异步数据传输通信协议设计

版本: V6.4 修改时间：2014-2-28

# 系统结构



如上图所示，系统由数据源、数据采集器、本地缓存、通信模块和服务器访问接口5个部分构成。

数据源是手机、PC操作系统或其他平台数据库存储短信、电子邮件、联系人、即时通信软件聊天记录等的数据库，以及文档、照片等文件数据。

数据采集器负责定时从数据源读取需进行异步上传的数据，数据上传方式采用增量更新方式进行。数据采集器包括了系统的基础数据采集程序，也包括日后新增的一系列扩展程序。各个数据采集器采用事件或定时器触发的方式决定何时进行数据读取。

数据采集器读取的数据，暂存至本地缓存，等待通信模块上传至服务器。本地缓存是可以是一个SQLite数据库,也可以是按某种规则产生的存储目录，具体格式可视实际情况而定，不在本文档的设计范围内。

通信模块检测可用的网络连接，尝试与服务器建立连接。一旦连接成功，将本地缓存数据库中的数据传输至服务器，并从本地缓存中删除已传送的数据。通信模块中的命令与控制功能执行来自服务器的指令，包括程序配置的更新，以及插件的安装与移除。

通信模块与服务器的通信采用HTTPS连接，服务器构建在Web服务器上提供一系列Web API。

# 基本概念

标识 用于识别协议、设备和数据来源等信息

* **Devid**:设备唯一标识，由设备随机产生唯一标识，在设备上生成以后，不再改变。该字段类型为二进制数据，建议采用通用唯一识别码UUID，包含32个[16进位](http://zh.wikipedia.org/wiki/16進位)数字。
* **Ver**:协议版本号，标识通信协议的版本，整型表示，本协议版本号为6
* **Source**:标识数据来源的不同平台或系统，用于区分如iOS、Android、Windows、Mac等，使用整型表示，目前已指定的来源标识如下表所示：

|  |  |
| --- | --- |
| *Source* | *描述* |
| 11 | iOS |
| 12 | Android |
| 13 | Windows Phone |
| 21 | Windows |
| 22 | Mac OSX |
| 31 | Email Account |
| 41 | Router |

* **Uid**:所属用户标识，采用BSON中的ObjectId表示，包含24个字母，示例：507f1f77bcf86cd799439011。它将会被预先指定在设备程序中。

事件 用于识别请求类型

* **Action**:请求类型，服务器根据请求类型判断需要进行的操作，类型为自然数，目前已指定的请求类型如下表所示：

|  |  |  |
| --- | --- | --- |
| *Action* | *接口* | *描述* |
| 101 | 注册（init） | 设备注册请求 |
| 102 | 用户登录（login） | 登录到服务器 |
| 111 | 数据上传（upload） | 上传数据到服务器 |
| 121 | 获取指令（comm） | 获取指令 |
| 122 | 获取配置（config） | 获取配置 |
| 123 | 下载数据（download） | 从服务器下载数据 |
| 124 | 下载安装（install） | 从服务器下载插件安装 |

* **Nonce**:随机数，每次请求都产生随机数，以当前timestamp时间戳和一个两位随机数相加构成，使用整型表示，如1392696510

认证 用于鉴别请求数据，防止恶意请求

* **Secretkey:**服务器密钥，用于数字签名，静态字段，长度15位，如：

8YRIJ41NK9PLOT6G5MZ7BVUE2XHWCF03DSQA

* **Sig**:数字签名。为了保证设备的合法性，根据**devid、ver、source、action、nonce**五个字段生成一个数字签名，生成算法如下：
* 将字段按上述顺序以字符串拼接，再附上服务器秘钥**secretkey**，生成一个明文字符串。计算明文字符串的md5值，32位小写。

sig=md5(devid|ver|source|action|nonce|secretkey)

举例：

devid:”550e8400-e29b-41d4-a716-446655440000” ver:6

source:11 action:101 nonce:1392696510

secretkey:8YRIJ41NK9PLOT6G5MZ7BVUE2XHWCF03DSQA

sig :

md5(“550e8400-e29b-41d4-a716-44665544000061110113926965108YRIJ41NK9PLOT6G5MZ7BVUE2XHWCF03DSQA”)= 38dae07177b85deabacb790bcceaed6a

数据类型

* **TypeId**:数据类型。以整型表示向服务器上传数据的类型，定义如下表中的ID与Type所示。
* **Data**:对不同的记录类型，其记录的数据格式也是不同的。大部分数据采用数组格式，再进行数据序列化编码。对于数组，并不是所有的字段都是必须的。数据中的时间戳（TS）一定是事件原始的时间（比如短信接收的时间、电话接通的时间等）。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *ID* | *Module* | *Type* | *Format* | *Data (Fields)* |
| 101 | basic | contact | Array | FirstName, LastName, Number |
| 102 |  | sms | Array | Number, TS, Direction, Content |
| 103 |  | callhistory | Array | Number, Name, TS, Direction, Duration |
| 104 |  | notes | Array | TS, Content |
| 105 |  | email | Array | Folder,Content |
| 106 |  | password | Array | Protocol, Server, Title, Username, Domain, Password |
| 107 |  | cellgps | Array | TS, MMC, MNC, LAC, CI, Latitude, Longitude, Accuracy, Confidence |
| 108 |  | wifigps | Array | TS, MAC, Latitude, Longitude, Accuracy, Confidence |
| 109 |  | webhistory | Array | TS\_LastVisit, Title, URL, NumVisits |
| 110 |  | bookmarks | Array | TS\_LastVisit, Title, URL, NumVisits |
| 111 |  | cookie | Array | Domain, Path, Name, Content |
| 112 |  | webform | Array | URL, InputName, Value |
| 113 |  | gps | Array | TS, Latitude, Longitude, Accuracy, Confidence |
|  | *Module* | *Type* | *Format* | *Data(Fields)* |
| 201 | cmd | cmdresp | Plain Text | 命令执行结果，多行文本 |
| 202 |  | dirlist | Plain Text | 目录及子目录文件列表，多行文本，格式有待确定 |
| 203 |  | fileul | Array | Filename,bindata上传文件 |
| 301 | Conf | conf | Array | ServerAddress,Intv,sms,contact,.... |

数据序列化格式

考虑到传输内容（例如Email内容）可能包含二进制数据，因此，选取支持二进制格式的BSON（http://bsonspec.org/）进行数据编码。

编码结果示例如下：

原始数据：

array(3) {

["name"] => string(5) "admin"

["email"] => string(15) "user@domain.com"

["bindata"] => string(16) 0x000102030405060708090a0b0c0d0e0f

}

BSON编码（78字节）：

0x4e000000026e616d65000600000061646d696e0002656d61696c00100000007573657240646f6d61696e2e636f6d000262696e646174610011000000000102030405060708090a0b0c0d0e0f0000

错误代码

其中，*error\_id*应该是确定的，能够被客户端程序识别的，*message*仅供调试用，不是必须的。

|  |  |  |
| --- | --- | --- |
| *Error ID* | *ID* | *Description* |
| dup\_devid | 1000 | 重复的devid |
| invalid\_devid | 1001 | 无效的devid |
| invalid\_sig | 1002 | **sig**验证失败 |
| unknown\_action | 2000 | 无效的操作类型 |
| dataul\_fail | 3001 | 数据上传失败 |
| invalid\_req | 3002 | 未知的客户端请求错误 |
| server\_fail | 3003 | 未知的服务器错误 |

出现dup\_devid，invalid\_devid错误，应当重新生成devid，尝试重新在服务器上注册。

注意：

1.所有字段全部使用小写。

2.devid、ver、source、action、nonce、sig是每次请求必须包含的字段。

# 通信设计

这一节描述设备端程序与服务器的通信方式。

所有的客户端与服务器通信均为HTTP POST请求，连接应使用SSL/TLS加密，初期的实现中客户端不对服务器的证书进行验证。

上传到服务器的数据均以**数据序列化BSON编码。**

服务器返回的数据以json编码直接在HTTP回答的数据中提供。

注册

* POST /api/init

设备首次运行，生成**devid**，并向服务器注册。

如果devid在服务器上已经存在，返回出错信息，客户端应重新生成一个devid并重复上述过程。

请求数据格式：

{

devid:{guid}, //设备标识

ver:{int}, //协议版本号

source:{int}, //来源

action:101,

nonce:{int},

sig:{string}, //根据上面五个字段生成的签名

uid:{guid}

}

响应数据格式：

{

action:101, //响应对应的请求类型

error:{int}, //错误类型，0为成功

errormsg:{string}, //可选，出错信息，调试用

}

用户登录

* POST /api/login

用户登录服务器，成功返回用户标识uid。

请求数据格式：

{

devid:{guid}, //设备标识

ver:{int}, //协议版本号

source:{int}, //来源

action:102,

nonce:{int},

sig:{string}, //根据上面五个字段生成的签名

Username:{string}

Password:{string} // Sha1-512

}

响应数据格式：

{

action:102, //响应对应的请求类型

error:{int}, //错误类型，0为成功

errormsg:{string}, //可选，出错信息，调试用

uid:{string} //uid

}

数据上传

* POST /api/upload

如果有数据上传，一次上传多条数据，采用多维数组（在数据内容中），并进行数据序列化编码。每一条数据，都应包含记录id、数据类型（typeid）和数据内容。服务器成功接收数据后，返回dataack消息，对收到数据的id进行确认。

请求数据格式：

{

devid:{guid}, //设备标识

ver:{int}, //协议版本号

source:{int}, //来源

action:111,

nonce:{int},

sig:{string}, //根据上面五个字段生成的签名

data:[{

id:{int}, //缓存数据库中的id

typeid:{int}, //数据类型

data:{bson}

},

]

}

响应数据格式：

{

action:111, //响应对应的请求类型

error:{int}, //错误类型，0为成功

errormsg:{string}, //可选，出错信息，调试用

ack:[int,], //上传成功的id数组

}

data根据不同的记录类型，可能是纯文本、二进制数据或一个数组，具体定义见**上节数据类型**。

命令执行

* POST /api/comm

由于服务器返回消息可包含多行文本，因此，服务器一旦有需要向客户端发送的命令，即包含在当前任意一类客户端请求（如upload）的响应消息中，但是，在失败的响应中不应该附带命令。

在没有其他任务时，客户端也应定时检查是否有新指令（发送本comm请求）。

请求数据格式：

{

devid:{guid}, //设备标识

ver:{int}, //协议版本号

source:{int}, //来源

action:121,

nonce:{int},

sig:{string}, //根据上面五个字段生成的签名

}

响应数据格式：

{

action:121, //响应对应的请求类型

error:{int}, //错误类型，0为成功

errormsg:{string}, //可选，出错信息，调试用

cmd:[{ //命令数组

cmd:{string},

param:[{string},], //参数数组

}

]

}

目前已定义的命令包括：

|  |  |  |
| --- | --- | --- |
| *Command* | *Parameters* | *Description* |
| install\_plugin | <*URL*> | 下载并安装插件 |
| update\_conf | <*CnfId*> | 下载配置文件 |
| download\_file | <*URL*>,<*SavePath*> | 下载文件到指定路径 |
| upload\_file | <*FilePath*> | 上传指定文件到服务器 |
| list\_dir | <*DirPath*> | 列出目录 |
| list\_dir\_rec | <*DirPath*> | 递归地列出目录（及子目录） |
| shell\_exec | <*ShellCommand*> | 执行系统命令 |

有执行结果的命令，执行结果采用与其他数据相同的上传方式，用相应的数据类型标识(201或202)。

配置更新命令

* POST /api/config

从命令执行请求中获取**update\_conf**指令后，根据CnfId下载配置文件。

请求数据格式：

{

devid:{guid}, //设备标识

ver:{int}, //协议版本号

source:{int}, //来源

action:122,

nonce:{int},

sig:{string}, //根据上面五个字段生成的签名

}

响应数据格式：

{

action:122, //响应对应的请求类型

error:{int}, //错误类型，0为成功

errormsg:{string}, //可选，出错信息，调试用

data: {bson}, //配置文件二进制数据

}

下载文件

* POST /api/download

从命令执行请求中获取**download\_file**指令后，根据URL下载文件，并保存到SavePath。

请求数据格式：

{

devid:{guid}, //设备标识

ver:{int}, //协议版本号

source:{int}, //来源

action:123,

nonce:{int},

sig:{string}, //根据上面五个字段生成的签名

}

响应数据格式：

{

action:123, //响应对应的请求类型

error:{int}, //错误类型，0为成功

errormsg:{string}, //可选，出错信息，调试用

data: {bson}, //文件二进制数据

}

下载插件并安装

* POST /api/install

从命令执行请求中获取**install\_plugin**指令后，根据URL下载文件，并运行。

请求数据格式：

{

devid:{guid}, //设备标识

ver:{int}, //协议版本号

source:{int}, //来源

action:124,

nonce:{int},

sig:{string}, //根据上面五个字段生成的签名

}

响应数据格式：

{

action:124, //响应对应的请求类型

error:{int}, //错误类型，0为成功

errormsg:{string}, //可选，出错信息，调试用

data: {bson}, //文件二进制数据

}