# 概述

监控服务有三个个组件：监控客户端和服务器端、集成到航显控制台的监控页面。监控客户端和服务器端采用wpf, 监控页面使用asp.net。航显监控客户端和服务器端采用C/S架构。客户端部署在工控机上，每台工控机部署一个客户端。服务器端只有一个，部署在服务器上。

客户端和服务器端通过tcp协议进行通信。服务器端扫描数据库获取到部署到工控机上的客户端的ip和端口。服务器端连接到客户端，通过接口获取客户端工控机的运行情况。服务器缓存最新的监控结果并将其保存到数据库中。 监控页面是asp.net的web页面，负责展示监控结果。监控页面连接到服务器端，并定时查询航显设备的运行情况。当设备异常时，航显监控页面通过声音等方式进行报警。

目前航显屏有两种：有工控机和无工控机。有工控机的航显屏上安装监控客户端，无工控机的航显屏的监控，直接由服务器通过航显屏提供的sdk进行监控。

# 设计

客户端部署在工控机上，工控机连接航显屏，控制航显屏的显示。客户端的ip和端口保存在数据库中。服务器端扫描数据库获取所有客户端的ip和端口号。

目前客户端和服务器端的通信采用UTF-8格式编码，并支持长连接。客户端和服务器端每10s进行一次监控数据传输，传输时间间隔配置。由于客户端和服务器端数据传输的时间间隔较短，因此无需心跳测试。

客户端检测打开的连接，如果连接3分钟内没有数据传输，客户端有权将其关闭，并无需通知服务器端。该时间间隔也可配置。

服务器端主动连接客户端，并负责保持连接。服务器端有两种关闭连接的方式：1）服务器端主动关闭与客户端连接；2）服务器端认为连接已从客户端关闭，从而被迫关闭与客户端连接。第1种情况通常发生在用户主动关闭服务器时或者用户认为没必要监控某个客户端时。第2种情况发生时，服务器端需要关闭当前与客户端连接，并重建与客户端连接。重建最多重试2次，并且在连续1分钟内均无法连接则认为该客户端失联，发出级别较高的警告。

## 服务器-🡪客户端数据协议

<?xml version="1.0"?>

<Request >

<RequestId>ccd0d5a7-b824-40ab-9d8d-e14e4f5ad10f</RequestId>

<RequestTime>2014-12-29T14:13:02.5591449+08:00</RequestTime>

<RequestContent>0</RequestContent>

</Request>\r\n\r\n\r\n##

各字段含义如下：

|  |  |
| --- | --- |
| 字段名 | 字段含义 |
| RequestId | 表示请求的id，使用GUID来标识一次请求 |
| RequestTime | 表示请求的时间 |
| RequestContent | 表示请求的内容,<= 0:不请求任何内容，可用于心跳测试；1:请求cpu；2：请求内存；4：请求硬盘数据；如果请求多组数据，可使用组合，如3表示请求cpu和内存情况，8重启，16关闭 |

由于使用TCP协议通信，存在粘包问题，可考虑使用\r\n\r\n\r\n##作为数据的边界。（或者考虑使用SuperSocket，从而忽略内部tcp协议）。

默认客户端接收的每个数据包不超过32k，支持的同时连接不超过1024。

注：从我们的数据来看远小于32k，64k大约可存储1w-3w个字符。有时黑客会发送大量无效的数据包，从而导致内存不足，系统崩溃。

对应的类

public class Request

{

public Guid RequestId { get; set; }

public DateTime RequestTime { get; set; }

/\*\*

\* 0：nothing

\* 1: cpu

\* 2：内存

\* 3：cpu + 内存

\* 4：硬盘数据

\*/

public int RequestContent { get; set; }

}

客户端—>服务器端数据协议

<?xml version="1.0"?>

<Response>

<IsValid>false</IsValid>

<ResponseCode>0</ResponseCode>

<MemoryInfo>

<UsedMemory>0</UsedMemory>

<AllMemory>0</AllMemory>

</MemoryInfo>

<DriverInfo>

<DriverStatus>

<DriverName>C</DriverName>

<UsedSpace>0</UsedSpace>

<AllSpace>0</AllSpace>

</DriverStatus>

</DriverInfo>

<CpuInfo>

<CpuCore>0</CpuCore>

<UsedPercent>0.28</UsedPercent>

</CpuInfo>

</Response>\r\n\r\n\r\n

由于使用TCP协议通信，存在粘包问题，可考虑使用\r\n\r\n\r\n作为数据的边界。（或者考虑使用SuperSocket，从而忽略内部tcp协议）

各字段含义

|  |  |
| --- | --- |
| 字段名 | 字段含义 |
| IsValid | 必填字段，标志请求是否是有效请求。 |
| ResponseCode | 必填字段，0，处理请求失败；1，表示成功处理监控请求，生成有效响应；2，表示接收到重启命令；4，表示接收到关闭命令。 |
| MemoryInfo | 可选，有两个字段：UsedMemory：表示使用的内存，AllMemory：表示所有的内存 |
| DriverInfo | 可选，一个DriverStatus的列表，每一项有DriverName\UsedSpace\AllSpace组成 |
| CpuInfo | 可选，有两个字段：CpuCore：表示cpu核数，UsedPercent表示使用的比例。 |

注：IsValid是检测请求是否有效的。端口扫描器或者黑客会发送嗅探数据，如果客户端能够确认是非法请求，客户端有权在不发送响应的情况下，关闭连接。从而避免拒绝服务攻击。

如果客户端接收到的命令是重启或者关闭命令，需要立刻返回Response数据包，并重启或者关闭。如果是重启或者关闭，则将设备状态置为重启或者关闭，轮询设备状态，但不报警。

对应的类

public class Response

{

/// <summary>

/// 请求是否有效处理

/// </summary>

public Boolean IsValid { get; set; }

/// <summary>

/// 返回处理结果码

/// 0：处理成功

/// 1：无效请求

/// </summary>

public int ResponseCode { get; set; }

/// <summary>

/// 内存状况

/// </summary>

public MemoryStatus MemoryInfo { get; set; }

/// <summary>

/// 硬盘情况

/// </summary>

public List<DriverStatus> DriverInfo { get; set; }

public CpuStatus CpuInfo { get; set; }

}

public class MemoryStatus

{

/// <summary>

/// 使用的内存

/// </summary>

public long UsedMemory { get; set; }

/// <summary>

/// 所有的内存

/// </summary>

public long AllMemory { get; set; }

}

/// <summary>

/// 硬盘状况

/// </summary>

public class DriverStatus

{

/// <summary>

///

/// </summary>

public String DriverName { get; set; }

public long UsedMomery { get; set; }

public long AllMemory { get; set; }

}

/// <summary>

/// cpu状况

/// </summary>

public class CpuStatus

{

/// <summary>

/// CPU核数

/// </summary>

public int CpuCore;

public double UsedPercent;

}

## 服务器与页面监控

服务器端从众多客户端获取到监控信息会缓存最新的一份，这是需要传递给监控页面的信息。服务器端和监控页面使用tcp协议进行通信。从监控页面到服务器传递的信息的长度是可预测的，并且长度有限。从服务器到监控页面的数据量是变长的，长度取决于客户端设备的数量。

监控页面是一个asp.net程序，只有在监控页面打开时，才与服务器端进行通讯。默认通讯的时间间隔是10s.该时间间隔可配置。

服务器端检测打开的连接，如果连接3分钟内没有数据传输，服务器有权将其关闭，并无需通知监控页面。该时间间隔可配置。

监控页面定时轮询服务器，当首次请求时，创建一个job，并定时与服务器通信，将获取的结果缓存。在页面轮询停止后5分钟内，job自动停止。

从监控页面关闭或者重启客户端设备，需要经过服务器，而不是直接连接客户端设备。这种设计主要是统一管理。

## 监控页面—>服务器接口

<?xml version="1.0"?>

<WebRequest>

<RequestId>2d4f56d9-8d0c-4b67-b6de-e6243a5625b7</RequestId>

<RequestTime>2014-12-31T14:48:15.1926968+08:00</RequestTime>

<RequestType>ALL</RequestType>

<RequestList />

</WebRequest>

各字段含义如下

|  |  |
| --- | --- |
| 字段名 | 字段含义 |
| RequestId | 请求的GUID |
| RequestTime | 请求的时间 |
| RequestType | 枚举值，ALL，代表所有客户端监控数据；LIST，指定客户端监控数据；RESTART，重启命令；SHUTDOWN，关闭命令，NONE，测试链接 |
| RequestList | 设备的列表，设备由guid标志。当RequestType为ALL时，可忽略该字段。其它情况，不可忽略。 |

对应的类

public class WebRequest

{

public Guid RequestId { get; set; }

public DateTime RequestTime { get; set; }

public RequestType RequestType { get; set; }

public List<Guid> RequestList { get; set; }

}

public enum RequestType

{

ALL,

LIST,

RESTART,

SHUTDOWN

}

## 服务器—>监控页面接口

<?xml version="1.0"?>

<WebResponse>

<Responses>

<Response>

<IsValid>false</IsValid>

<ResponseCode>0</ResponseCode>

<DeviceId>00000000-0000-0000-0000-000000000000</DeviceId>

<MemoryInfo>

<UsedMemory>0</UsedMemory>

<AllMemory>0</AllMemory>

</MemoryInfo>

<DriverInfo />

<CpuInfo>

<CpuCore>0</CpuCore>

<UsedPercent>0</UsedPercent>

</CpuInfo>

</Response>

</Responses>

</WebResponse>

|  |  |
| --- | --- |
| 字段名 | 字段含义 |
| IsValid | 必填字段，标志请求是否是有效请求。 |
| ResponseCode | 必填字段，0，处理请求失败；1，表示成功处理监控请求，生成有效响应；2，表示接收到重启命令；4，表示接收到关闭命令。 |
| DeviceId | 必填字段，设备id |
| MemoryInfo | 可选，有两个字段：UsedMemory：表示使用的内存，AllMemory：表示所有的内存 |
| DriverInfo | 可选，一个DriverStatus的列表，每一项有DriverName\UsedSpace\AllSpace组成 |
| CpuInfo | 可选，有两个字段：CpuCore：表示cpu核数，UsedPercent表示使用的比例。 |

对应的类：

public class WebResponse

{

public List<Response> Responses;

}

public class Response

{

/// <summary>

/// 请求是否有效请求

/// </summary>

public Boolean IsValid { get; set; }

/// <summary>

/// 返回处理结果码

/// 0：处理成功

/// 1：无效请求

/// </summary>

public int ResponseCode { get; set; }

public Guid DeviceId { get; set; }

/// <summary>

/// 内存状况

/// </summary>

public MemoryStatus MemoryInfo { get; set; }

/// <summary>

/// 硬盘情况

/// </summary>

public List<DriverStatus> DriverInfo { get; set; }

public CpuStatus CpuInfo { get; set; }

}