艾伟玛生产线管理系统嵌入式接口定义

一般约定

- 数字在传输中可能被包装成字符串字符串.请注意转换.
- 字符集为UTF-8
- 示范代码和返回值中的空格仅仅是语法格式(PEP8).并不是有效的信息载体.请自行去除空白/空字符.

服务器信息

- 主机 47.100.23.19
- tcp服务端口 32000
- http服务端口 7012

接口定义

1. 查询条码状态
2. 查询条码详情
3. 条码状态重置
4. 申请条码
5. 替换条码
6. 回传作业文件

查询条码状态

查询条码的使用状态.

约定

- 协议 tcp
- 端口 32000
- 请求参数 字符串,格式为: "CheckTraceCodeCanUse," + code (逗号和条码之间有无空格不重要,系统会自动去除)
- 返回值 字符串,格式为: code + "," + 状态位 (逗号和条码之间可能有空格,请自行去除)

行为

服务器在接收到条码信息后会根据条码查询数据库.并返回查询的使用状态.

状态位定义

- 系统检测条码合格返回数据格式: code,1
- 系统检测条码重复返回数据格式: code,2
- 系统检测条码非当前生产数据格式: code,3
- 系统检测条码格式错误: code,4

Example

```
client = TCPClient()
client.connect("47.99.105.196:32000")
client.send("CheckTraceCodeCanUse, 10401911001201805011536541033317")
result = client.revice()
print(result)
>> 10401911001201805011536541033317, 1    // 状态位1,标识可用.
```

查询条码详情

查询条码的详细信息.

约定

- 协议 tcp
- 端口 32000
- 请求参数 字符串,格式为: "code_details," + code (逗号和条码之间有无空格不重要,系统会自动去除)
- 返回值 json字符串

行为

服务器在接收到条码信息后会根据条码查询数据库.并返回查询的详细信息.

返回值定义

- **_id** 条码,字符串,格式类似: 23132102805841430218730720125819577
- **status** 条码状态 1/已使用, 0未使用.
- **level** 条码级别, 整形数字,也可能表现为字符串格式的数字.
- **file_id** 表示条码导入系统时关联的文件,对嵌入式意义不大.
- **print_id** 表示条码打印时对应的批次,对嵌入式意义不大.
- **print_time** 条码打印的时间, 字符串类型,格式:2018-12-09 06:14:06
- **product_id** 表示条码导入系统时关联的产品id
- **product_info** 产品信息,字符串类型, 四段内容分别是 awma/商品名称 100g/规格 450/净含量 1:5:100/包装比例
- **task_id** 表示条码对应的生产任务id
- **batch_sn** 生产批号, 字符串类型
- **sync_id** 表示条码从嵌入式回传时的任务,
- **sync_time** 同步时间,是值嵌入式在(每日)工作结束后,回传使用数据的时间, 字符串类型,格式:2018-12-11 10:24:43
- **output_id** 条码最终导出时的批次id
- **output_time** 导出时间,有这个字段的条码已经完成了整个生产流程. 字符串类型,格式:2018-12-11 10:28:14

Example

```
client = TCPClient()
client.connect("47.99.105.196:32000")
client.send("CheckTraceCodeCanUse, 10401911001201805011536541033317")
result = client.revice()
print(result)   // 打印json字符串
>> {"output_time": "2018-12-11 10:28:14", "sync_time": "2018-12-11 10:24:43", "product_info": "awma 100g 450 1:5:100", "b
data = json.loads(result)   // 将json字符串转换成键值对格式
for key, value in data.items():
    print(key: values)  // 逐行打印键值对
>> output_time: 2018-12-11 10:28:14
>> sync_time: 2018-12-11 10:24:43
>> product_info: awma 100g 450 1:5:100
>> batch_sn: 34655689094266456
>> print_time: 2018-12-09 06:14:06
>> _id: 2313210280584143021873072012581957 7
>> status: 1
>> file_id: 5c08dfb99f0a5e7eb80144d2
>> print_id: 5c0c422edbea6235b8cc5514
>> product_id: 5bff66ecb8a0d3467b6a8dfa
>> level: 1
>> task_id: 5c0c348bdbea622793ee0d14
>> sync_id: 5c0f1feb9f0a5e43c6ce21a9
>> output_id: 5c0f20be9f0a5e451963f59d
```

条码状态重置

重置条码使用状态

约定

- 协议 tcp
- 端口 32000
- 请求参数 字符串,格式为: "reset_code," + code (逗号和条码之间有无空格不重要,系统会自动去除)
- 返回值 字符串,格式为: code + "," + 状态位 (逗号和条码之间可能有空格,请自行去除)

行为

服务器在接收到条码信息后会重置对应条码的状态信息.恢复到使用之前的状态.

状态位

- 返回值1. 表示重置成功
- 返回值0. 表示此条码处于未使用状态.无需重置.
- 返回值2. 表示此条码不存在
- 返回值3. 表示此条码尚未打印

Example

```
client = TCPClient()
client.connect("47.99.105.196:32000")
client.send("reset_code, 10401911001201805011536541033317")
result = client.revice()
print(result)
>> 10401911001201805011536541033317, 1    // 状态位1,重置成功.
```

条码申请

临时申请一个条码(用作箱码等)

约定

- 协议 tcp
- 端口 32000
- 请求参数 字符串,格式为: "apply_code," + example_code (逗号和条码之间有无空格不重要,系统会自动去除)
  - 参数example_code是一个已使用的条码信息(比如上一个已使用的条码信息).系统会根据example_code所关联的产品,申请一个关联相同产品的可用条码.
- 返回值 字符串,
  - 服务器响应正确,直接返回一个条码信息
  - 否则会返回状态位提示错误信息

行为

服务器在接收到申请后,先根据example_code的值查询其对应的产品信息.然后申请一个空白条码,设置其关联的产品和example_code关联的产品一致.然后返回这个空白条码.

状态位

- 返回值1. 表示example_code无效
- 返回值0. 表示空白条码已用尽.
- 返回值-1. 表示程序出错

Example

```
client = TCPClient()
client.connect("47.99.105.196:32000")
client.send("apply_code, 10401911001201805011536541033317")  // 10401911001201805011536541033317是一个已经使用过的条码,使用它作
result = client.revice()
print(result)
>> 23132100917116379735071455644638667   // 申请到了一个新条码.
```

条码替换

交换2个条码除条码内容,关联产品和打印批次之外的所有信息

约定

- 协议 tcp
- 端口 32000
- 请求参数 字符串,格式为: "replace_code," + code1 + "," + code2 (逗号和条码之间有无空格不重要,系统会自动去除)
  - 参数code1和code2代表2个待交换信息的条码.
- 返回值 数字/字符串

行为

服务器在接收到申请后:

1. 交换2个条码的status(使用状态)
2. 交换2个条码的父级条码信息(如果有)
3. 交换2个条码的码级别(如果有)
4. 交换2个条码的任务信息(如果有)
5. 交换2个条码的同步信息(如果有)
6. 交换2个条码的导出信息(如果有)

状态位

- 返回1. 表示执行正确
- 返回2.表示条码2不可用.

- 返回3.表示条码1未使用
- 返回4.表示条码1未打印
- 返回5.表示条码1未打印
- 如果程序执行出错.返回 -1

Example

```
client = TCPClient()
client.connect("47.99.105.196:32000")
client.send("replace_code, 10401911001201805011536541033317, 10401911001201805011536541033318")
result = client.revice()
print(result)
>> 1   // 替换成功
```

数据回传

每日工作结束后,把嵌入式设备里存储的条码信息打包成压缩文件回传到服务器

约定

- 协议 **http**
- 端口 **7012**
- url: **/upload**
- 方法: **POST**
- 请求参数 参数名: file, 参数值: 回传的文件内容, 格式: 二进制
- 返回值 json字符串, 可以转换为键值对格式
  - 服务器响应正确,返回{"message": "success"}
  - 服务器响应错误,返回{"message": "错误的原因"}

行为

嵌入式向服务器的/upload端口发起一个http的请求(请使用相关类库以简化开发工作),把待上传的文件,以file作为参数名,放入请求头中的files对象中.服务器在接收到请求时, 会检查http请求头部的文件载荷.如果发现有一个名字为file的二进制内容,就会取出这个文件并保存.否则程序出错.

强烈建议文件在上传之前使用zip压缩. 有望大大缩小文件的尺寸.服务端目前使用的压缩算法是ZIP_DEFAULT,印象中应该是zlib库提供的算法.对应的百度百科的内容在此https://baike.baidu.com/item/zlib/2432726?fr=aladdin, 这里是zlib的官网

Example

```
"""
data(数组)是一个包装比例1:3的条码数据, data的每个元素都是一个键值对对象.
code 是条码
level 是代表者是几级码
children 是改条码的下属条码(相当于箱码下面的瓶码)
"""
data = [
        {"code": "1234", "level": 2, "children": ["3343,", "2211", "5112"]},
        {"code": "1235", "level": 2, "children": ["3344,", "2212", "5113"]},
        {"code": "1236", "level": 2, "children": ["3345,", "2213", "5114"]},
    ]
with open("task3.json", "w", encoding="utf-8") as f:
    json_data = json.dumps(data)            # 把数据转换为json格式.
    f.write(json_data)                      # 把数据写入task3.json文件
    f.close()
    z = zipfile.ZipFile(file="task3.zip", mode="w", compression=zipfile.ZIP_DEFLATED) # 新建一个压缩文件task3.zip
    z.write(filename=file_name)             # 把task3.json添加到task3.zip文件中
    z.close()                               # 保存文件
    file_data = open("task3.zip", mode='rb')   # 读取压缩文件
    files = {"file": file_data}                # 生成一个http请求头载荷,注意参数名必须是字符串"file"
    r = requests.post("http://47.99.105.196:7012/upload", files=files)  # 发送http请求,上传文件
    print(r)

 >> {"message": "success"}             # 成功
```

有关上传文件的说明

客户端上传的步骤:

1. 组装数据
2. 转成json格式
3. 写入json文件
4. 压缩成zip文件
5. 上传文件

组装数据

- 数据只能由字符串,整数, 数组和键值对这四种格式的类型构成.
- 除一级码之外,单个条码的基本元素是键值对形式的. 组织形式如下: {"code": 条码, "level": 码级, "children": 子码}
- 如果是一级码,那么单个条码的表现形式可以简化为字符串.比如"10401911001201805011536541033317"
- 条码之间的关系是层层嵌套的.就像产品的包装嵌套层级一样:

```
[
    {
        "code": "3234",
        "level: 3,                                      # 三级码信息
        "children": [
                        {
                            "code": "2235",
                            "level: 2,                  # 二级码信息
                            "children":[
                                        "1236",         # 一级码信息
                                        "1237",
                                        "1238",
                                        ......
                                        ]
                        },
                        {
                            "code": "2238",
                            "level: 2,                  # 二级码信息
                            "children":[
                                        "1239",         # 一级码信息
                                        "1240",
                                        "1241",
                                        ......
                                        ]
                        },
                        ......
                    ]
    },
    ......
]
```

- code的值是字符串格式, level的值可以是字符串也可以是数字,服务端会自动进行类型转换.

转成json格式

尽量使用已有的类库进行json转换.不过由于本例的数据都是数字和英文.如果找不到相关类库,也可以自行按照json的标准进行转换.

写入json文件

这一步的要求是文件的后缀名是json就行了.

压缩成zip文件

要求是压缩成zip格式的文件. 算法建议是ZIP_DEFAULT之类的开源算法(zlib库).

上传文件

要求请求头中的文件对象的参数名必须是"file",使用http协议post方法发送