Python による ia-cloud パケージ、及びラズベリーパイ用のサンプルモジュール

1　目的

本ファイルは、ia-cloud プロジェクトで策定された、「Web 端末型 IoT プラットフォーム ia-cloud 」の REST JSON API 仕様に基づく Python 言語ライブラリである ia-cloud パッケージと、そのパッケージを使用したアプリケーションのサンプル実装コードについて解説するものである。

2　利用環境

本パッケージとサンプル実装の実行環境は以下の通りである。
ハードウエア：Raspberry Pi 3 Model3
OS：Raspbian GNU/Linux 8
Python：ver.3.4.2

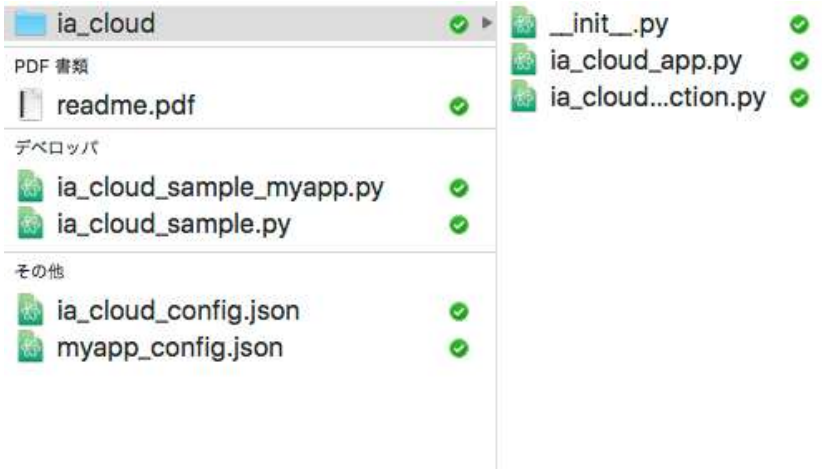
他のハードウエアプラットフォームや他の Linux 系 OS での動作は確認されていないので、使用者の責任において改変されるようお願する。

2.1 命名規則についての補足

ia-cloud の表記は、カタカナ表記の場合「IA クラウド」、英語表記の場合は「ia-cloud」を標準としている。しかし、URL に含まれる場合は「ia_cloud」を用いることもある。
ia-cloud REST JSON API 仕様においては、原則 camelCase（単語の区切りを大文字にする）が用いられており、したがって ia-cloud オブジェクトのアトリビュートの表現は camelCase となる。さらに Python では、メソッド名や引数名にはアンダースコア区切りの命名規則を使うことが推奨されており、本 ia-cloud パッケージやサンプルモジュールのコードは、これらが入り混じった印象を与えることが予想されるがご容赦いただきたい。

3　内容

本圧縮ファイルは、以下のファイルで構成されている



3.1 ia-cloud アプリケーションのサンプル実装

ia-cloud パッケージを利用して、データ収集を実現するためのサンプルアプリケーションのモジュール 2 種と、そのモジュールが参照するアプリケーションの設定情報が格納された JSON ファイル 2 種が格納されている。

- readme.pdf
  この説明ファイルである。

- ia_cloud_sample.py
  一定間隔で Raspbian の bash コマンドを実行し、その戻り値を CCS へ格納するサンプルアプリケーションモジュール。

- ia_cloud_sample_myapp.py
  bash コマンドによるデータ取得に加えて、Raspberry Pi に搭載されている GPIO のデジタル入力データを取得する機能を実装したサンプルである。定周期収集と非同期のデマンド収集に対応している。
  GPIO の出力にアサインされた Pin を一定周期でトグルするスレッドも実行するので、これを入力 pin にループバックすることで、入力の非同期収集をシミュレートできる

- ia_cloud_config.json
  CPU 温度、CPU 使用率、空きメモリ容量、を収集するサンプル設定情報ファイルである。

- myapp_config.json
  CPU 温度の定周期収集に加えて、GPIO 入力 Pin の定周期収集と状態の変化時の非同期収集を実行するための設定情報ファイルである

3.2 ia-cloud パッケージ

REST JSON API に基づき、ia-cloud クラウドセンタサーバ（CCS）に接続し、データの格納などのサービスリクエストを実行する IaCloudConnection クラスを含んだモジュール ia_cloud_connection と、このクラスを利用するフィールドデータサーバ（FDS）のアプリケーションのための基底クラス IaCloudApp クラスを含んだ、ia_cloud_app モジュールを提供する。

- __init__.py
  Python のパッケージの初期設定ファイル（現在は空ファイル）
- ia_cloud_app.py
  データ収集アプリケーションの基底クラス。このクラスでは、Raspbian の bash コマンドによるデータ取得の機能を実装している。必要に応じてこのクラスの get_data()メソッドをオーバーライドして必要なデータ取得機能を実装できる。
- ia_cloud_connection.py
  ia_cloud_app.py モジュールで使用されている、ia-cloud の CCS への接続 Http 通信を実行するクラスと、CCS のレスポンスエラークラスを定義している。

4 ia-cloud パケージの使い方

ia-cloud パッケージを利用したサンプルアプリケーションとして

- CPU 情報を ia-cloud サービスへ格納するアプリケーション
- RaspberryPi の GPIO デジタル入力データを ia-cloud サービスへ格納するアプリケーション

の 2 種類のサンプル実装のモジュールがある。

4.1 CPU 情報を ia-cloud サービスへ格納するサンプル実装

RaspberryPI の CPU 温度データや、vmstat などの OS シェルコマンドで取得できるデータを、ia-cloud サービスへ格納するサンプルアプリケーションの実装である。このアプリケーションを実現するため、以下のような実装を行なっている。詳細は Python のサンプルコードを確認してほしい。

4.1.1 Signal 割り込みハンドラの定義

回復不可能なエラーが発生した場合や、キーボード割り込みでアプリケーションの実行を停止する場合の、Signal 割り込みハンドラをモジュールレベルの関数として定義する。

4.1.2 ia-cloud パッケージの import

標準的に import するモジュール・パッケージに加えて、アプリケーションモジュールの実行には、ia-cloud パッケージを import することが必要である。ia-cloud パッケージがアプリケーションモジュールがあるディレクトリー直下に存在する前提で実装している。

```
# ia-cloud のパッケージをインポート
from ia_cloud.ia_cloud_app import IaCloudApp
```

4.1.3　設定情報ファイルの読み込み

CCS への接続に必要な設定情報、データの格納に必要なデータオブジェクト情報などが記載された JSON ファイルを読み込み、内部の辞書型オブジェクトに格納する。
JSON ファイル名は、モジュールの第 1 引数で指定する。もし、引数が省略された場合、自動的にモジュールパス内の、ia_cloud_config.json が検索され読み込まれる。
設定ファイルの形式は別項（6. 設定情報ファイルの構成）を参照のこと。

4.1.4　IaCloudApp クラスのインスタンスを生成

読み込んだ設定情報オブジェクトをパラメータとして、ia-cloud パッケージの IaCloudApp クラスのインスタンスを生成する。

4.1.5　Signal 割り込みハンドラの設定

アプリケーション終了時に、スレッドの停止と CCS とのコネクション切断実行するため、IaCloudApp クラスのインスタンスをパラメータとして、Signal 割り込みハンドラを設定する。

4.1.6　定期データ収集の開始

IaCloudApp クラスのインスタンスの定期データ収集機能を開始する。具体的には、start()メソッドをコールする。start()メソッドでは、収集データオブジェクト毎に別スレッドが生成され、各スレッドがデータ取得・オブジェクトの作成・CCS への格納を行い、次の収集タイミングまで wait()するように実装されている。

4.2 RaspberryPi の GPIO デジタル入力データを ia-cloud サービスへ格納するサンプル実装

CPU 情報を ia-cloud サービスへ格納するサンプル実装の機能（RaspberryPI の CPU 温度データや、vmstat などの OS シェルコマンドで取得できるデータを、ia-cloud サービスへ格納）に加えて、RaspberryPi の GPIO を操作しデジタル出力と入力を行うサンプルアプリケーションの実装である。このアプリケーションを実現するため、以下のような実装を行なっている。詳細は Python のサンプルコードを確認してほしい。

4.2.1　GPIO 機能の初期設定関数の定義

Raspberry PI のデジタル入出力を設定し、default 設定や Pull-Up 等の設定を行う関数で、アプリケーション起動時に実行される

4.2.2　GPIO の出力をトグルする関数の定義

Raspberry PI のデジタル出力 Pin をトグルする機能の関数。アプリケーション起動時に生成されるスレッドとして並列実行される。
トグルする出力 Pin を入力 Pin にループバックすることで、定期的に変化するデジタル入力としてデータ収集するサンプルアプリケーションとなっている。

4.2.3　Signal 割り込みハンドラの定義

回復不可能なエラーが発生した場合や、キーボード割り込みでアプリケーションの実行を停止する場合の、Signal 割り込みハンドラをモジュールレベルの関数として定義する。
4.1 CPU 情報を ia-cloud サービスへ格納するサンプル実装での Signal 割り込みハンドラに比較し、後述する GPIO の出力をトグル（反転）するスレッドを停止するために必要なパラメータが追加されている。

4.2.4　IaCloudApp クラスを拡張し、個別のアプリケーションの機能を実装した MyIaCloudApp クラスを定義する。

4.1 RaspberryPi の GPIO デジタル入力データを ia-cloud サービスへ格納するサンプル実装では、

- GPIO から設定ファイルに基づきデジタル入力の ON/OFF 状態を取得する機能を付加したメソッドで、get_data()をオーバライドしてる。
- GPIO イベントハンドラとして GPIO の入力変化をトリガに呼び出され、その GPIO の入力状態を取得し CCS へ格納するスレッドを起動する機能のメソッド

GPIO_event_handler()を定義する。モジュールレベルの関数として定義しても良いが、本実装では、IaCloudApp を拡張した子クラス myiaClouApp の新たなメソッドとして定義した。

4.2.5 ia-cloud パッケージの import

4.1 CPU 情報を ia-cloud サービスへ格納するサンプル実装と同様に import

4.2.6 RPi.GPIO パッケージの import

Raspbian の標準モジュールとして組み込まれている GPIO の操作のためのライブラリーパッケージを import する。

4.2.7 設定情報ファイルの読み込み

4.1 CPU 情報を ia-cloud サービスへ格納するサンプル実装と同様に設定情報ファイルを読み込む。JSON ファイル名は、モジュールの第 1 引数で指定する。もし、引数が省略された場合、自動的にモジュールパス内の、myapp_config.json が検索され読み込まれる。

設定ファイルの形式は別項（5．設定情報ファイルの構成）を参照のこと。

4.2.8 IaCloudApp クラスを拡張した MyIaCloudApp のインスタンスを生成

4.1 CPU 情報を ia-cloud サービスへ格納するサンプル実装と同様に、読み込んだ設定情報オブジェクトをパラメータとして、IaCloudApp クラスを拡張した MyIaCloudApp クラスのインスタンスを生成する。

4.2.9 GPIO の初期設定を行い、定期的に GPIO の出力をトグル（反転）させるためのスレッドを生成し起動する。

4.2.10 GPIO の入力変化をトリガに呼び出される、GPIO イベントハンドラーを設定する。ハンドラーとしては上述の MyIaCloudApp のイベントハンドラーを設定する。

4.2.11 Signal 割り込みハンドラの設定

4.1 CPU 情報を ia-cloud サービスへ格納するサンプル実装と同様に 、IaCloudApp クラスのインスタンスと GPIO をトグルするスレッドを停止させるための Event オブジェクトをパラメータとして、Signal 割り込みハンドラを設定する。

4.2.12 定期データ収集の開始

4.1 CPU 情報を ia-cloud サービスへ格納するサンプル実装と同様に 、IaCloudApp クラスのインスタンスの定期データ収集機能を開始する。具体的には、start()メソッドをコールする。start()メソッドでは、収集データオブジェクト毎に別スレッドが生成され、各スレッドがデータ取得・オブジェクトの作成・CCS への格納を行い、次の収集タイミングまで wait()する用実装されている。

5 設定情報ファイルの構成

設定情報ファイルは JSON 形式のファイルであり、以下の基本構成をとる。

```
{
    “userAppConfig”:{JSONobject},          // ユーザアプリで自由に利用できる設定情報
    “iaCloudConfig”:{JSONObject},          // ia-cloud の CCS との接続に関する設定情報
    “収集 object”:{JSONObject},            // 個別の収集オブジェクトに関する設定情報
    “収集 object”:{JSONObject},            // これらの収集オブジェクトの名称
            ・                             // は”iaCloudConfig”でリストされている
            ・
            ・
}
```

個別のアプリケーションに依存する設定情報は、上述の JSONObject 内の Optin アトリビュートに記述

できるようになっている。
詳細は、別紙１、２のサンプル設定ファイルを参照のこと。

6　ia_cloud パッケージについて

ia-cloud パッケージで提供されているクラスの内部構成について簡単に解説する。詳細は Python コードとコメントを参照してもらいたい。

6.1 iaCloudConnection クラス

REST JSON API に基づき、ia-cloud クラウドセンタサーバ（CCS）に接続し、データの格納などのサービスリクエストを実行するクラスである。

内部メソッドとして、__init__()コンストラクターと、CCS への Https POST リクエスト処理を実行する_post_request()、外部メソッドとして ia-cloud の CCS リクエストコマンドを処理する、connect()、store()、retrieve()、get_status()、terminate()を実装している。

6.1.1内部メソッド

- __init__()
  人類のために作られた、エレガントでシンプルな Python の Http ライブラリーとされる、requests パッケージを利用して、requests の Session オブジェクトを生成し、、設定情報から取得した、CCS の URL や接続オプション（認証情報や、timeout,、proxy など）を設定する。

- _post_request()
  辞書オブジェクトで渡された情報を JSON に変換し、Http の Post リクエストボディとして CCS へ送出するメソッド。レスポンスボディの JSON はパースし、辞書オブジェクトを返す。requests.Session が投げた Exception（ConnectionError、HTTPError、Timeout）は、そのまま呼び出し元へ再 raise している。またレスポンスコードエラー（200OK 以外）も CCSBadResponseError として呼び出し元へ raise する。

6.1.2外部メソッド

- connect()
  設定ファイル情報に基づき、ia-cloud の CCS へ接続しサービス ID を取得するメソッド。
- store()
  引数として渡された辞書オブジェクトを JSON に変換し、store コマンドを使用して CCS へ格納するメソッド。受け取った新サービス ID を次に使用するサービス ID として設定する。
- retrieve()
  指定された objectKey と timeStamp を利用して、CCS から retrieve コマンドを使用して iaCloudObject を取得するメソッド。
- get_status()
  CCS に getStatus コマンドを送り、新しいサービス ID を（変更するかは CCS 側の判断）を取得するメソッド。
- terminate()
  CCS との接続を遮断する terminate コマンドを送るメソッド。terminate しても IaCloudConnection オブジェクトは破棄されない。

6.2 CCSBadResponseError クラス

ia_cloud_connection モジュール内で、CCS からの Http レスポンスのステータスコードが 200 ok ではない場合とレスポンスボディの JSON が不正な場合の Exception クラスを定義している。

6.3 IaCloudApp クラス

フィールドデータサーバ（FDS）のアプリケーションのための基底クラス IaCloudApp クラスである。IaCloudConnection オブジェクトを生成し、CCS との接続を行い。定周期データ収集のスレッドを起動管理し、非同期デマンドデータ収集のスレッドを起動する。
ユーザ定義のサブクラスにてオーバーライドすることを想定したサンプル実装として、bash コマン

ドを実行しデータを取得する内部メソッドを実装している。

6.3.1内部メソッド

- __init()__
  設定情報から必要なオブジェクト変数を設定し、IaCloudConnection オブジェクトを生成して、CCS への接続を行うコンストラクター。
  また、エラー情報や動作情報のロギングを司るロギングオブジェクトを取得し設定する。

- _get_data()
  データを取得するための内部メソッド。acq_loop()や_acq_async()から実行される。
  データ取得元を表す引数 source が、"CPU_info" の場合、設定情報に含まれる dataItem の option 情報に基づき、bash を起動してその戻り値を取得データとして返す。
  ユーザアプリケーションのサブクラスにて、実際のデータ取得のメソッドにオーバーライドして使用することを想定している。
  オーバーライドの具体的な例は ia_cloud_sample_myapp.py モジュールにある_get_data()を参照のこと。
- _acq_loop()
  定周期のデータ収集を実行するメソッドで、対象オブジェクトごとに別スレッドとして生成・起動される。収集が終了すると、次の周期まで event.wait()を実行しウエイトする。スレッドはアプリケーション終了まで生存し続ける。

- _acq_async()
  非同期デマンドデータ収集を実行するメソッドで、非同期に別スレッドとして生成・起動され、収集を終了すると、スレッドも終了する。

- _createlogger()
  エラー情報や動作情報のロギングを司るロギングオブジェクトを取得するメソッドで、コンストラクターから実行される。Python の標準ロギングパッケージを利用している。
  設定情報に記述されるデバッグモードで、詳細のデータ収集内容などをログ出力することができる。

6.3.2外部メソッド

- stop()
  CCS へ terminate コマンドを送り、起動されている定周期データ収集スレッドをすべて停止するメソッド。アプリケーション終了時に実行されることを想定している。
- start()
  設定情報に基づいて、定周期データ収集スレッドを生成・起動するメソッド

- async_trigger()
  非同期デマンド収集のスレッドを生成・起動するメソッド。

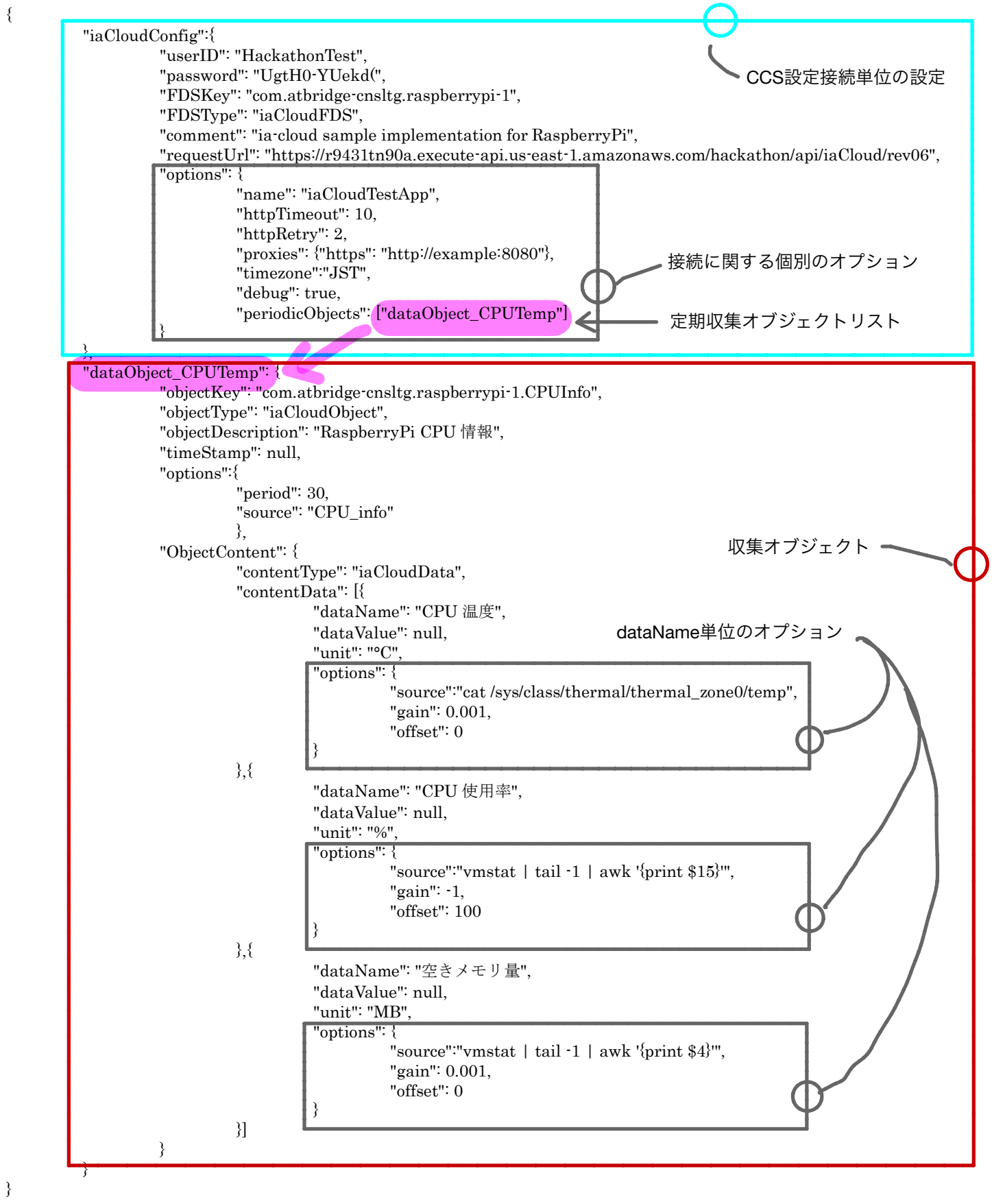
6.3.3通信エラー処理

Requests の Session オブジェクトは、Session オブジェクトのパラメータとして設定される、
・max_retries、・timeout
などの処理の後、発生する TCP/IP コネクションレベル、Http レベルのエラーである、
・ConnectionError、・Timeout、・HTTPEror
の通信エラーを raise する。これらの Exception は、最終的に iaC;oudApp オブジェクトでキャッチされ処理される。
また ia-cloud の API 仕様レベルでのエラーは、CCSBadResponseError として raise される。
IaCloudApp クラスの実装では、これらの Exception 発生時は３回リトライされたあと、エラーログを出力し、アプリケーションは実行を継続する。したがって、次の定周期データ収集タイミングで次のデータ収集が試みられ、次の非同期のデータ収集デマンドで非同期データ収集が試みられる。

7　シーケンスチャート

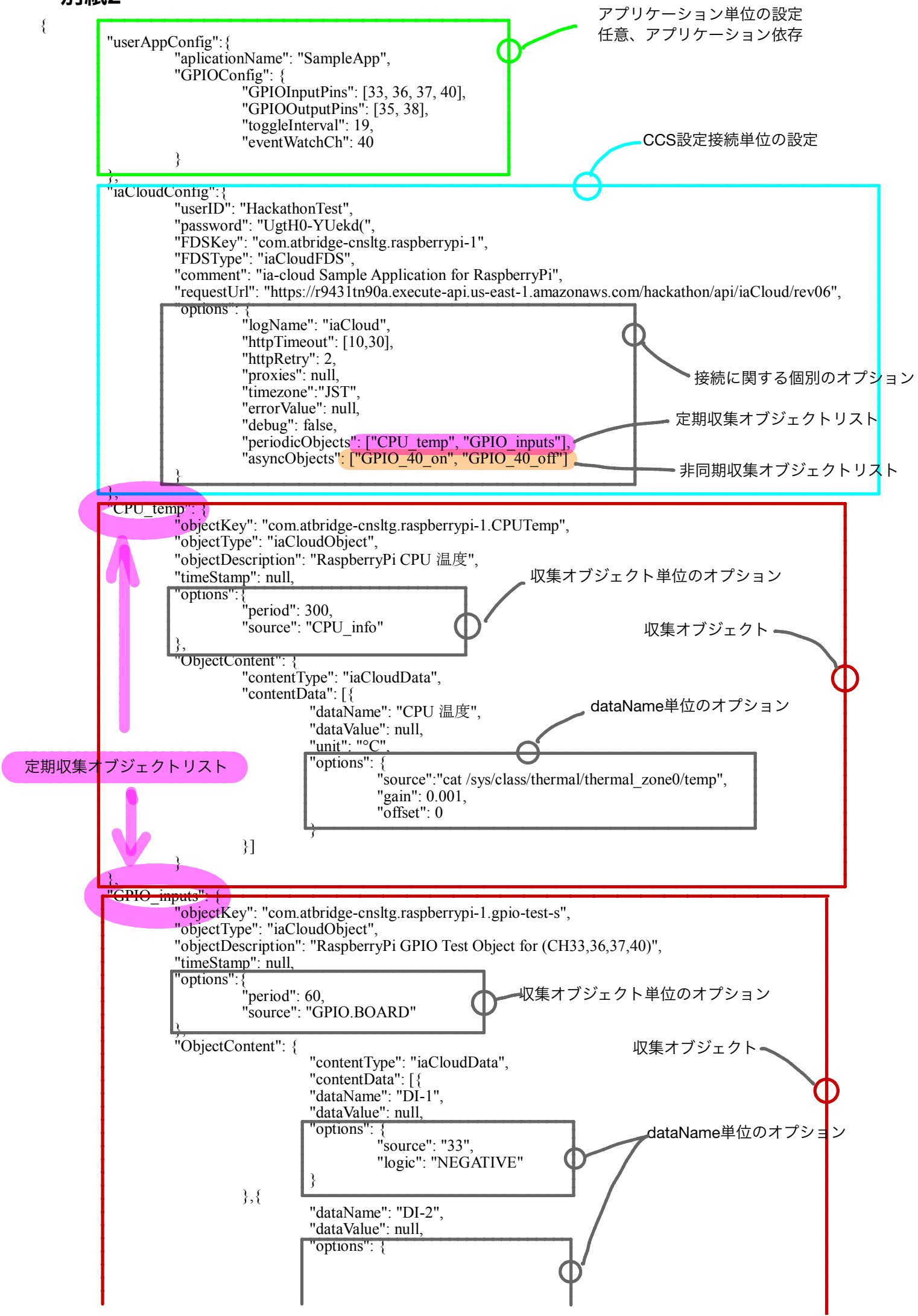
別紙３に、参考として ia_cloud_sample_myapp のシーケンスチャートを示す。

別紙1

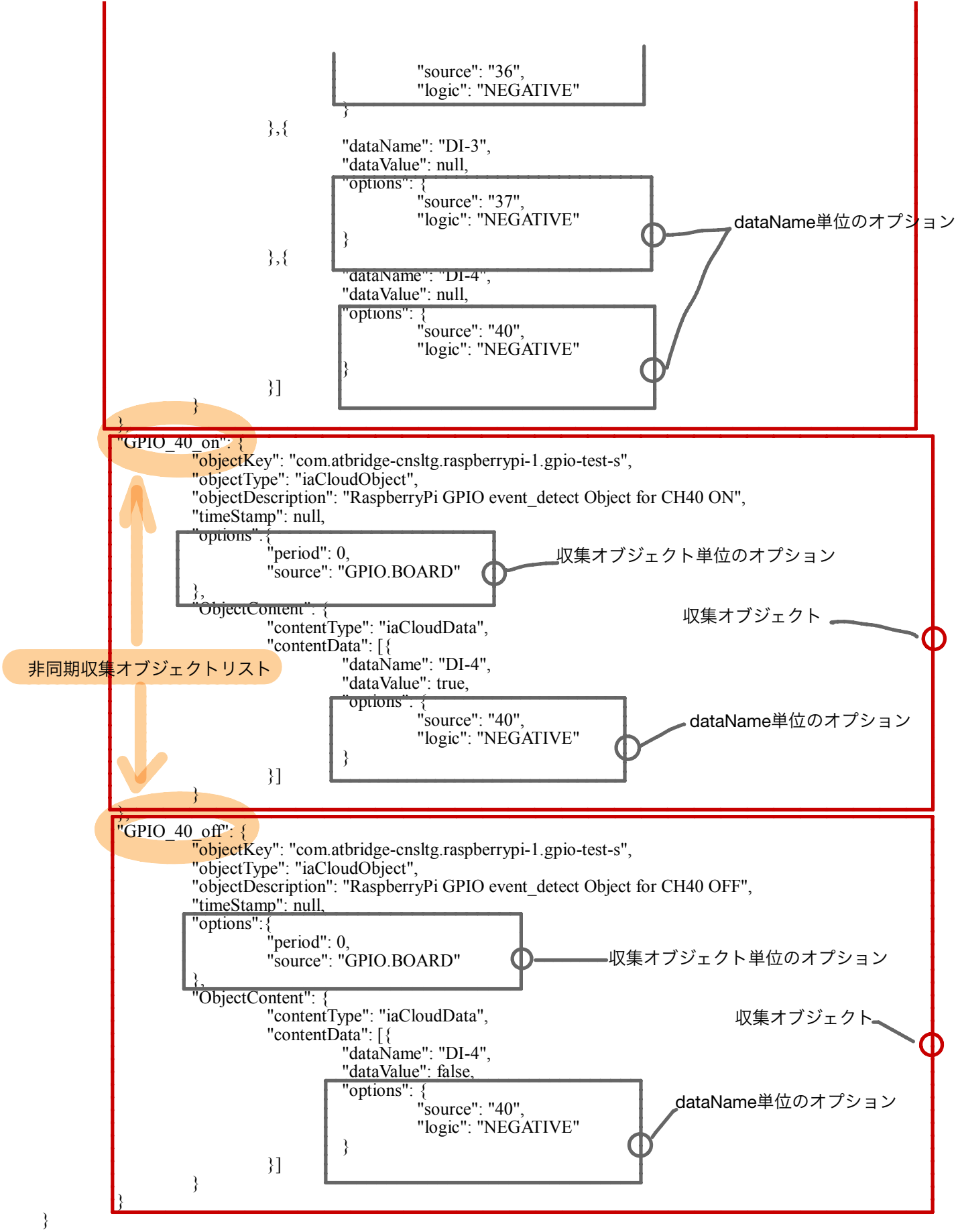



```
{
  "iaCloudConfig":{
        "userID": "HackathonTest",
        "password": "UgtH0-YUekd(",
        "FDSKey": "com.atbridge-cnsltg.raspberrypi-1",
        "FDSType": "iaCloudFDS",
        "comment": "ia-cloud sample implementation for RaspberryPi",
        "requestUrl": "https://r9431tn90a.execute-api.us-east-1.amazonaws.com/hackathon/api/iaCloud/rev06",
        "options": {
                "name": "iaCloudTestApp",
                "httpTimeout": 10,
                "httpRetry": 2,
                "proxies": {"https": "http://example:8080"},
                "timezone":"JST",
                "debug": true,
                "periodicObjects": ["dataObject_CPUTemp"]
        }
  },
  "dataObject_CPUTemp": {
        "objectKey": "com.atbridge-cnsltg.raspberrypi-1.CPUInfo",
        "objectType": "iaCloudObject",
        "objectDescription": "RaspberryPi CPU 情報",
        "timeStamp": null,
        "options":{
                "period": 30,
                "source": "CPU_info"
                },
        "ObjectContent": {
                "contentType": "iaCloudData",
                "contentData": [{
                        "dataName": "CPU 温度",
                        "dataValue": null,
                        "unit": "°C",
                        "options": {
                                "source":"cat /sys/class/thermal/thermal_zone0/temp",
                                "gain": 0.001,
                                "offset": 0
                        }
                },{
                        "dataName": "CPU 使用率",
                        "dataValue": null,
                        "unit": "%",
                        "options": {
                                "source":"vmstat | tail -1 | awk '{print $15}'",
                                "gain": -1,
                                "offset": 100
                        }
                },{
                        "dataName": "空きメモリ量",
                        "dataValue": null,
                        "unit": "MB",
                        "options": {
                                "source":"vmstat | tail -1 | awk '{print $4}'",
                                "gain": 0.001,
                                "offset": 0
                        }
                }]
        }
  }
}
```

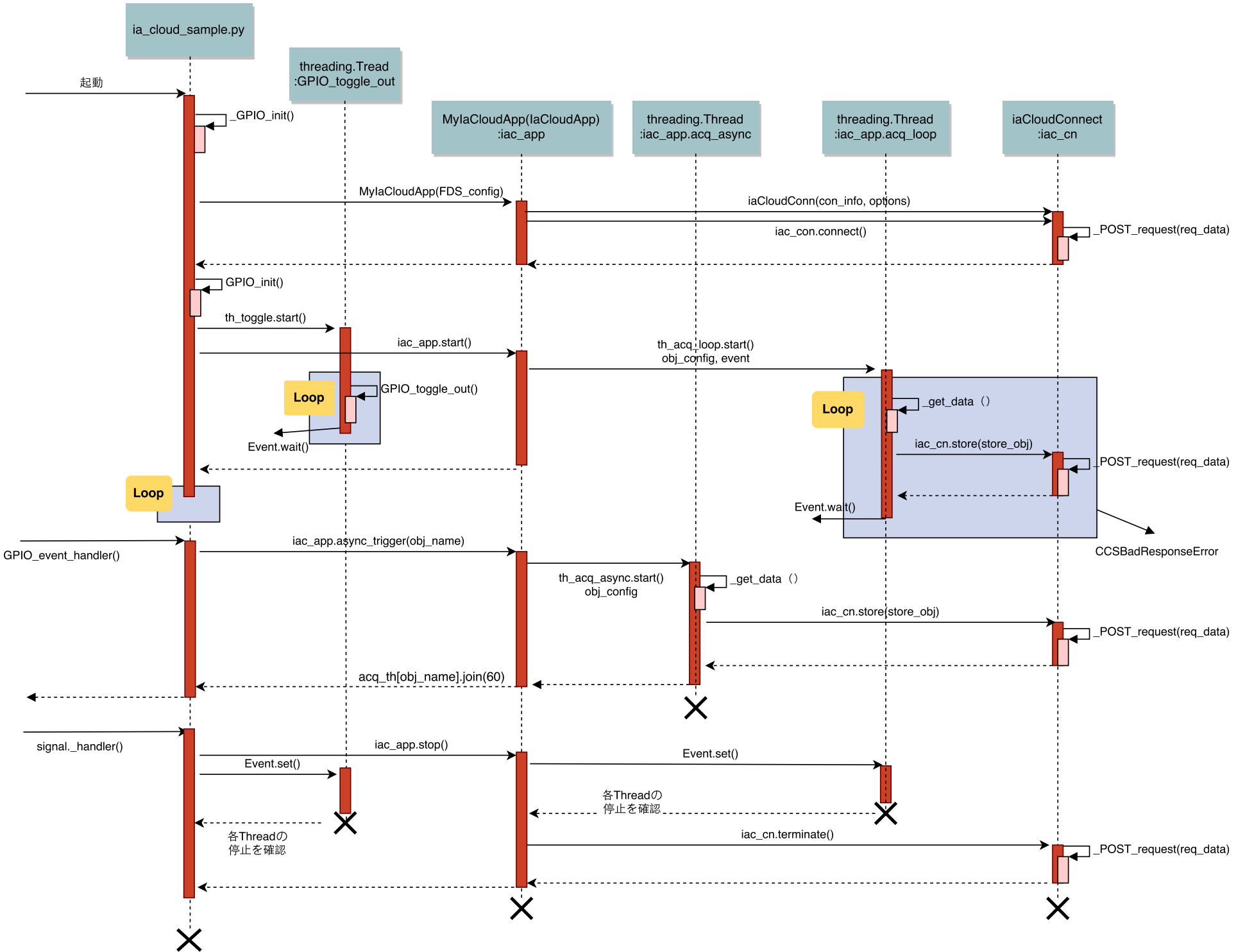
別紙2




```
{
    "userAppConfig":{
            "aplicationName": "SampleApp",
            "GPIOConfig": {
                    "GPIOInputPins": [33, 36, 37, 40],
                    "GPIOOutputPins": [35, 38],
                    "toggleInterval": 19,
                    "eventWatchCh": 40
            }
    },
    "iaCloudConfig":{
            "userID": "HackathonTest",
            "password": "UgtH0-YUekd(",
            "FDSKey": "com.atbridge-cnsltg.raspberrypi-1",
            "FDSType": "iaCloudFDS",
            "comment": "ia-cloud Sample Application for RaspberryPi",
            "requestUrl": "https://r9431tn90a.execute-api.us-east-1.amazonaws.com/hackathon/api/iaCloud/rev06",
            "options": {
                    "logName": "iaCloud",
                    "httpTimeout": [10,30],
                    "httpRetry": 2,
                    "proxies": null,
                    "timezone":"JST",
                    "errorValue": null,
                    "debug": false,
                    "periodicObjects": ["CPU_temp", "GPIO_inputs"],
                    "asyncObjects": ["GPIO_40_on", "GPIO_40_off"]
            }
    },
    "CPU_temp": {
            "objectKey": "com.atbridge-cnsltg.raspberrypi-1.CPUTemp",
            "objectType": "iaCloudObject",
            "objectDescription": "RaspberryPi CPU 温度",
            "timeStamp": null,
            "options":{
                    "period": 300,
                    "source": "CPU_info"
            },
            "ObjectContent": {
                    "contentType": "iaCloudData",
                    "contentData": [{
                            "dataName": "CPU 温度",
                            "dataValue": null,
                            "unit": "°C",
                            "options": {
                                    "source":"cat /sys/class/thermal/thermal_zone0/temp",
                                    "gain": 0.001,
                                    "offset": 0
                            }
                    }]
            }
    },
    "GPIO_inputs": {
            "objectKey": "com.atbridge-cnsltg.raspberrypi-1.gpio-test-s",
            "objectType": "iaCloudObject",
            "objectDescription": "RaspberryPi GPIO Test Object for (CH33,36,37,40)",
            "timeStamp": null,
            "options":{
                    "period": 60,
                    "source": "GPIO.BOARD"
            },
            "ObjectContent": {
                    "contentType": "iaCloudData",
                    "contentData": [{
                    "dataName": "DI-1",
                    "dataValue": null,
                    "options": {
                            "source": "33",
                            "logic": "NEGATIVE"
                    }
                    },{
                    "dataName": "DI-2",
                    "dataValue": null,
                    "options": {
```


"source": "36",
"logic": "NEGATIVE"
}
},{
"dataName": "DI-3",
"dataValue": null,
"options": {
"source": "37",
"logic": "NEGATIVE"
}
},{
"dataName": "DI-4",
"dataValue": null,
"options": {
"source": "40",
"logic": "NEGATIVE"
}
}]
dataName単位のオプション
"GPIO_40_on": {
"objectKey": "com.atbridge-cnsltg.raspberrypi-1.gpio-test-s",
"objectType": "iaCloudObject",
"objectDescription": "RaspberryPi GPIO event_detect Object for CH40 ON",
"timeStamp": null,
"options": {
"period": 0,
"source": "GPIO.BOARD"
},
収集オブジェクト単位のオプション
"ObjectContent": {
収集オブジェクト
"contentType": "iaCloudData",
"contentData": [{
"dataName": "DI-4",
"dataValue": true,
"options": {
"source": "40",
"logic": "NEGATIVE"
}
dataName単位のオプション
}]
}
},
非同期収集オブジェクトリスト
"GPIO_40_off": {
"objectKey": "com.atbridge-cnsltg.raspberrypi-1.gpio-test-s",
"objectType": "iaCloudObject",
"objectDescription": "RaspberryPi GPIO event_detect Object for CH40 OFF",
"timeStamp": null,
"options":{
"period": 0,
"source": "GPIO.BOARD"
},
収集オブジェクト単位のオプション
"ObjectContent": {
"contentType": "iaCloudData",
収集オブジェクト
"contentData": [{
"dataName": "DI-4",
"dataValue": false,
"options": {
"source": "40",
"logic": "NEGATIVE"
}
dataName単位のオプション
}]
}
}
}

別紙3


ia_cloud_sample.py
threading.Tread
:GPIO_toggle_out
MyIaCloudApp(IaCloudApp)
:iac_app
threading.Thread
:iac_app.acq_async
threading.Thread
:iac_app.acq_loop
iaCloudConnect
:iac_cn
起動
_GPIO_init()
MyIaCloudApp(FDS_config)
iaCloudConn(con_info, options)
iac_con.connect()
_POST_request(req_data)
GPIO_init()
th_toggle.start()
iac_app.start()
th_acq_loop.start()
obj_config, event
Loop
GPIO_toggle_out()
Event.wait()
Loop
_get_data（）
iac_cn.store(store_obj)
_POST_request(req_data)
Event.wait()
CCSBadResponseError
Loop
GPIO_event_handler()
iac_app.async_trigger(obj_name)
th_acq_async.start()
obj_config
_get_data（）
iac_cn.store(store_obj)
_POST_request(req_data)
acq_th[obj_name].join(60)
signal._handler()
iac_app.stop()
Event.set()
Event.set()
各Threadの
停止を確認
各Threadの
停止を確認
iac_cn.terminate()
_POST_request(req_data)