# contents

# COVER Illustration

illustrations :tea

# Page XX ~ XX

*GPU*で機械学習をやってみる

edit :ksakiyama

# Page XX ~ XX

*Arduino*に恋する夏

edit :MofuMofu

# Page XX ~ XX

*"What's IoT ?"* --初めに知っておきたい*"IoT"*のコト

edit :MofuMofu

# Page XX ~ XX

*Arduino*に恋する夏

edit :MofuMofu

# Page XX ~ XX

*title*

edit :keigodasu

# Page XX

*Afterword*

<<<

# はじめに

## この本をお買い上げいただきありがとうございますーーーーー！！！！

どうもこんにちは、暑すぎて夏バテ気味のMofuMofuと申します。
今回初めてコミケに申し込んだのですが、普段全くくじ運がないのに当選してしまい
本人が一番驚いている次第です。

初サークル参加がコミケとか、正直結構不安です……。
でも、ノリと気合いでなんとか本を仕上げることができましたー！ぱちぱちぱち〜〜！！

なんとなんと、今回は初めてなのに合同誌になってます、みんな自分の好きな分野について
あつーく語ってくださってるみたいです！
いやー楽しみになってきたなーワクワクもんだあ！

……というわけで前置きが長くなりましたが、楽しんでいただければ幸いです！

# GPUで機械学習～うわっ・・・私のモデル、正解率低すぎ・・・？～

# はじめに

昨年にGoogleがTensorFlowをリリースしたのをきっかけに、Deep Learning（以下DLと略）関連の記事がQiitaやはてなブログで増えてきました。しかしそこまで普及しても、人によっては「GPU？低レイヤ？C++なの？難しそう」といった印象を持たれている人も多いようです。なので本記事では、GPUのわかりやすーい説明とChainerを使ったモデル判定をやってみようと思います。

# GPUと機械学習

# GPU

「そもそもGPUってよくわかってないんだけど？」という方もけっこういるみたいです。GPUとはGraphics Processing Unitの略称です。「CPUと何が違うの？」みたいなこともよく聞かれます。簡単に書くと、GPUは画像処理専用のプロセッサのことです。ここでは、画像処理とはCG（コンピュータグラフィックス）とか3Dゲームとかと思ってください。CPUがコンピュータ全体を制御する中枢で、GPUがお絵かきやゲームを表示したりしてくれる部品です。よく「グラボ」と言われているパーツがあると思いますが、あのグラボの中にGPUが組み込まれています。


*Figure 1.* グラフィックスボード

GPUはゲームなどのCGを高速に描画するために進化してきました。例えば物体の移動・光の反射による色の変化など、滑らかに表示するためにそれ

らを高速に計算する必要があるからです。これらはベクトルの計算で、ベクトルの計算は数式だと行列の加算や乗算で表現できます。つまり、GPUは行列の演算を高速に行うことができるのです。また、行列の計算は行列の各要素に対して同じ計算を実施することとなります。つまり、GPUは同じ計算を大量のデータに対して実施することに特化しています（図２）。
もちろんCPUでも演算は可能ですが、GPUほど特化した構造になっていないため、3Dゲームをプレイしようとしても、計算が間に合わずカクついたり、そもそもプレイできないこともあります。

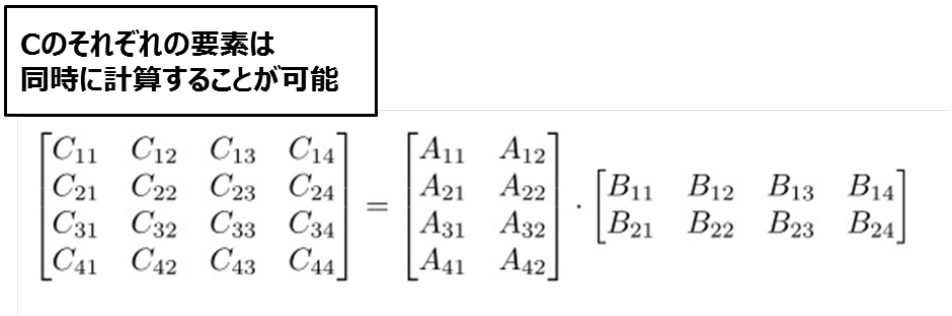



*Figure 2.* 行列の演算

# GPGPU

「そんな3Dゲームに使う代物が、なんで機械学習とかDLの分野でよく出てくるの？」と思っている方もいるでしょう。その前に、GPGPUという技術について説明します。
GPGPUとは、General Purpose computing on GPUの略です。GPUコンピューティングとも呼ばれたりします。これは、GPUにCG以外の計算を実施してもらう応用技術です。前節で述べたように、GPUは大量のデータに同じ計算を素早く実施することに特化しています。ゲームだけにこの演算パワーを使うのはもったいないので、数値計算にも使ってしまえば、並列数が多い計算ではCPUより高速に結果を得られることがあります。
GPGPUは、最初はCGの命令を上手く使って計算を行うという、変わったプログラミング方法で実現していました。しかし、NVIDIAがCUDAというGPGPU用の開発環境を提供してくれたおかげでGPGPUの敷居が下がり、様々な分野でGPUを利用する場面が増えました。

## CUDAとOpenCL

ちなみに、筆者がよくGPGPUのアプリケーションを開発するときによく使っていたのは、OpenCLというフレームワークです。CUDAがNVIDIA専用の開発環境というのに対し、OpenCLはNVIDIA・AMDの両GPUでGPGPUが使えるというメリットがありました。しかし、OpenCLのプログラミングは非常に難易度が高く、容易に使えるものではありませんでした。また、GPGPUではAMDよりNVIDIAのほうがパフォーマンスチューニングがしやすかったり、デバッグツールなどが整っていたため、あまり普及することなく今はCUDA一強となってしましました。（FPGAなど組込の分野ではOpenCLは使われたりするそうです。）

# 機械学習とGPUの関係

機械学習になぜGPUがよく使われるのか簡単に説明します。
機械学習や流行りのDLの計算のほとんどは行列の演算になります。前節で述べたように、GPUは行列のような各要素に同じ計算を実施するような処理が得意です。

# Chainer

Chainer（http://chainer.org/）とは、Pythonのニューラルネットワーク用フレームワークです。国産のOSSとして知られていて、DLのコーディングがシンプルに書けるのが特徴です。　　もう1つ有名なのは、2015年11月にGoogleが公開したTensorFlow（https://www.tensorflow.org/）です。Googleがサポートしていることや、TensorBoardという可視化ツールの存在もあり、Githubのスター数はTensorFlowの一人勝ちです。
「なんで流行ってるTensorFlowの紹介じゃないの？Githubのスター数もTensorFlowの方がめちゃ上じゃん」という意見もあると思います。ここでは敢えて、TensorFlowを紹介しない理由を述べます。ただ、筆者の主観もあるので一意見として捉えてください。

- TensorFlowの微妙なところ
  - コーディングしたらわかるイライラ感。
  - メモリをたくさん消費する（気がする）。
  - Qiitaが「チュートリアルやってみた」の記事ばかりでうざい。

# Chainer with GPUをはじめよう

ここでは実際にChainerで開発をするための、開発環境構築を説明します。インストールされる方は参考になさってください。ここでインストールするソフトウェアは以下です。

- NVIDIA GPU Driver 367.27
- CUDA Toolkit 7.5
- pyenv v20160629
- Chainer 1.11.0

# 筆者のPC環境

*Table 1. PCスペック*

| | |
|---|---|
| OS | Ubuntu 14.04(LTS) |
| GPU | NVIDIA GeForce GTX 980 |

# GPUドライバのインストール

まず、NVIDIA GPUドライバをインストールします。
Ubuntuはデフォルトではプロプライエタリのドライバがインストールされていますが、NVIDIA公式ドライバをここではインストールします。
（プロプライエタリでも構わないのですが、ここは筆者の好みです）
NVIDIAのサイトに行って、GPUに適したドライバをダウンロードします。
http://www.nvidia.co.jp/Download/index.aspx
まずプロプライエタリなドライバを削除しないといけません。以下のコマンドを実行します。

```
$ sudo apt-get --purge remove nvidia*
$ sudo apt-get --purge remove xserver-xorg-video-nouveau
```

再起動してUbuntuのログイン画面が表示されたら、「Ctrl + Alt + F1」を入力して仮想コンソールに入ってください。そうするとCUIでログ

インできる画面が開きます。下記のコマンドでGUIを停止します。

```
$ sudo /etc/init.d/lightdm stop
```

ここまでできたら準備完了です。以下のコマンドを実行して、GPUドライバをインストールします。

```
$ chmod +x NVIDIA-Linux-x86_64-367.27.run
$ sudo ./NVIDIA-Linux-x86_64-367.27.run
```

ドライバのインストール事項が聞かれますが、基本的にACCEPTとかにしとけばOKです。インストールが終わったらまた再起動をします。GUIが立ち上がり、ログインできたら「NVIDIA X Server Setting」というアプリケーションがインストールされているので、起動してみてください。

# GUIがうまく起動しなくなったら

Ubuntuなどは頻繁にソフトウェアアップデートがありますが、ときどきアップデートした後にOSを再起動すると、GUIがうまく動作しなくなったりします。こういったときは、だいたいGPUドライバが原因のため、GPUドライバの再インストールをしてみてください。手順としては以下です。

- ログイン画面で「Ctrl + Alt + F1」を入力して仮想コンソールにログインする。
- GUIを停止させる。
  - sudo /etc/init.d/lightdm stop
- GPUドライバにuninstallオプションを指定してアンインストールする。
  - sudo ./NVIDIA-Linux-x86_64-367.27.run --uninstall
- 再インストールする。
  - sudo ./NVIDIA-Linux-x86_64-367.27.run
- OSを再起動する。

# CUDAのインストール

NVIDIAのサイトからCUDA Toolkitをダウンロードします。
https://developer.nvidia.com/cuda-downloads
「Installer Type」はrunfile(local)を選択してください。ダウンロードが終わったら、以下のコマンドを実行すればインストールが実施されます。
注）最初に「GPUドライバをインストールするか？」と聞かれるが、ここだけnを選択してスキップする。それ以外はyでOKで、ディレクトリパスもデフォルトでよい。

```
$ chmod +x cuda_7.5.18_linux.run
$ sudo ./cuda_7.5.18_linux.run
```

# CUDAの動作チェック

CUDAのインストールが完了したらサンプルを動かしてみて、動作チェックをしよう。「PASS」の文字が出たらOKです。

```
$ cd /usr/local/cuda/samples/
$ sudo make
$ ./1_Utilities/deviceQuery/deviceQuery
./1_Utilities/deviceQuery/deviceQuery Starting...
<省略>
Result = PASS
```

# CUDNNのインストール

Chainerは標準のCUDAのみではライブラリが足りません。CUDNNというCUDA用の追加ライブラリが必要なので、これも公式サイトからダウンロードしてきます。ダウンロードには登録が必要ですが、無料です。
https://developer.nvidia.com/accelerated-computing-developer

```
$ tar zxvf cudnn-7.0-linux-x64-v4.0-prod.tgz
$ sudo cp cuda/include/* /usr/local/cuda/include/
$ sudo cp -R cuda/lib64/* /usr/local/cuda/lib64/
```

# Python環境の構築（pyenv）

Pythonは2系と3系が混在したりするなど、プログラミング言語としての環境がちょっとややこしいです。そのためpyenvを使って、Pythonのバージョンをコントロールするのがおすすめです。pyenvのインストールに関しては、公式READMEを読めばすべて書いてあります。

```
$ git clone https://github.com/yyuu/pyenv.git ~/.pyenv
$ git clone https://github.com/yyuu/pyenv-virtualenv.git
~/.pyenv/plugins/pyenv-virtualenv
```

~/.bashrcの末尾に以下を追記します。

```
export PYENV_ROOT="$HOME/.pyenv"
export PATH="$PYENV_ROOT/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"
```

~/.bashrcを再読み込みします。

```
$ source ~/.bashrc
```

pyenvを実行してみましょう。systemと表示されていることがわかります。これは、現在はUbuntuにデフォルトインストールされているPythonを使用しているという意味です。

```
$ pyenv global
system
```

ではここで、pyenvを使ってAnacondaをインストールしてみましょう。Anacondaとは Pythonの数値計算系ライブラリをまとめたディストリビューションです。 通常のPythonを利用するより、各種ライブラリのインストールなどが楽になるためこちらを導入します。以下のコマンドを実行したら環境構築は完了です。試しに、Pythonのバージョンを確認してみてください。

```
$ pyenv install anaconda3-4.0.0
$ pyenv global anaconda3-4.0.0
$ python -V
Python 3.5.1 :: Anaconda 4.0.0 (64-bit)
```

# Chainer用のPython環境を構築

Anacondaの機能を使って、Chainer用の環境を準備します。pipを使って簡単にインストールできます！

```
$ conda create -n chainer python=3.5
$ source activate chainer
$ pip install chainer
```

# Chainerでたぬき顔かきつね顔か判定させる（正解率90%くらい）

# たぬき顔orきつね顔

現在、女性の顔には たぬき顔 と きつね顔 の2種類の存在が確認されています。人間の男性ならこれを見分けるのは容易なことでしょう。これをChainerで学習させ、判別できるかやってみましょう。
例えば代表的なたぬき顔の女優さんといえば、 石原さとみさん や長澤まさみさん ですね。反対にきつね顔の女優さんは、 北川景子さん 柴咲コウさん などですね。以下を参考にしております。

- タヌキ顔とキツネ顔どっちが好き？
  http://blog.livedoor.jp/nwknews/archives/5029437.html
- あなたはどっちがタイプ??たぬき顔ときつね顔芸能人を比べてみた。
  http://matome.naver.jp/odai/2145264775798931501?

また、今回のソースコードはGithubで公開するので、そちらを参考にしていただければと思います。 https://github.com/ksakiyama/tanuki-kitsune

# データを集める

## データのターゲットを決める

代表的なたぬき顔ときつね顔の画像サンプルがほしいため、各代表の女優さんをリストアップします。今回は以下の方々で実施しました。

*Table 2.* 代表の女優様

| ラベル | 分類 | 代表の方々 |
|---|---|---|
| 0 | たぬき | 石原さとみさん,長澤まさみさん,おののののかさん,有村架純さん<br>橋本環奈さん,木村文乃さん,宮崎あおいさん,永作博美さん |
| 1 | きつね | 柴咲コウ,北川景子さん,加藤あいさん,吉瀬美智子さん<br>黒木メイサさん,大政絢さん,佐々木希さん,小島瑠璃子さん |

## Bing Search API

DLにとって、教師データを集める作業が一番時間がかかる作業かもしれません。できれば作業時間はDLモデルのチューニングにあてたいので、なるべく楽をしてデータを集めましょう。手っ取り早い手段としては、Google画像検索などを使う方法ですが、ここではBingのAPIを使って画像を集めることにしました。BingはGoogleより許容されているAPIトランザクショ

ン数が多いためです。とりあえず、一人につき1000枚の画像を収集しました。

# 集めた画像を加工する

DLで学習させるための前処理でいろいろと画像処理のテクニックが必要となります。今回はOpenCVとdlibというライブラリを使用してデータを加工します。OpenCVとdlibは非常に有名な画像処理ライブラリです。両方ともPythonからも使えるAPIがあります。すばらしい。 http://opencv.org/ http://dlib.net/

## 重複画像の削除

上記のAPIを使うと画像を大量に同じ画像がヒットしてしまうので、重複している画像を排除します。アルゴリズムとしては、ヒストグラムを用いた画像類似推定を実施し、類似度が高い場合に重複画像とみなします。OpenCVにはcompareHistというAPIがあるので、それを使います。

## 顔写真だけ切り抜く

dlibにはget_frontal_face_detectorという顔認証のAPIがあるため、それを利用して顔画像をどんどん出力していきます。

# 学習させる

ChainerのサンプルにImageNetがあります。これのインプットを女優の顔写真に変更して、たぬきさんかきつねさんかの分類をやってみましょう。画像は256x256のサイズで統一し、以下のようなファイルリストを作成します。1列目にファイルパス、半角スペースで区切って2列目にラベルです。

```
0_arimurakasumi/0000_face.jpg 0
0_arimurakasumi/0022_face.jpg 0
...
1_katoai/0002_face.jpg 1
1_katoai/0013_face.jpg 1
...
```

実行コマンドは以下です。

```
python train_imagenet.py --gpu -0 -j 1 --root data --epoch
150 --test data/train.txt data/test.txt
```

# 教師データの数

前節でいろいろやって、約8,000枚の画像を集めることができたので、ここから学習用データとテスト用データに分割して、学習用データを使ってトレーニングさせます。まぁ分割する割合は適当に20:1としました。

# 結果

150stepほど実行させると、91〜95%の正解率で収束しました。どうやらこれ以上は正解率が伸びないようです。
この結果を高いか低いか判断するのはその用途で決まると思うのですが、たぬきorきつねの2択なので、もう少し正解率がほしいところですね。適当に分類しても50%で当たるわけですし。

```
epoch       iteration   main/loss   validation/main/loss
main/accuracy  validation/main/accuracy
1           254         1.08134     0.698075
0.51624        0.5
2           507         0.70439     0.736036
0.52001        0.5
...
146         36943       0.0269541   0.149054
0.990119       0.95625
147         37196       0.0142548   0.296433
0.99543        0.91875
148         37449       0.025054    0.215606
0.990983       0.9125
149         37702       0.015178    0.307153
0.994442       0.93125
150         37955       0.0214843   0.279545
0.992218       0.925
```

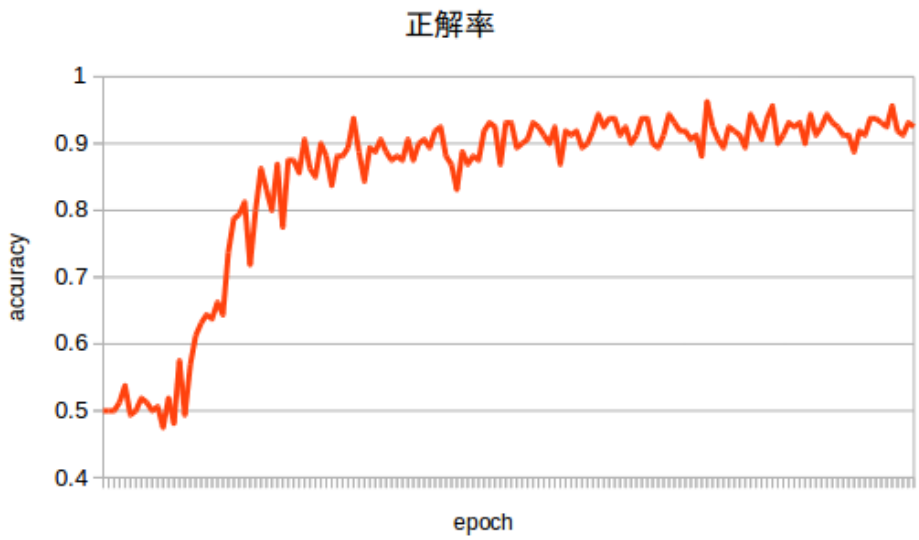



*Figure 3.* 実行結果

# 終わりに

今回、ほとんどは画像収集や顔認識のコードを書いていて、メインのDLのコードは1行も書きませんでした笑。でもChainerのサンプルを使うだけでも90%以上の正解率が出て驚きました。デフォルトのninモデルを使っただけですが、同ディレクトリにGoogleLeNetなどの素晴らしいサンプルもあるので、そちらでも試してみると面白いかもしれませんね。

# ひと夏の恋を貴方に　～この夏はじめる　ストリーム処理ことはじめ～

# What’s "IoT"?

この本では「IoT」を大きなテーマとして掲げつつ、個人個人で好きな物についてぐーたら、
でも真面目に書いています。が！そもそもIoTってなんだっけ？というのは大事かなと思いますので
IT初めての女子でも分かるように定義しておこうかなと思います。

# IoTとは

「そもそもIoTって何？」「っていうか言葉すら初めて聞いた」というあなた。
IoTとは”Internet of Things”の略称です。直訳すると「インターネットに繋がるもの」となります。
パソコンやサーバ（Google検索などインターネット上で動作するアプリを動かすためのパソコン）など
ITに関係しているもの以外の”もの”がインターネットに繋がっている状態を”IoT”と言っているんですね。

# 全ての”もの”をインターネットに

“もの”がインターネットに繋がるとはどういうことでしょうか。
むしろそんなものあるのか、と考えた方もいらっしゃるかもしれません。
ものだけに。
“もの”の定義は人によって曖昧なことが多いのが現状です。
今のところは、この世の中に存在する道具や生き物は”もの”として捉えて良いでしょう。
人も“もの”にあたるのか？と聞かれるとちょっと答えに迷ってしまうところはありますが。

# 生き物すらインターネットに繋がる時代

生き物がインターネットに繋がると聞いて驚かれる方もいらっしゃるかも

しれません。
そうです、IoTを駆使すれば生き物の生態や状態を観察することができる時代になったのです。

植物を例にとって考えてみましょう。
今までの植物のお世話といえば、土が乾いていないか自分で確かめることが一般的でした。
水やりをしても大丈夫か判断するためには、直接プランターやポットの土を触って確かめますよね。
水やりを忘れた挙句、土がカラカラに乾いていることに気づいたときにはもう手遅れ…。
こうして花の命ははかなく散ったと言う経験をされた方は多いと思います。

しかし、IoTを駆使すれば花の水やりを効率よく進めることができるようになります。
例えばプランターに「土壌湿度センサー」を取り付けます。
そうすると、「土壌湿度センサー」は土の中に含まれている水の量を測ってくれます。
水の量が少なくなった場合またはスマホにメール送信するといった仕組みを作っておけば、
水やり忘れで植物を枯らすということも無くなります。なんだか便利そうな感じがしてきましたね。

また、動物のお世話にIoTを活用している事例もあります。
ペットの首にGPSセンサー付きの首輪をつけていれば、迷子になった時もスマホで探すことができます。

他にも動物の動きに法則性があれば、動物がいつ、どんなことをしたか分かるようになります。
最近では牛の首に加速度センサー（ものがどの位の速さで動いたか検知できるもの）を巻きつけておき、
牛の首振り運動が早くなったら発情期と判断するような研究を進めているところもあるそうです。

# とりあえず物が繋がっていれば、それはもうIoT

IoTの守備範囲は意外と広いということは感じていただけたでしょうか。
重くて大きいパソコンに頼らずとも、”もの”が持っている情報をやり取りすることができるのです。
ITの世界に普段触れない人からすると、何ができるのかさっぱり分からないと感じるでしょう。
IoTでは”もの”・IoT用のデバイス（センサーなど）・パソコンと3つの世界を使い分けています。
実際に目に見えるものを動かして楽しむことができるのがIoTの魅力です。

だんだん興味出てきたと思いません？

# 女の子の生活にもIoTは存在する

「でもーITなんで枯れたおっさんがやるもんでしょ」「アキバはオタクがいっぱいいるからなー」
そんなことを思う流行追っかけ女子もいらっしゃるかもしれません。

確かに秋葉原のパーツショップのお客さんは大半が男性で構成されているのは間違いありません。
女性にとって居心地がいい！と言い切るのは正直...難しいです。
しかし、IoT技術は女子にとって身近なところに使われています。アキバに行かなくてもIT体験はできる！

自動販売機

JRをよく利用する方はちょっと黒くて大きめの自動販売機を見たことがあるのではないでしょうか。
新しめの駅なら必ず置いてあります。
若い子が行き交う渋谷や、流行に敏感な女子が沢山いる原宿にもありますね。
この自販機は一見普通の自販機のように思えますが、IoT技術がフル活用されています。
この自販機は、人が近くにいるかを自分で判断することができます。
人感センサーやカメラ認証で人が近くにいることを検知し、自販機の画面の切り替えを行います。

さらに、インターネット上にある天気の情報やカメラに映る人の姿からオススメの飲み物を判断し、
画面に映し出す機能も備えています。今までの自販機では考えられなかったことです。
これもIoT技術によって自販機という”もの”とインターネットが繋がっているからこそできるサービスと言えるでしょう。

商業ビル内のパネル
もっと女子の身近な例で何かないの？と思われる方もいらっしゃるかもわかりませんが、
実は女子がよく行くスポットにもIoT技術が使われています。
ルミネやパルコ、マルイなどの商業ビルに行くと、たまに人が通ると反応する案内板を見たことがあるかと思います。
人が通ると反応するだけではなく、パネルをタッチするとフロアマップが出てきたり、オススメの洋服を提示したりしてくれます。
これも人感センサーやカメラを使って人の存在を検知し、インターネットを介してフロアや服の情報を取得していると言えそうです。

# とりあえずセンサー買っちゃお

兎にも角にも、意外と身近な世界にIoTが存在していることはお分かりいただけたでしょうか。
モノがインターネットに繋がっているのはもはや当たり前の時代というわけです。
ただ、「IoTが身近にあるのはわかったけれど、結局自分じゃコントロールできないし？」とお考えになってらっしゃるかもしれません。
が、そんなことは全くありません。
最近のIoT体験グッズはとても充実しているので、初期投資4000円くらいで始めることができます。

何を準備したらいいの？
自分で好きにカスタマイズ・作成できるIoTグッズの代表格と言えばセンサーです。
で、そのセンサーと組み合わせて使うのが「マイコン」と言われる小さなコンピューターだったりします。
代表的なものとして「Raspberry Pi」「Arduino」などがありますが、個人的なオススメのものは「Arduino」です。 理由は2つあります。

- 安い
- セットアップが簡単

安い

Arduino本体は1つ3000円前後で買うことができます。
対するRaspberry Piですが、4000円から6000円程度かかります。
1000円あれば同人誌を２冊買えます。初期費用はやっぱり安く抑えたいですよね。

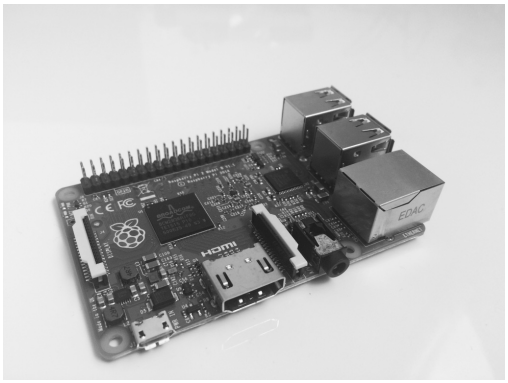
セットアップが簡単

Arduinoは基盤とIDE（統合開発環境）のみで動作させることが可能です。
ちなみにIDEとは自分で書いたプログラムが合っているか確かめたり、
パソコンが読めるような言語に書き換えたりしてくれる素晴らしいソフトウェアのことです。
こう書くとなんだか一種の宗教みたいですね。どうでもいいんですけど。

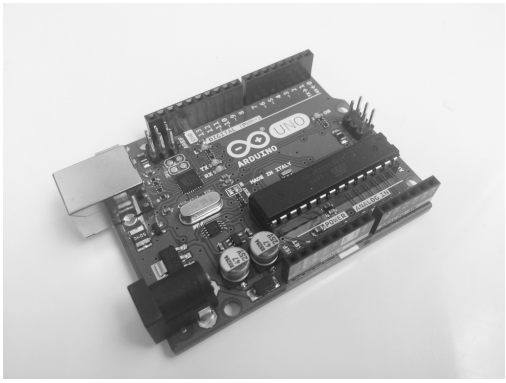
それに対してRaspberry　PiはRaspberry　Pi専用OS（WindowsとかMac OSみたいなもの）を
SDカードに焼き付けるところから始まります。
Raspberry
Pi自体の操作をするためにはキーボードやディスプレイを準備して繋ぐ必要があります。
さらに無線環境を準備して、OSインストールして、rpmコマンドで最新化もやって…なんてなるともう気が遠くなってきます。

初めて触る人にとってはハードルが高すぎてクラクラしてしまいそうです。
普段ITと無縁な世界で過ごしている人や、これからIoTやってみようかな～なんて考えている方は
Raspberry Piの利用は避けた方が良さそうです。

*RaspberryPi 2*。手のひらからはみ出るくらいのサイズです



こっちは*Arduino UNO*。中身は全然違うのに、結構似ているかも...



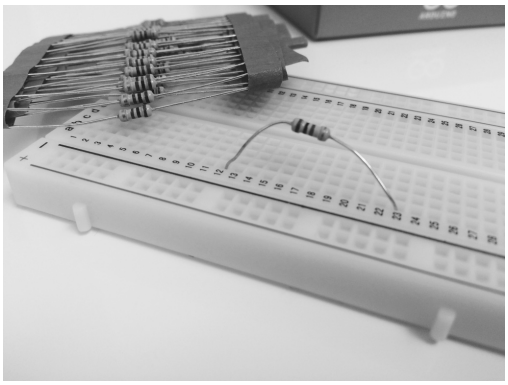
...というわけで、今回はArduinoを使うことにしましたが、他にも準備するものは色々あります。
代表的なものをいくつか挙げてみます。

*USB*ケーブル

> Arduinoとパソコンを繋ぐのに必要です。Arduino用と書いてあるものを購入すれば大丈夫です。 +

抵抗（とりあえず*10・220Ω*）
電気の強さを調整するのに使います。電子パーツを扱っているお店に行くと大量に売っています。

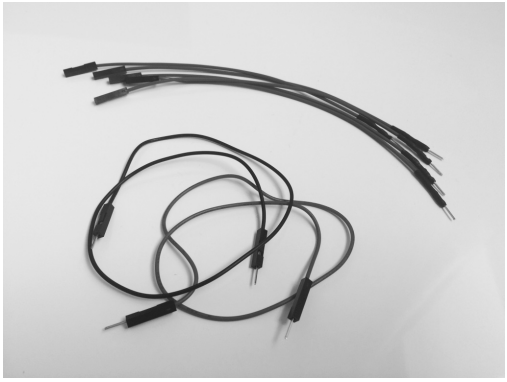
抵抗。絶対直角に曲げるのなんて無理。



ジャンパーワイヤ（オスーオス）

Arduinoとセンサーを繋ぐ回路として使います。
オスというのは先っぽが尖っていて何かに刺せるもの、メスは尖った部品がないもののことを指します。
なんでオスメスで識別するのかは永遠の謎です。
誰がこんな識別方法にしようって言い出したのか200時間くらい問い詰めたい。

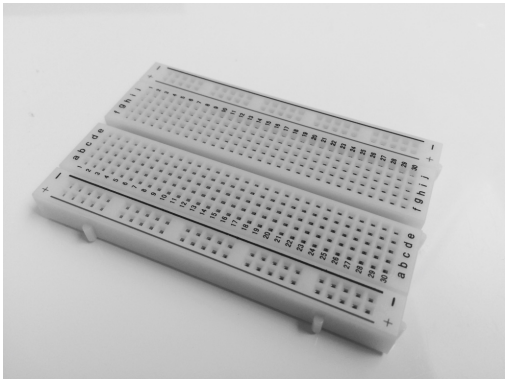
手前がオス-
オスのジャンパーワイヤ、奥はオスーメスのジャンパーワイヤです。



ブレッドボード

Arduinoとセンサーを繋ぐ回路として使います。
これを使うと半田付けなしでセンサーとArduinoを繋ぐことができます。
何回でも、いつでも、どんな時でも抜き差しできるので便利。

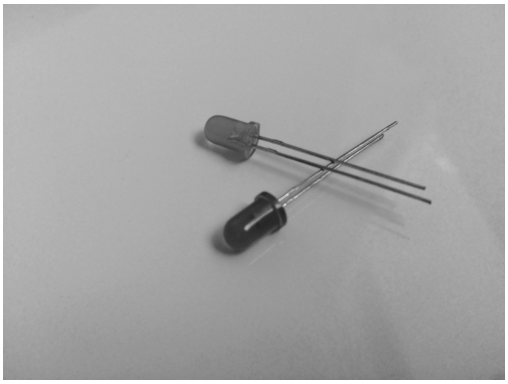
ものによりけりだけど、*Arduino*より一回り大きい。数字があったほうが使いやすいと思います。



*LED*ライト

一番初めの動作確認や、回路の状態を確かめるのに使うことができます。
赤・白・緑など、色々な色（決してダジャレではない）があるので好きなものをどうぞ。

ものによっては結構お値段が張る*LED*。みんな光り物好きだよね。



使いたいセンサー

やりたいことに合わせて好きなセンサーを買いましょう。
っていうかこれがないと始まらないっす！今回は温度を測るセンサーを使います。
「準備する物が多くてめんどくさ〜い」という人はArduinoのスターターキットを買うのがオススメです。
Amazonで15000円だすとセンサー等20パーツ＋説明書（英語/日本語）を一緒に購入できます。

センサーがいっぱいです。こんな感じで袋に入っていることが多いですが、小さいので無くさないように。

どこで買ったらいいの？

秋葉原が近い人は秋月電子にいけば間違いありません。
が、平日でも満員電車並みに混んでいるのであまりオススメできません。
安いんですけど。
なのでIoTグッズに強いネット通販を利用するのがいいと思います。
スターターキットを購入するか、Amazonで入門パックを探してみるのが良いでしょう。

# まとめ

いかがでしたでしょうか。IoT技術によって“もの”がインターネットに繋がっていることと、
それを活用して色々なサービスができていることがお分かりいただけたかと思います。
日本だとブームが終わると技術も衰退するような印象があります。
でも、IoT技術は今後色んなところで使われていくこと間違いなしです。

IoTを駆使すれば自作ラジコンや自動水やり機など色々なものが作れます。
まだまだ可能性は無限大です！この本を読んでくださった方が秋葉原の電気街に行ってみよー、
とかIoTと得意なスキルを組み合わせてみよー、とか思ってくれたら嬉しいです。

# はじめてのIoT

やっぱりーせっかく買ったら何かしてみたいと思いませんか？思いますよね？
なのでーーーArduino使ってIoTの世界を体験してみましょうよ。そうしましょう。
あ、そうそう、このネタははじめて本格的にITやる人に向けて書いているので、（特にわたし）
デキる皆さんはこのセクションはすっ飛ばしてもらうといいと思いますよ！

# 今回の目標

とは言いましても、何も目標がないと「うわー動いたー」で終わってしまいます。
ものが動くのもそれはそれで楽しいですが…。
自分があったら嬉しいものとか、思いつきのアイディアでもいいからなんか計画立ててみましょう。

というわけで！今回の目標はこちら！！

センサーで温度を測ってみたーーい！

…だって、なんかIoTっぽくない？普段も使えるだろうし。
立ち上げるのがめんどくさい？そんな考えだから何もできないんじゃないか。

# はじめてのセットアップ

何事も準備は大事です。やっぱり。
今回はArduino UNOを使ってみたいと思います。
早速なので、繋げてみましょう。

…とは言っても、Arduino用のUSBケーブルの四角っぽい端子と普通のUSB端子を繋げるだけです。
簡単でしょ？

あとは、Arduinoを動かすツールをインストールしてやればOKです。

こちらのホームページにアクセスの上、Arduinoのプログラムをインストールします。

```
https://www.arduino.cc
```

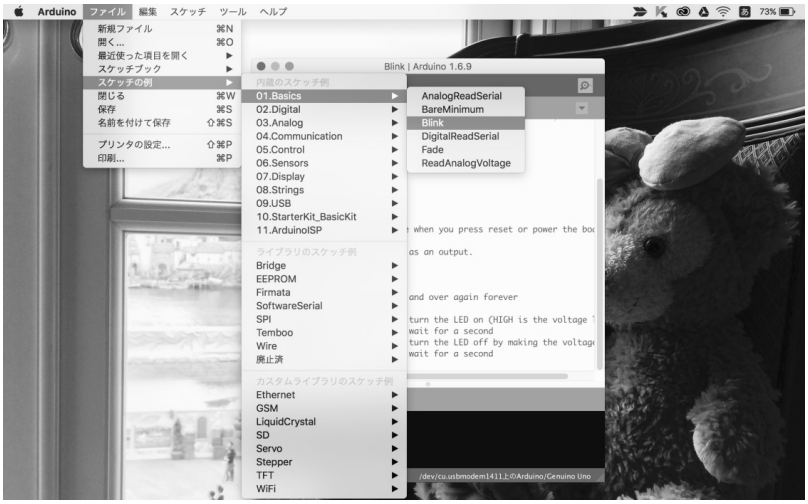
MacだったらMac用のもの、WindowsならWindows用のもの、LinuxはLinux用のものをどうぞ。
だいたいダウンロードしたのものをダブルクリックして、ひたすら「次へ」を押すだけです。Windowsなら。
Arduinoのプログラムがインストールできたら、いよいよ何かやってみたいと思います！

# はじめての**LED**

じゃあ早速温度測ってみよーとか思っちゃいます？
はやる気持ちはわかりますが、あまりオススメできません。
せっかく買ってきたArduinoが壊れているか確かめておいたほうが良いからです。
みんなに等しくコミケのサークル参加に当たる確率が与えられているのと同じくらい、
壊れているArduinoを引く可能性もありえるわけです。
ArduinoでLEDを光らせることができれば、壊れていないことが分かるわけです。

### *Arduino*をパソコンに読み込む

まずはArduinoをセットアップしてやります。まず始めにArduinoのIDE（統合開発環境）を立ち上げます。
「ファイル」→「スケッチの例」→「01.Basics」→「Blink」を選択します。




次に「ボード」→「Arduino
Uno」の順に選択します。この時自分の持ってるArduinoの種類を選択してあげてください。
こう見るといっぱい種類があるんですね。


で、最後に右向きの矢印ボタンを押してあげると、Arduinoを認識してくれるはずです。 できなかった場合は以下を疑ってみてください。

1. ボードの種類は間違っていないか
   ボードの種類が違っていれば、Arduinoを読み取ることはできません。
   「そんなことない！」とか言わずに確認してみましょう。

2. ケーブルはしっかり刺さっているか
   これも意外とあります。一度抜き差ししてみてはいかがでしょう。
   ちなみにArduinoの電源はPCから供給されているので、ケーブルを抜くと電源も切れます。

*LED*を光らせてみる

通称「Lチカ」とかいうらしいです。
Arduinoの「新規ファイル」を選択し、以下を書いてみます。

```
//LEDを光らせる

//LEDを13番のデジタルポートに接続
const int LED = 13;

void setup()

{
  //電気の出力はデジタルピンから送る
  pinMode(LED, OUTPUT);
}

void loop()
{
  //LEDを1000秒間隔で強く光らせる
  digitalWrite(LED, HIGH);
  delay(1000);
  //LEDを1000秒間隔で弱く光らせる
  digitalWrite(LED, LOW);
  delay(1000);
}
```

"//"はコメントアウトです。忘れないようにやっていることをメモしておくためにあります。
人間忘れる生き物ですので、記録をつけることは週間にした方がいいと思います。
久しぶりにプログラムを開けてみたときに、「これは何やってるんだろう」とかいうのはザラにあります。

次に"void setup()"の部分ですが、ここにはArduinoからどうやって電気を通してあげるか、など
始めにArduinoに対して言っておかないといけないことを書いておきます。
今回だと"LED"という変数は電気を送る先ということを示しています。
初めの行で変数LEDを宣言しているので、そこに対する処理を書いているわけですね。なるほどー。

"void loop"の部分ではLEDに対して何をするかを書いています。
loop（ループ）と書いているくらいですので、中に書いてあることが繰り返し実行されます。
今回は1000秒の間隔を開けてLEDを光らせます。digitalWriteの部分が光らせる処理というわけです。
プログラムは上から下に実行されていくので、始めに強く光り、次に弱く光るはずです。

*LEDとArduinoを繋げてみる*

やっぱり繋げてみないと始まりません。
LEDは先が長い方がプラス（電気が入ってくるほう）、短い方がマイナス（電気が出て行くほう）になっています。
プラスは動脈で、マイナスは静脈ってことですね。例えがグロいとかいう苦情は受け付けません。

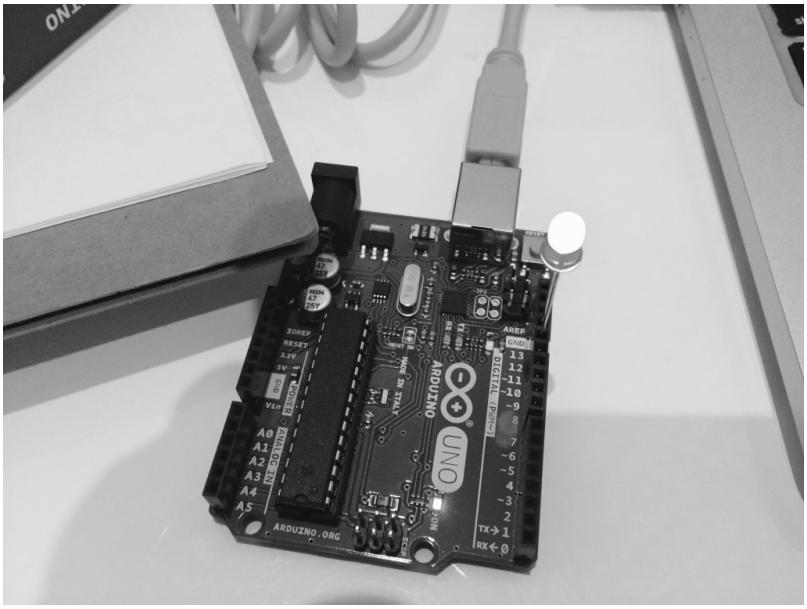
で、LEDのプラス側をArduinoの"DIGITAL"の13番に差し込みます。マイナスは"GND"と書いてあるところに差し込みます。
さっきのプログラムと比べてみると、この部分が当てはまりますね。

```
//LEDを13番のデジタルポートに接続
const int LED = 13;

void setup()

{
  //電気の出力はデジタルピンから送る
  pinMode(LED, OUTPUT);
}
```

GNDは「グランド」と言って、電気のマイナス方向のことをいいます。

電気は入ったら出て行く場所が必要なのですが、その部分のことを指します。
泥棒だって入ったら出て行くってことです。関係ないか。

実行してみる
最後にArduinoの画面左端の「検証」ボタンをクリックしてみると…
あら不思議、LEDが光ります！イメージはこんな感じ。




……わかんないって？やっぱり？
だってテキストが白黒なんですもの。しょうがないって。
とにもかくにも、これでArduinoは壊れていないことがわかりましたーぱちぱちぱち！！！
次はいよいよ温度を測ってみたいと思います。

# Hello,Johnny!

ただ、ここでちょっと気になることが出てきます。
Arduinoだけだと画像とか表示できなくない？とか思うわけです。
Processingという言語を使うと、Arduinoからもらったセンサーの値などで絵を描けたりします。
でも、**Arduino**と**Processing**はそれ専用の言語でできているんです。ベースはC言語ですが。

新しいもの始めるんだったら、最近はやりの他でも使える言語がいいよなーとか思っちゃうわけですよ。

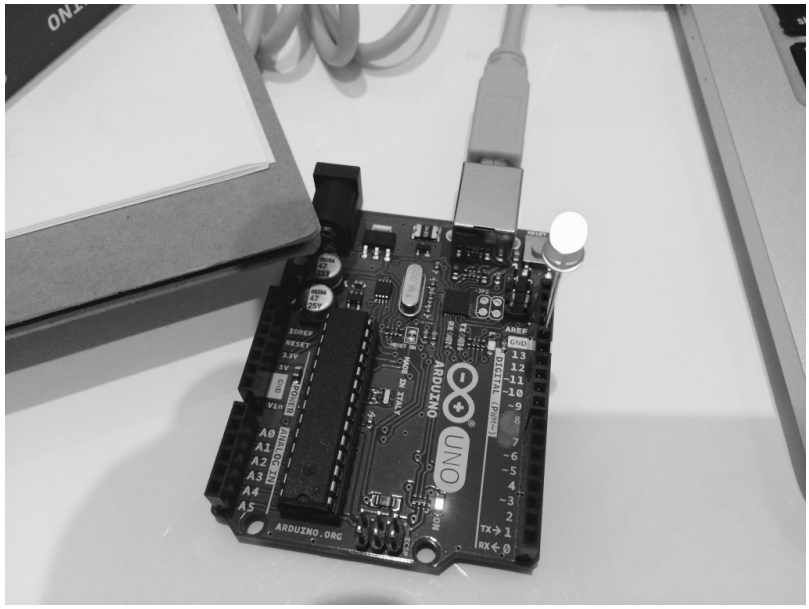
で、ここで登場するのがJohnny-fiveなわけです。
Johnny-fiveは芸能人の名前ではありません。今アツいJavascriptを使ってArduinoを動かすことができます。
こりゃお買い得もんだぁ！！！
というわけで、ここはJohnny-fiveを使うことにします。
もちろんArduino言語でセンサー操作もできます。その方が情報量は多かったりします。

### *Node.jsとJohnny-five*のセットアップ

まずはNode.jsをインストールします。Johnny-fiveはNode.jsというサーバサイドJavascriptエンジン上で動くからです。
エンジンっていうのは、パソコンをなんか動かすためのプログラムセットのことです。
マックでいうところのハッピーセットってことです。

Node.jsのインストールのために下記ホームページにアクセスし、"Recommented For Most Users"と書いてある方をクリックします。
最新版を落とすと不具合に出くわす確率が高いためです。

https://nodejs.org/en/



黒枠のほうを落とします。

で、あとは次への繰り返しでインストールを進めていきます。
この方法を取るとバージョンアップ対応は面倒ですが…しゃーない、今回は「やってみよう」のノリなので！
別に本格的な開発もやんないし！！と、心の中で言い訳をしながら「次へ」をひたすら叩きます。
ちなみにWindowsもMacもやることは一緒です。
できたらコマンドプロンプト、もしくはターミナルを立ち上げて以下のコマンドを打ってみましょう。
こんな感じで返ってくれば上出来です。インストールできてます。

```
# 打つコマンド
node -v
#出て欲しいもの
v4.4.7
```

次に、Johnny-fiveをインストールします。
まず、Javascriptをおいてあげる場所（ディレクトリといいます）を作ります。
その次にJhonny-fiveをインストールしてあげればOKです！

```
// Johnny-five用の作業場所作成
mkdir johnny5
// johnny5 に移動
cd johnny5
// Jhonny-fiveのインストール
npm install johnny-five

// こんな感じの結果が返って来ればOK！
MofuMofu-no-MacBook-Pro:jhonny5 MofuMofu$ npm install
johnny-five
-
> serialport@3.1.2 install
/Users/MofuMofu/jhonny5/node_modules/johnny-
five/node_modules/serialport
> node-pre-gyp install --fallback-to-build

[serialport] Success:
"/Users/MofuMofu/jhonny5/node_modules/johnny-
five/node_modules/serialport/build/Release/serialport.node"
is installed via remote
johnny-five@0.9.60 node_modules/johnny-five
 ├─── lodash.debounce@4.0.6
 ├─── ease-component@1.0.0
 ├─── color-convert@1.2.2
 ├─── browser-serialport@2.0.3
 ├─── temporal@0.5.0
 ├─── nanotimer@0.3.10
 ├─── lodash.clonedeep@4.3.2 (lodash._baseclone@4.5.7)
 ├─── es6-shim@0.35.1
 ├─── firmata@0.12.0 (es6-shim@0.33.13)
 ├─── chalk@1.1.3 (supports-color@2.0.0, escape-string-
regexp@1.0.5, ansi-styles@2.2.1, strip-ansi@3.0.1, has-
ansi@2.0.0)
 └─── serialport@3.1.2 (bindings@1.2.1, es6-promise@3.2.1,
nan@2.4.0, commander@2.9.0, debug@2.2.0,
object.assign@4.0.4)
MofuMofu-no-MacBook-Pro:jhonny5 MofuMofu$
```

Johnny-five（もう次からJohnny5にします。）とArduinoは

”Firmata"というプロトコルで通信します。
プロトコルとはコンピューターの間で話をするための言語です。
ArduinoのIDEを立ち上げ、「ファイル」→「スケッチの例」→「Firmata」→「StandardFirmata」を選択します。
あとは右矢印をクリックしてArduinoにプログラムを焼き付けるだけです。

これでJohnny5とArduinoがおしゃべりできるようになりました！

*Johnny*で温度を測る

今度はJohnny5を使ってセンサーを動かしてみたいと思います。先ほども述べましたが、
Johnny5はJavascriptを使ってArduinoを動かすことができます！なのでJavascriptファイルを作ってあげます。
先ほどJohnny5をインストールしたディレクトリに"tmpTest.js"というファイルを保存します。
中身はこんな感じで。

```
//johnny-fiveを使うことを宣言する
var five = require("johnny-five");

five.Board().on("ready", function(){
  var temperature = new five.Thermometer({
    //気温センサー
LM35を使う（専用のプログラムがここで適用される）
    controller: "LM35",
    //A0から入力があることを宣言
    pin: "A0"
    //5000ミリ秒たったら繰り返し
    freq: 5000
  });
  //入力された値を気温に変換
  temperature.on("change", function(){
    //コンソールに摂氏・華氏で温度出力
    console.log(this.celsius + "  ");
  });
});
```

で、プログラムを実行してみましょう。コマンドはこんな感じです。
sudo node tmpTest.js

うまくコンパイルできていれば、こんな感じで結果が返ってくるはずです。
やめたくなったときはCtl+Cを2回押します。

```
MofuMofu-no-MacBook-Pro:jhonny5 MofuMofu$ sudo node
tmpTest.js
1469108128497 Device(s) /dev/cu.usbmodem1411
1469108128509 Connected /dev/cu.usbmodem1411
1469108132328 Repl Initialized
// 出力結果
>> 36
```

...暑くね？(今エアコンが壊れてます)
とにかく、Johnny5を使ってArduinoを操作できました！ぱちぱちぱち！！！
HTMLやAPIと組み合わせれば、できることが広がっていきそうです。わたしも勉強してみよーーっと！

# 参考になる公式サイト集

本を買うと高いしーというあなた。公式サイトがあるので載っけておきますね。
ただし 英語 です。ほら、国際交流のために勉強しよっ！（適当）

1. Arduinoの公式サイト
   Arduinoとは？はもちろん、事例とか具体例がいっぱいあります。
   公式にアカウントを作ると、実装例やサンプルコードを閲覧可能です。
   https://www.arduino.cc
2. Johnny-fiveのGitlab　　オープンソースといえばGitlabなイメージですね。Johnny-fiveもオープンソースなので。
   関係ないけどGitとGitlabとGithubって紛らわしいから名前変えて欲しいです。
   あ、そうそう、documentsってやつがサンプルコードになってます。
   今回もそれを参考にやってますーー

https://github.com/rwaldron/johnny-five

3. Node.jsの公式サイト
   ダウンロードはここからどうぞ。実は日本語サイトもあったりします。
   https://nodejs.org/en/

4. Johnny-fiveの公式サイト　Jhonny-fiveで使えるAPIがまとまっています。APIっていうのはある一定の処理をまとめているプログラムのことです。
   あとサイトがおしゃれですね。やっぱり海外のサイトは色使いが綺麗です。
   http://johnny-five.io

5. Node.bots　Jhonny-fiveを盛り上げちゃうぞー的なサイトです。今後はJavascriptでIoTするのが盛り上がりそうですね。
   https://github.com/rwaldron/johnny-five

# 終わりに

今回は触ってみよー的なところで終わってしまいました...。ポンコツですみません。
サーバからHTMLや他サービスに温度とかを出せるといいですよねー。
IT初めての人にはハードルがぐっと上がりますが、今後もちょっとずつ勉強してみたいと思います。
で、また本をかけるように頑張ります。

# ストリーム処理入門以前　～この夏はじめる　ストリーム処理ことはじめ～

## はじめに

本誌ではストリーム処理処理について取り上げます。　Apache　Sparkや Apache　Stormをはじめ、ストリーム処理を実現するOSSが最近注目を集めています。
そのため、なんとなくストリーム処理について聞いたことがあるといった方も多いのではないでしょうか。

実際にストリーム処理をシステムとして実現するためにはこの処理ならではの様々な考慮ポイントがあります。

そこで本誌ではそもそもストリーム処理とはどういったものなのかから始め、ストリーム処理システムを構築するにあたってのポイントを中心にみていきます。

## 対象読者

本誌はこれからストリーム処理について勉強してみようという方を対象にしています。
そのため、ストリーム処理を実現する特定のプロダクトには特化せず、ストリーム処理そのものについて記載しています。

# ストリーム処理ってなんだろう

## ストリーム処理ってなに？

まずはこれから扱うストリーム処理がどういったものなのかを整理してみましょう。
本誌ではストリーム処理について以下の定義で話を進めていこうかと思います。

> 刻々と生成されるデータ(ストリームデータ)をその都度、もしくは短い時間幅で取り込んで処理する

ストリーム処理が対象とするストリームデータには以下のような特徴があります。(Figure1)

- 連続的/継続的に発生
- ひとつひとつのデータサイズは少量(なことが多い)

ストリームデータについて、センサーデータを例に考えてみます。
温度センサーや湿度センサーなどはその時々の温度/湿度データを連続して生成します。
これらのセンサーが生成するデータはひとつひとつはとても小さいものです。

センサーデータ以外にも、ECサイトなどのWebサイトで生成されるログデータもストリームデータとしての特性があります。
日々多くのユーザによって行われる様々な操作はログとして連続的/継続的に生成されます。
こうしたログデータもひとつひとつのデータサイズは小さいものです。

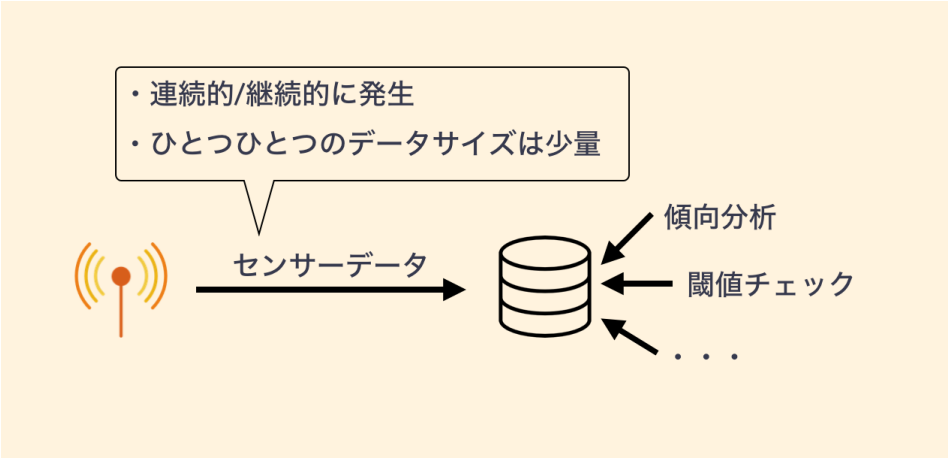



*Figure 4.* ストリームデータの特徴

ストリーム処理ではこうしたストリームデータに対して処理を施していきます。(Figure2)

センサーデータの例でいうと、センサーから送信させるデータに対してその都度閾値をチェックしたり、
その都度の直近の平均値を算出したりといったことが可能となります。
はたまた、ECサイトの例では、ユーザの購買履歴のログからより早く世間のトレンドを把握できたり、
ユーザーの不正操作などを早期に検知することができるようになります。

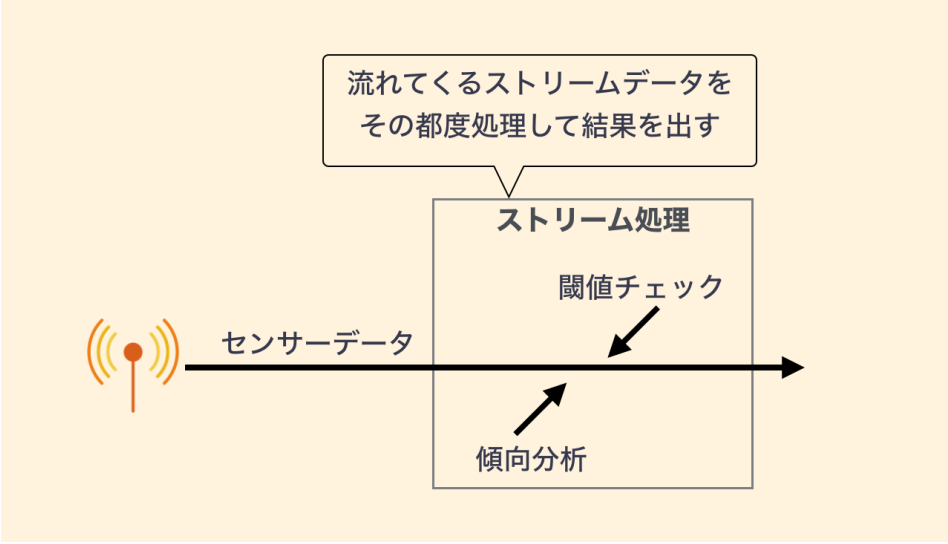



*Figure 5.* ストリーム処理の特徴

# バッチ処理とは何がちがうの？

ストリーム処理とは逆の処理アプローチを行うのがバッチ処理です。
もちろん、バッチ処理が対象とするデータにはストリームデータも含まれます。
ただしバッチ処理はそうしたストリームデータも貯めたのち、まとめて処理するといった特徴があります。
そのため、その都度処理するストリーム処理に対して、まとめて処理を行うためストック型の処理といったりすることもあります。

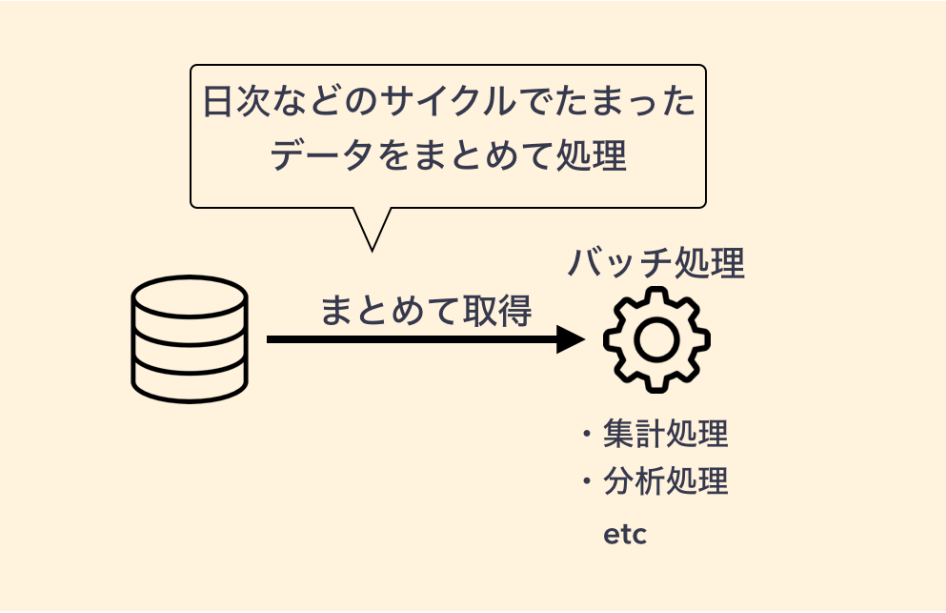



*Figure 6.* バッチ処理の特徴

バッチ処理と比較したストリーム処理のメリットは、先ほども述べたとおりより早く結果を得て、フィードバックにつなげられることです。

それではバッチ処理のメリットとはなんでしょうか。
厳密に比較を行うと多くの違いはありますが、後述するストリーム処理ならではの考慮点からすると、ある静止点をもってあとは一気に処理ができる点がひとつあげられます。

ストリーム処理でポイントとなってくるのが、いかに確実に正確に断続的に発生するストリームデータを処理できるのかといった点があります。
ストリームデータの特性上、連続的/断続的にデータが発生し続けるため、ストリーム処理ではそれらのストリームデータをどこまで処理できたか、そもそもストリームデータを受け取れたのかといった様々な
状態を管理する必要があります。

一方バッチ処理では、数時間に1回、日次など処理のサイクルが長いため、ある程度時間の余裕をもって確実にデータをシステムへ届けられればよいですし、バッチ処理開始時点までのデータを対象として/処理を実施すればよいので、ストリーム処理のように細々と状態を管理しなくてもよくなります。
(実際のバッチシステムを構築するとなると、一概にはこのようにきっぱりとはいかないことも多いです・・・)

| 処理方式 | 処理の間隔 | 処理データ量 |
| --- | --- | --- |
| ストリーム処理 | 短い(発生データをより逐次的に処理) | 少量 |
| バッチ処理 | 長い(データを貯めてまとめて処理) | 大量 |

ここまではストリーム処理とはどういったものなにかについてみてきました。
それではつぎに、よりシステム的な観点でストリーム処理の特徴についてみていきましょう。

# ストリーム処理を実現するためのシステム的観点

ストリーム処理はそれを実現するための処理方式やシステム制約など、様々な考慮ポイントが存在します。
ここでは、ストリーム処理をシステムとして実現するためのポイントについてみていきます。

世の中には数多くのストリーム処理エンジンと呼ばれるようなOSSがありますが、これから見ていくポイントに対応する様々な特徴の違いがあります。
採用するプロダクトを選ぶにあたって、着目すべきポイントにもなってくる内容でもあります。

# ストリーム処理のシステム的な考慮ポイント

ストリーム処理のシステム観点でのポイントとしては主に以下のようなものがあります。

- ストリームデータの処理方式
- ストリームデータのバッファリング/キューイング
- ストリームデータの発生時間と処理実施時間
- ストリームデータ処理の信頼性

それではひとつずつ見ていきましょう。

## ストリームデータの処理方式

さきほどストリーム処理とは時事刻々と発生するデータ(ストリームデータ)を短い時間幅で処理すると説明しました。
実はそのデータのストリーム処理方式には大きく2つのやり方があります。

| 処理のやり方 | 説明 | 主要プロダクト |
|---|---|---|
| 逐次処理 | データが到着するつど処理を行う | Storm |
| マイクロバッチ処理 | 短い間隔でデータをまとめて処理を行う | Spark Streaming |

逐次処理はストリームデータをその都度、各ストリームデータ単位に処理

を実施します。
こうした特徴から、ひとつひとつのストリームデータを低レイテンシで処理して結果をえることが得意です。
こうした処理を行う代表的なOSSプロダクトにはTwitter社が開発したApache Stormがあります。

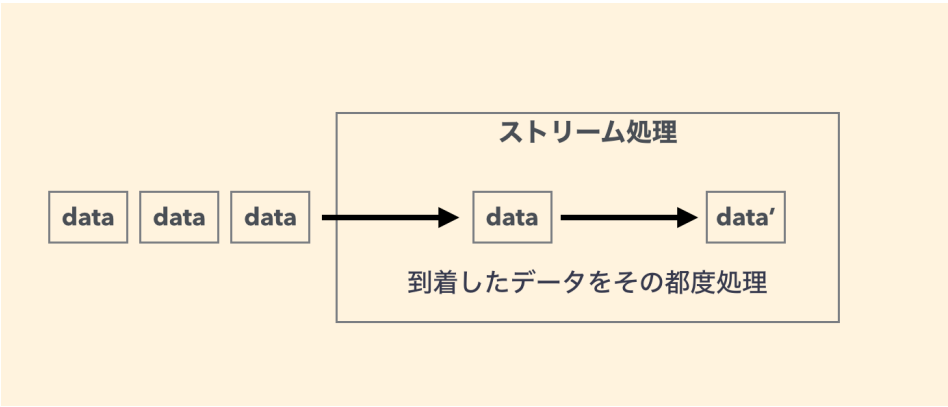



*Figure 7.* 逐次処理

一方マイクロバッチ方式では、ストリームデータをその都度処理するのでなく、データをまとめて処理します。
これだけきくと通常のバッチ処理と変わらないようにも思えます。
しかし、マイクロバッチ処理はより短いサイクルでこのバッチ処理をまわしていきます。
このように短いサイクルで高頻度にバッチ処理を繰り返していくことでストリーム処理を実現していきます。
マイクロバッチ処理では、直近発生したデータを塊で処理していくので、複数データをまとめて取り扱うような集計処理を得意としています。
マイクロバッチでストリーム処理を実現するOSSプロダクトにはApahce SparkのSpark Streamingがあります。

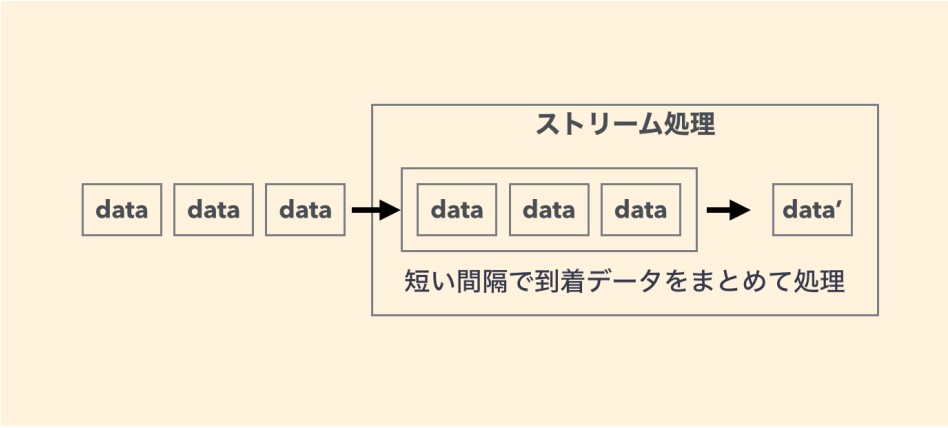



*Figure 8.* マイクロバッチ処理

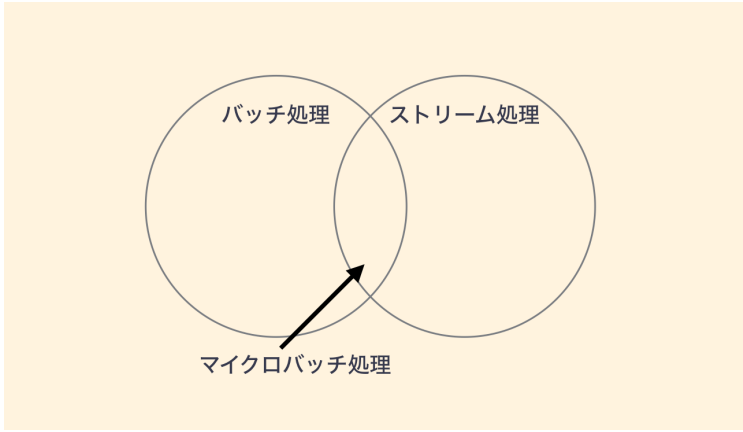
このように、ストリーム処理エンジンによって処理方式が異なるため、どういった処理を実現したいかに
応じて、適切なプロダクトを選定していくことになります。

NOTE

コラム　マイクロバッチ処理　≠　ストリーム処理

本誌ではストリーム処理の処理方式としてマイクロバッチについて説明しました。
上述した通り、マイクロバッチ方式では短い間隔でバッチ処理を実施し、擬似的にストリーム処理を実現しています。
そのため、正確にはストリーム処理ではなく、バッチ処理に分類されます。
ただ、その性質からもストリーム処理とバッチ処理の中間的な位置づけとも捉えることができます。

最近ではApache Sparkのようなマイクロバッチ方式でストリーム処理を実現するようなプロダクトも
多く利用されているため、本誌ではマイクロバッチもストリーム処理として取り扱っています。




## ストリームデータのバッファリング/キューイング

ストリームデータには以下の特性があることについて説明しました。

- 連続的/継続的に発生する
- ひとつひとつのデータのサイズは少量

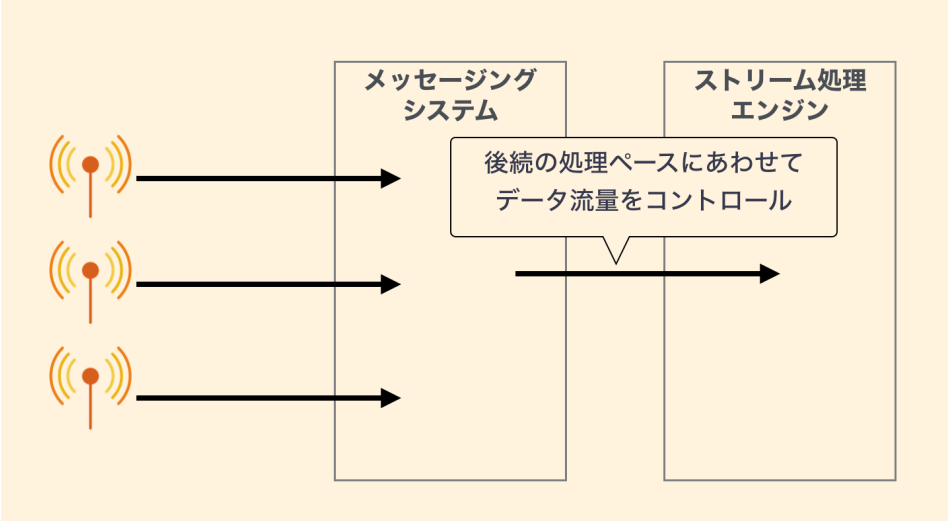
センサーデータのようにひとつひとつのデータサイズは小さいものの、多くのセンサーデバイスからデータが送信されてくるような場合、ストリーム処理エンジン側では直接受けきれない可能性があります。

一つ一つのデータは小さくとも、それが同時にしかも大量に押し寄せてくるとストリーム処理エンジン側でシステムリソースの割り当てが間に合わ

ず、しだいに未処理データが蓄積されていって、処理遅延が次第に大きくなっていきます。

処理が遅延するだけでなく、あまりにさばききれない場合、ストリーム処理エンジン自体がダウンして しまうおそれもあります。

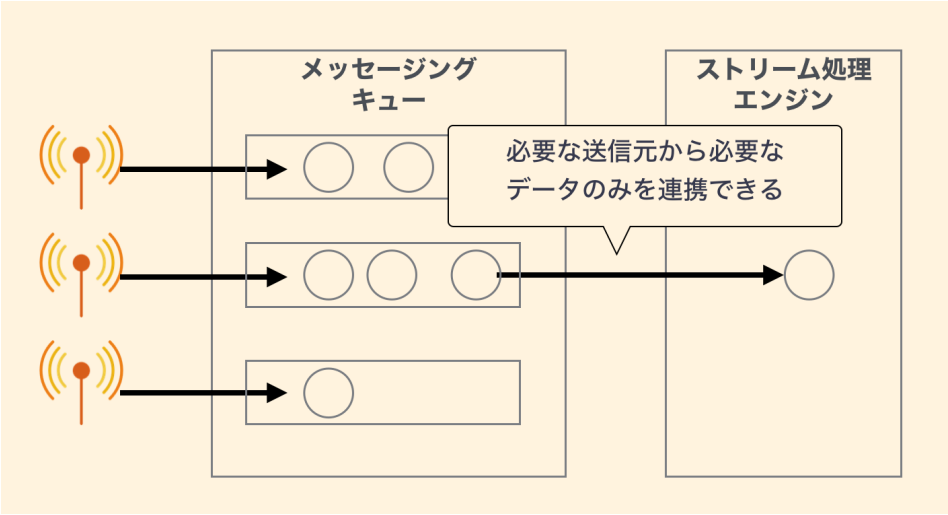
こうした事態を防ぐ方法としてストリーム処理でもバッファリング/キューイングを考慮する必要があります。




例えばApache
Kafkaはバッファリングとしても機能するメッセージングシステムです。
Kafkaは様々なプロダクトと連携するためのコネクタプラグインが充実しているため、最近ではKafka+何かしらのストリーム処理エンジンのセットでシステムが構築される事例も多いです。

本誌ではメッセージングシステムの詳細についてはふれませんが、このようなシステム構成を考慮することで大量のストリームデータが発生してしまってもストリーム処理システムの負荷を緩和させることが可能です。

またメッセージングシステム本来の機能として、処理に必要なデータのみをストリーム処理エンジン側で取得して処理することも可能となります。

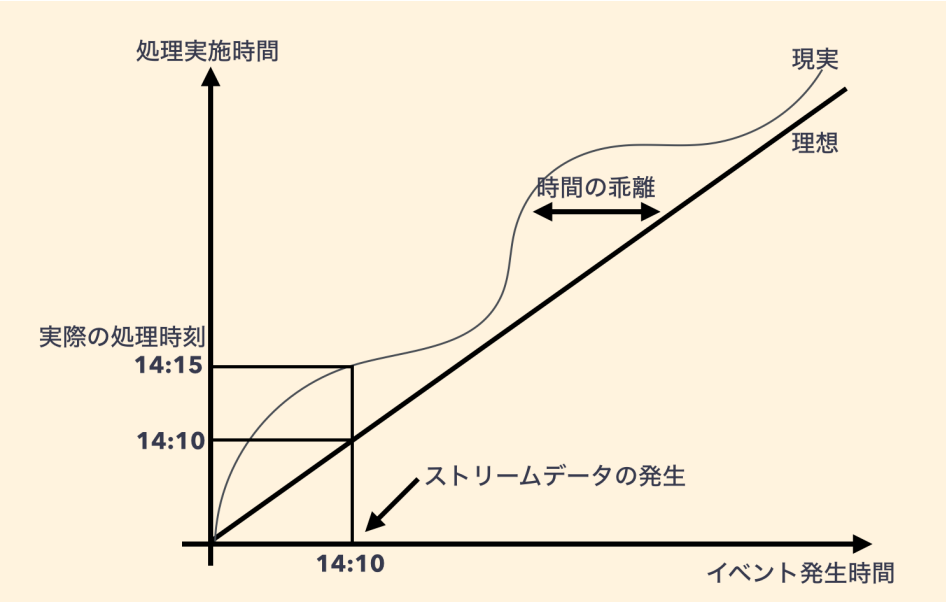



## ストリームデータの発生時間と処理実施時間

ストリーム処理は流れてくるストリームデータを短い時間幅で処理することで、より早く結果を得ることができるというメリットについてはすでに述べた通りです。

しかし、実際にストリーム処理によるシステムを構築する際は、あらゆるシステム制約により必ずしも発生したストリームデータを即時に処理できないケースも起こりえます。

例えば、ストリームデータ送信されている途中でネットワーク障害等により一時的に配信遅延が発生してしまような場合、ストリームデータが発生した時刻とそれが実際にストリーム処理されて結果えられた時間に乖離がでてしまいます。




上の図の横軸は実際にストリームデータが発生した時刻、縦軸は発生したストリームデータがストリーム処理エンジンで受信して処理が完了した時間です。

理想は発生したストリームデータは即時にストリーム処理エンジンに届き処理されて結果が得られることです。
しかし先ほどの例のように、実際のシステムには様々なそれを妨げる多くの制約が起こりえます。

例えば、

- システム間のネットワークが不安定
- データ送信元でデータが滞留
- ストリーム処理エンジン内部で処理遅延

などなど、ストリームデータの発生時間とそれがストリーム処理されるまでのタイムラグにつながる要因は様々です。

どの程度までこうした時間的な差異が許容されるのかといった業務要件や、また差異を縮めるためにどの程度までシステムに落とし込めるのかといったか考慮しつつ、システムを構築していく必要があります。

## ストリームデータ処理の信頼性

それでは最後にストリームデータ処理そのものの信頼性について考えてみましょう。
ここでいう信頼性とはストリームデータをいかに確実/正確に処理することができるのかの度合いを意味します。

こうした信頼性は一般的にメッセージの信頼性といわれたりします。
このメッセージの信頼性はストリームデータを処理する側だけでなく、そのストリームデータを送信する側でも考慮が必要となってきます。

一般的にはメッセージの信頼性には以下の3種類あります。

| 名前 | 説明 |
| --- | --- |
| at most once | 最大1回のみ送信する。そのため、1回のみ送信するがそれが相手に届くかどうかは保証されない。 |
| at lest once | 少なくとも1回は相手に届ける。そのため、相手に確実に1回は届けるが、<br>同じメッセージが重複して送られてしまうかもしれない。 |
| exactly once | 正確に1回のみ届ける。そのため、上記の2つとは異なり、重複もなく確実に相手に1回<br>メッセージを届けることが可能。 |

下にいくほどより確実に相手にデータを届けることができるため、データ送信の確実度は高くなっていきます。
ただし、「at least once」の場合、確実にデータが届けられるものの、そのために実は相手に届いているのに、同じデータを何度か送信してしまい、ふたたびストリーム処理エンジン側で同じデータが処理されてしまうことが考えらます。
そのため、処理の正確さという観点では唯一条件を満たせるのは「exactly once」に基づいた処理です。

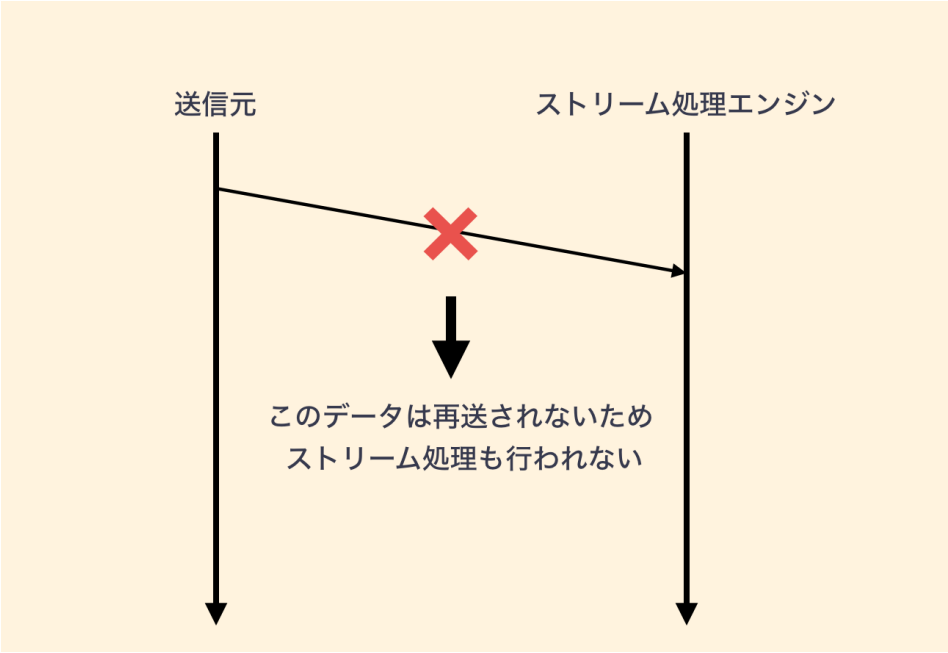



*Figure 9. at most once*

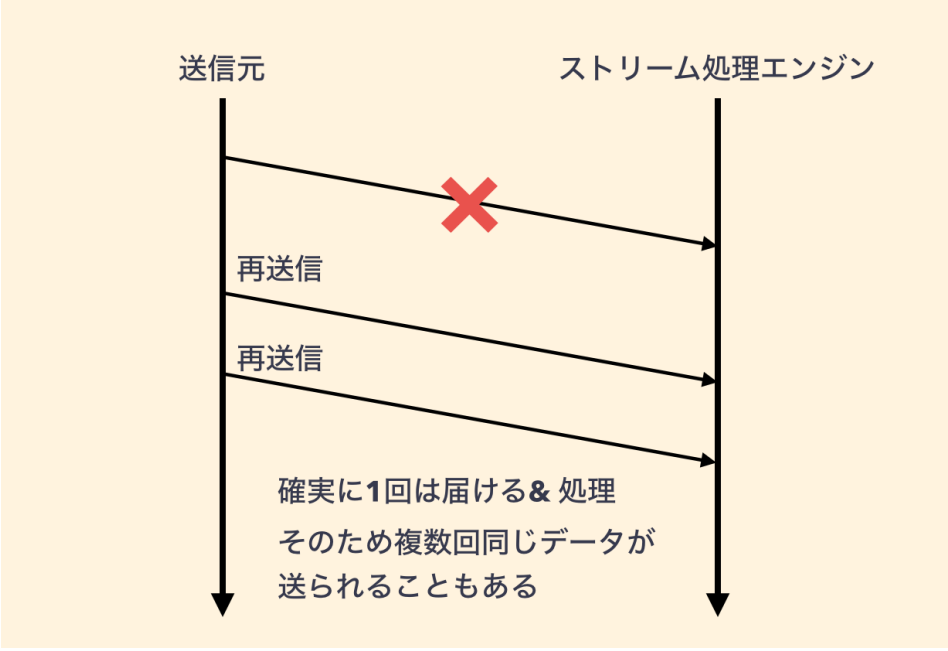



*Figure 10. at least once*

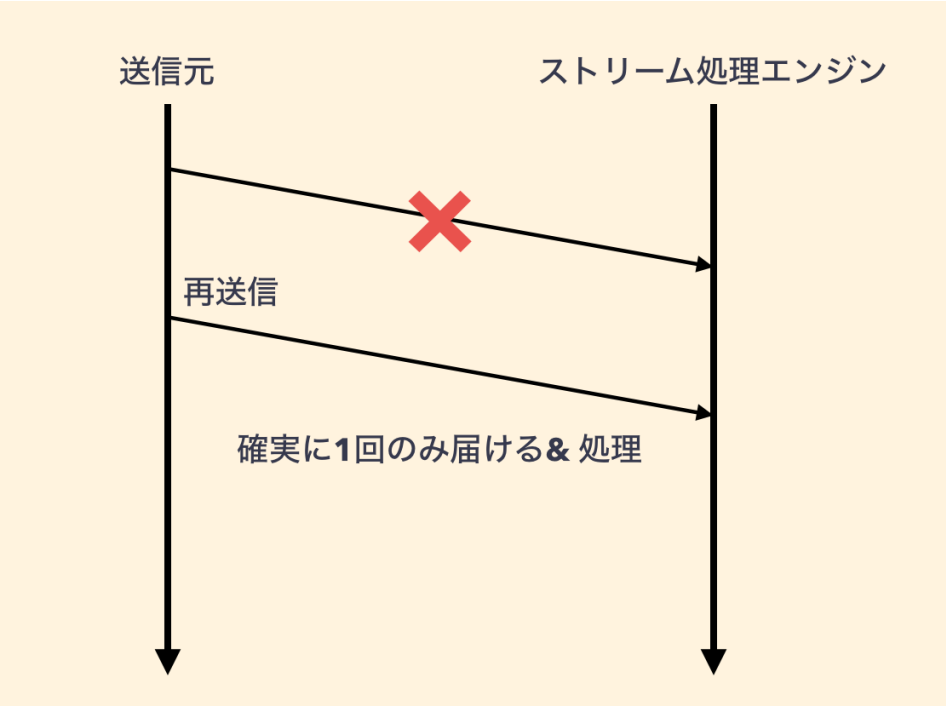



*Figure 11. exactly once*

ここまでの特徴だけで比較すると常にexactley
once方式に基づいた処理が望ましいように思えます。
もちろんそれが理想なのですが、システム観点だと正確さ、確実さを追求していくと実装コストが大きくなりがちです。

**at most once**の場合、ストリームデータの送信側はとりあえずデータをストリーム処理エンジンめがけて送信するだけでよく、それが確実に届いたかについては気を払う必要はありません。
また、ストリーム処理エンジン側も届いたデータだけを処理していけばよいだけです。

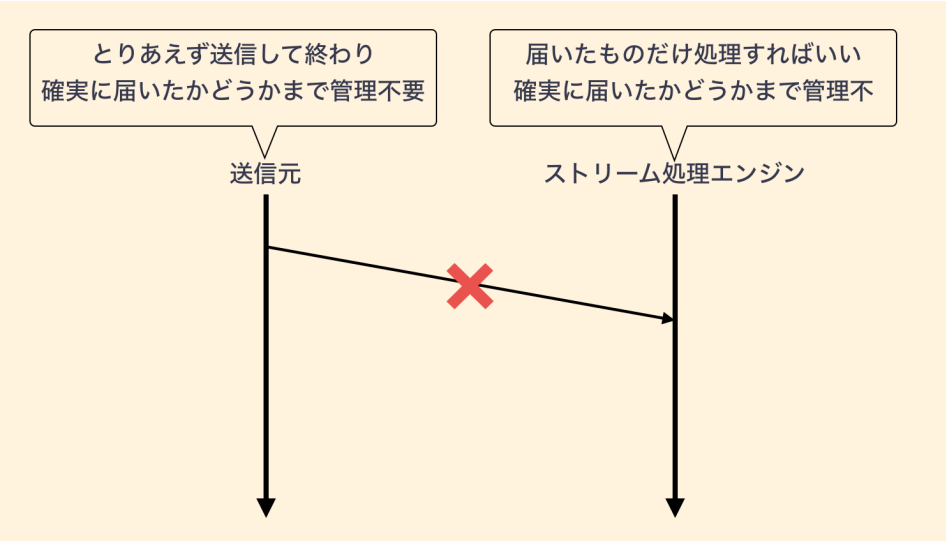



*Figure 12.「at most once　」の処理フロー*

一方「at least once」の場合、ストリーム処理エンジンへ確実にデータを届けるためには、正常にやりとりが完了したかをお互いに知る必要があります。
そのため、ストリーム処理エンジンは正常にデータを受け取れた場合にそのデータの送信元へその旨を通知する必要があります。

また、万が一ストリーム処理エンジンにデータが届かなかった場合、再度同じデータを送信してもらう(リトライ)の考慮も必要となってきます。
さらに、「exactly once」を実現するためには、上記の仕組みに加えて、ストリーム処理エンジン側で全く同一のデータが送信されてきても、そうした重複を排除する仕組みも必要となります。
重複を排除するためには、各ストリームデータを一意に特定できるようなIDなどをもとにして、各ストリームデータの状態も管理しておく必要があります。

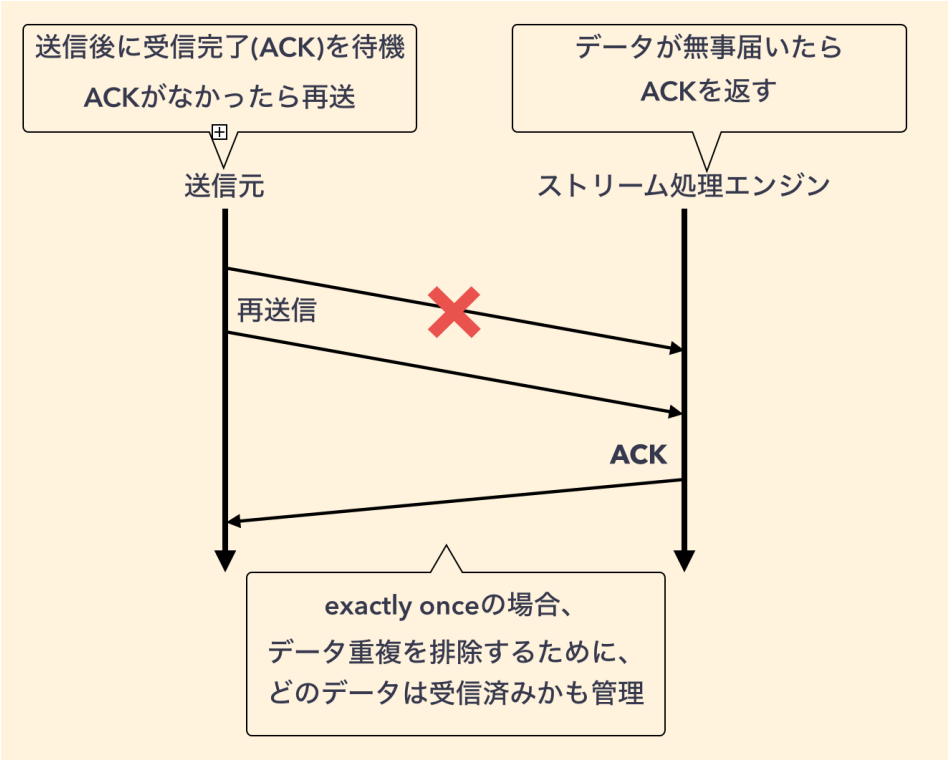



*Figure 13.* 「*at least once/exactly once*」の処理フロー

このようにより確実性/正確性を求めるためには、その分ストリーム処理の実装難易度や考慮ポイントもどんどん増えていくこととなります。
また、管理する状態や整合性を維持しなくてはいけない対象が増えるほど、ストリーム処理そのものの性能も落ちていってしまいます。

ストリーム処理エンジンのOSSごとにサポートするメッセージの信頼性は異なってくるため、実現したい要件とも照らし合わせて、適切なシステム設計を行う必要があります。

# さいごに

本誌では具体的なプロダクトの説明よりかは、それ以前のストリーム処理の考え方や、実装にあたって考慮すべき要件などのポイントを中心にみてきました。
こうした観点をもとに業務要件に適したシステム設計と最適なプロダクト選びに少しでも力になれましたら光栄です。

# Afterword

## tea

*illustrations*

お手に取っていただき、ありがとうございます。
表紙のイラストを担当させていただきました。
なんだかんだで楽しく描けたのでよかったです。

## ksakiyama

*Page XX ~ XX*

本誌を手にとっていただきありがとうございます。
はじめてこういう類を書きましたが、ブログとかQiitaとは違った面白さがありました。
次があれば、もっと面白い内容をかけるように頑張りますm(_ _)m

## keigodasu

*Page XX ~ XX*

お手にとっていただきありがとうございます。
ストリーム処理の情報っていろいろ外に出てるしなーどうしようかなー、ページ数も限られてるしなーってと思い、入門以前ってことでポイント絞って書いてみました。
また機会があればもう少し突っ込んだ話ができればと思います。

## MofuMofu @froakie0021

*Page XX ~ XX*

ITの会社で働いてますが、ITよりも文章書く方が得意なMofuMofuです。
最近はインフォグラフィックとプリキュアが好きです。モフルンは可愛いですね。
とにもかくにも、読んでいただきありがとうございました！
もうちょっと技術系分野も多様性社会になればいいなーなんて思いつつ、もし次の機会があればもっと頑張ります。

発行
超未来工房

発効日
2016年8月14日　初版発行

印刷・製本
有限会社　ねこのしっぽ