**项目编号:XXXXXXX**

需求描述书

**2014年03月**

1. 文档属性

|  |  |
| --- | --- |
| 文档属性 | 内容 |
| 项目/任务名称 | 硬件设备服务化 |
| 项目/任务编号 |  |
| 文档名称 |  |
| 文档版本号 | V1.2.0 |
| 文档状态 | 正在编写 |
| 文档编写完成日期 |  |
| 作 者 |  |
| 内部发布范围 |  |

1. 文档变更历史清单

|  |  |  |  |
| --- | --- | --- | --- |
| 文档版本号 | 变更日期 | 修改人 | 备注 |
| 1.0 | 2014-3-20 | 徐强 | 初稿,描述了各个组件及其交互 |
| 1.1 | 2014-4-3 | 徐强 | 细化ab硬件部分(满足最基本功能) |
| 1.1.1 | 2014-4-16 | 徐强 | 添加操作后硬件状态及实现厂商名单 |
| 1.1.2 | 2014-4-22 | 徐强 | 修正打印机接口 |
| 1.1.3 | 2014-4-24 | 徐强 | 添加调试接口 |
| 1.1.4 | 2014-5-6 | 徐强 | 细化调试配置说明 |
| 1.1.5 | 2014-5-9 | 徐强 | 详细描述更新策略 |
| 1.1.6 | 2014-5-21 | 徐强 | 描述打印格式 |
| 1.1.7 | 2014-6-24 | 徐强 | 添加KVM |
| 1.1.8 | 2014-7-8 | 徐强 | 添加音视频录制 |
| 1.2.0 | 2014-10-24 | 徐强 | 实际检验后,增加若干接口 |
| 1.2.1 | 2014-11-1 | 徐强 | 细化后续设计 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[第 1 章 总体说明 6](#_Toc401924177)

[1.1 项目背景 6](#_Toc401924178)

[1.2 基本定义和术语 7](#_Toc401924179)

[1.3 相关文档人员 7](#_Toc401924180)

[第 2 章 设计思路 7](#_Toc401924181)

[2.1 上述问题的解决方法 7](#_Toc401924182)

[2.1.1 DLL调用不稳定 7](#_Toc401924183)

[2.1.2 跨B/S页面调用 8](#_Toc401924184)

[2.1.3 跨网络共享设备 9](#_Toc401924185)

[2.1.4 硬件配置统一更新 9](#_Toc401924186)

[2.1.5 支持服务调试 10](#_Toc401924187)

[第 3 章 软件架构描述 10](#_Toc401924188)

[3.1 系统组成部分 10](#_Toc401924189)

[3.2 总体设计原则 11](#_Toc401924190)

[3.2.1 组件间解耦 11](#_Toc401924191)

[第 4 章 需求描述 11](#_Toc401924192)

[4.1 自动更新程序 11](#_Toc401924193)

[4.1.1 规划 11](#_Toc401924194)

[4.1.2 需求场景描述 12](#_Toc401924195)

[4.1.2.1 自动更新客户端 12](#_Toc401924196)

[4.1.2.2 执行更新脚本 12](#_Toc401924197)

[4.2 更新服务器 12](#_Toc401924198)

[4.2.1 规划 12](#_Toc401924199)

[4.2.2 具体例子 13](#_Toc401924200)

[4.2.3 更新流程 13](#_Toc401924201)

[4.2.4 需求场景描述 14](#_Toc401924202)

[4.2.4.1 获得设备的配置 14](#_Toc401924203)

[2 发送请求部署配置的服务(注：如果为了快速加载,可以把一部分数据存到本地数据库中) 14](#_Toc401924204)

[4.2.4.2 保存设备配置 15](#_Toc401924205)

[4.2.4.3 新建更新类别 15](#_Toc401924206)

[2 使用者新建更新类别 15](#_Toc401924207)

[4.2.4.4 新建更新内容 15](#_Toc401924208)

[1 使用者选择要更新的类别 16](#_Toc401924209)

[4.2.4.5 查看更新列表 16](#_Toc401924210)

[4.2.4.6 比对版本服务 16](#_Toc401924211)

[4.3 监视程序 16](#_Toc401924212)

[4.3.1 规划 16](#_Toc401924213)

[4.3.2 需求场景描述 16](#_Toc401924214)

[4.3.2.1 启动程序 16](#_Toc401924215)

[4.4 设备服务器 17](#_Toc401924216)

[4.4.1 规划 17](#_Toc401924217)

[4.4.2 需求场景描述 17](#_Toc401924218)

[4.4.2.1 硬件服务 17](#_Toc401924219)

[4.4.2.2 可视化界面配置 17](#_Toc401924220)

[4.4.2.3 访问权限配置 17](#_Toc401924221)

[4.4.2.4 提供硬件设备服务 18](#_Toc401924222)

[2.a 调用设备错误,返回给客户端 18](#_Toc401924223)

[4.4.2.5 开始JOB 18](#_Toc401924224)

[4.4.2.6 结束JOB 18](#_Toc401924225)

[4.4.2.7 提供启动情况服务 18](#_Toc401924226)

[4.4.2.8 列举所有设备 19](#_Toc401924227)

[4.4.2.9 列举正在运行设备 19](#_Toc401924228)

[4.4.2.10 列举正在运行JOB服务 19](#_Toc401924229)

[4.4.2.11 结束某项设备服务 19](#_Toc401924230)

[4.4.2.12 提供运行情况 19](#_Toc401924231)

[4.5 请求客户端程序 19](#_Toc401924232)

[4.5.1 规划 19](#_Toc401924233)

[4.5.2 需求场景描述 19](#_Toc401924234)

[4.5.2.1 调用硬件请求 19](#_Toc401924235)

[3 解析jsonresponse结构 20](#_Toc401924236)

[4.5.2.2 调用状态查询请求 20](#_Toc401924237)

[4.5.2.3 结束某项设备请求 20](#_Toc401924238)

[2 发出结束请求。 20](#_Toc401924239)

[4.5.2.4 用户选择设备 20](#_Toc401924240)

[第 5 章 技术选型的考虑 20](#_Toc401924241)

[5.1 实际需求场景 20](#_Toc401924242)

[5.2 编程语言选型 21](#_Toc401924243)

[5.3 架构 21](#_Toc401924244)

[5.4 几个抽象要点 21](#_Toc401924245)

[5.4.1 设备 21](#_Toc401924246)

[5.4.2 更新元素 21](#_Toc401924247)

[5.5 实现上的考虑 22](#_Toc401924248)

[5.6 编码规范 22](#_Toc401924249)

[第 6 章 附录 22](#_Toc401924250)

[6.1 更新脚本描述 22](#_Toc401924251)

[6.1.1 例子 22](#_Toc401924252)

[6.1.2 解释 22](#_Toc401924253)

[3 Close:表示关闭程序 23](#_Toc401924254)

[6.2 更新服务器数据库描述 23](#_Toc401924255)

[6.2.1 更新类别表 23](#_Toc401924256)

[6.2.2 级别表 23](#_Toc401924257)

[6.2.3 更新记录表 24](#_Toc401924258)

[6.3 硬件服务 24](#_Toc401924259)

[6.3.1 DLL格式规范 24](#_Toc401924260)

[6.3.2 主数据结构描述 25](#_Toc401924261)

[6.3.2.1 Json返回结构 25](#_Toc401924262)

[6.3.3 错误代码表 25](#_Toc401924263)

[6.3.3.1 错误代码 25](#_Toc401924264)

[6.3.4 密码键盘 25](#_Toc401924265)

[6.3.4.1 调用接口 25](#_Toc401924266)

[6.3.4.2 驱动实现 26](#_Toc401924267)

[6.3.5 磁卡刷卡器 26](#_Toc401924268)

[6.3.5.1 调用接口 26](#_Toc401924269)

[6.3.5.2 驱动实现 26](#_Toc401924270)

[6.3.6 IC刷卡器 26](#_Toc401924271)

[6.3.6.1 调用接口 26](#_Toc401924272)

[6.3.6.2 驱动实现 26](#_Toc401924273)

[6.3.7 身份证 27](#_Toc401924274)

[6.3.7.1 调用接口 27](#_Toc401924275)

[6.3.7.2 驱动实现 27](#_Toc401924276)

[6.3.8 指纹仪 27](#_Toc401924277)

[6.3.8.1 调用接口 27](#_Toc401924278)

[6.3.8.2 驱动实现 27](#_Toc401924279)

[6.3.9 评价器 27](#_Toc401924280)

[6.3.9.1 调用接口 27](#_Toc401924281)

[6.3.9.2 驱动实现 28](#_Toc401924282)

[6.3.10 扫描仪 28](#_Toc401924283)

[6.3.10.1 调用接口 28](#_Toc401924284)

[6.3.11 打印机 28](#_Toc401924285)

[6.3.11.1 调用接口 28](#_Toc401924286)

[6.3.11.2 例子 28](#_Toc401924287)

[6.3.11.3 标签解释 29](#_Toc401924288)

[6.3.12 KVM 30](#_Toc401924289)

[6.3.12.1 调用接口 30](#_Toc401924290)

[6.3.13 音视频录制 30](#_Toc401924291)

[6.3.13.1 调用接口 30](#_Toc401924292)

[6.4 状态服务 31](#_Toc401924293)

[6.4.1 启动信息描述 31](#_Toc401924294)

[6.4.2 状态信息描述 31](#_Toc401924295)

[6.5 调试服务 31](#_Toc401924296)

[6.5.1 环境定义 31](#_Toc401924297)

[6.5.2 设备状态定义 31](#_Toc401924298)

[6.5.3 设备类别定义 31](#_Toc401924299)

[6.5.4 远程查看日志 31](#_Toc401924300)

[6.5.5 查看系统支持的设备(本驱动程序支持的设备) 33](#_Toc401924301)

[6.5.5.1 返回报文 33](#_Toc401924302)

[6.5.5.2 报文解释 34](#_Toc401924303)

[6.5.6 查看正在使用的设备(指已经装载驱动的设备) 34](#_Toc401924304)

[6.5.6.1 返回报文 34](#_Toc401924305)

[6.5.6.2 报文解释 34](#_Toc401924306)

[6.5.7 改变日志级别 34](#_Toc401924307)

[6.5.7.1 参数解释 34](#_Toc401924308)

[6.6 配置服务 35](#_Toc401924309)

[6.6.1 配置内容范围 35](#_Toc401924310)

[6.6.2 获取某个驱动配置 35](#_Toc401924311)

[6.6.2.1 参数解释 35](#_Toc401924312)

[6.6.2.2 返回报文 35](#_Toc401924313)

[6.6.3 获取所有驱动配置 35](#_Toc401924314)

[6.6.3.1 返回报文 35](#_Toc401924315)

[6.6.4 获取所有可用硬件列表配置 36](#_Toc401924316)

[6.6.4.1 返回报文 36](#_Toc401924317)

[6.6.5 保存某个驱动配置 37](#_Toc401924318)

[6.6.5.1 参数解释 37](#_Toc401924319)

[6.6.5.2 上送报文 37](#_Toc401924320)

[6.6.5.3 返回报文 37](#_Toc401924321)

[6.6.6 保存所有驱动配置 37](#_Toc401924322)

[6.6.6.1 参数解释 37](#_Toc401924323)

[6.6.6.2 上送报文 37](#_Toc401924324)

[6.6.6.3 返回报文 37](#_Toc401924325)

[6.6.7 获得更新服务器地址 38](#_Toc401924326)

[6.6.7.1 返回报文 38](#_Toc401924327)

[6.6.7.2 解释 38](#_Toc401924328)

[6.6.8 保存更新服务器地址 38](#_Toc401924329)

[6.6.8.1 参数解释 38](#_Toc401924330)

[6.6.8.2 上送报文 38](#_Toc401924331)

[6.6.8.3 返回报文 38](#_Toc401924332)

[6.7 服务启动 38](#_Toc401924333)

[6.7.1.1 程序参数 39](#_Toc401924334)

[6.7.1.2 运行界面 39](#_Toc401924335)

# 总体说明

## 项目背景

随着实施客户的增多及项目的不断进展,原有的硬件调用方式已经凸显出各种问题。主要问题如下：

1 某些厂商提供的DLL驱动不稳定。某些情况下会造成DLL出错,从而使得客户端进程崩溃,严重的影响了客户体验。

2 在某些特殊的需求中(如：需要跨越BS页面进行音视频录制),原有的调用方式不能解决跨浏览页调用问题。

3 不支持设备跨机器共享。

4 不支持硬件配置统一批次更新。

旧有的调用方式已经不能解决问题了。鉴于此,提出了硬件设备服务化项目。在新版本中,不仅要解决上述的四个问题。而且为了更好的进行测试工作,还需要增强调试功能。

本项目着力于解决上述几个问题。并把硬件服务,更新服务两个模块抽象出来,争取能为其它项目提供基础服务。

## 基本定义和术语

|  |  |
| --- | --- |
| 硬件设备服务化 | 将硬件调用看成是一种服务。由服务提供者提供服务。 |
| 串口 | 主机与外部设备的一种通信接口。通常是9针。常用于交换小数据。如：密码键盘 |
| 并口 | 主机与外部设备的一种通信接口。通常是25针。常用于交换较大量数据。如：打印机 |
| 转接口 | 当主机通信接口不足时的一种扩展手段。通常是一拖四口。 |
| Usb口 | 主机与外部设备的一种通信接口。 |
| B/S activex调用 | Activex的生命域是在某张页面内。如果跳出这张页面,则会注销此DLL。 |
| JSON RPC | RPC变种。支持HTTP,TCP方式。 |
| ACL | 访问权限控制。常用于操作系统的权限控制。 |

## 相关文档人员

|  |  |
| --- | --- |
| 项目开发者 | 调用接口,使用服务提供者提供的服务。 |
| 运维人员 | 使用系统提供的操作界面,进行各种操作 |

# 设计思路

## 上述问题的解决方法

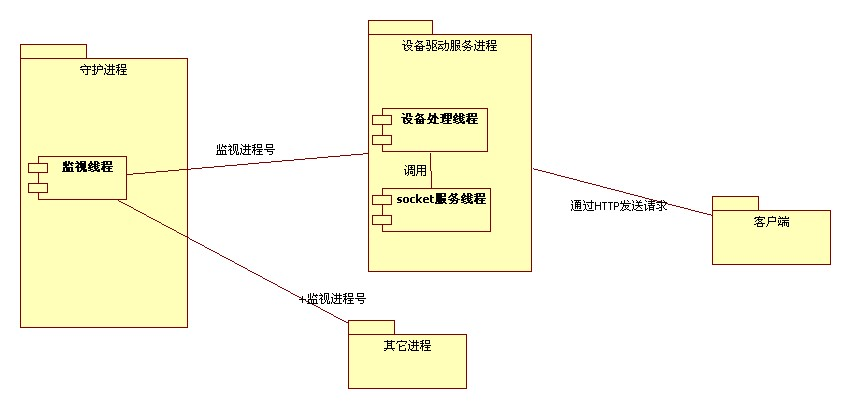
### DLL调用不稳定

通过隔离进程的方法使其隔离。需要涉及到三个实体：

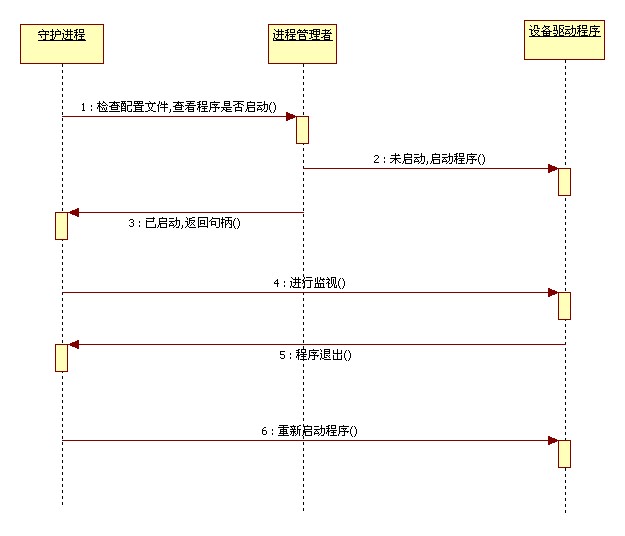
1 调用服务者：负责启动监视服务

2 监视服务者：负责监视服务提供者。如果服务提供者发生异常,则启动它。

3 服务提供者：负责提供服务。



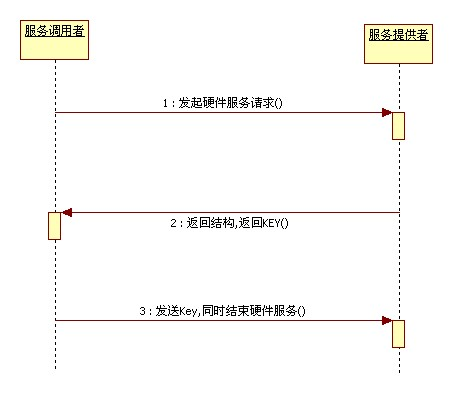
(组件间的静态关系)



(组件间的动态关系)

### 跨B/S页面调用

把硬件设备进行服务化。同时,返回给客户端KEY,标识着这次服务过程。



### 跨网络共享设备

服务通过HTTP服务发布。只要能访问此机器,就能共用其服务。同时,设备服务器需要管理好各种资源。

### 硬件配置统一更新

在服务程序中,配置文件的具体规则如下：

1 支持硬件的本地化配置(如：某台机器com口坏了,需要换一个口,这时候程序应该支持。即：在更新配置时应该忽略过这台机器)

2 新发的驱动配置不应该覆盖旧的驱动配置(可以有个选项：是否强行覆盖)

3 不同类型的设备的驱动配置应该互不干涉(即：可以只更新一种硬件的驱动配置。而这个驱动配置更新后,不应该对其他设备的配置造成影响)

因此,最后的配置文件设计如下：



描述文件采用XML格式。

### 支持服务调试

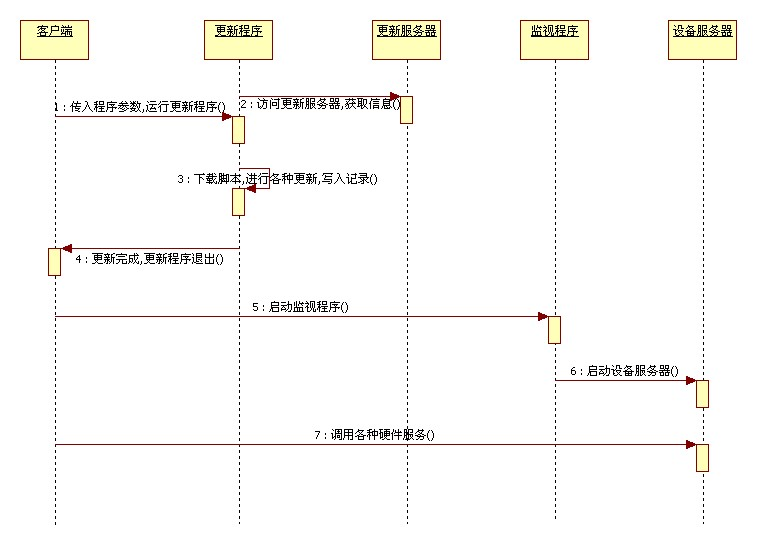
客户端和服务器的交互。共分为几种信息： 1 服务调用信息 2 状态查询信息

注：此客户端并不仅仅意味着服务客户端。状态信息通过HTTP服务发布。就是说,调试程序可以remote attach一个设备服务,然后查看各种状态。

# 软件架构描述

## 系统组成部分

下图是系统调用的总的一个序列图。其中涉及了五部分组件。



(注意：如果要支持域更新。则有必要标识(网点号之类的)。这部分信息可以是外部调用程序传入,也可以是读配置文件。)

以下章节的内容就是围绕着这五部分展开阐述的。

## 总体设计原则

### 组件间解耦

1 解耦服务调用端和硬件细节。要求：所有的硬件配置都要放在硬件设备服务端。

2 解耦自动更新程序和设备服务。更新程序不仅仅能更新设备服务,而且能够支撑其它应用的更新。

# 需求描述

## 自动更新程序

### 规划

1 可以保存本地的版本更新记录

2 可以更新本机的 普通文件/exe文件/各种bat文件/diff文件/注册表文件。以及一些通用操作(如移动文件,改名之类)。

3 更新服务器的部署地点：支持部署在不同的物理地点(如：网点,支行,分行,总行)。每台机器可以连接到不同的更新地点。

4 支持多更新源。

### 需求场景描述

#### 自动更新客户端

主执行者：外部程序或随系统启动

范围：

层次：

主成功场景：

1. 由程序调用者传入http连接。自动更新客户端根据本机版本的情况,加上版本号。

1.a 本地找不到更新记录,则退出。

1. 上送http报文。

2.b 无法访问http链接。报错。

1. 获得返回报文。

3.a 已经是最新版本了,则退出。

1. 下载更新脚本

4.a 如果下载失败,则提示更新出错。

1. 调用Use case:执行更新脚本

#### 执行更新脚本

主执行者：程序

范围：

层次：

主成功场景：

1. 获取更新脚本
2. 解析脚本内容,执行相应的操作(相应的更新脚本内容,请参见附录)

## 更新服务器

### 规划

1 支持更新 普通文件/exe文件/ bat文件/diff文件/注册表文件

2 支持更新脚本(简单的拷贝/替换只能满足简单场景,复杂场景就需要更新脚本)

1. 支持域(指的是归属于不同逻辑组的机器。这个逻辑组可以是机构,可以是自定义组。但组间关系必须是树形结构。系统默认有一个ROOT节点)更新,局部更新。
2. 支持手工配置

从更新内容的关联性来看,大致如下：



此时的配置策略应该如下：

1 对于公用性的部分,如程序本身和其它资源,应该使用级别上最高的服务器进行更新。

2 对于每台机器不一样的更新内容,可以通过手工更新(配置服务器提供更新接口,程序只需调用即可)和批量更新(提供配置文件脚本)。

### 具体例子

1. 如果需要更新程序本身更新的,此时可以把这部分更新放到总行服务器。
2. 如果需要更新配置的,可以在网点内通过配置服务器访问。(此时直接覆盖配置文件)

### 更新流程



### 需求场景描述

#### 获得设备的配置

主执行者：运维人员

范围：

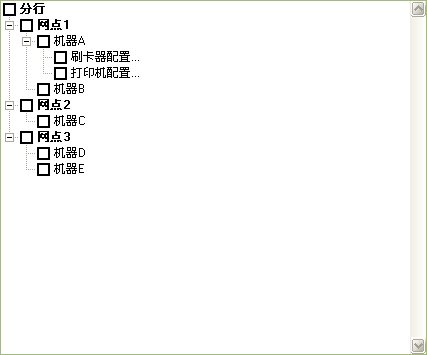
层次：用户

主成功场景：

1. 获得部署级别内所有的配置
2. 发送请求部署配置的服务(注：如果为了快速加载,可以把一部分数据存到本地数据库中)

2.a 如果是网点,则请求网点内所有配置

2.b 如果是分行,则获得分行内所有配置



原型图(可能不太准,但已经能够清楚表明意思了)

#### 保存设备配置

主执行者：运维人员

范围：

层次：用户

主成功场景：

1. 修改某一台机器的配置后,点击”保存”

2 发送配置更新服务到机器

2.a 更新成功,则提示”更新成功”

2.b 更新失败,则提示”更新失败”

#### 新建更新类别

主执行者：运维人员

范围：

层次：用户

主成功场景：

1. 使用者新建更新类别

1.a 如果重名,则进行提示

#### 新建更新内容

主执行者：运维人员

范围：

层次：用户

主成功场景：

1. 使用者选择要更新的类别
2. 使用者上传要更新的脚本文件
3. 使用者选择所需要更新的 级别
4. 使用者填写更新描述
5. 使用者点击”分发更新”,系统自动生成一个版本号(注意是递增且不可重复的)

#### 查看更新列表

主执行者：运维人员

范围：

层次：用户

主成功场景：

1. 使用者选择 更新类别,更新级别,更新日期 进行查询

#### 比对版本服务

主执行者：WEB服务(由客户端调用)

范围：

层次：用户

主成功场景：

1. 获取更新URL中的更新字符串。获取组号,获取版本号 。
2. 如果没有版本号,则报错返回。
3. 获取组号

3.A 如果有组号,则进行层级检索（此域或其归属）。找出大于此版本号的更新。

3.B 如果没有组号,视作所有级别。找到大于版本号的更新。

1. 组装返回报文

## 监视程序

### 规划

能够监视指定程序。如果被监视程序退出,则此程序能够自动启动它。

### 需求场景描述

#### 启动程序

主执行者：外部程序调用。或者随操作系统启动

范围：

层次：系统

主成功场景：

1. 读取配置文件,获取要监视的所有程序

1.a 如果没有配置文件,则属于严重错误。需要提示使用者。

1. 遍历内存中的所有程序

2.A 如果已经执行,则进行监视。

2.B 如果没有执行,则启动此程序,并且进行监视

3 如果程序在执行过程中错误,则需要启动此程序。并且保存日志。

## 设备服务器

### 规划

1 对调用者隐藏硬件细节,包括硬件配置

2 提供各种硬件服务

1. 提供各种调试服务
2. 支持设备共享,但要能提供隔离服务(如：打印机从开始第一条打印指令,到执行指令结束。中间不应该插入其他客户端的打印请求)
3. JOB：使用于以下场景。如：一项服务是由若干步骤组成,在这些步骤间不能接收其它客户请求。打印机的分页打印就是这么一个例子。

### 需求场景描述

#### 硬件服务

设备服务程序所能提供的服务,参见附录：硬件服务。

#### 可视化界面配置

提供可视化界面,配置各种硬件参数。注：此配置需要是http方式的。



#### 访问权限配置

1ACL配置

2返回报文

#### 提供硬件设备服务

设备服务程序所能提供的服务,参见附录：硬件服务。

主执行者：程序

范围：

层次：用户

主成功场景：

1 获取要调用的设备

1.a 检查此设备上有无job.如果有,则返回错误。

1.b 检查此设备是否正在执行。如果在执行,则返回错误。

2 调用此设备进行服务,同时阻塞住客户端。

2.a 调用设备错误,返回给客户端

3 获得硬件输入,或者超时后,返回信息。

#### 开始JOB

主执行者：程序

范围：

层次：用户

主成功场景：

1接收到客户端提出的申请Job指令

2 检查所需要锁定的硬件

2.a 如果硬件空闲,则新建jobid,返回

2.b 如果硬件忙,则返回JOB错误信息

#### 结束JOB

主执行者：程序

范围：

层次：用户

主成功场景：

1接收到客户端提出的退出Job指令,获得jobid。

1.a 没有jobid,返回报文错误。

2 解除对资源的锁定

#### 提供启动情况服务

主执行者：程序

范围：

层次：用户

主成功场景：

1程序接收到请求。

2 返回启动时的日志

#### 列举所有设备

1接收到客户端提出的列举所有设备服务

2 返回报文

#### 列举正在运行设备

1接收到客户端提出的列举运行设备服务

2 返回报文

#### 列举正在运行JOB服务

1接收到客户端提出的列举运行job服务

2返回报文

#### 结束某项设备服务

1接收到客户端提出的结束设备服务

2返回报文

#### 提供运行情况

1接收到客户端提出的运行时情况服务

2返回报文

## 请求客户端程序

### 规划

1 能够调用设备服务器程序提供的各种服务。

2 在出错时,能够有清晰的描述。并且能够继续处理。

### 需求场景描述

#### 调用硬件请求

设备服务程序所能提供的服务,参见附录：硬件服务。

主执行者：程序

范围：

层次：用户

主成功场景：

1 程序发起各种硬件服务。

2 获取返回。(注：该调用程序被阻塞住,直到服务器端有返回。如果客户端需要无阻塞模式,请自行开启线程调用)

2.a 权限不允许,立即返回错误。

2.b 网络连接错误。可能是网络问题,也有可能是服务器失灵。

3 解析jsonresponse结构

#### 调用状态查询请求

主执行者：程序

范围：

层次：用户

主成功场景：

1 程序发起获得运行设备服务。

2 获取返回并且展示。

#### 结束某项设备请求

主执行者：程序

范围：

层次：用户

主成功场景：

1 程序发起获得运行设备服务。

1. 发出结束请求。

#### 用户选择设备

由于有可能有共享设备的需求(如：一台打印机被多个用户共享),这时候程序应该提供一种机制来支持此种操作。可以放在两个地方： 1 把此配置放在设备服务端(通过技术人员来更新) 2 把此配置放到界面端(由终端用户选择ip地址,以指定使用那台机器硬件)。个人认为第二种方案比较好,因为硬件需要人的操作。

主执行者：用户

范围：

层次：用户

主成功场景：

1 更改硬件请求地址

2 更改成功。

# 技术选型的考虑

## 实际需求场景

1 硬件设备服务：硬件设备最多服务于一个网点。假设有5台机器。那么,最大的并发量不会超过5。硬件设备的并发量并不是最重要的,稳定性才是最重要的。另外：一次服务所花费的时间主要在硬件上,因此,编程语言的运行效率高低并不是主要要考虑的。

2 更新服务：如果把更新服务部署在分行。假设所有分行有200台机器。那么,并发量可以达到200.而且,更新服务器主要提供的是文件下载服务。最好,这个服务器能够提供以下功能：

* 1. 高并发
  2. 能够控制下载流量

## 编程语言选型

1 此项目需要和硬件打交道,这部分最好用c语言。

2 同时,上层还需要提供rpc,http,各种资源管理服务,这些用高级语言写比较好。考虑到还需要和dll紧密调用,需要调用操作系统资源。Go语言能够在语言级别支持C语言,和操作系统整合比较好。因此,开发语言初步定为go+c语言。

## 架构

采用动态库还是静态库：因为涉及到众多厂商的DLL,建议采用动态库架构。

## 几个抽象要点

### 设备

需要仔细考虑软件设计模式的地方：硬件设备服务。

大致变动如下：

1同一个设备有后续设备加进来。

2会增加新的设备类别。

因此,这部分的实现需要仔细考虑,大致草图如下：



### 更新元素

更新元素如同设备一样,以后也会加入新的更新类型。和上述抽象图一样设计即可。

## 实现上的考虑

如同Unix的设备管理一样,首先划分出category。然后按category进行管理。

## 编码规范

1 所有的模块功能,必须包含单元测试例子。

# 附录

## 更新脚本描述

### 例子

<?xml version='1.0' encoding='utf-8'?>

<root>

<Section exe="c:/Program Files/a.exe" preupdate="close" updateend="open">

<File srclocation='update/ylkz/WaveLibMixer.dll' dstlocation='c:/Program Files/a/WaveLibMixer.dll' />

</Section>

<Section>

<File srclocation='update/patch/office.bat' dstlocation='c:/patchout/office.bat' runmethod='run'/>

<File srclocation='update/patch/ylkz.bat' dstlocation='c:/patchout/ylkz.bat' runmethod='run' />

</Section>

<Section exe="c:/Program Files/short\_cut/short\_cut.exe" preupdate="close" updateend="closeie" >

<File srclocation='update/short\_cut/ktr.dll' dstlocation='c:/Program Files/short\_cut/dll/ktr.dll' modifytime='2010 09 15 15 45'/>

<File srclocation='update/short\_cut/short\_cut.exe' dstlocation='c:/Program Files/short\_cut/short\_cut.exe' modifytime='2011 06 17 19 06'/>

</Section>

<Section updateend="reboot">

<File srclocation='update/short\_cut/config.ini' dstlocation='C:/Program Files/short\_cut/conf/config.ini' />

</Section>

</root>

### 解释

Srcloction:表示HTTP更新服务器上的路径。使用相对路径。

Dstlocation:表示本地机器上的路径。可以使用相对路径和绝对路径。相对路径指：对于update.exe的路径。

预定义section动作：

1 reboot:重启机器

2 closeie:关闭IE

1. Close:表示关闭程序
2. open:表示打开程序

预定义file动作：

1 run:主要用于运行bat之类的脚本

2 reg:主要用于ACTIVEX的注册

Section：表示按更新内容组成的节。描述了在此节中该如何更新脚本。

属性：

Exe:表示section是关联到这个exe文件。后续的都是针对这个exe文件。

Preupdate:表示更新前需要做的动作。

Updateend:表示更新后需要做的动作。

Userexit:表示需要哪个用户退出。主要用于有用户登录到此系统时。

File:定义了要更新的一个文件。

属性：

Srclocation:在http服务器上的源文件地址

Dstlocation:表示要拷贝到的地址

Modifytime:表示文件更新的时间。本地文件若小于这个更新时间,则更新文件

Version:表示文件版本(这个比较少用。一般而言,使用文件更新时间就已足够)

Runmethod:表示后续动作。见预定义file动作。

## 更新服务器数据库描述

### 更新类别表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **表代码** | Sys\_update\_category | | | | |
| **表名称** | 更新类别表 | | | | |
| **序 号** | **字段代码** | **字段名称** | **字段长度** | **是否非空** | **备 注** |
| 1 | ID | id | NUMBER(10) | Y | 主键自增 |
| 2 | category\_name | 类别名称 | varchar2(60) | Y |  |
| 3 | description | 描述 | varchar2(255) | N |  |

### 级别表

注：此表可能和其它数据库联动,建议在上面再套一层视图。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **表代码** | sys\_update\_grouping | | | | |
| **表名称** | 更新级别表 | | | | |
| **序 号** | **字段代码** | **字段名称** | **字段长度** | **是否非空** | **备 注** |
| 1 | ID | id | NUMBER(10) | Y | 主键自增 |
| 2 | parent\_id | 上级ID | NUMBER(10) | N |  |
| 3 | code | 代码 | varchar2(20) | Y |  |
| 4 | name | 名称 | varchar2(20) | N |  |
| 5 | create\_time | 创建时间 | date | Y |  |
| 6 | note | 备注 | varchar2(200) | N |  |

### 更新记录表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **表代码** | Sys\_update\_record | | | | |
| **表名称** | 更新记录表 | | | | |
| **序 号** | **字段代码** | **字段名称** | **字段长度** | **是否非空** | **备 注** |
| 1 | ID | id | NUMBER(10) | Y | 主键自增 |
| 2 | category\_id | 对应的类别ID | NUMBER(10) | Y |  |
| 3 | group\_id | 对应的组ID | NUMBER(10) | Y |  |
| 4 | create\_user | 创建人 | varchar2(255) | Y |  |
| 5 | create\_time | 创建时间 | date | Y |  |
| 6 | update\_file | 更新脚本的服务器路径 | varchar2(255) | N |  |
| 7 | version | 版本号 | NUMBER(10) | N |  |
| 8 | note | 备注 | varchar2(200) | N |  |

## 硬件服务

### DLL格式规范

DLL用c或c++编写,windows平台上的DLL按照stdcall方式进行函数调用。

超时时间：一律设置为毫秒。

### 主数据结构描述

#### Json返回结构

Type struct JsonResponse{

int32 code; //code:表示返回代码

int32 key;//key:指明一个key

string ErrMsg;//表示出错信息

string ResMsg;//表示正常返回信息,由各个操作解释

}

注：如果遇到字符串,统一是UTF-8格式。

Code：0表示正常。ResMsg表示返回信息

若不为0,则表示各种错误。ErrMsg记录了出错信息。

Key:如果有连续性的请求,key表示这个会话的标识符。

### 错误代码表

#### 错误代码

|  |  |  |
| --- | --- | --- |
| 常量定义 | 常量值 | 常量说明 |
| DEVICE\_OPER\_SUCCESS | 0 | 设备操作成功 |
| DEVICE\_NOT\_FOUND | 1 | 未找到设备 |
| DEVICE\_OPEN\_ERROR | 2 | 设备打开错误 |
| DEVICE\_IN\_USE | 3 | 设备正在使用 |
| DEVICE\_OPER\_TIMEOUT | 4 | 设备操作超时 |

### 密码键盘

#### 调用接口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 | 操作硬件状态 |
| 让小键盘发声“您好，请输入密码”，并亮绿灯,并且获得输入值 | PinProtocol. Readonce | Timeout:int32 | JsonResponse | 操作完成后,端口关闭 |
| 让小键盘发声“请再输入一次”，并亮红灯,并且获得输入值 | PinProtocol. Readtwice | TimeOut:int32 | JsonResponse | 操作完成后,端口关闭 |

#### 驱动实现

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 驱动类型 | 厂商 | 接口 | 实现方式 | 实现文件 |
| 密码键盘 | 哈诚 | 串口 | 往串口写指令 | Pingeneral.go |

### 磁卡刷卡器

#### 调用接口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 | 操作硬件状态 |
| 读磁卡 | MsfProtocal. Read | Timeout:int32  Readtype：int32 | JsonResponse | 操作完成后,端口关闭 |

#### 参数说明

Timeout:超时时间

Readtype:1 二磁道 2 三磁道 3 二三磁道

#### 驱动实现

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 驱动类型 | 厂商 | 接口 | 实现方式 | 实现文件 |
| 磁条卡 | 通用 | USB口 |  | 类似键盘,用户直接读取即可(在获取焦点的输入设备中获得) |
| 磁条卡 | 通用 | COM口 | 往串口写指令 | Bkgeneral.go |

### IC刷卡器

#### 调用接口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 | 操作硬件状态 |
| 获得ic卡信息 | IcProtocol. GetICCardInfo | Timeout:int32  Taglist:String  LpicAppData:string | JsonResponse | 操作完成后,端口关闭 |
| 获得交易详细信息 | IcProtocol. GetTransDetail | Timeout:int32  Path:string | JsonResponse | 操作完成后,端口关闭 |
| 获取ARQC | IcProtocol. GenARQC | Timeout:int32 | JsonResponse | 操作完成后,端口关闭 |
| 执行脚本 | IcProtocol. CtrScriptData | Timeout:int32  Taglist:String  LpicAppData:string  ARPC:String | JsonResponse | 操作完成后,端口关闭 |
| 下电 | IcProtocol. PowerOff | Timeout:int32 | JsonResponse | 操作完成后,端口关闭 |

#### 驱动实现

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 驱动类型 | 厂商 | 接口 | 实现方式 | 实现文件 |
| IC卡 | 通用 | COM口,usb口 | 调用winscard.dll进行操作(现在只适用于windows平台) | Icgeneral.go |

### 身份证

#### 调用接口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 | 操作硬件状态 |
| 获得所有文字信息 | IdcProtocol. ReadData | Timeout:int32 | JsonResponse | 操作完成后,端口关闭 |

#### 驱动实现

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 驱动类型 | 厂商 | 接口 | 实现方式 | 实现文件 |
| 身份证 | 通用 | usb口,com口 | 通用实现 | IDCGeneral.go |

### 指纹仪

#### 调用接口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 | 操作硬件状态 |
| 比对指纹数据 | FinProtocol. MatchFinger | Timeout:int32  FingerTZ:string | JsonResponse | 读取指纹,和存储的指纹进行对比。硬件回到初始状态 |
| 获取指纹数据 | FinProtocol. GetFingerPrinter | Timeout:int32 | JsonResponse | 读取指纹,并且生成特征码返回 |

#### 驱动实现

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 驱动类型 | 厂商 | 接口 | 实现方式 | 实现文件 |
| 指纹仪 | 杭州中正 | usb口 | 调用特定厂商提供的DLL实现 | zzfinger.go |

### 评价器

#### 调用接口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 | 操作硬件状态 |
| 开始评价 | PingjiaProtocol. StartEstimate | Timeout:int32 | JsonResponse | 操作完成后,端口关闭 |
| 取消评价 | PingjiaProtocol. CancelEstimate | Timeout:int32 | JsonResponse | 操作完成后,端口关闭 |

#### 驱动实现

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 驱动类型 | 厂商 | 接口 | 实现方式 | 实现文件 |
| 评价器 | 航泰 | com口 | 调用公司封装的DLL实现 | htpingjia.go |

### 扫描仪

#### 调用接口

|  |  |  |  |
| --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 |
| 同步扫描，并返回第1张扫描图片 | ScanProtocol.scan | showDialog:  adjust:  outputFileName:  quality: | JsonResponse |
| 异步扫描 | ScanProtocol. asyncScan | showDialog:  adjust:  useDefaultSource: | JsonResponse |

### 打印机

#### 调用接口

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 | 操作硬件状态 | |
| 打印(要打印的内容通过Data传入) | Pr2Protocol.Print | Timeout:int32  Data:string | JsonResponse | | 操作完成后,端口关闭 |

#### 例子

<?xml version='1.0' encoding='utf-8'?>

<Root>

<Setting>

<Pagewidth>100</ Pagewidth>

<Pageheight>60</ Pageheight>

<Leftmargin>5</ Leftmargin >

<RightMargin>60</ RightMargin >

<Topmargin>60</ Topmargin >

<Bottommargin>60</ Bottommargin >

</Setting>

<Content>

<table trheight=’2’>

<tr width=”60”>

<td width=”20”><br>1111</br></td>

<td width=”40”>2222</td>

</tr>

<tr width=”60”>

<td width=”20”>3333</td>

<td width=”40”>4444</td>

</tr>

</table>

<br>

</Content>

</Root>

#### 标签解释

长度,坐标的单位：均以标准ASCII码字符长度表示。1个ASCII标准字符长度=1/10inch=0.24cm。

定位方式：分为两种定位方式。一种是绝对定位,另一种是相对定位。

绝对定位：绝对定位下,以Leftmargin,Topmargin为起始坐标(0,0)。

相对定位：相对于当前打印头所在位置的定位。注：为了在各种型号的打印机间保持兼容。不支持打印头回退定位(即：在此种定位方式下,left和top属性不能出现负数)。

Setting:表示纸张的设置。

Pagewidth:纸张的宽度。

Pageheight: 纸张的高度。

Leftmargin:纸张的左边界。默认为2.

Rightmargin: 纸张的右边界。默认为纸张宽度-4。

Topmargin: 纸张的上边界。默认为2.

Bottommargin: 纸张的下边界。默认为纸张高度-4。

Lineseperate:行距。默认为1/6inch。

Content:表示需要打印的内容。

打印格式控制：

Br:表示略过一行

Turnpage:表示手动设置翻页。

Text:表示输出文字

Table:表示定义一个表格

Tr:定一行

Td:定义列

Rowspan:定义行

Colspan:定义列

打印内容控制：

Underline:表示加下划线

Bold:表示加重

DoubleHeight:表示倍高打印

DoubleWidth:表示倍宽打印

属性定义:定义各种需要打印内容的修饰信息。

pos:表示定位的方式。Absolute:绝对定位。Relative:相对定位

left:左边坐标。

Top:上边坐标。

Width:定义宽度。

Height:定义高度。

Align:表示对齐方式。

### KVM

#### 调用接口

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 | 操作硬件状态 | |
| 外屏同屏内屏 | KvmProtocol. OutSyncInner | Timeout:int32 | JsonResponse | | 操作完成后,端口关闭 |
| 内屏同屏外屏 | KvmProtocol. InnSyncOuter | Timeout:int32 | JsonResponse | | 操作完成后,端口关闭 |
| 断屏 | KvmProtocol. DeSync | Timeout:int32 | JsonResponse | | 操作完成后,端口关闭 |

### 音视频录制

#### 调用接口

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 | 操作硬件状态 | |
| 外屏同屏内屏 | AVProtocol. StartRecord | Timeout:int32 | JsonResponse | | 操作完成后,端口关闭 |
| 内屏同屏外屏 | AVProtocol. EndRecord | Timeout:int32 | JsonResponse | | 操作完成后,端口关闭 |

## 状态服务

### 启动信息描述

### 状态信息描述

## 调试服务

### 环境定义

本系统现在支持两种运行环境。这两项可以通过改变日志级别来切换。具体见6.5.7节。

切换日志系统后,不需要重启应用,实时生效的。

典型应用：

生产上出问题了,可以先切换到开发环境.等排除完问题,可以切换到生产环境。

1 开发环境：日志级别为trace。支持远程查看日志。

2 生产环境：日志级别为info。支持远程查看日志。

所有的日志均记录在程序的目录中,日志文件最大不超过40MB。

### 设备状态定义

0设备不可用

1设备空闲

2 设备正在执行任务

### 设备类别定义

Pin:密码键盘

Ic:Ic卡设备

Msf：磁条卡

Idc:身份证

Fin:指纹仪

Pingjia:评价器

Scn:扫描仪

Pr2prt:PR2打印机

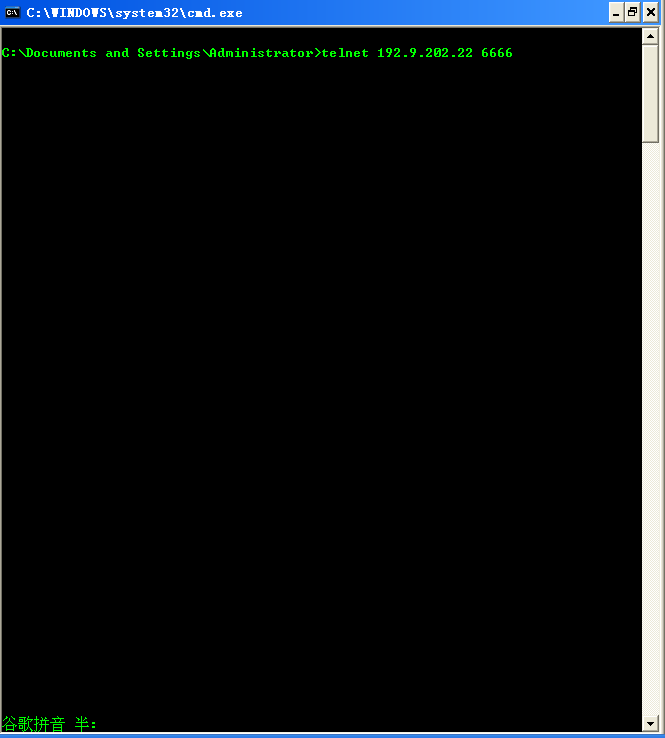
Kvm:通道切换器

AV:音视频录制

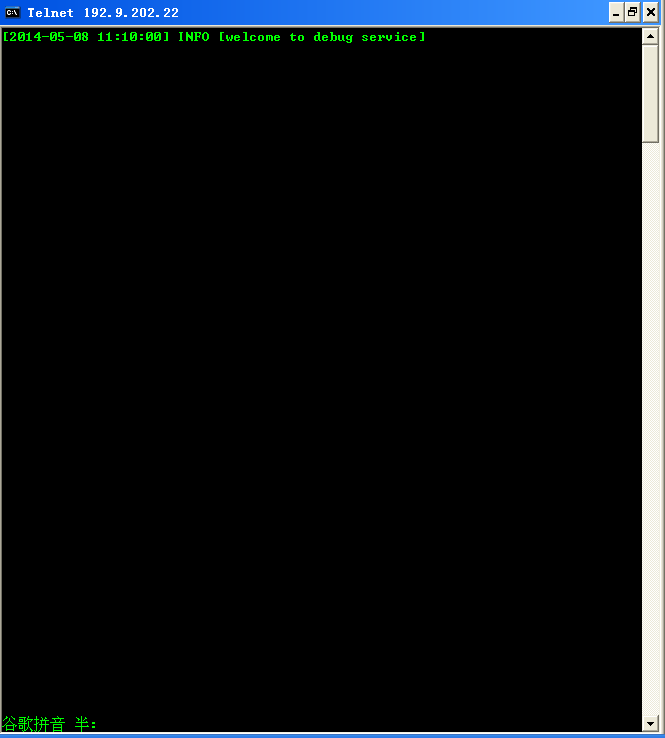
### 远程查看日志

通过telnet。如：机器的IP地址为192.9.202.22,端口为6666(默认)。

telnet 192.9.202.22 6666



连接成功后,出现下图：



注意：

目前指令流是通过调试协议进行的,现阶段不能通过telnet进行控制。

### 查看系统支持的设备(本驱动程序支持的设备)

|  |  |  |  |
| --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 |
| 获取所有设备 | DebugProtocol. GetAllDevice | Timeout:int32 | JsonResponse |

#### 返回报文

<?xml version='1.0' encoding='utf-8'?>

<root>

<res>

<oper>success</oper>

</res>

<content>

<Device Category=”Pin” State=”2” Drivername=””/>

<Device Category =”Ic” State =”2” Drivername=””/>

</content>

</root>

#### 报文解释

Category：表示驱动类别。

State:表示设备状态。见6.5.1.

Drivername:驱动厂商名称

### 查看正在使用的设备(指已经装载驱动的设备)

|  |  |  |  |
| --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 |
| 获取正在运行的设备 | DebugProtocol. GetUsableDevice | Timeout:int32 | JsonResponse |

#### 返回报文

<?xml version='1.0' encoding='utf-8'?>

<res>

<oper>success</oper>

</res>

<content>

<Device category=”Pin” status=”1” drivername=”nantian”/>

</content>

</root>

#### 报文解释

Category：表示驱动类别。

State:表示设备状态。见6.5.1.

Drivername:驱动厂商名称

### 改变日志级别

|  |  |  |  |
| --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 |
| 设置日志运行级别 | DebugProtocol.ChangeLogEnv | Timeout:int32  kind:int32 | JsonResponse |

#### 参数解释

Kind: 1为开发环境 2为生产环境

## 配置服务

### 配置内容范围

1本驱动支持的各种硬件设备类型

配置项的为：param,value类别。不支持局部更新(局部更新是指仅更新一个节。需要把所有参数组成xml报文传入)。

### 获取某个驱动配置

|  |  |  |  |
| --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 |
| 获取设备配置 | ConfigProtocol. GetDevConfig | Timeout:int32  Dev:string | JsonResponse |

#### 参数解释

Dev:表示设备类别。见7.5.2节。

#### 返回报文

<Device Category="Ic">

<Param Name="Factory" Value="nantian"/>

<Param Name="Port" Value="COM4"/>

</Device>

解释：

Category:表示硬件类别。

Param:表示这个硬件设备的配置。以Name和Value的形式给出。

### 获取所有驱动配置

|  |  |  |  |
| --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 |
| 获取所有设备配置 | ConfigProtocol. GetAllDevConfig | Timeout:int32 | JsonResponse |

#### 返回报文

<Root>﻿<Device Category="Pin">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM4"/>

</Device>﻿<Device Category="Pr2prt">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM4"/>

</Device>﻿<Device Category="Fin">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM4"/>

</Device>﻿<Device Category="Pingjia">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM4"/>

</Device><Device Category="Ic">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM5"/>

</Device>

﻿<Device Category="Idc">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM4"/>

</Device>﻿<Device Category="Msf">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM4"/>

</Device></Root>

### 获取所有可用硬件列表配置

|  |  |  |  |
| --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 |
| 获取所有设备配置 | ConfigProtocol. GetDeviceList | Timeout:int32 | JsonResponse |

#### 返回报文

<Root>﻿<Device Category="Pin">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM4"/>

</Device>﻿<Device Category="Pr2prt">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM4"/>

</Device>﻿<Device Category="Fin">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM4"/>

</Device>﻿<Device Category="Pingjia">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM4"/>

</Device><Device Category="Ic">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM5"/>

</Device>

﻿<Device Category="Idc">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM4"/>

</Device>﻿<Device Category="Msf">

<Param Name="Factory" Desc="厂商" Value="nantian"/>

<Param Name="Port" Desc="端口号" Value="COM4"/>

</Device></Root>

### 保存某个驱动配置

|  |  |  |  |
| --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 |
| 获取所有设备 | ConfigProtocol. SaveDevConfig | Timeout:int32  Dev:string  Content:string | JsonResponse |

#### 参数解释

Dev:表示设备类别。见6.5.2节。

Content:表示更新后的内容

#### 上送报文

如把COM口改为5,则Content字段为：

<Device Category="Ic">

<Param Name="Factory" Value="nantian"/>

<Param Name="Port" Value="COM5"/>

</Device>

#### 返回报文

Code值为0表示成功。

注：成功修改配置后,系统并不会自动应用配置。需要通过命令进行重启。

### 保存所有驱动配置

|  |  |  |  |
| --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 |
| 保存所有设备配置 | ConfigProtocol. SaveAllDevConfig | Timeout:int32  Content:string | JsonResponse |

#### 参数解释

Timeout:超时参数

Content:表示更新后的内容

#### 上送报文

#### 返回报文

Code值为0表示成功。

注：成功修改配置后,系统并不会自动应用配置。需要通过命令进行重启。

### 获得可用硬件列表

|  |  |  |  |
| --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 |
| 获得可用硬件列表 | ConfigProtocol. GetAllHardWare | Timeout:int32 | JsonResponse |

#### 参数解释

Timeout:超时时间

#### 上送报文

#### 返回报文

Resmsg中表示返回的列表

### 获得更新服务器地址

|  |  |  |  |
| --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 |
| 获取更新地址列表 | ConfigProtocol. GetUpdateUrl | Timeout:int32 | JsonResponse |

#### 返回报文

<update>

<Url UpdateUrl="192.168.1.2" Desc="更新服务器地址1"/>

<Url UpdateUrl="192.168.1.6" Desc="更新服务器地址2"/>

</update>

#### 解释

Ip:表示更新服务器地址。

Desc:描述

### 保存更新服务器地址

|  |  |  |  |
| --- | --- | --- | --- |
| 方法说明 | RPC方法 | 输入参数 | 返回参数 |
| 保存更新服务器地址 | ConfigProtocol. SaveUpdateUrl | Timeout:int32  Content:string | JsonResponse |

#### 参数解释

Content:表示更新后的内容

#### 上送报文

<update>

<Url Ip="192.168.1.5" Desc="分行服务器"/>

<Url Ip="192.168.1.6" Desc="支行服务器"/>

</update>

#### 返回报文

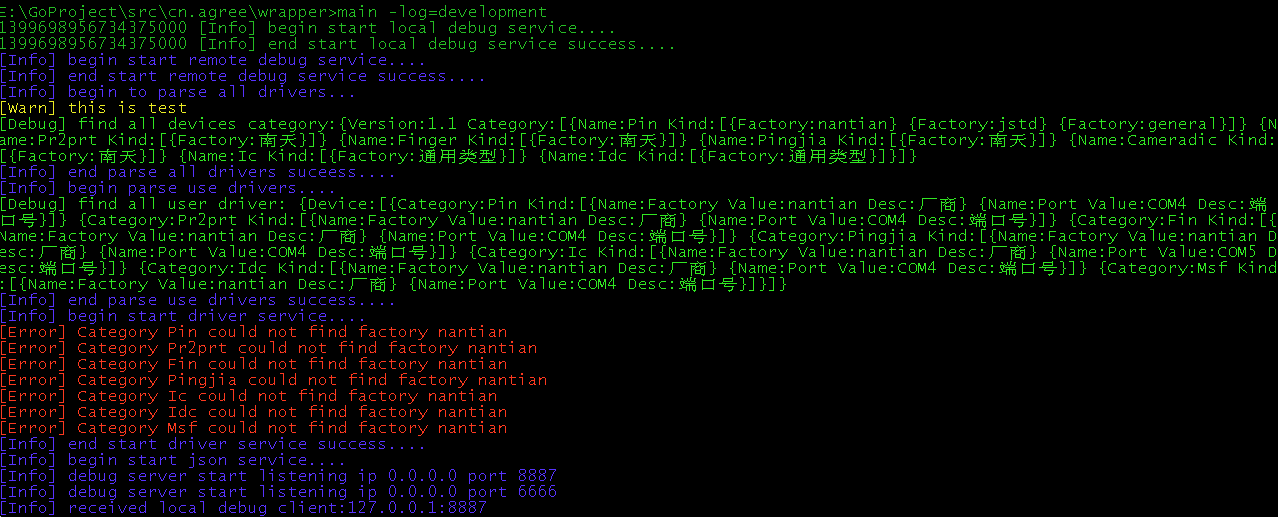
Code值为0表示成功。

## 服务启动

#### 程序参数

-log:表示日志环境。默认是生产环境。-log=development:表示开发环境-log=production:表示生产环境

#### 运行界面



解释：

红颜色ERROR:如果在程序启动时有此错误,必须进行处理。

黄颜色WARN:仅仅是警告,不处理也不会有什么大问题。

蓝颜色INFO:程序关键信息。

绿颜色DEBUG：程序调试信息。用于辅助调试。

褐颜色TRACE:程序内部信息,一般不必关注。

生产环境的日志是到INFO级别。

开发环境的日志是到DEBUG级别。

注：

在实际的生产环境中,会隐藏此界面。但功能是一样的。