的选择

2.1 被控制端运行的操作系统

本远程控制软件的被控制端需要在被控制主机上运行，经过简单设计目前大多数人使用的操作系统是 Windows XP SP3 32 位以及 Windows 7 SP1 32 位和 Windows 7 SP1 64 位，于是将被控制端的运行平台定为任意版本 Windows XP 和任意版本 Windows 7 甚至包括 Windows 8。

此外，为了最大的发挥控制软件的跨平台特性，让使用 MAC OS X 或者是 Linux 的用户也能使用本系统，保证被控制端能方便快速的移植到类 Unix 系统，在开发过程中应该注意代码的可重用性，使之只需要做少许修改就能工作在类 Unix 平台上，让使用类 Unix 操作系统的用户同样能方便的进行远程控制，更能体现此软件存在的价值。

2.2 控制端运行平台的选择

基于目前流行的广泛用于实际开发的编程语言，有 3 种平台可供选择。

（1）使用 IIS + C# + MSSQL Server：由于国内互联网被严格审查，导致网上已经找不到稳定，免费，高速的支持这个搭配的虚拟主机了，VPS 价格过于昂贵，与设计初衷相违背，所以只好排除。

（2）使用 Python + HRD(High Replication datastore)构建于 Google App Engine 之上：这本身同样也是一个非常不错的选择，另外 Google App Engine 所提供的免费流量和数据存储量也是足够个人使用的，并且速度和稳定性也非常有保障。但同样因为国内的特殊原因，Google App Engine 也被屏蔽，很难直接访问，这对普通用户来说也显得不实用。

（3）使用 Apache + PHP + MySQL：这个组合开源并且免费，能在任意操作系统上快速搭建完成，本地环境也容易配置，性能也完全能满足本系统的需求。并且国内的 Sina App Engine 也提供了在开发和测试阶段完全够用的免费使用时间，普通环境下的 PHP 项目不用做太多修改就能很好的运行在 SAE 之上。

综上所述，这里选择第三种方案，具体的本地环境为：

Apache 2.2.22 + PHP 5.3.13+ MySQL Server 5.5

2.3 编程语言及开发工具

2.3.1 C++语言

考虑到软件需要在 Windows XP 和 Windows7 都能正常工作，并且不是所有用户的计算机上都安装了.NetFramework 平台或者是 Java 运行时环境，所以直接排除使用 C#和 Java 等语言。以及之后可能移植到 MAC 或者 Linux 系统，还有程序的可扩展和灵活性，决定使用 C++语言进行开发，并且尽量使用标准模板库

（STL），而不是只有某个平台才支持的框架，比如 MFC。在 Windows 上也直接使用 Windows 提供的 API 进行开发，这样在移植到其它系统时只需要对不同的 API 进行修改即可。

2.3.2 Microsoft Visual Studio 2010 10.0.40219.1 SP1Rel

本论文主要讨论在 Windows 系统下的开发过程，使用的集成开发环境（IDE）为 Visual Studio 2010。它包含基于组件的开发工具（如 Visual C#、Visual J#、Visual Basic 和 Visual C++），以及许多用于简化基于小组的解决方案的设计、开发和部署的其他技术。

2.3.3 Notepad++ 6.1.5

除了 VS2010 外，开发过程中还使用 Notepad++做为代码和文件编辑器。

Notepad++是一套自由软件的纯文本编辑器，由台湾人侯今吾基于同是开放源代码的 Scintilla 文本编辑组件并独力研发。不仅有语法高亮度显示，也有语法折叠功能，并且支持宏以及扩充基本功能的外挂模块。

2.3.4 Google Chrome 23.0.1271.64

在控制端的前端开发中，使用 Chrome 浏览器来进行 CSS 样式与 Javascript 代码的调试。

2.4 其它辅助工具

2.4.1 Visual Assist X 10.7.1903.0 built 2012.04.03

为了加速应用程序的开发，开发过程中使用 VS 插件 VAX，它提供自动识别各种关键字，系统函数，成员变量，自动给出输入提示，自动更正大小写错误，自动标示错误等各种实用的功能。

2.4.2 VMware Workstation 8.0.1 build-528992

为了方便对多台主机进行控制的测试，使用安装了 Windows XP SP3 以及 Windows 7 的 VMware 虚拟机以辅助开发。

2.4.3 SVN 1.6.17 (r1128011)

本远程控制系统使用的版本控制软件为 Subversion，简称 SVN，是一个开放源代码的版本控制系统，相对于的 RCS、CVS，采用了分支管理系统，它的设计目标就是取代 CVS。并且这是位于 Google Code 的开源项目地址所使用的代码管理工具

2.4.4 GIT 1.7.5.1

Git 是一个分布式版本控制 / 软件配置管理软件，原来是 linux 内核开发者林纳斯·托瓦兹（Linus Torvalds）为了更好地管理 linux 内核开发而创立的。本系统位于 Github 的开源项目地址所使用的代码管理工具选用 Git。

第三章　设计

3.1 数据库的设计

在整个系统中，由于是使用 HTTP 服务器做为中介，也就是说命令以及执行结果的存储与转发都是由 HTTP 服务器完成，而这其中的存取过程是由数据库服务器来完成，所以说，数据库系统部分设计的好坏，直接关乎到整个系统的性能、重要功能是否能实现、以及程序编写的复杂度，开发周期等重要问题。若数据库部分没有明确设计结果，整个系统的开发则无从谈起。又或是设计有太多缺陷，以后的修改过程将是代价惨痛的，对程序员来说更是致命的。

3.1.1 要考虑的问题

作为一个功能较为完整的远程控制系统，数据库的设计部分需要考虑的问题有很多：

（1）命令的类型繁多，如屏幕控制的截图，键盘鼠标指令，文件管理，DOS 或 Shell 命令等。

（2）执行结果的类型也种类繁多，可能是图像，文本，树型结构的数据，或者是格式未知的二进制文件等，这些东西要如何在数据库中体现出来，是否应该为其分别建立字段，而这样的话数据库的设计又过分冗余。

（3）如何设计才能实现一个用户可以控制多台主机，一台主机也可以被多个用户控制。

（4）是否需要存储 shell 命令的历史命令表，以便用户能快速的回溯到之前执行过的命令。

3.2 整个系统的总体结构

在本系统中控制端与被控制端都是三层结构，如图 3.1 所示：

图 3.1 控制系统结构图

3.3 数据库的最终设计

在经过一系列测试后确定了最终数据库系统的设计方案，整个方案中决定回

归到最原始，最直接的设计方式，将复杂的命令类型，以及各种繁杂的命令执行结果的格式进行一个抽象，也就是说所有的命令执行结果都按相同的方式存储，并且提供统一的读取接口，这样能很好的避免各种未知数据格式的存储问题，同样也不用担心关系型数据库不适合用作树形结构存储等问题。

3.3.1 实体以及属性分析

图 3.2 给出了控制系统数据库的基本实体及必需属性。

图 3.2 数据库实体及属性

3.3.2 控制系统数据库 E-R 图

控制系统数据库 E-R 图

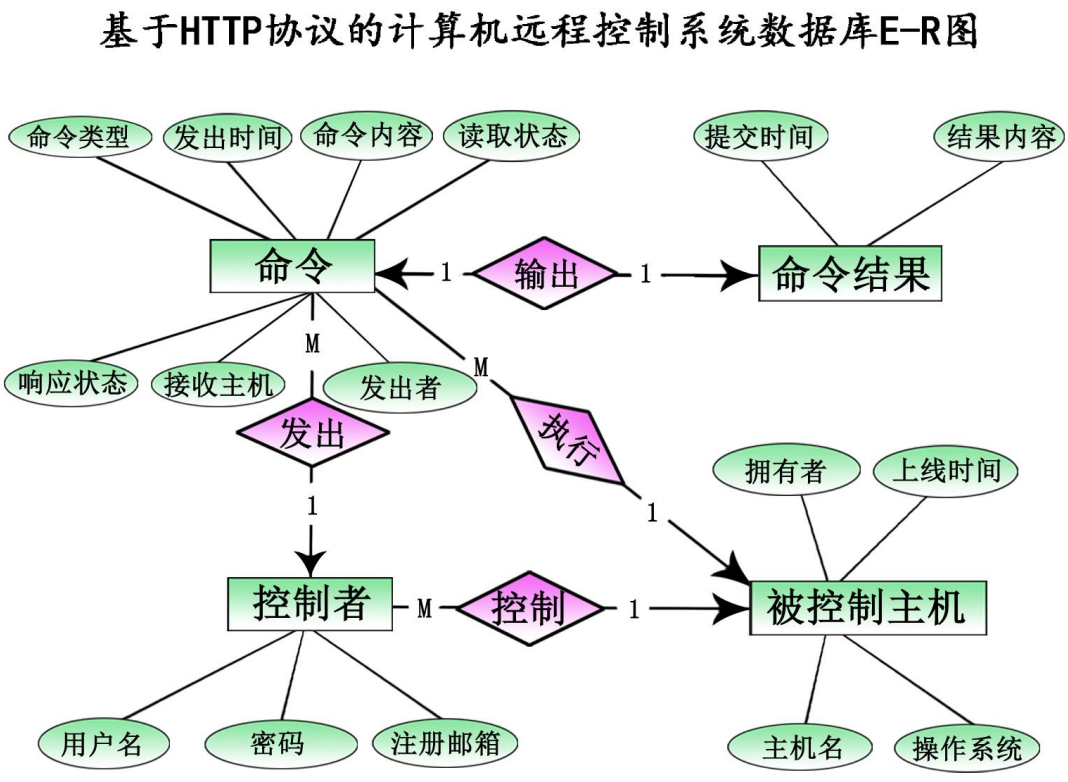

图 3.3 数据库 E-R 图

3.3.3 将 E-R 图转换为数据库表

数据库表关系图示。

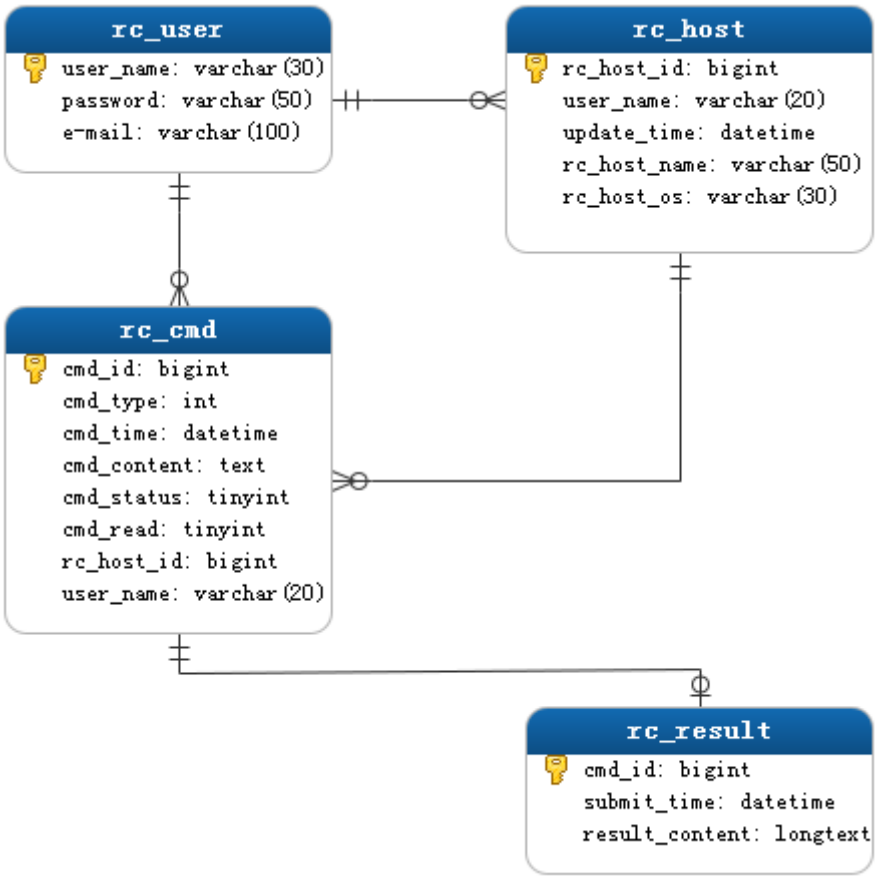

图 3.4 数据库表图示

3.3.4 应用三大范式对多张表进行审核，规范表结构

(1)表中每列都是不可再分的最小数据单元满足第一范式（1NF）

(2)除主键以外的其他列都完全依赖主键，满足第二范式（2NF）

(3)除了主键以外的其他列都不传递依赖于主键列，满足第三范式（3NF）

3.3.5 创建数据库的 SQL 语句以及解释

（1）创建并选择数据库，暂时忽略外键检查约束，以免创建出错。

```
Create Database remotecontrol;
use remotecontrol;
SET FOREIGN_KEY_CHECKS=0;
```

（2）创建用户表，主键为用户名，邮箱这里只用作找回帐号用，使用 PHP 检查是否是已经存在邮箱，所以这里没有将邮箱一同做为主键。使用 utf8 作为编码方式，对中文用户名进行支持。

```
-- ----------------------------
-- Table structure for `rc_user`
-- ----------------------------
DROP TABLE IF EXISTS `rc_user`;
```

```
CREATE TABLE `rc_user` (
  `user_name` varchar(30) NOT NULL,
  `password` varchar(50) NOT NULL,
  `e-mail` varchar(100) NOT NULL,
  PRIMARY KEY (`user_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

（3）创建主机表，以主机 ID 做为主键，无符号整型自动增长，用户名为外键，参考表（外键表）为上面的用户表，参考字段为 user_name（用户名），更新时间用来表示主机最后在线的活动时间，用于确定主机是否还在线，接下来是主机名和主机的操作系统类型，这由被控制端来完成获取。

```
-- ----------------------------
-- Table structure for `rc_host`
-- ----------------------------
DROP TABLE IF EXISTS `rc_host`;
CREATE TABLE `rc_host` (
  `rc_host_id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `user_name` varchar(20) NOT NULL,
  `update_time` datetime NOT NULL,
  `rc_host_name` varchar(50) DEFAULT NULL,
  `rc_host_os` varchar(30) DEFAULT NULL,
  PRIMARY KEY (`rc_host_id`),
  KEY `fk_user_name_h` (`user_name`),
  CONSTRAINT `fk_user_name_h` FOREIGN KEY (`user_name`) REFERENCES
`rc_user` (`user_name`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=0 DEFAULT CHARSET=utf8;
```

（4）创建命令表，命令 ID 为主键自动增长，cmd_type（命令类型）具体意义由程序员自定义，进行命令的分发处理。cmd_time（命令发出的时间），cmd_content（命令内容）具体含义也由程序员自定义。cmd_status（命令的执行状态），cmd_read（命令是否已经被取出），rc_host_id（主机标识号）来自主机表中的外键，用来表示此条命令是发送给哪个主机的，user_name（用户名）来自用户表中的外键，用来表示此条命令由哪个用户发出。

```
-- ----------------------------
-- Table structure for `rc_cmd`
-- ----------------------------
DROP TABLE IF EXISTS `rc_cmd`;
CREATE TABLE `rc_cmd` (
  `cmd_id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `cmd_type` int(11) NOT NULL,
  `cmd_time` datetime NOT NULL,
  `cmd_content` text NOT NULL,
  `cmd_status` tinyint(4) NOT NULL,
  `cmd_read` tinyint(4) NOT NULL,
```

```
  `rc_host_id` bigint(20) unsigned NOT NULL,
  `user_name` varchar(20) NOT NULL,
  PRIMARY KEY (`cmd_id`),
  KEY `fk_host_id_c` (`rc_host_id`),
  KEY `fk_user_name_c` (`user_name`),
  CONSTRAINT `fk_host_id_c` FOREIGN KEY (`rc_host_id`) REFERENCES
`rc_host` (`rc_host_id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `fk_user_name_c` FOREIGN KEY (`user_name`) REFERENCES
`rc_user` (`user_name`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=0 DEFAULT CHARSET=utf8;
```

（5）创建结果表，cmd_id（命令 ID）来自命令表的外键，标示这是哪条命令的执行结果，submit_time（结果提交的时间），result_content(命令结果)是 longtext 类型，因为决定将所有类型的命令结果都以 base64 密文来表示，这样无论是文本，二进制文件都能保存在其中，树型结构也可以以 xml 来描述，然后在控制端进行解密，解释还原，再呈现给用户即可，整个过程对用户来说是透明的，同时也可以编写一个统一的读写接口，而不用为不同的文件类型分别编写存取方法，大大方便了程序编写。

```
-- ----------------------------
-- Table structure for `rc_result`
-- ----------------------------
DROP TABLE IF EXISTS `rc_result`;
CREATE TABLE `rc_result` (
  `cmd_id` bigint(20) unsigned NOT NULL,
  `submit_time` datetime NOT NULL,
  `result_content` longtext NOT NULL,
  PRIMARY KEY (`cmd_id`),
  CONSTRAINT `fk_cmd_id_r` FOREIGN KEY (`cmd_id`) REFERENCES `rc_cmd`
(`cmd_id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

（6）在结果表中创建一个插入触发器，当结果表中插入数据时，将命令表中对应命令的执行状态置 1，表示这条命令已经完成执行。在实际使用的过程中，由于没有权限在 SAE 上创建触发器，所以在 SAE 上的系统这个功能由 PHP 来模拟完成。

```
DROP TRIGGER IF EXISTS `t_after_submit_result`;
DELIMITER ;;
CREATE TRIGGER `t_after_submit_result` AFTER INSERT ON `result` FOR
EACH ROW begin
UPDATE `cmd` SET `cmd_status`='1' WHERE (`cmd_id`=new.cmd_id);
end
;;
DELIMITER ;
```

3.4 远程控制系统数据流图

3.4.1 顶层数据流图:

控制系统顶层数据流图。

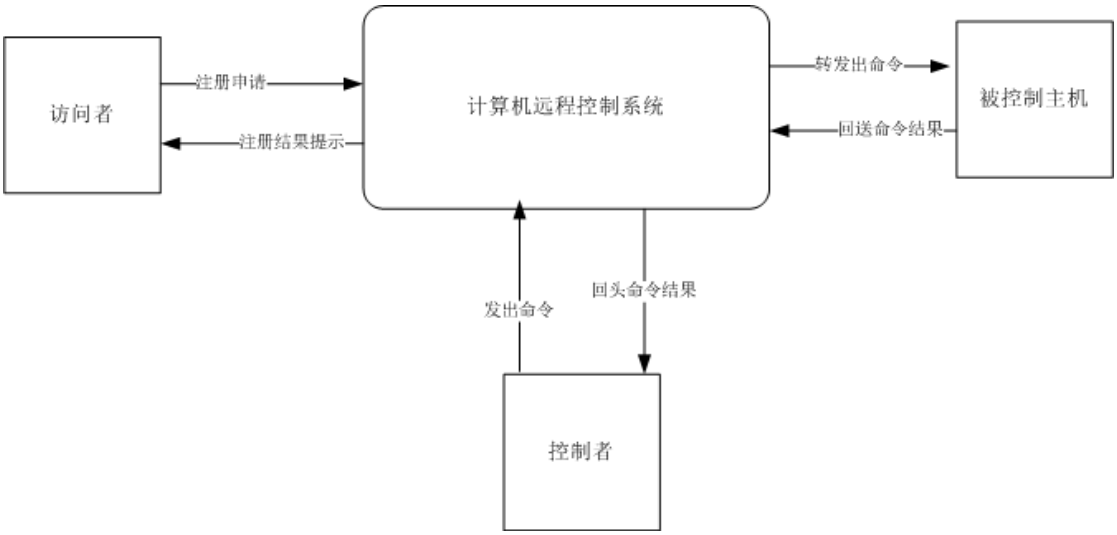

图 3.5 控制系统顶层数据流图

3.4.2 控制系统中层数据流图

控制系统中层数据流图，描述数据在各模块间的流动。

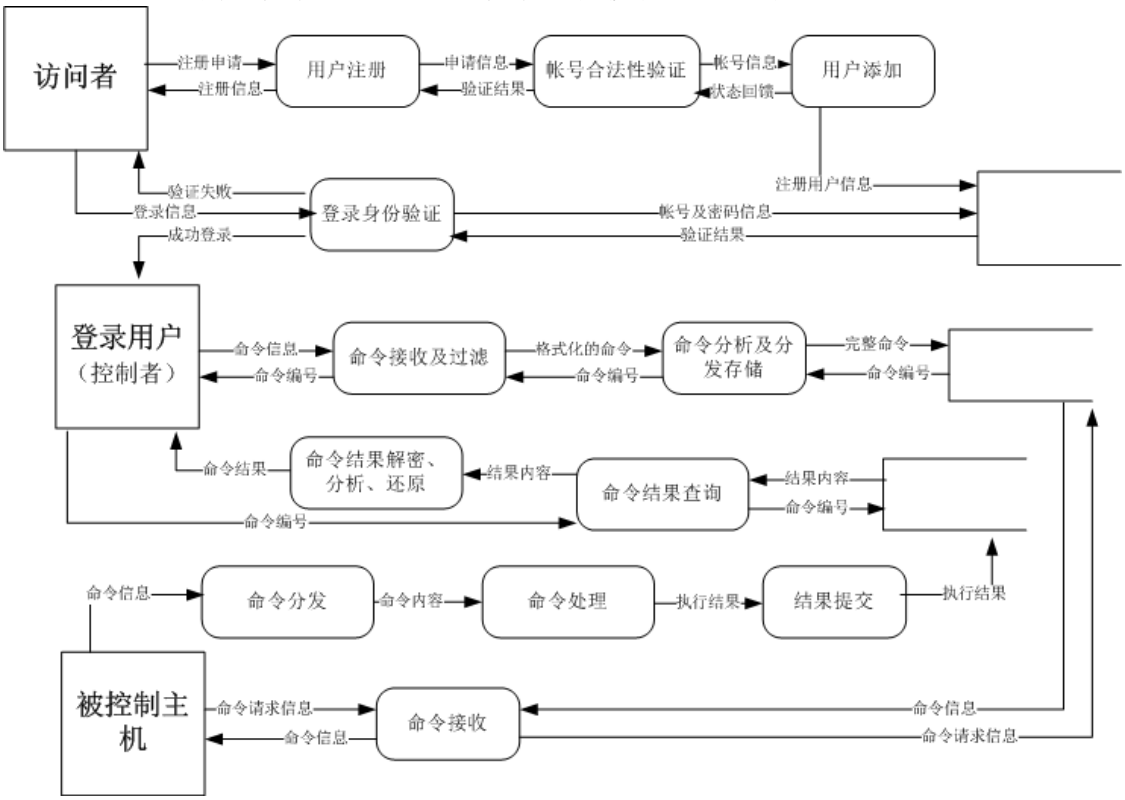

图 3.6 控制系统中层数据流图

第四章　基础模块的实现

4.1 被控制端 HTTP 通信类的实现

4.1.1 HTTP 通信类概述

在本系统中被控制端接受命令以及提交命令执行结果的过程都是一次次的 HTTP 请求，那么实现一个稳定强壮的 HTTP 通信类就非常有必要了，之所以不使用诸如 WinInet 系列的函数，是因为这些函数是 Windows 独有，为了方便程序能很快的移植到类 Unix 平台，同时也为了程序拥有更强的控制能力，这里基于基本的 socket 编程，实现了一个 HTTP 通信类，名称为 HttpMessenger。

经过作者长时间的测试和修改，已经能同诸如 IIS，nginx，Apache 等流行的 HTTP 服务软件进行稳定的通信，并且能轻松改写用于类 Unix 平台。

4.1.2 HttpMessenger 类关键成员函数及变量的介绍和使用

(1)构造函数：`HttpMessenger(char *szHostName="127.0.0.1",int nPort=80);`

这是HttpMessenger类的构造函数，共两个参数，第一个为要访问的服务器的地址，第二个参数为通过哪个端口进行访问，由于是用于HTTP通信，默认值为80端口。

(2)错误查询函数：`string HttpMessenger::GetErrorLog();`

此函数用于在某个步骤失败时查询具体的出错原因和出错位置，以便于及时发现错误并修正，或用于显示提示。

(3)创建并发送HTTP请求：

```
bool CreateAndSendRequest(char *RequestType,
        char *ResourcePath,char *Host=NULL,char *PostData=NULL,
        bool OnlyGetLength=false,char *Save2File=NULL);
```

此函数根据输入的参数创建HTTP请求头，并自动向服务器发送请求和接收服务器的响应，共有6个参数。

RequestType为向服务器发送的HTTP请求的类型，通常有GET和POST等。

ResourcePath为请求的资源的路径。

Host为提交请求的服务器地址。

PostData，这个参数在使用POST类型的请求时使用，是向服务器发送的数据，如果是GET请求，请将此参数传为NULL。

OnlyGetLength，这个参数为true时，仅仅只获取到服务器返回的资源长度后，函数就马上返回，并不会接收服务器传回的数据。

Save2File，此参数不为NULL时，会把传入的字符串当做一个文件名，或者是文件路径，来创建一个文件，在接收数据的过程中，把接收到的数据写入文件当中，可以用于较大文件下载。

返回值：如果请求成功则返回 true，连接失败则返回 false,此时可以调用 GetErrorLog()函数获取具体失败的原因。

(4)获取HTTP响应的状态：**int GetResponseState();**

此函数用于查看服务器返回的HTTP响应状态，常见的返回值有200：请求成功，302：资源被移动，404：请求的资源未找到等等，更多内容请参考HTTP协议标准。

(5)初始化套接字函数：**static bool InitialSocket();**

此静态方法需要在程序的初始的地方调用一次，用于初始化套字，否则类不能正常工作。

(6)HTTP请求头的内容：**string m_RequestHeader;**

访问此成员变量可以得到所生成的HTTP请求头。

(7)HTTP响应头的内容：**string m_ResponseHeader;**

访问此成员变量可以得到服务器返回的HTTP响应头。

(8)HTTP 响应的全部内容：**string m_ResponseText;**

访问此成员就是可以得到服务器返回的 HTTP 响应内容，如果已经指定在函数 CreateAndSendRequest 中将请求内容保存为文件，此成员内容为空，请读取文件进行获取。

(9)向服务器发起连接请求：**bool CreateConnection();**

此函数用于向服务器发起连接请求，如果连接成功则返回 true，连接失败则返回 false,此时可以调用 GetErrorLog()函数获取具体失败的原因。

4.1.3 HttpMessenger 使用示例

(1)在程序开始处，使用 HttpMessenger::InitialSocket();加载套接字，此操作只需要做一次。

(2)创建 HttpMessenger 对象，例如：

```
HttpMessenger *hm_GetTotalXML=new HttpMessenger(server_addr);
```

(3)向服务器发起连接，例如：

```
if (!hm_GetTotalXML->CreateConnection())
{
    Msg("检测更新时，连接服务器失败，错误日志:%s\r\n",
    (char *)hm_GetTotalXML->GetErrorLog().c_str());
       delete[] hm_GetTotalXML;
       return;
}
```

(4)连接成功后，创建 HTTP 请求头，并向服务器发送请求，例如：

```
if (!hm_GetTotalXML->CreateAndSendRequest("GET",
          "/mov/xml/Total.xml",server_addr,NULL,true))
    {
```

```
        Msg("检测更新时，发送请求失败，错误日志:\r\n%s",
            hm_GetTotalXML->GetErrorLog().c_str());
        delete[] hm_GetTotalXML;
        return;
    }
```

(5)整个请求过程完成，根据需要来处理得到的响应内容。

4.2 控制命令与执行结果的传输过程

4.2.1 并发的命令执行

在控制端的运行过程中，为了能并发的执行命令，采用多线程机制。当登录完成后创建命令接收线程接收命令，即 ReceiveCommand 线程函数，当有新的命令到达时调用 AnalyzeCommand 函数对命令进行解析。

对于命令具体格式的表示，决定采用类似于 XML 的表示方式，形如：

<cmd_id>1</cmd_id>

<cmd_type>0</cmd_type>

<cmd_time>2012-03-21 14:42:35</cmd_time>

<cmd_content>ipconfig /all</cmd_content>

命令分析的过程就是解释上述格式的指令，并生成一个 cmdinfo 类型的结构体，当分析完接收到的命令后发出 WM_CMDRCVD_MESSAGE 消息，并把已经填充好的 cmdinfo 类型的结构体做为参数传送，此时消息环接收到消息，按命令类型进行分发，创建相应的线程对命令内容进行处理，最后提交命令执行结果，并释放 cmdinfo 类型的结构体，如图 4.1 所示:

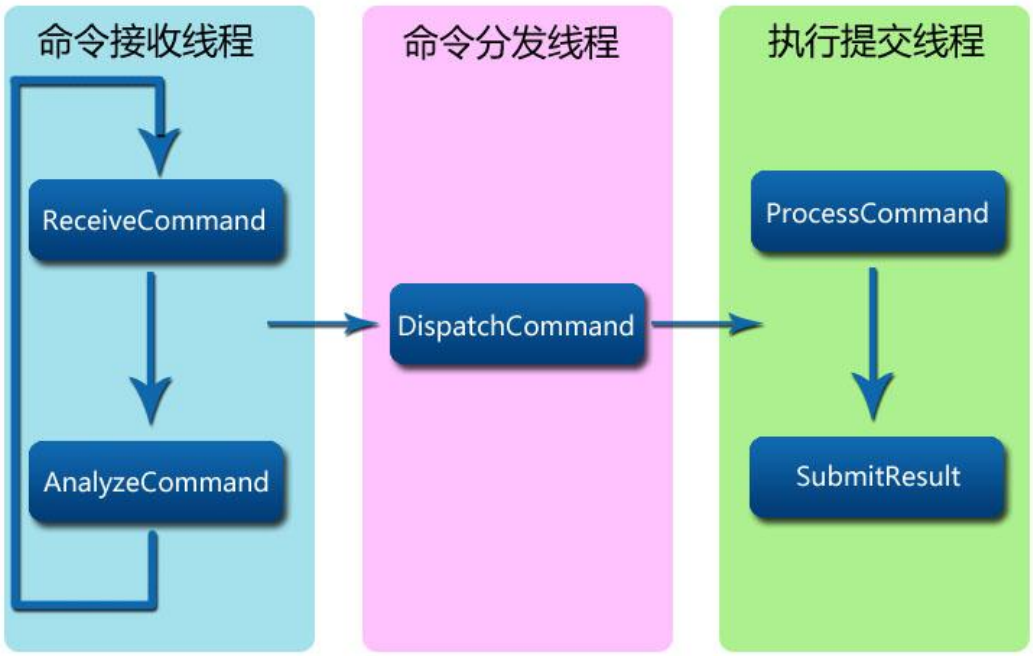

图 4.1 被控制端命令处理流程

4.2.2 命令与命令执行结果的具体获取方式

由于 HTTP 是无状态单向的协议，也就是客户端需要向服务器发送请求,服务器处理请求并传送所请求的数据，也就是说被控制端需要不停的向服务器发送请求，而且为了命令较实时的响应，每次请求的间隔还不能够太长，但命令不是时时都有，就会产生大量的无用请求，即增加了服务器负担，同时也浪费网络资源。为了缓解这个问题，实现在线聊天和服务器推送等功能，出现了相对于这种“短轮询”(polling)方式的“长轮询”(long polling)的方式，Web 开发中也称为 Comet 或者是反向 Ajax 技术。在本系统中被控制端接收命令的方式就是模仿这种 Web 技术来实现。

具体过程是，被控制端向 HTTP 服务器发起一个 HTTP 请求，在服务器收到这个请求后，并不马上对这个请求进行响应，而是进入一个等待命令的循环中，此时被控制端的连接就会一直阻塞，而服务器端可以一直等待有控制命令到达，才对这个请求进行响应，此时客户端也从阻塞中恢复并收到响应的命令进行处理，然后被控制端再重新发起命令请求。当然，服务器端的脚本应该有个最大执行时间，以避免被控制端已经下线还在继续循环等待命令，造成服务器资源浪费。超过最大执行时间后返回超时信息，客户端收到超时信息，释放连接并重新建立请求。在本系统中服务器端超时时间为 1 分钟，这是依据 PHP 和 SAE 的默认脚本运行超时时间而决定的，使用者同样可以自行定制。

另外，在控制端发出命令并等待命令执行结果的过程中也是使用的长轮询，虽然使用长轮询能大大减少了无效的 HTTP 请求数据，并且降低了命令服务器的被动性,但还不是最完善的解决方法,新兴的 WebSockets 技术在 HTML5 中出现，是比长轮询更加优秀的反向 Ajax 技术。WebSockets 支持双向、全双工通信信道，但是目前浏览器对其支持尚不完善，另外还有 WebRTC(Web Real-Time Communication)等支持浏览器进行实时语音对话或视频对话的软件架构,也是由于尚不完善，便没有在目前版本的控制系统中使用。

4.2.3 服务器端命令和执行结果的等待方式

使用长轮询虽然可以大大减少被控制端的无效请求，但是服务器在循环等待命令的过程中实际上又是不断对数据库进行轮询操作，以查询是否有新命令。所以长轮询并不能减轻对数据库服务器的压力,在只有少量被控制端时对服务器还不会形成太大压力，但当被控制端较多时数据库的并发查询数瞬间就会被占满，导致系统不稳定。

为了缓解数据库压力，在系统中使用了 memcache，memcache 是一套分布式的高速缓存系统，以键值对的方式来存取数据，并且 XAMP 环境中容易安装，SAE 也支持。memcache 通常的使用情景为，将需要经常使用的数据从数据库中取出

放入 memcache，也就是内存中，当下次再需要这个数据时就直接从内存中取出，而不用再去查询数据，这样来达到减轻数据库压力的目的。可在远程控制系统中命令与命令结果都是需要实时获取，且每次相同命令的执行结果也可能是不一样的，就不可能以这种普通的方式来使用 memcache。在本系统中 memcache 实际上是充当了信使的作用，用来通知是否有新命令到达。

细节是这样的，首先在被控制主机登录到服务器时，服务器会以这台主机的 ID 加上登录者用户名为键的名称来创建一个 memcache 键，该键的初始值为 0，表示这台主机并没有收到命令，那么在被控制端请求命令时，服务器只需要检查这个被控制端所对应 memcache 键的值是否为 0，就可以知道这个控制端有没有命令了，而不必去查询数据库。当控制者在前端发出命令，命令以 ajax 方便提交到后端时，命令会添加到数据库，并且把命令的目标主机对应的 memcache 键的值执行加一操作，表示有一条新命令加入。此时获取命令的脚本会发现主机对应的键值不为 0 了，才去数据库进行查询取出命令，成功取出命令后对键值进行减一操作，并且向被控制端发送命令，这样数据库查询的命中率就几乎高达 100%了，极有效节省了数据库连接资源。

命令执行结果的等待过程原理也是相同的，区别在于，执行结果的 memcache 键是以命令 ID 为键名的，因为命令 ID 具有唯一性，并且一个命令对应的键在前端成功获取到结果后是会被立即删除的，以节省 memcache 资源。

4.2.4 添加命令的关键 PHP 代码

完整代码位于./function/AddCommand.php

```
$add_command="insert into `rc_cmd`
(`cmd_type`,`cmd_time`,`cmd_content`,`cmd_status`,`cmd_read`,`rc_host
_id`,`user_name`) values
('".$cmd_type."','".$cmd_time."','".$cmd_content."','0','0','".$rc_ho
st_id."','".$user_name."');";

$get_cmd_id="SELECT LAST_INSERT_ID()";

//以用户名加主机ID为构建memcache键名
$mckey=$user_name."-".$rc_host_id;
//将命令插入数据库
$queryresult = mysql_query($add_command,$database);
if($queryresult!=false)
{
    //获取刚插入命令的命令ID
    $cmd_id=mysql_insert_id();
    echo $cmd_id; //返回命令ID
    incrMC($memcache,$mckey);    //通知有新命令被加入
```

```
    $memcache->set($cmd_id,0);//以命令ID创建一个Memcache键用于等待命令结果
    $memcache->close();
}
else
{
    die('Query Error A2:'.mysql_error());
}
mysql_close($database);
```

4.2.5 命令获取的 PHP 代码

完整代码位于./function/GetCommand.php

```
//以用户名加主机ID为构建memcache键名
$mckey=$user_name."-".$rc_host_id;
//循环访问memcache，若有命令则返回，没则睡眠200毫秒
for($i=1;$i<290;$i++)
{
    if(memcache_get($memcache,$mckey)!="0")
    {
        //有新命令被添加，连接数据库并取出命令
    require("ConnectSAEDB.php");
    $sql_get_cmd="select `cmd_id`,`cmd_type`,`cmd_time`,`cmd_content`
from `rc_cmd` where (`user_name`='".$user_name."' and
`rc_host_id`='".$rc_host_id."' and `cmd_status`='0' and
`cmd_read`='0');";
    $queryresult = mysql_query($sql_get_cmd,$database);
    if($queryresult!=false)
    {
        if(mysql_num_rows($queryresult)!=0)
        {
            $cmdarray=mysql_fetch_assoc($queryresult);
            $mark_as_read="update `rc_cmd` set `cmd_read`='1' where
(`cmd_id`='".$cmdarray['cmd_id']."');";
            mysql_query($mark_as_read,$database);
            decrMC($memcache,$mckey);
            mysql_close($database);
            //返回一条命令
            ////////////////////以下部分用于LAMP/WAMP环境////////////////
            echo "<cmd_id>".$cmdarray['cmd_id'].
                    "</cmd_id><cmd_type>".$cmdarray['cmd_type'].
                    "</cmd_type><cmd_time>".$cmdarray['cmd_time'].

    "</cmd_time><cmd_content>".stripcslashes($cmdarray['cmd_content']
)."</cmd_content>";

            ////////////////////以下部分用于SAE环境////////////////
```

```
            //  echo "<cmd_id>".$cmdarray['cmd_id'].
            //      "</cmd_id><cmd_type>".$cmdarray['cmd_type'].
            //      "</cmd_type><cmd_time>".$cmdarray['cmd_time'].
            //
    "</cmd_time><cmd_content>".$cmdarray['cmd_content']."</cmd_conten
t>";
            }
            else
            {
                echo "False\r\n\r\n";
            }
        }
        else
        {
            echo "Query False G1:".mysql_error();
        }
        $memcache->close();
        exit(0);
    }
    usleep(200000);
}
$memcache->close();
```

4.3 被控制端与控制端的关联

在本系统中控制端与被控制端使用用户名加主机 ID 作为关联凭据，用户在被控制端登录时，会为当前用户名分配一个唯一的主机 ID。此用户在控制端登录时会查询此用户下所有主机的记录，让用户选择进行控制。这里需要考虑的是主机 ID 与用户名在被控制端，控制前端，控制后端要如何传递。

4.3.1 关联凭据的具体传递方式

主要是出于安全与效率方面的因素考虑，最终实现时全部采用 session 来传递这些关联凭据。对于控制端来说，如果是采用 POST 或 GET 参数来传递，或者是使用 cookie 来传递，这些信息都是用户能够修改的，不具有可信性，要保证被控制主机不被越权控制，需要进行非常多的验证，甚至是每条命令都必须验证当前使用者是否有权向该主机发送命令，这也就降低了整个系统的效率。对于被控制端来说，使用参数或 cookie 来传递，就必须告诉被控制端，当前被控主机在数据库中的 ID 编号，用户可以修改这个编号来仿冒其它被控制端，或者是通过枚举的方式，截获其它用户的控制命令，虽然对整个系统没多大害处，但会对其它用户造成影响，同时对用户也不够透明。

而采用 session 时，关联凭据是保存在服务器上的，用户不能修改，对控制端来说，只需要在登录和选择主机时对控制者进行身份验证即可，对被控制端来

说，只传递一个 session ID，用户并不知道这台主机在数据库中的编号，而且想影响其它用户就必须构造一个正确的 session 碰撞，这似乎是不可能的。另外，在源代码中保留的使用 POST 参数和 cookie 的代码是为了开发时避免反复输入用户名而使用的。

4.3.2 被控制端登录时，服务器处理的 PHP 代码

完整代码位于,./function/AddUserHost.php

```
session_start();
require("ConnectSAEDB.php");
require("ConnectMC.php");
$user_name = strtolower(addslashes($_POST['user_name']));
$rc_host_name= addslashes($_POST['rc_host_name']);
$rc_host_os = addslashes($_POST['rc_host_os']);
date_default_timezone_set('PRC');
$now = getdate();
$update_time=$now[year]."-".$now[mon]."-".$now[mday]."
".$now[hours].":".$now[minutes].":".$now[seconds];
//增加新的主机
$add_host="INSERT INTO `rc_host`
(`user_name`,`update_time`,`rc_host_name`,`rc_host_os`) VALUES
('".$user_name."', '".$update_time."', '".$rc_host_name."',
'".$rc_host_os."');";
//获取已经存在主机的ID
$get_host_id="select `rc_host_id` from `rc_host` where
(`user_name`='".$user_name."' and `update_time`='".$update_time."' and
`rc_host_name`='".$rc_host_name."' and
`rc_host_os`='".$rc_host_os."');";
//判断主机是否已经存在
$exist_host="select `rc_host_id` from `rc_host` where
(`user_name`='".$user_name."' and `rc_host_name`='".$rc_host_name."'
and `rc_host_os`='".$rc_host_os."');";
//更新主机的上线时间
$update_exist_host="UPDATE `rc_host` SET
`update_time`='".$update_time."' WHERE (`user_name`='".$user_name."'
and `rc_host_name`='".$rc_host_name."' and
`rc_host_os`='".$rc_host_os."');";
//检查用户名是否存在
$check_user_name="select * from `rc_user`
where(`user_name`='".$user_name."');";
$exist_user=mysql_query($check_user_name,$database);
if($exist_user)
{
    if(mysql_num_rows($exist_user)==0)
    {
```

```
    echo $user_name."并不是已注册用户.";
    session_destroy();
    exit;       }
}
else
{
    die('Query Error:'.mysql_error());
}
    $exist_host_id = mysql_query($exist_host,$database);
    if($exist_host_id)
    {
        if(mysql_num_rows($exist_host_id)!=0)
        {
            $host_id=mysql_fetch_assoc($exist_host_id);

            if(mysql_query($update_exist_host,$database))
            {
                $_SESSION["rc_host_id"]=$host_id[rc_host_id];
                $_SESSION["user_name"]=$user_name;
                echo "OK";
            }
            else
            {
                echo "Update Error or Get Host ID Error!";
            }
        }
        else
        {
            $addresult = mysql_query($add_host,$database);
            if($addresult)
            {
                $new_host_id = mysql_query($get_host_id,$database);
                if($new_host_id)
                {
                    if(mysql_num_rows($new_host_id)!=0)
                    {
                        $host_id=mysql_fetch_assoc($new_host_id);
                    //将主机ID存入session
                    $_SESSION["rc_host_id"]=$host_id[rc_host_id];
                        $_SESSION["user_name"]=$user_name;
                        echo "OK";
                    }
                    else
                    {
```

```
                        echo "Add Error or Get Host ID Error!";
                    }
                }
                else
                {
                    die('Query Error:'.mysql_error());
                }
            }
            else
            {
                die('Query Error:'.mysql_error());
            }
        }
    }
    else
    {
        die('Query Error:'.mysql_error());
    }
//以"用户名-主机ID"为key，创建Memcache以进行命令通知,可加入salt防止仿冒MCkey
$mckey=$user_name."-".$host_id[rc_host_id];
$memcache->set($mckey,0);
$memcache->close();
$mark_as_read="update `rc_cmd` set `cmd_read`='1' where
(`rc_host_id`='".$host_id[rc_host_id]."' and
`user_name`='".$user_name."' and `cmd_read`='0');";
mysql_query($mark_as_read,$database);
mysql_close($database);
```

第五章　各个功能模块的实现

5.1 被控制端界面的实现

由于被控制端在登录的过程中，只是需要向服务器声明当前主机属于哪个用户即可，是一个归属过程，而不是身份认证过程，所以无需输入密码，只输入用户名即可重新登录。为了方便程序调试还创建一个文本框用于显示信息。

图 5.1 被控制端界面

5.2 控制端的各种功能模块

5.2.1 用户登录及用户注册界面

（1）用户登录及注册界面

图 5.2 控制端用户登录及注册界面

5.2.2 被控制主机选择界面

选择被控制主机进行控制的选择界面。

图 5.3 被控制主机选择界面

5.2.3 控制主机的虚拟桌面环境

控制端的控制界面。

图 5.4 控制端虚拟桌面

5.2.4 命令行及摄像头监控

使用命令行执行命令，以及摄像头查看。

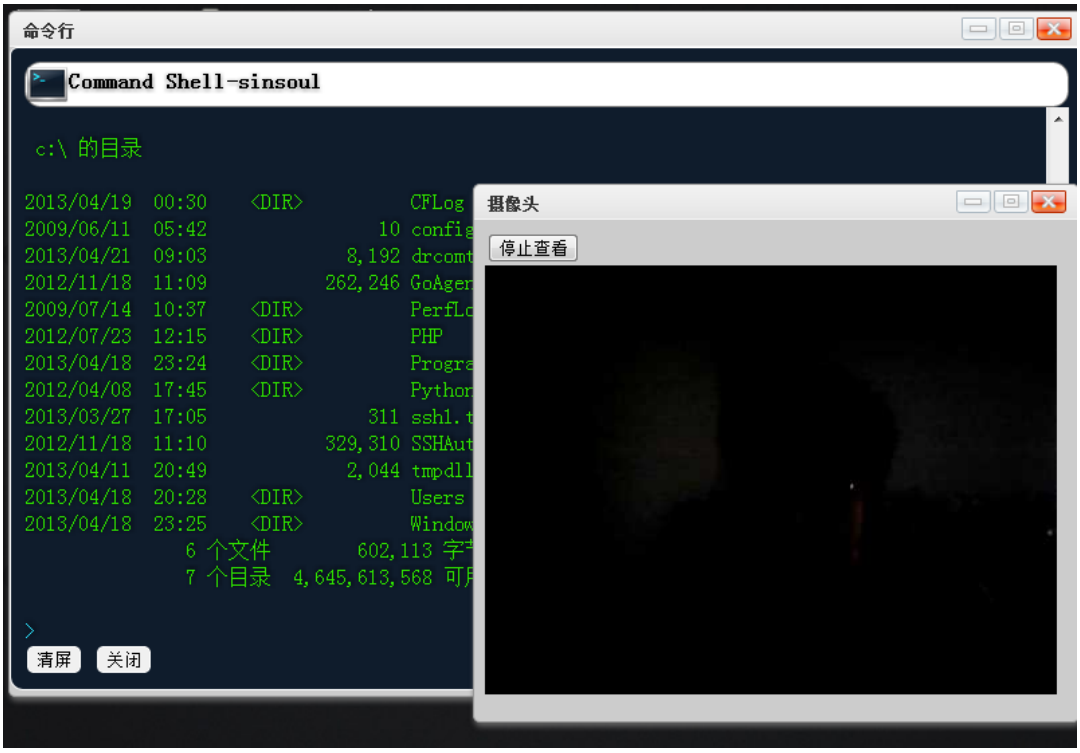

图 5.5 命令行及摄像头查看界面

5.2.5 文件管理功能

文件管理功能包括左边树状目录浏览，及基本复制、粘贴、重命名及删除操作等。

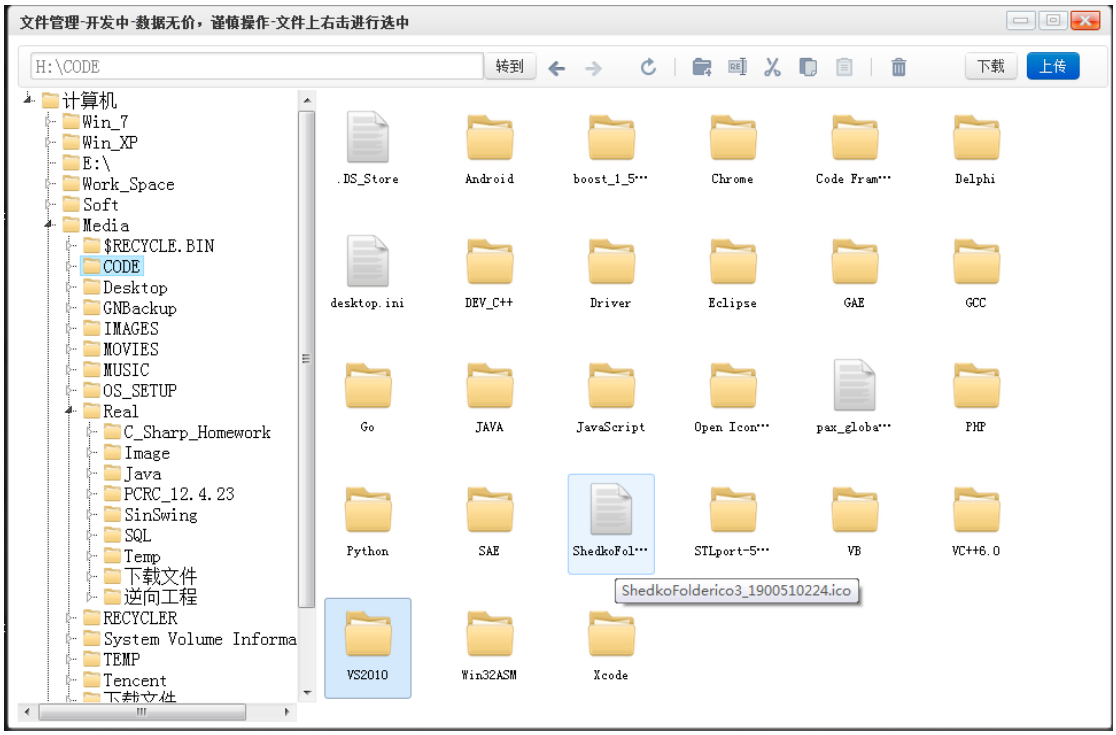

图 5.6 文件管理界面

第六章　　测试及总结

6.1 测试

6.1.1 局域网内测试 WAMP 环境

在修复完已知的所有BUG后，进行局域网内测试。测试效果理想，被控制端在Windows XP，Windows 7等操作系统中都能稳定正常的长时间工作，软件工作效率满足实际环境使用需求，但屏幕控制部分效率较低，还有改进空间。

6.1.2 广域网环境 SAE 平台测试

在确定本系统在局域网环境中能够稳定工作后，将控制系统在SAE平台上线，并邀请广域网中的网友进行测试，控制软件同样能够稳定的工作。发现的问题是被控制端在在线几十个小时后，再请求命令时，SAE平台会返回一个跳转页面，以验证发来请求的客户端是否是浏览器，在后续的版本中会增加对这种情况的处理。

6.1.3 控制端在移动设备上的测试

在手机及平板电脑上，使用自带浏览器，及Chrome浏览器进行测试，控制端都能正常地进行控制，但应该针对移动平台进行优化，并且适配更多不同的浏览器。

6.2 关于开源

6.2.1 开源的原因

目前已经实现了对 Windows 平台的基本控制功能，但还只是基本的框架，或者说只是一个雏形，与最终实现一个跨平台的远程控制系统还有很大的差距，再加之本系统所涉及的技术众多，包括了各种操作系统中应用软件的编写，Web 应用的前端呈现和后端处理，而要让本系统更精致和实用，还是需要专注某一方面的程序员分工合作，所以为了更快的让系统在实际环境中被使用，让更多的开发者加入一起完善。

6.2.2 开源的方式

（1）位于 Google Code 的开源项目地址

项目地址：

https://code.google.com/p/pc-remote-control/

项目源代码在线查看：

https://code.google.com/p/pc-remote-control/source/browse/

完整源代码检出：

```
# Non-members may check out a read-only working copy anonymously over HTTP.
svn checkout http://pc-remote-control.googlecode.com/svn/trunk/
pc-remote-control-read-only
```

（2）位于 Github 的开源项目地址

项目地址：

https://github.com/sinsoul/SinSoulWebRemoteControl

项目源代码在线查看

https://github.com/sinsoul/SinSoulWebRemoteControl/tree/master/

完整源代码检出:

git clone git@github.com:sinsoul/SinSoulWebRemoteControl.git SSWRC

6.3 总结

虽然此系统只是我写过的程序中很普通的一个，但它所涉及的需要了解的知识却是最多的，在开发的过程中也遇到的很多困难，但最终都较好的解决了，对我自身也是一个很大的提高。

参考文献：

[1]Andrew Curioso，Ronald Bradford. Expert PHP and MySQL[M]. US:Wiley Publishing,inc.,2010

[2]吴翰清.白帽子讲 Web 安全[M].北京：电子工业出版社.2012

致谢

在程序编写的过程中，参考了很多优秀程序员的文章和代码，以及非常优秀的设计方式，这将使我终身受益。

转载说明