

# 修士論文

題目

IoT機器からの通知に基づいた機器監視サービスの開発

学籍番号・氏名

15006・宮坂 虹櫻

指導教員

横山 輝明

提出日

2017年1月28日

神戸情報大学院大学  
情報技術研究科 情報システム専攻

# 目 次

第 1 章 はじめに	1
第 2 章 IoT サービスの維持における問題	3
2.1 IoT サービス . . . . .	3
2.2 IoT サービスの構造 . . . . .	3
2.3 IoT サービス維持の問題 . . . . .	4
2.4 従来の機器監視手法 . . . . .	5
2.4.1 サーバからの問い合わせによる監視 . . . . .	5
2.4.2 監視対象機器からの通知による監視 . . . . .	5
2.4.3 ネットワークによる監視 . . . . .	6
2.4.4 既存手法のまとめ . . . . .	6
2.5 岡本商店街での事例 . . . . .	6
2.5.1 実験結果 . . . . .	8
2.5.2 課題点 . . . . .	10
第 3 章 IoT 機器からの通知に基づく機器監視サービスの提案	11
3.1 IoT 機器の監視 . . . . .	11
3.2 IoT 機器監視の重要性と問題点 . . . . .	11
3.3 IoT 機器からの通知に基づく機器監視サービスの提案 . . . . .	11
3.4 IoT 機器の監視に必要な機能 . . . . .	12
3.5 IoT 機器の監視における要件 . . . . .	13
第 4 章 機器監視サービスの実装	14
4.1 機器監視サービスの構成 . . . . .	14
4.1.1 エージェントプログラム . . . . .	15
4.1.2 エージェントプログラム用インターフェース . . . . .	15
4.1.3 機器状態データベース . . . . .	15
4.1.4 機器情報データベース . . . . .	15

4.1.5	Web アプリケーションサーバ	15
4.1.6	Web アプリケーション	17
4.2	機器監視サービスの実装	18
4.2.1	エージェントプログラムの実装	18
4.2.2	エージェントプログラム用インターフェース「かおりちゃん」の実装	23
4.2.3	機器情報データベース	23
4.2.4	機器状態データベース	23
4.2.5	Web サーバーアプリケーション	24
4.2.6	Web アプリケーション	25
4.2.7	エージェントプログラムと「かおりちゃん」間の通信の実装	25
4.2.8	Web アプリケーションサーバと Web アプリケーション間の通信の実装	26
4.3	サービスによる監視のイメージ	26
<b>第 5 章 機器監視サービスのユーザーテストと考察</b>		<b>28</b>
5.1	機器監視機能のテスト	28
5.1.1	実験のシナリオ	28
5.1.2	実験結果	29
5.2	IoT 機器を監視までの手順の比較	29
5.2.1	機器監視サーバーの構築	29
5.2.2	各 IoT 機器の状態を可視化する	29
5.3	考察	30
<b>第 6 章 おわりに</b>		<b>31</b>
<b>第 7 章 謝辞</b>		<b>32</b>
<b>参考文献</b>		<b>33</b>

# 図 目 次

2.1 IoT サービスの構成図	4
2.2 岡本商店街人流観測 使用した機器	7
2.3 岡本商店街人流観測 動作イメージ？	7
2.4 岡本商店街人流観測可視化アプリケーションスクリーンショット	8
2.5 岡本商店街人流観測 観測 1	9
2.6 岡本商店街人流観測 観測 2	9
2.7 岡本商店街人流観測 観測 3	9
3.1 IoT サービスの構成図	12
4.1 システムのブロック図	14
4.2 ログイン画面	17
4.3 機器状態一覧画面	18
4.4 機器状態一覧画面（小さく表示）	19
4.5 機器状態詳細表示	19
4.6 機器追加ダイアログ	20
4.7 機器情報編集ダイアログ	21
4.8 過去の状態表示ページ	22
4.9 エージェントプログラムと「かおりちゃん」の間のメッセージシーケンス図	25
5.1 IoT サービスの構成図	28

## 内容梗概

近年，IoT が注目を集めている。IoT とは，コンピュータをさまざまなモノに取り付けることで，利便性の向上を図る概念である。近年の半導体技術の進歩により，コンピュータが安価・小型になったこと，インターネットへの通信が様々な場所で安価に行えるようになったことにより，注目が集まっている。

それらのモノが連携して提供するサービスは IoT サービスと呼ばれ，より生活に身近なサービスの登場が期待されている。IoT サービスは，IoT 機器とサーバーがインターネットを介して通信し合うことで，成り立っている。IoT 機器は，モノにコンピュータが取り付けられた機器で，周囲の状況を検知，または，周囲へ働きかける機能を持つ。サーバーは，IoT 機器からの情報を蓄積・分析し，IoT 機器へ指示を送るか，ユーザーへ分析結果を表示する機能を持つ。これら IoT 機器とサーバーが連携することで，IoT サービスは利便性をユーザーへ提供している。

IoT サービスを円滑に提供するには，IoT 機器とサーバーの連携を正常に維持しなければならない。そのため，IoT 機器の動作状態や通信状態の監視が重要となる。数多く，さまざまなネットワークを介して接続される IoT 機器の監視は困難な問題である。IoT 機器が設置される様々なネットワークの構成を把握することは，IoT 機器が多量であることを考えると現実的ではない。また，従来の監視手法はパーソナルコンピュータを対象としたもので，利用しにくい。現状としては IoT サービス開発者が，IoT サービス毎に実装しているため負担が大きいといった問題がある。そのため，設置されるネットワークに関係なく状態が監視できることが求められる。また，IoT 機器の状態を一覧して確認できることや，IoT 機器の過去の動作状態や通信状態を確認できることが必要である。IoT サービスの開発者の負担を減らすためにも，IoT 機器の監視サービスが必要である。

そこで，我々は，IoT 機器からの通知に基づいた機器監視サービスを提案する。IoT 機器が自身の過去の動作状態や通信状態を記録することで，設置されるネットワークに関係なく状態監視をすることを可能にする。また，サービスを機器監視に特化させ独立させることで，IoT サービスに変更を与えること無く，機器を監視することを可能にする。この仕組みを用いることで，IoT 機器が設置されるネットワークに関係無く状態を監視することや，IoT 機器の過去の状態や通信状態を確認することを容易にし，IoT サービス開発者はサービスの開発に専念できる。本研究では，IoT 機器からの通知による機器の設置環境によらない機器の監視を行うことにより，IoT サービスの維持を容易にするシステムの開発に取り組む。

# 第1章 はじめに

近年，IoT が注目を集めている。IoT とは，コンピュータをさまざまなモノに取り付けることで，利便性の向上を図る概念である。近年の半導体技術の進歩により，コンピュータが安価・小型になったこと，インターネットへの通信が様々な場所で安価に行えるようになったことにより，注目が集まっている。

それらのモノが連携して提供するサービスは IoT サービスと呼ばれ，より生活に身近なサービスの登場が期待されている。IoT サービスは，IoT 機器とサーバーがインターネットを介して通信し合うことで，成り立っている。IoT 機器は，モノにコンピュータが取り付けられた物で，周囲の状況を検知，または，周囲へ働きかける機能を持つ。サーバーは，IoT 機器からの情報を蓄積・分析し，IoT 機器へ指示を送るか，ユーザーへ分析結果を表示する機能を持つ。これら IoT 機器とサーバーが連携することで，IoT サービスは利便性をユーザーへ提供している。

IoT サービスを円滑に提供するには，IoT 機器とサーバーの連携を正常に維持しなければならない。そのため，IoT 機器の動作状態や通信状態の監視が重要となる。数多く，さまざまなネットワークを介して接続される IoT 機器の監視は困難な問題である。IoT 機器が設置される様々なネットワークの構成を把握することは，IoT 機器が多量であることを考えると現実的ではない。多量の IoT 機器を個々に識別し，異常を検知することも難しい。そのため，設置されるネットワークに関係なく状態が監視できることが求められる。また，IoT 機器の状態を一覧して確認できることや，IoT 機器の過去の動作状態や通信状態を確認できることが必要である。

そこで，我々は，IoT 機器からの通知に基づいた機器監視サービスを提案する。IoT 機器が自身の過去の動作状態や通信状態を記録することで，設置されるネットワークに関係なく状態監視をすることを可能にする。また，サービスを機器監視に特化させ独立させることで，IoT サービスに変更を与えること無く，機器を監視することを可能にする。この仕組みを用いることで，IoT 機器が設置されるネットワークに関係無く状態を監視することや，IoT 機器の過去の状態や通信状態を確認することを容易にする。本研究では，IoT 機器からの通知による機器の設置環境によらない機器の監視を行うことにより，IoT サービスの維持を容易にするシステムの開発に取り組む。

本論文では，IoT 機器の監視困難の問題を取り上げ，その問題解決のための監視サービスを開発し，効果を報告する。第 2 章では，IoT サービスの維持に関する背景と，IoT 機器の監視に関する問題を述べる。第 3 章では，第 2 章で述べた問題を分析し，IoT 機器監視サービスの機能要件について述べる。第 4 章では，IoT 機器監視サービスの実装の詳細について述べる。第 5 章では，実験により IoT 機器監視サービスがもたらす効果

を検証し，考察を述べる．第6章では，本研究に関する評価について述べる．第7章では，本研究を通して得られた知見や今後の課題について述べる．

## 第2章 IoT サービスの維持における問題

### 2.1 IoT サービス

IoT とは、Internet of Things の略で「モノのインターネット」とも呼ばれる概念である。IoT では、様々な物がインターネットにつながり、相互に情報をやり取りすることで、多様な自動化を行う。IoT サービスとは、ユーザーに対し IoT による利便性を提供するものである。

IoT サービスは、半導体技術の進歩によりコンピューターが小型且つ安価になったこと、通信ネットワークの整備が進み様々な場所から安価に通信が利用可能になったことで登場した。

例えば、次のような物がある。

- 駐車場の検索・予約・決済サービス [1]
- 太陽光発電の監視

駐車場の検索・予約・決済サービスとは、ドライバーが空いている駐車場を探す手間を省くためのサービスである。周囲の空いている駐車場の検索や、予め駐車場を予約しておくことで、駐車場を探す手間を省いている。このサービスの実現のために、駐車場の駐車スペースにコンピュータを取り付ける。これらコンピュータが、駐車場が空いているか否か・予約が入っているか否か等をサーバーとやり取りする。それにより、サーバーは空いている駐車場の一覧や、利用情報に基づく決済を、ユーザーに提供している。

太陽光発電の監視とは、太陽光発電所の発電量や機器の異常を確認しに行くための手間を省くためのサービスである。このサービスの実現の為に、太陽光発電所の機器にコンピューターを取り付ける。これらコンピュータが発電量や機器の異常の情報をサーバーとやり取りする。それにより、サーバーは、発電量や機器の異常をユーザーに知らせる。

このように、IoT サービスは、IoT 機器とサーバーが連携し、ユーザーに利便性を提供するものである。今後も数多くサービスが登場すると考えられている。

### 2.2 IoT サービスの構造

IoT サービスは、IoT 機器とサーバーが連携し利便性を提供するものである。IoT サービスの構造として、多数の IoT 機器とサーバーがインターネットを介し連携する事が挙げられる。

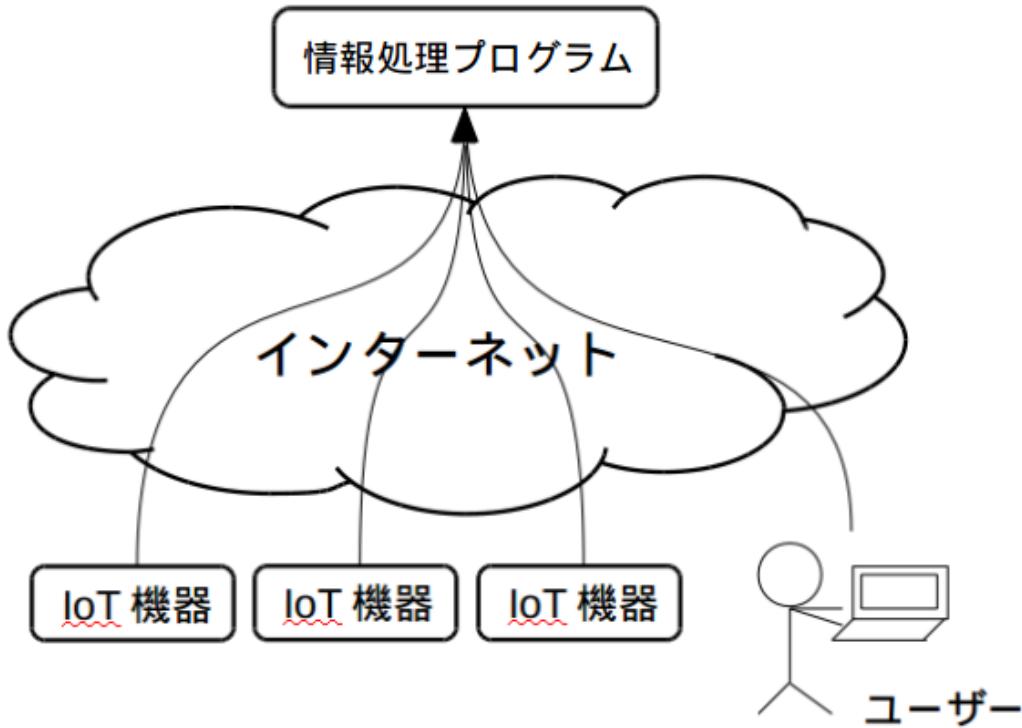


図 2.1: IoT サービスの構成図

IoT 機器は、様々な環境へ設置され、周囲の状況を検知することや、周囲へなんらかの働きを行う為に使用される。駐車場の例では、駐車スペースに車が止まっているか否かを検知している。また、予約された駐車スペースであることを別のユーザーへ知らせている。

この機器からの情報を収集し、処理しているのがサーバーである。サーバーは、IoT 機器からの情報を蓄積・分析し、IoT 機器やユーザーに対し何らかの働きかけを行う。駐車場の例では、駐車場の利用情報を蓄積・分析し、ユーザーへ対し可視化を行っている。サーバーにて動作するプログラムが、IoT 機器と通信することで、IoT サービスを構成する。この通信に利用されるのがインターネットである。様々な通信リンクを用いて IoT 機器とサーバー上のプログラムが連携する。

このように、IoT サービスの構造は、IoT 機器とサーバー上のプログラムがインターネットを介し通信し、連携することで成り立っている。図 2.1 は、IoT サービスの構造図である

### 2.3 IoT サービス維持の問題

IoT サービスは、IoT 機器とサーバー上のプログラムがインターネットを介し通信し合うことで成り立っている。IoT サービスを維持するためには、これらの構造を維持する必要がある。そのため、IoT 機器が正常に動作しているのか、通信が途切れていないか、監視する事が重要である。ところが、IoT 機器が多量に存在することや、IoT 機器が接続するネットワークが多様であることから、その監視には技術的困難がある。また、

個別の IoT サービスに組み込まれた監視システム等を別として、一般的な監視サービスも存在しない。

IoT サービスでは、多量の IoT 機器を使用する。そのため、個々の IoT 機器を識別し、適切に管理することが困難である。また、家庭内や屋外等、様々な環境下に置かれる IoT 機器が接続されるネットワークを予め予期することは困難を極める。接続先のネットワークでもプライベートアドレスを付与されるなどの他、IoT 機器とサーバーとの通信に制限がかかる場合もある。さらに、IoT 機器によっては、移動することを前提とした物もある。そのような場合、接続されるネットワークが頻繁に切り替わるため、機器の IP アドレスを利用した既存の機器監視手法は適応することができない。そのため、遠隔から機器の状態を確認することが難しい。

このように、IoT サービスの維持において、IoT 機器の動作や通信状態を監視することは重要であるが、その監視には技術的な困難がある。IoT サービスの円滑な提供や今後の発展の為には、機器が設置されるネットワークに関係なく、機器が多量であっても容易に監視可能な、IoT 機器の監視の実現が重要である。

## 2.4 従来の機器監視手法

### 2.4.1 サーバからの問い合わせによる監視

従来から機器監視に用いられてきた手法として、定期的に機器監視サーバから対象機器に状態を問い合わせる手法がある。機器監視サーバから、監視対象機器上のエージェントプログラムに、現在の状態を問い合わせる事で機器の監視を実現している。

この形では、機器監視サーバは、監視対象機器の IP アドレスを覚えておかねばならない。そのため、監視対象が接続されるネットワークが切り替わった場合、監視サーバ上に記録された IP アドレスを変更しなくては監視できない。監視対象が、多量且つ移動する IoT 機器の監視において、頻繁に監視サーバ上に記録された IP アドレスを書き換えるのは大変である。また、プライベートアドレスの利用時には、サーバーから IoT 機器への到達性が失われる可能性がある。

### 2.4.2 監視対象機器からの通知による監視

機器監視手法として、定期的に監視対象機器から機器監視サーバへ状態を送信するという手法がある。監視対象機器上のエージェントプログラムが、指定した時間毎に機器監視サーバへ状態を送信することで、機器の監視を実現している。

この形だと、機器監視サーバが監視対象の IP アドレスを覚えておく必要がなくなるため、監視対象が接続されるネットワークが切り替わった場合でも、追跡が可能である。ところが、エージェントプログラムの導入には、技術スキルと時間が必要である。また、サーバーの構築・設定も必要となる。

#### 2.4.3 ネットワークによる監視

そもそも監視対象機器がつながる為のネットワークを作ってしまおうという動きもある。既設の携帯電話網を利用して、IoT 機器用ネットワークサービスが展開されている。これは、IoT 機器向けのデータ通信を提供するサービスで、Web 画面から通信量や接続状況を確認することができる。携帯電話網を利用しているので、接続状況から機器の状態を推測できる。

しかし、そのネットワークを利用している場合に限り機器の状況を推測できるので、異なるネットワークに接続された機器と、専用ネットワークを用いた機器とを 1箇所で監視することが難しい。

#### 2.4.4 既存手法のまとめ

既存の手法として、サーバーからの問い合わせによる監視、監視対象機器からの通知による監視、ネットワークによる監視といった手法がある。

### 2.5 岡本商店街での事例

2015 年 12 月 8 日から 2016 年 2 月 26 日まで、NPO 法人コミュニティリンクへのインターンシップの一環として、実験を行った。人流観測を行い、結果を分析し商店街の活性化に役立てるという趣旨で行った。人流観測は、2016 年 2 月 7 日から 2016 年 3 月 14 日まで行った。岡本商店街とは、神戸市東灘区にある阪急岡本駅と JR 摂津本山駅の間にある商店街のことである。

人流観測とは、各地点から各地点迄をある時に移動した人数を観測するものである。通常は、観察員がカウンタを用いて数えるが、それでは各地点間を移動した人数はわかるが、その人が以前どの地点に居たのかはわからない。そこで、携帯電話についている Wifi 機能を利用し、観測を行うこととした。携帯電話の Wifi 機能は、無線 LAN の接続に使われるが、接続毎に Wifi 機能を有効にすることが手間なため、常時 ON にしている人も少なくない。携帯電話の Wifi 機能を有効にしている場合、携帯電話から接続可能な無線 LAN を探す為、プローブパケットというものが定期的に送出される。プローブパケットには、そのプローブパケットを送出した機器の物理アドレスが含まれている。そのプローブパケットを複数地点で観測し、含まれている物理アドレスを照合することで、携帯電話端末を持った人がどのように移動をしたのかが分かる。岡本商店街では、この原理を利用して、人流観測を行った。

本事例では、岡本商店街の 5 店舗を開発した観測機器を設置し、サーバにて蓄積・分析・可視化を行った。開発した観測機器は、RaspberryPi とampsence、rsync というソフトウェアを利用した。図 2.2 は、開発した観測機器である。

RaspberryPi とは、小型 PC の一つで安価入手が可能である。ampsence とは、プローブパケットを受信しファイルへ記録するソフトウェアである。図 2.3 は、携帯電話と観測機器、サーバーの関係を表している。



図 2.2: 岡本商店街人流観測 使用した機器

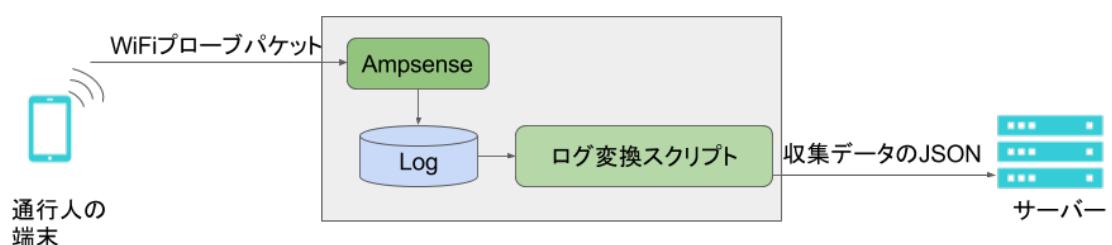


図 2.3: 岡本商店街人流観測 動作イメージ？

rsync とは、ディレクトリの中を同期させるプログラムで、遠隔にあるコンピュータの指定したディレクトリの中と、手元のコンピュータの指定したディレクトリを同期することができる。開発した観測機器では、定期的に rsync を実行するシェルプログラムを置き、起動時に自動で実行するよう設定した。それによって、各所に設置された観測機器の情報をサーバー上に集約・蓄積する。分析・可視化については、Web アプリケーションを作成した。分析については、作成したプログラムがとりおこない、可視化については、作成した Web アプリケーションが行っている。図 2.4 は作成した Web アプリケーションである。

## 岡本商店街動態情報可視化システム

2016年 1月 1日 現在の動態情報



図 2.4: 岡本商店街人流観測可視化アプリケーションスクリーンショット

実験としては、個々の RaspberryPi 内に、XX 分おきに次のような形式のデータをファイルとして書き出し、定期的に実行される rsync によって、サーバー側のディレクトリと個々の RaspberryPi のディレクトリを同期させることで収集した。

ampsence が書き出すデータは次のような形式の連なりとなっており、XX 分おきに新たなファイルとして書き出される。

---

このようなデータファイルを RaspberryPi に保存し、サーバー側のディレクトリと rsync を使用して同期することで、データの転送を実現した。

### 2.5.1 実験結果

2月 7 日から 3月 14 日の 35 日間観測を行ったが、全地点で問題なく観測できたのは、18 日間だけであった。

図 2.5、図 2.6、図 2.7 は、期間中の観測人数の推移を示す。

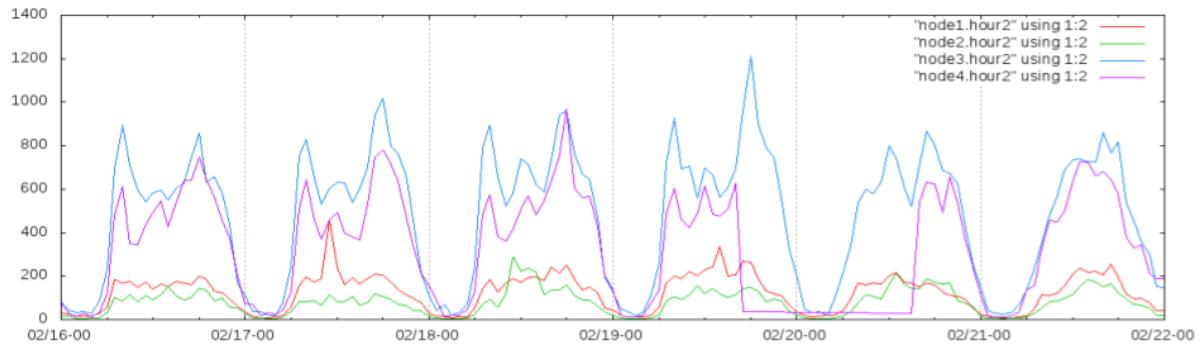


図 2.5: 岡本商店街人流観測 観測 1

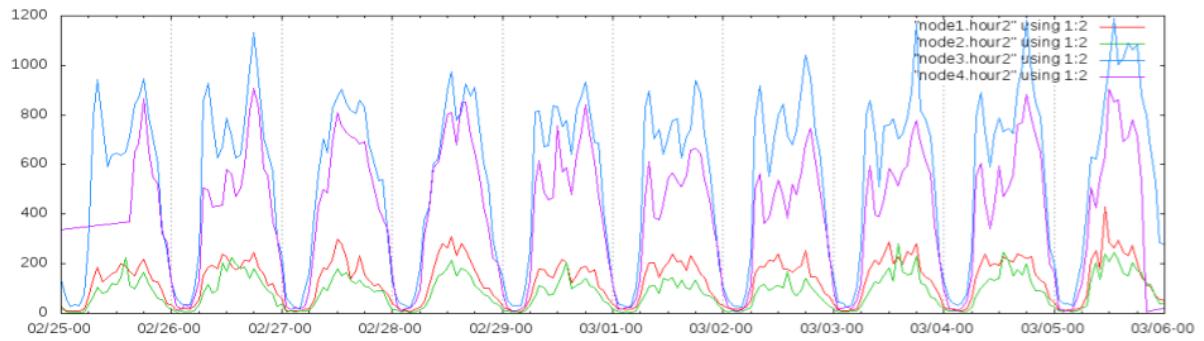


図 2.6: 岡本商店街人流観測 観測 2

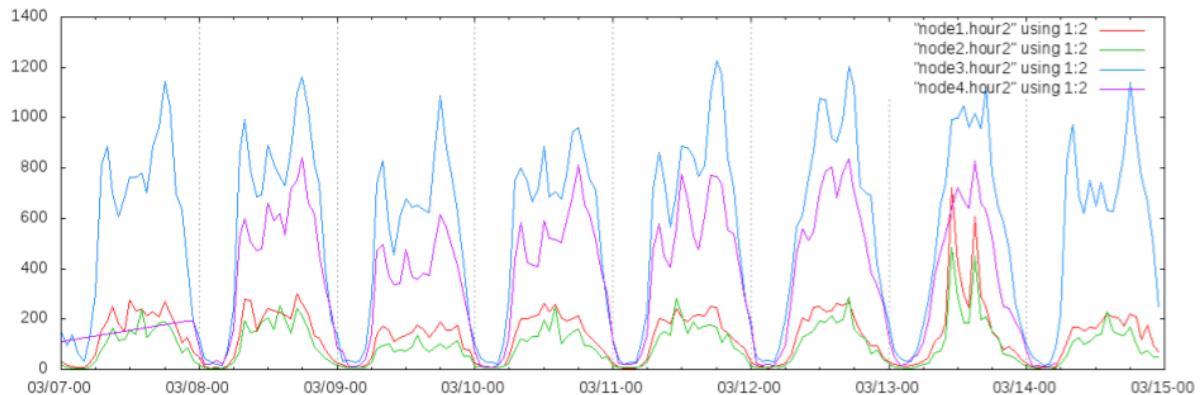


図 2.7: 岡本商店街人流観測 観測 3

この実験を通して、このような結果が得られた。どのような結果が得られたのか実験結果

## 2.5.2 課題点

しかし、実験を始めると、いくつか不具合が見つかった。まず、機器がネットワークに繋げていない事があつた。また、機器の電源そのものが入っていない事があった。そのため、分析が上手く動作せず、可視化が行えない事があった。

機器がネットワークに繋げていない場合、観測データは同期されず、機器内に蓄積されていくことになる。しかし、機器自体は、そもそもインターネットを介してサーバーに順次送信することを想定していたため、蓄積できるスペースは少ない。こうして溢れてしまった観測データは、蓄積されず捨てられてしまつたため、分析の際に観測データが揃わないという自体が発生した。

機器の電源がそもそも入っていなかつた場合、そもそも観測することができない。そのため、分析することもできない。

これら事象が発生した場合、分析アプリケーションでは、何も観測されなかつたものとして処理していたため、その分析データを可視化するアプリケーションでは、何も観測されていなかつたのか、機器に何か不具合があつたのか判断がつきにくかつた。

本実験では、定期的に可視化アプリケーションの画面を確認し、他地点では観測されているのに、該当地点では観測されていないと言つた場合に、何か不具合があつたものとして、現地に赴き対処した。

本実験としては、商店街に報告する義務があつたので、実験期間終了後、機器を回収し、機器内に保存されたデータを吸い上げ、分析するという作業をすることで、ある程度は補完できたものの、観測データが存在しない・不十分な曜日や時間帯については分析を行えず、最低限の分析結果しか出すことができなかつた。

しかし、各機器の動作状態がすぐに知る事ができれば、夜間等を除き、より良い分析が行えたのではないかと考える。

# 第3章 IoT 機器からの通知に基づく機器監視サービスの提案

## 3.1 IoT 機器の監視

IoT 機器の監視とは、動作の状態・通信の状態を確認することである。IoT 機器の動作の状態・通信の状態を確認し、記録することは、IoT サービスを維持する上で重要である。動作状態とは、少なくとも電源が入っているのかどうかで、通信状態とは、インターネットに接続され、サーバーと通信ができるのかどうかである。

このように、個々の IoT 機器の動作状態、通信状態を確認し、記録することが、IoT 機器の監視である。

## 3.2 IoT 機器監視の重要性と問題点

IoT 機器の監視は、IoT サービスを停止させないために重要である。IoT 機器が正常に動作していない事が分かった場合、正確に分析できないため、サービスを提供することができなくなる。そのために、常に IoT 機器を監視し、異常が有れば、復旧作業を行う必要がある。また、IoT 機器が正常に動作していない場合、分析を正確なものとするため、分析時に該当の IoT 機器を除外するか、全ての IoT 機器が動作している時間帯を選び分析を行う必要がある。そのため、IoT 機器を監視し、IoT 機器が正常に動作していない時間を記録しておく必要がある。

このように IoT 機器の監視は、IoT サービスを維持する上で重要である。しかし、従来より用いられてきた手法は、様々なネットワークに接続される IoT 機器の特性上用いることは難しい。また、機器が非常に安価になっていく事を考えると、機器の設定に手間を掛けたくない。

## 3.3 IoT 機器からの通知に基づく機器監視サービスの提案

そこで私は、IoT 機器の監視に特化した IoT 機器監視サービスのを提案する。IoT 機器から機器監視サーバに通知を送ることで、IoT 機器が接続されるネットワークによらない機器監視が行えると考えた。IoT 機器の監視に特化することで、機器の監視に手間をかけずにすむようになると考える。図 3.1 は提案サービスの図である。

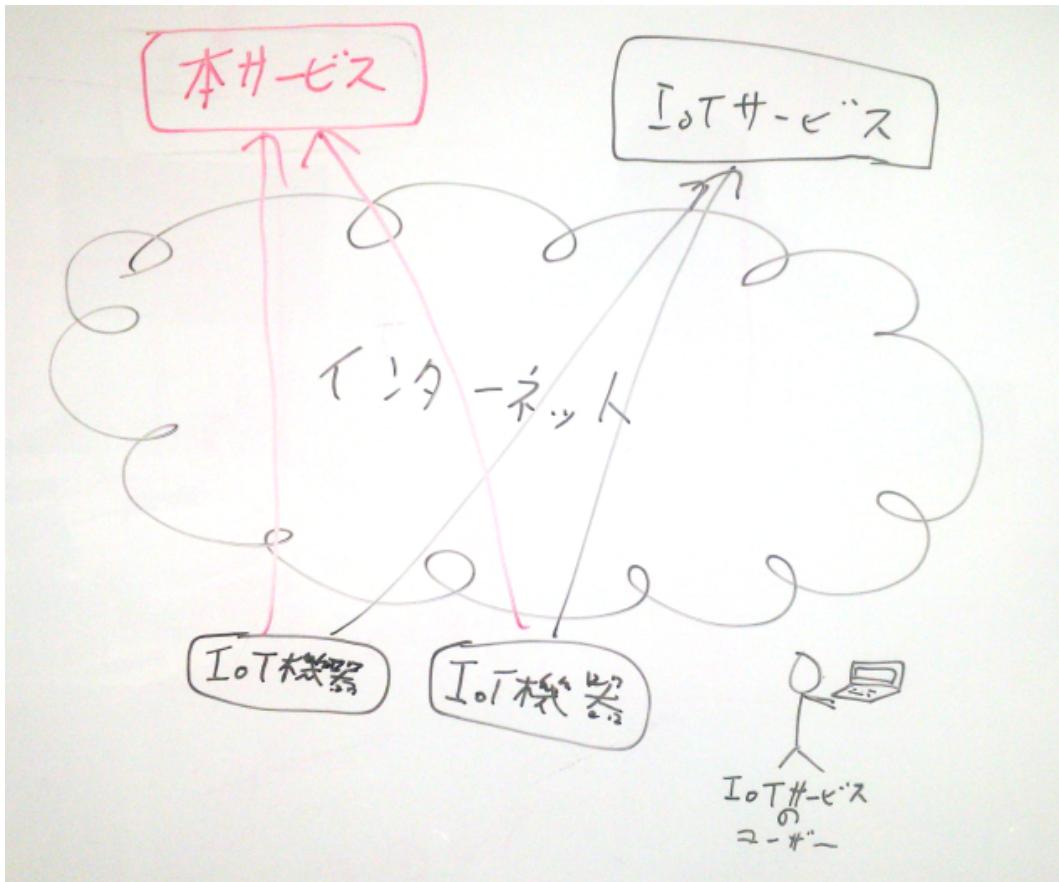


図 3.1: IoT サービスの構成図

### 3.4 IoT 機器の監視に必要な機能

IoT 機器の監視とは、個々の IoT 機器の動作状態、通信状態を確認することである。そのため、次のような機能が必要となる。

- IoT 機器の動作状態、通信状態を検知する機能
- IoT 機器の動作状態、通信状態を記録する機能
- IoT 機器の動作状態を一覧して表示する機能
- IoT 機器の過去の状態を確認する機能

IoT 機器の動作状態、通信状態を検知する機能とは、IoT 機器の現在の状態を検知する機能である。

IoT 機器の動作状態、通信状態を記録する機能とは、IoT 機器の状態を時刻と共に記録する機能である。

IoT 機器の動作状態を一覧して表示する機能とは、現在の IoT 機器の動作状態を見ることができる機能である。ある IoT 機器が動作しなくなった時に、動作していないことを明確に示す必要がある。

IoT 機器の過去の状態を確認する機能とは、過去の IoT 機器の動作状態と通信状態を確認することができる機能である。過去の動作状態や通信状態を整理し、いつ動作していて、いつ動作していなかったのか、各 IoT

機器ごとに示す必要がある .

このように , IoT 機器の監視には , IoT 機器の動作状態を一覧して表示する機能 , IoT 機器の過去の状態を確認する機能を持つ必要がある .

### 3.5 IoT 機器の監視における要件

IoT 機器の監視には , 上記機能が必要である他に , IoT 機器が設置されるネットワークにかかわらず監視ができることが求められる . 何故ならば , IoT 機器が設置されるネットワーク環境は多様であるからである .

IoT 機器が設置されるネットワーク環境として , 次のような環境が挙げられる .

- IoT 機器に対しプライベートアドレスが与えられる環境
- IoT 機器とサーバーの通信が制限される環境

多くの場合 , IP アドレスの節約の観点から , IoT 機器に対し , プライベートアドレスのみ与えられる . この場合 , インターネット側からは , IP アドレスを指定することが出来ないため , 通信は常に IoT 機器から始まる必要がある . また , IoT 機器とサーバーとの通信がセキュリティの観点から制限される場合もある . HTTP 等一般的に広く使用される通信を除いてブロックされる事がある .

このように IoT 機器の監視には , IoT 機器から通信が始まることや , HTTP 等頻繁に利用される通信を利用しなければならないといった制約がある .

これらを踏まえて , IoT 機器の監視には次のような機能要件が求められる .

- IoT 機器の動作状態 , 通信状態を検知する機能
- IoT 機器の動作状態 , 通信状態を記録する機能
- IoT 機器の動作状態を一覧して表示する機能
- IoT 機器の過去の状態を確認する機能

また , 非機能要件として , 次のような制約がある .

- プライベートアドレスが与えられたとしても動作する事
- インターネットの通信として一般に広く利用されている HTTP 等の通信を用いる事
- IoT サービスへの変更が無い事
- IoT 機器への設定が簡単である事

# 第4章 機器監視サービスの実装

## 4.1 機器監視サービスの構成

第3章にて述べた要件に基づき、システムを構築した。システムは、エージェントプログラム、機器状態データベース、機器情報データベース、エージェントプログラム用インターフェースプログラム、Web アプリケーションサーバ、Web アプリケーションから成り立っている。エージェントプログラムとエージェントプログラム用インターフェース、Web アプリケーションサーバと Web アプリケーションは、インターネットを介して通信しあう。図 4.9 は、システムのブロック図である。

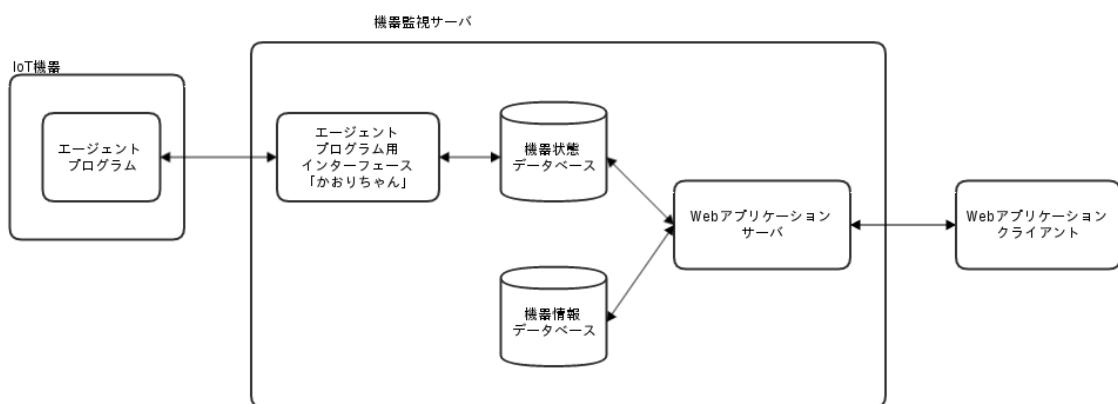


図 4.1: システムのブロック図

エージェントプログラムは、IoT 機器上で動作するプログラムである。定期的に自身の状態を状態蓄積システムへ送信する役割を果たす。定期的かつ自発的に状態を送信することで、ネットワーク環境によらない機器の監視を可能にした。IoT 機器として頻繁に使用される RaspberryPi を想定し作成した「かおりちゃん」とは、エージェントプログラム用インターフェースである。各 IoT 機器上で動くエージェントプログラムから送られてきた状態を、時刻と共に機器状態データベースへ書き込む役割を果たすプログラムで、機器監視サーバー上で動作する。機器状態データベースとは、機器の状態を時系列に沿って蓄積するデータベースである。機器監視サーバー上で動作する。機器情報データベースとは、機器 ID や、機器名、ユーザー名を記録するデータベースである。機器監視サーバー上で動作する。

Web アプリケーションサーバーは、Web ページや Web アプリケーション自体を配信する。ユーザーが利用するブラウザからの要求に答え、現在の機器の状態や、機器名等を返答する。

Web アプリケーションとは、ブラウザ上で動作するプログラムである。ユーザーからの入力を受け付け、ユーザーへ表示する他、必要な機能を問い合わせる。

これら各要素が連携することで、機器の監視を実現している。

#### 4.1.1 エージェントプログラム

エージェントプログラムとは、IoT 機器上にインストールされるプログラムである。エージェントプログラムの役割は、定期的に送信失敗回数を「かおりちゃん」へ報告することである。送信失敗回数とは、ネットワークの不具合等により、機器監視サーバーへ送信されなかった報告の数である。自発的に状態を報告するため、IoT 機器にプライベートアドレスが付与されても、状態を検知することができる。また、HTTP を用いるため、間のネットワークにてブロックされることがない。

#### 4.1.2 エージェントプログラム用インターフェース

エージェントプログラム用インターフェースとは、サーバー上で動くプログラムである。名前を「かおりちゃん」とした。エージェントプログラム用インターフェースの役割は、エージェントプログラムから送信されたメッセージを受け取り、現在の時刻と正常である旨を機器状態データベースへ書き込む。また、エージェントプログラムから送られた送信失敗回数から、IoT 機器がインターネットから切断された時刻を逆算し、機器状態データベースへ書き込む事も行う。

#### 4.1.3 機器状態データベース

機器状態データベースとは、サーバ上で動作するデータベースである。各 IoT 機器の状態を時刻とともに記録する。機器状態監視システムの中心にあるデータベースである。

#### 4.1.4 機器情報データベース

機器情報データベースとは、サーバー上で動作するデータベースである。各 IoT 機器の機器 ID、機器名、機器詳細情報、ユーザーのメールアドレスとパスワードを記録する。

#### 4.1.5 Web アプリケーションサーバ

Web アプリケーションサーバとは、サーバ上で動作するプログラムである。Web アプリケーションや Web ページの配信、Web アプリケーションからの要求の処理などを行う。必要に応じて、機器状態データベースと

機器情報データベースへアクセスを行う。次に、Web アプリケーションとのインターフェースを挙げ、それについて説明する。

#### ログイン API

Web アプリケーションから、メールアドレスとパスワードを受け取り、機器情報データベースのユーザー テーブルと照合する。照合した結果、ユーザー名とパスワードが合致したユーザーが存在すれば、HTTP クッキーにセッションキーをセットし、機器状態一覧ページへのリダイレクトを返す。合致したユーザーが存在しなかった場合、エラーメッセージを返す。

#### ログアウト API

Web アプリケーションに、HTTP クッキーから該当のセッションキーを削除するよう要求する。

#### 機器情報・機器状態取得 API

ログインチェックを行った後、機器情報データベースと機器状態データベースより、全 IoT 機器の機器情報と機器状態を返す。

#### 機器 ID 生成 API

ログインチェックを行った後、機器情報データベースに存在しない、ランダムな機器 ID を返す。

#### 機器 ID 重複チェック API

ログインチェックを行った後、Web アプリケーションから機器 ID を受け取る。機器情報データベースに該当の機器 ID が存在するかしないかを返信する。

#### 機器作成 API

ログインチェックを行った後、Web アプリケーションから、機器 ID、機器名、機器の詳細と、機器の作成なのか編集なのかの指示を受け取る。機器の作成であった場合、機器 ID に重複が無いことを確認したうえで、機器情報データベースに該当のエントリを作成・機器状態データベースにメジャーメントを作成する。作成されたか、されなかったかを返信する。機器の編集であった場合、機器情報データベースから、受け取った機器 ID を持つものを探しだし、編集する。編集できたか否かを返信する。

## 機器削除 API

ログインチェックを行った後，Web アプリケーションから，機器 ID を受け取る．機器情報データベースから，受け取った機器 ID を持つものを削除する．その後，機器状態データベースから，メジャーメントを削除する．削除されたか否かを返信する．

## 過去の機器状態取得 API

ログインチェックを行った後，機器状態データベースから，全ての IoT 機器の過去の状態をまとめ，返信する．

### 4.1.6 Web アプリケーション

Web アプリケーションとは，ユーザーのブラウザ上で動作するアプリケーションである．ユーザーへグラフィカルインターフェースを提供する．必要に応じて，Web アプリケーションサーバへ必要な情報を要求する．Web アプリケーションは，3 つの Web ページから成り立っている．以下に各ページとそれぞれの役割を述べる．

- ログインページ

正当なユーザーであることを確認するために，メールアドレスとパスワードの入力を求める．メールアドレスとパスワードは，Web アプリケーションサーバへ送信される．図 4.2 は，ログインページのスクリーンショットである．

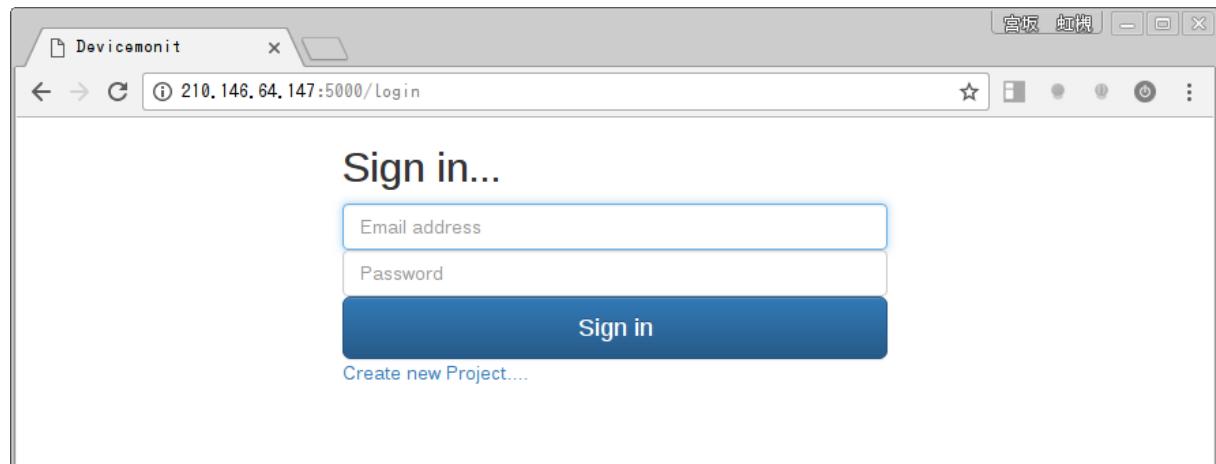


図 4.2: ログイン画面

- 機器状態一覧ページ

機器状態を一覧して表示する他，機器の作成や，機器情報の編集，機器の削除等を行う事ができる．現在の機器の状態を取得するため，定期的に Web アプリケーションサーバと通信する．また，機器の作成や機器情報の編集，削除の為，Web アプリケーションサーバと通信する．

図 4.3・図 4.4・図 4.5 はこのページのスクリーンショットである。それぞれ、一覧表示・縮小一覧表示・詳細な情報の表示をしている。図 4.6・図 4.7 は、それぞれ、機器追加ダイアログ・機器情報編集ダイアログのスクリーンショットである。

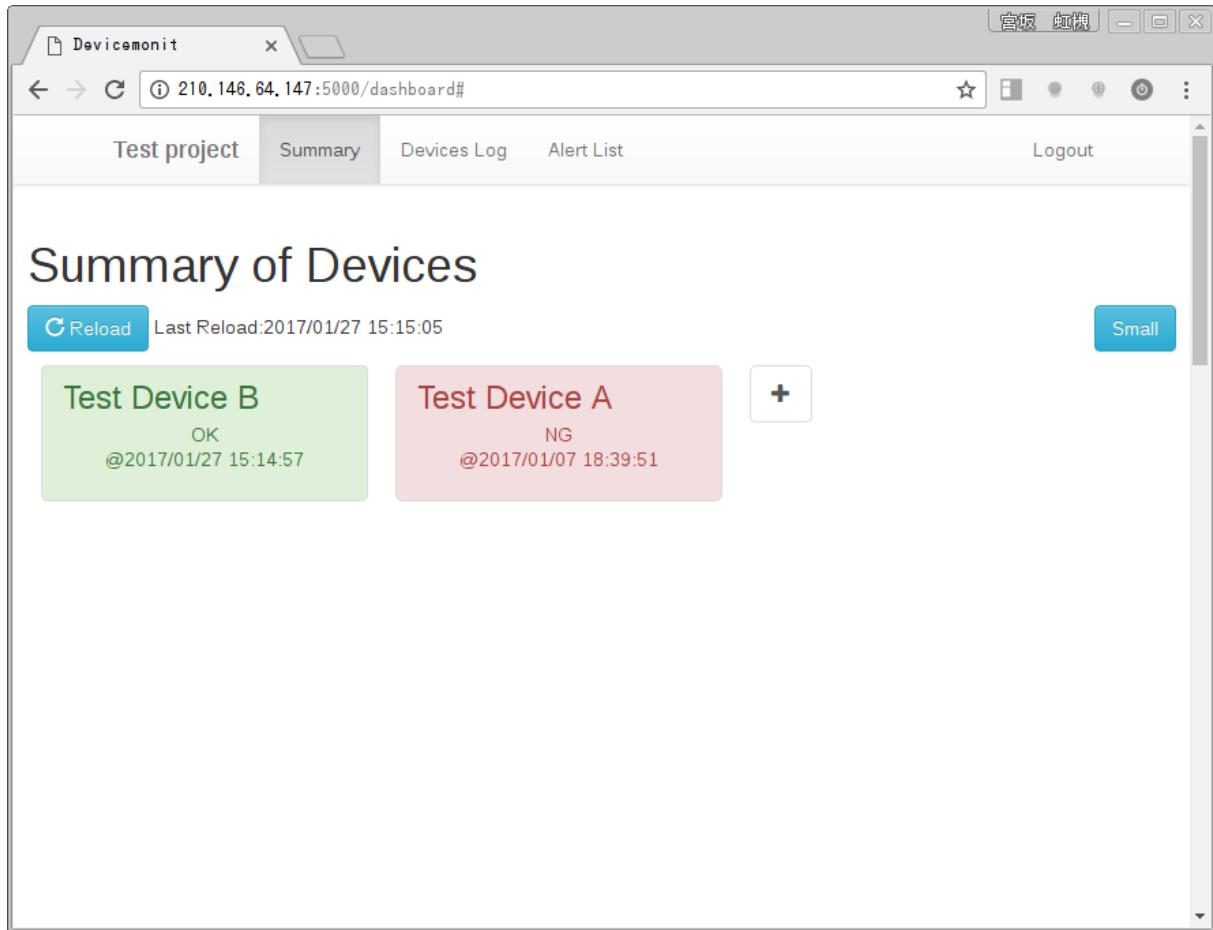


図 4.3: 機器状態一覧画面

- 過去の機器状態一覧ページ

過去の機器状態を時刻と共に整理し、一覧表示するページである。現在の機器の状態を取得するため、定期的に Web アプリケーションサーバと通信をする。図 4.8 は、スクリーンショットである。

## 4.2 機器監視サービスの実装

### 4.2.1 エージェントプログラムの実装

エージェントプログラムの役割は、送信失敗回数を定期的に IoT 機器監視サーバーに送信することにある。約 1 分おきに、現在の送信失敗回数と過去の送信失敗回数を「かおりちゃん」へ送信する。どのような Linux 環境でも動作することを考え、Shell スクリプトにて実装した。

エージェントプログラムの動作は次のようになる。

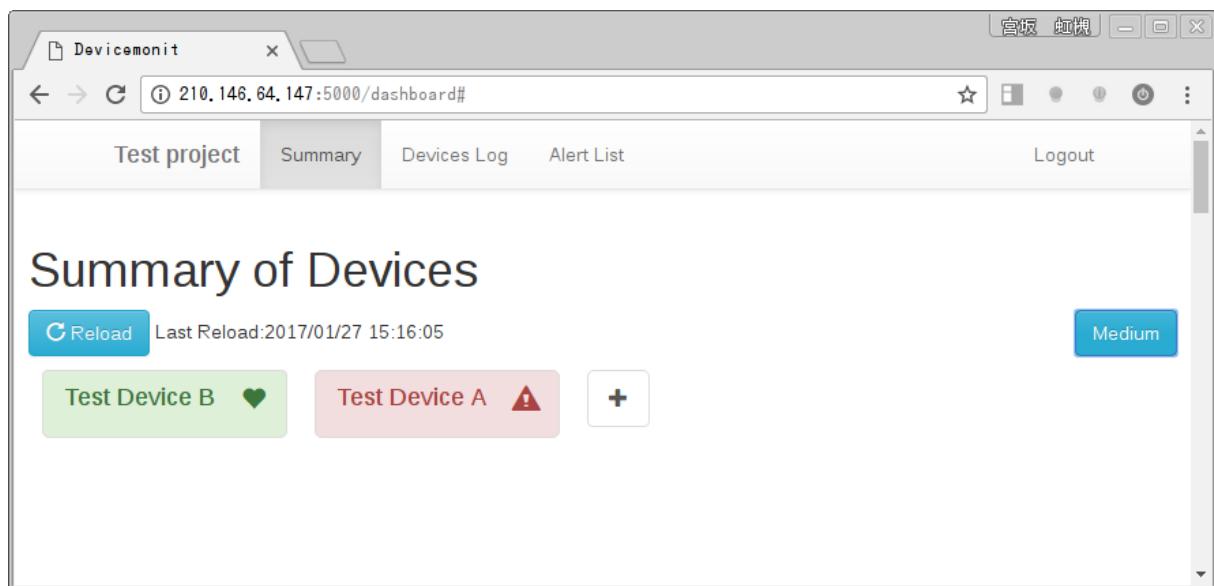


図 4.4: 機器状態一覧画面（小さく表示）

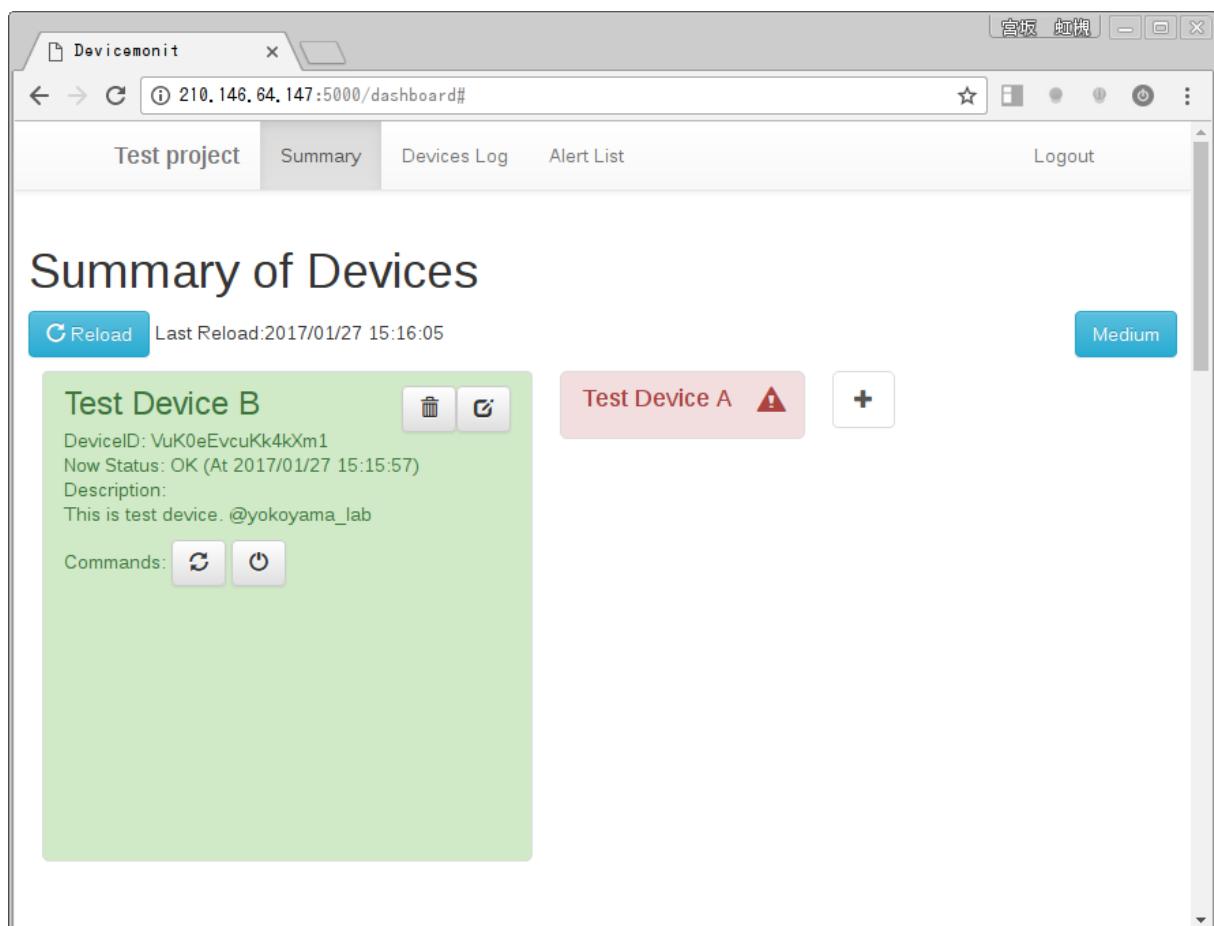


図 4.5: 機器状態詳細表示

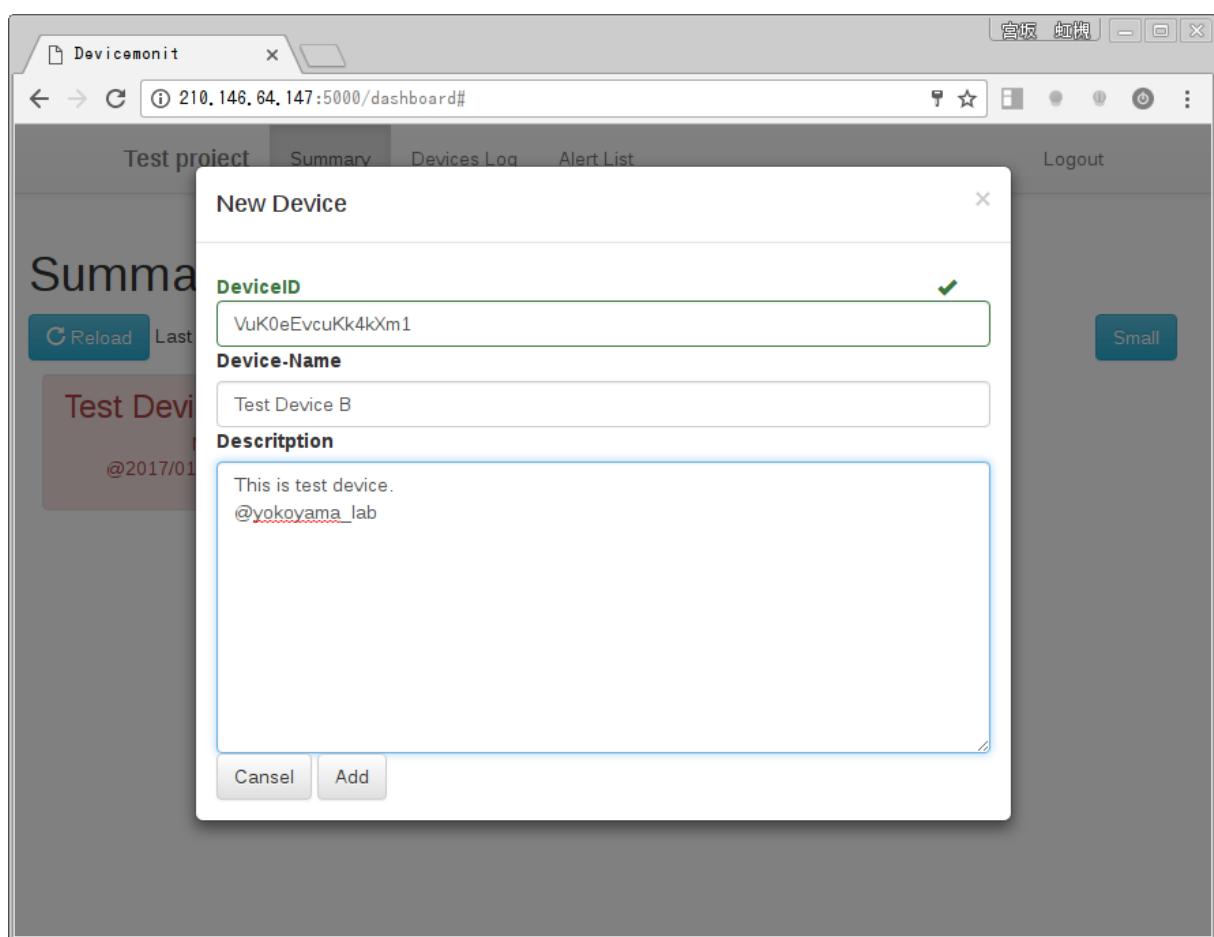


図 4.6: 機器追加ダイアログ

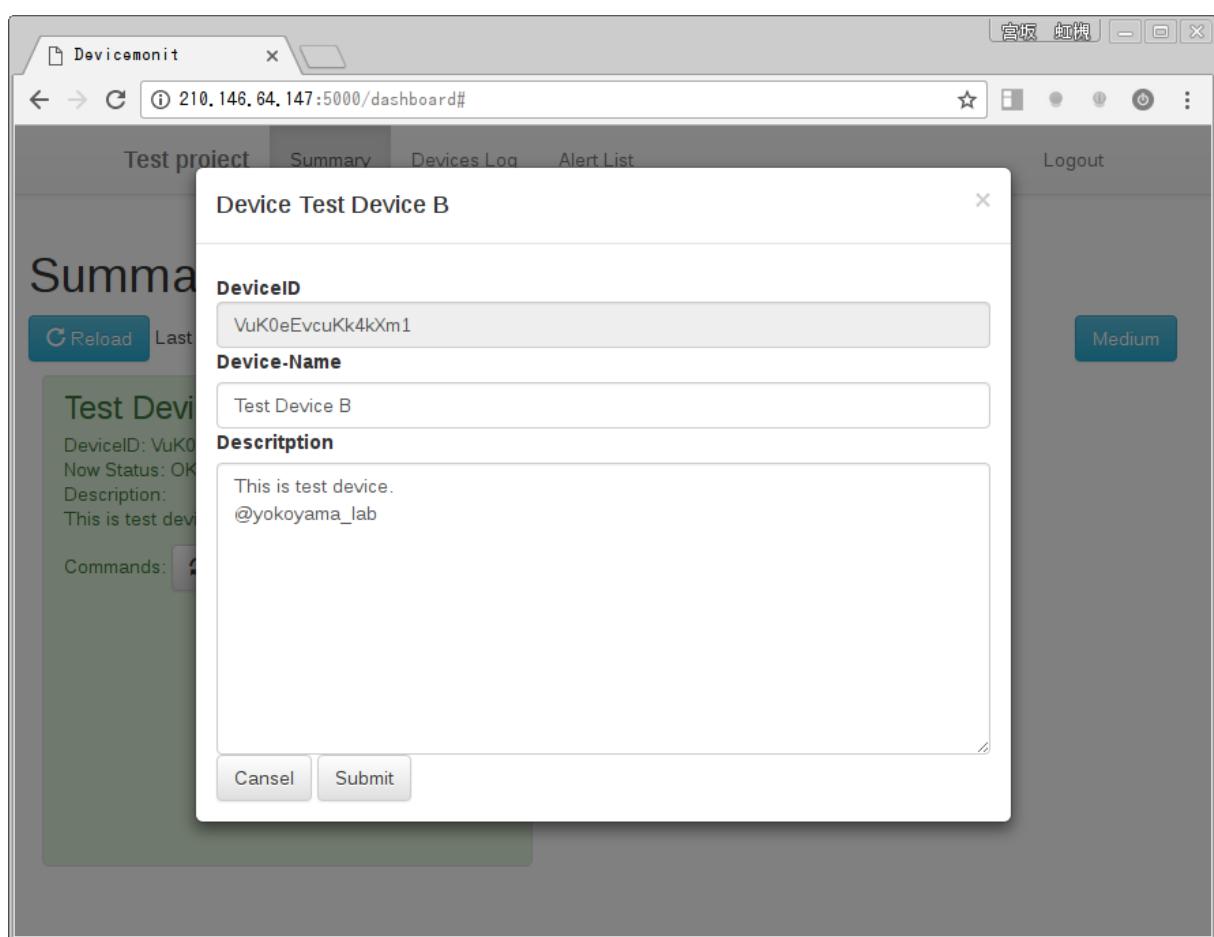


図 4.7: 機器情報編集ダイアログ

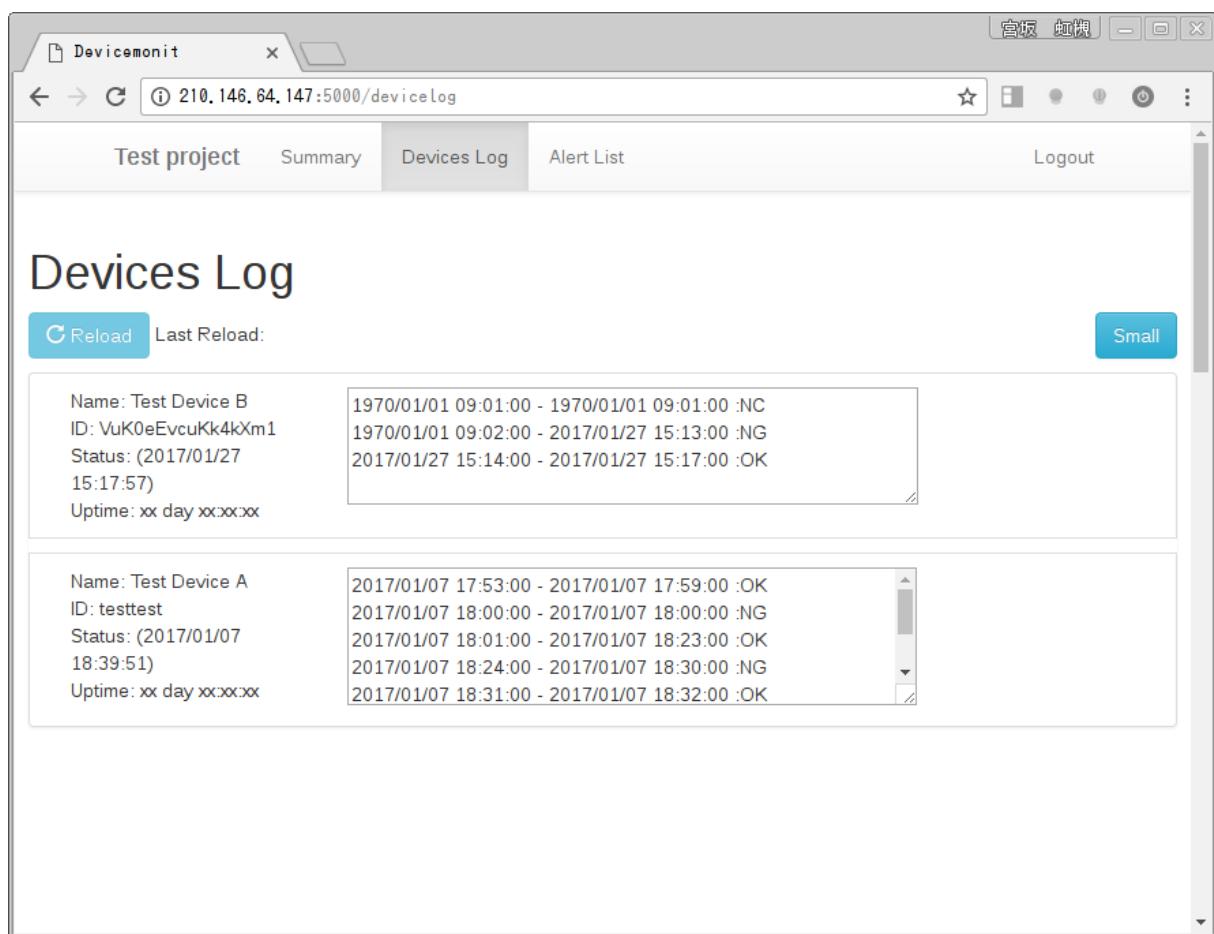


図 4.8: 過去の状態表示ページ

1. エージェントプログラムが起動されると，まず過去に記録された送信失敗回数を読み出す．
2. 「かおりちゃん」に対し，過去に記録された送信失敗回数と，現在の送信失敗回数を送信する．
3. サーバーから応答があった場合，過去に記録された送信失敗回数と，現在の送信失敗回数をクリアしする．  
サーバーから応答がない場合，現在の送信失敗回数をインクリメントする．
4. 1分間スリープし，再び2から繰り返す．

#### 4.2.2 エージェントプログラム用インターフェース「かおりちゃん」の実装

「かおりちゃん」の役割は，エージェントプログラムから送信失敗回数を受け取り，時刻と正常である旨を機器状態データベースへ書き込むこと，また，エージェントプログラムから送られた送信失敗回数から，インターネットより切断された時刻を逆算し，機器状態データベースへ書き込む事である．Falcon と呼ばれる API の作成に特化したフレームワークを使用し，Python にて実装した．また，機器状態データベースへの書き込みには，InfluxDBClient というライブラリを使用した．

#### 4.2.3 機器情報データベース

機器情報データベースの役割は，機器の名前，機器の詳細説明，機器 ID，ユーザー ID，ログイン用メールアドレスとパスワードを記録し保持する事である．ユーザ ID とは，システムにてユーザーを識別するための識別子である．機器 ID は，システムにて IoT 機器識別するための識別子である．機器情報データベースには，SQLite3 を用いた．機器情報データベースには，次のようなテーブルが用意されている．

##### 機器情報テーブル

機器 ID，ユーザー ID をキーとして，機器の名前，機器の詳細説明を記録し保持するテーブルである．

##### ユーザーテーブル

ユーザー ID をキーとして，ユーザー名とパスワードを記録し保持するテーブルである．

#### 4.2.4 機器状態データベース

機器状態データベースの役割は，機器の状態を時刻と共に記録・保持することである．機器状態データベースには，Influxdb を用いた．機器 ID をメトリクス名（テーブル名）とし，時刻をキーとして，機器の状態を記録している．

#### 4.2.5 Web サーバーアプリケーション

Web サーバーアプリケーションの役割は、与えられた HTTP リクエストを元に、Web ページや、各種情報を返却することにある。Flask と呼ばれる Web アプリケーションフレームワークを用いた。Python を使用している。Web サーバーアプリケーションの設計と Web アプリケーションの設計から、下記の様に実装した。先頭に付いている GET や POST は HTTP メソッド、/login 等は URL を示している。

**GET /login**

ログインページを返す。

**POST /login**

メールアドレスとパスワードを受け取る。クッキーからセッションキーを探し、存在すれば既にログインしているとして、/dashboard へのリダイレクトメッセージを返す。データベースにメールアドレスとパスワードの組が存在するか確認し、存在した場合、セッションキーを返す。存在しなかった場合、ログインエラーページを返す。

**GET /logout**

セッションキーを受け取り、該当のセッションを削除する。

**GET /dashboard**

ログインチェックをし、機器状態一覧ページを返す。

**GET /devicelog**

ログインチェックをし、過去の機器状態一覧ページを返す。

**GET /api/device/all**

ログインチェックをし、現在の状態、最後にメッセージを受け取った日時、機器 ID、機器名、機器詳細のデータを、JSON 形式にまとめたものを返す。

**GET /api/deviceID**

ログインチェックをし、ランダムにデバイス ID を生成し、JSON 形式にまとめたものを返す。

**POST /api/deviceID**

ログインチェックをし、受け取ったデバイス ID が既に存在するか確認し、その結果を JSON 形式にまとめ、返す。

**POST /api/device/{DeviceID}**

ログインチェックをし、デバイス ID と受け取った JSON データから、デバイスの新規作成・編集をし、結果を JSON 形式にまとめ、返す。

**DELETE /api/device/{DeviceID}**

ログインチェックをし，該当のデバイス ID を持つデバイスをデータベースから削除する。

#### 4.2.6 Web アプリケーション

Web アプリケーションの役割は，ユーザーインターフェースを提供することである。Bootstrap,JQuery というライブラリを用いて作成した。HTML,CSS,Javascript で書かれている。定期的に Web アプリケーション サーバから状態を取得し，HTML,CSS を用いて表示する。

#### 4.2.7 エージェントプログラムと「かおりちゃん」間の通信の実装

エージェントプログラムと「かおりちゃん」は，インターネットを介して，HTTP というプロトコルを用いて通信する。エージェントプログラムと「かおりちゃん」間の通信は次の様な形になる。

1. エージェントプログラムが「かおりちゃん」に対し，送信失敗回数を報告する。
2. 「かおりちゃん」は，時刻と正常である旨を機器状態データベースへ送信する。
3. 機器状態データベースより，正常に書き込んだというメッセージが返ってくる。
4. 「かおりちゃん」は，必要に応じて送信失敗回数からインターネットから切断された時刻を逆算し，その時刻と切断されていた旨を機器状態データベースへ送信する。
5. 機器状態データベースより，正常に書き込んだというメッセージが帰ってくる。
6. 「かおりちゃん」は，エージェントプログラムに対し，正常に受け付けたというメッセージを返す。

エージェントプログラムと「かおりちゃん」の間の通信は，約 1 分おきに繰り返される。

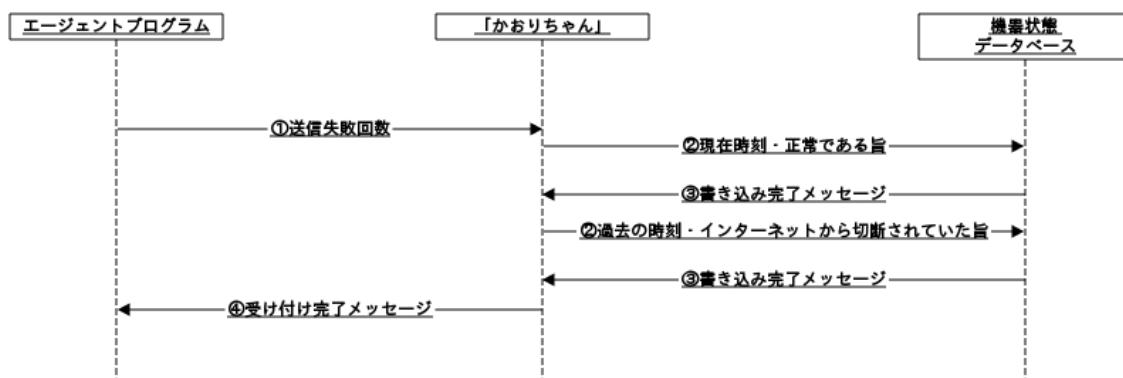


図 4.9: エージェントプログラムと「かおりちゃん」の間のメッセージシーケンス図

HTTP 上でやり取りするデータの形式としては、Javascript Object Notation(JSON) という形式を用いる。  
エージェントプログラムからは、次のような形式でメッセージが送られる。

---

ソースコード 4.1: エージェントプログラムから「かおりちゃん」に送られるメッセージの書式

---

```
1 {"seq":<NOW>, "stat":"OK", "log":{"seq":<PAST>}}
```

---

< NOW > には、現在の送信失敗回数が入る。< PAST > には、過去の送信失敗回数が入る。

また、「かおりちゃん」からは、次のような形式で応答がある。

---

ソースコード 4.2: 「かおりちゃん」からエージェントプログラムへの応答の形式

---

```
1 {"stat":"OK", "time":<NOWTIME>}
```

---

ここで、< NOWTIME > にはサーバー側の現在時刻が入る。

#### 4.2.8 Web アプリケーションサーバと Web アプリケーション間の通信の実装

Web アプリケーションサーバと Web アプリケーションは、インターネットを介し、HTTP というプロトコルを用いて通信する。HTTP 上でやり取りするデータの形式としては、Javascript Object Notation(JSON) という形式を用いる。

### 4.3 サービスによる監視のイメージ

本サービスで想定するユーザーの動きをまとめる。

- ユーザーが IoT 機器をサービスに登録する場合
  1. ユーザーは最初にサービスにログインし、機器一覧ページへ遷移する。
  2. ユーザーは機器一覧ページにある「+」ボタンを押す。すると、機器追加用ダイアログが表示される。
  3. 機器 ID が既に決まっている場合は、機器 ID 欄を書き換える。
  4. 決まっていない場合は、機器 ID 欄に表示されている機器 ID をメモしておく。
  5. 機器の追加ボタンを押し、機器一覧に追加した機器が表示されていることを確認する。
  6. ユーザーは、IoT 機器へエージェントプログラムをインストールし、自動でエージェントプログラムが起動するよう設定する。
  7. ユーザーは、IoT 機器をインターネットに繋ぎ、サービスの機器状態一覧ページの表示が変わったことを確認する。

- ユーザーが IoT 機器の情報を変更する場合

1. ユーザーは、サービスにログインし、機器一覧ページへ遷移する。
2. 該当の機器をクリックし開く。
3. 該当の機器の編集ボタンを押すと、機器情報編集ダイアログが表示される。
4. 機器名、機器詳細を編集し、OK ボタンを押す。

- ユーザーが IoT 機器を削除する場合

1. ユーザーは、サービスにログインし、機器一覧ページへ遷移する。
2. 該当の機器をクリックし開く。
3. 該当の機器の削除ボタンを押すと、確認ダイアログが表示されるので、OK を押す。
4. 機器一覧ページから、機器が消えたことを確認する。

- ユーザーが現在の機器の情報、機器の状態を確認する場合

1. ユーザーはサービスにログインし、機器一覧ページへ遷移する。
2. 該当の機器をクリックし開くと、機器の情報がと、現在の状態、最後に通信があった日時等が表示される。
3. また、機器一覧ページにて機器の背景が、緑色である場合は正常で、赤色である場合は異常である。

- ユーザーが過去の機器状態を確認する場合

1. ユーザーは、サービスにログインし、機器一覧ページへ遷移する。
2. ナビバーから過去の機器状態ページへ遷移する。
3. すると、機器と機器の過去の状態一覧が表示されるので、該当の機器を見つけ出し、確認する。

# 第5章 機器監視サービスのユーザーテストと考察

本システムの機能検証の為に，ユーザーテストを行った．本章では，ユーザーテストの結果を報告し，考察する．

## 5.1 機器監視機能のテスト

必要となる機能が実装されていることを確認するため，検証を行った．

### 5.1.1 実験のシナリオ

ユーザーテストは，1台のIoT機器で成り立っているIoTサービスを想定し行った．IoT機器には，RaspberryPiを使用する．図5.1は使用したRaspberryPi2と，IntelEdisonである．今回は、IntelEdisonは使用していないが，参考の為に上げた．



図 5.1: IoT サービスの構成図

RaspberryPi2は無線LANインターフェースを持たないので，バファロー製の無線LANインターフェースを使用した．

期間は2017年1月28日正午から1時間行い，途中何度かIoT機器(RaspberryPi)の電源を抜き，正常に検知することを確認する．その後，電源が抜けていた期間について，IoT機器の記録を確認する．

### 5.1.2 実験結果

IoT 機器の監視については問題なく動作した事を確認した。機器の電源を抜くと、1 分程度で表示状態が変わり、数分後、機器の電源を再び入れると、2 分程度で表示状態が戻ることを確認した。

## 5.2 IoT 機器を監視までの手順の比較

従来手法による監視の手間と、本サービスによる監視の手間を比較する。ここでは、従来の手法として、Telegraf と Influxdb、Grafana を用いた機器状態の監視システムを比較対象に挙げる。  
順に結果を報告する。

### 5.2.1 機器監視サーバーの構築

従来の手法では、機器監視をするために、次の様な手間があった。

1. 機器監視の為にサーバーを用意する
2. 機器監視サーバへ Influxdb をインストールする
3. 機器監視サーバへ Grafana をインストールする
4. Influxdb に対し、Telegraf との通信に用いるインターフェースの設定を行う
5. Grafana に対し、Influxdb と連携するための設定をする。

本サービスを用いる場合、これら準備は不要となる。

### 5.2.2 各 IoT 機器の状態を可視化する

従来の手法では IoT 機器の状態を可視化するまでに以下の手順を踏んでいた。

1. IoT 機器に Telegraf をインストールする
2. IoT 機器にインストールされた Telegraf の接続先の設定を行う
3. 各 IoT 機器固有の ID を Telegraf に設定する。
4. Grafana にログインする。
5. Grafana へ、現在の状態を表示する為のデータベースクエリを入力する。
6. Grafana へ、過去の状態を表示するためのデータベースクエリを入力する。

7. 各描画パネルに対し，機器名を設定する .

本サービスでは次の手順を踏む .

1. 本サービスにログインする
2. 本サービスの機器追加ボタンを押し，機器 ID をメモ（コピー）する .
3. 本サービスに対し，機器名と機器詳細を入力する .
4. IoT 機器にエージェントプログラムをインストールする .
5. IoT 機器にエージェントプログラムを自動で起動するよう設定する .

比較すると，Grafana ヘデータベースクエリを入力する作業が，なくなっている．代わりに，自動起動の為の設定ファイルを設定する手順が増えている .

### 5.3 考察

ユーザーテストから本システムは，ある程度有効であることが分かったが，以下の点について課題があることが分かった .

- 機器への設定について

設定ファイルの編集等の手間は削減されたが，エージェントプログラムを IoT 機器にインストールするのに手間がかかっていることが分かった．従来手法を用いて機器の監視をする際は，自動起動の設定を行う必要はなかったが，本プログラムでは必要としている．自動起動の為の設定ファイルを自動生成するか，あるいはエージェントプログラム配布の際，簡単なインストールスクリプト等と一緒に配布することで，大きな手間の削減になると感じている .

- ユーザーインターフェースについて

現在，過去の機器状態の記録については，文字記録として表示しているが，グラフ表示等の方が見やすいと感じた．また，期間を指定して閲覧できる機能も必要であることが分かった .

- エージェントプログラムについて

本サービスで提供しているエージェントプログラムは，その他のパッケージに依存しない単純な構成であるため，移植性が高い .

## 第6章 おわりに

IoT とは、様々なモノにコンピュータを取り付け、インターネットを介して相互に情報をやり取りすることで、様々な自動化を図ろうという概念である。IoT サービスとは、IoT による利便性をユーザーに提供するもので、IoT 機器とサーバーのプログラムがインターネットを介して通信し合うことで成り立っている。そのため、IoT サービスの円滑な提供のためには、IoT 機器の監視は不可欠である。

しかし、IoT 機器が多量に存在することや、IoT 機器が接続するネットワークが多様であることから、IoT 機器の監視には技術的困難がある。IoT サービスでは、多量の IoT 機器を使用するため、個々の IoT 機器を識別し、適切に管理することは難しい。また、IoT 機器が接続されるネットワークを予測することは困難である。そのため、機器の IP アドレスを使用した既存手法を適応することは現実的ではない。

そこで、本研究では、IoT 機器から通知を送ることで、IoT 機器が接続されるネットワークによらない機器の監視を可能にすることを提案した。また、機器の監視に必要な要件を抽出し、機器の監視に特化した監視サービスを提案した。この機器監視サービスを実現する為、IoT 機器にインストールされるエージェントプログラム、機器監視サーバ上で動作するエージェントプログラム用インターフェース「かおりちゃん」、Web アプリケーションを作成した。

必要な機能を持っているのか検証を行い、オープンソース利用の既存監視手法と手順について比較した。その結果、IoT 機器の動作状態について監視できていることを確認した。また、既存監視手法との比較では、手間がかからなくなった事を確認した。今後の課題として、IoT 機器への設定の簡略化や、ユーザインターフェースの向上、様々な IoT 機器への対応があることがわかった。

## 第7章 謝辞

本論文は、著者が神戸情報大学院大学情報技術研究科情報技術専攻在学中に、横山研究室にて行った研究をまとめたものです。本研究に関してご指導ご鞭撻を頂きました横山輝明講師に心より感謝申し上げます。また、本論文をご精読頂き、有用なコメントと励ましをくださった藤原明生准教授に深謝致します。

そして、論文を書く際にアドバイスをくださった、嶋教授と、嶋研究室の渡邊香織さん、田頭潤さん、笠谷拓伸さん、また、研究についてアドバイスを頂いた横山研究室OBの鄒曉明さんと、良き議論相手である京都産業大学大学院M1の石原真太郎さんに感謝致します。

## 参考文献

- [1] eCoPA <http://ecopa.in/>
- [2] 平成 27 年版 情報通信白書（総務省）「IoT の実現に向けたアプローチと我が国 ICT 産業の方向性」より  
<http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h27/html/nc254150.html>
- [3] 6 割が「IoT は流行語」－エスキューピズム調査（ZDBet Japan）<http://japan.zdnet.com/article/35093272/>
- [4] 先進テクノロジのハイブ・サイクル 2011 年 Gartner <https://www.gartner.co.jp/press/html/pr20110907-01.html>