目次

1

Node.js で 1 から作る 生放送サーバ

NaruseJun 著

Node.js を使って生放送サーバを書いていきます。コアとなる技術は **HTTP Live Streaming**(HLS) と呼ばれる HTTP ベースのストリーミングプロトコルで、特殊なストリーミングサーバを必要としないのが特徴です。
今回は、HLS を使ってニコニコ動画の Nsen[*1]の再現に挑戦してみます。

[*1] ユーザ参加型ラジオ風生放送サービス。ユーザが再生したい動画をリクエストすると、その動画が生放送される。

第 1 章

はじめに

1.1　前提環境

今回紹介するコードは、以下の環境で動かしています。簡単に構築方法も示しますので、適宜用意してください。

- Windows 10 (version 1709 build 16299.125)
  - Microsoft Edge 41.16299.15.0
- Node.js 9.3.0
  - npm 5.5.1
- FFMPEG 3.4.1

Node.js のインストール

Windows/macOS/Linux 向けのバイナリが `https://nodejs.org/` から入手できます。バイナリをインストールした場合は、パスを通すようにしてください。

macOS では、Homebrew が利用可能であれば `brew install node` を実行しても入手可能です。

Linux では、パッケージマネージャ経由でも入手可能ですが、ディストリビューションによっては古いバージョンがインストールされる場合があります。古い Node.js では動作しないコードを多数含みますので、必ず最新版を利用してください。[*1]

FFMPEG のインストール

Windows/macOS/Linux 向けのバイナリが `https://www.ffmpeg.org/` から入手できます。バイナリをインストールした場合は、パスを通すようにしてください。

macOS では、Homebrew が利用可能であれば `brew install ffmpeg` を実行しても入手可能です。

Linux では、パッケージマネージャ経由でも入手可能です。

[*1] 未検証ですが、恐らく 8.0.0 以上であれば動作します。

第 2 章

HTTP Live Streaming の概要

HTTP Live Streaming(HLS) と呼ばれる HTTP ベースのストリーミングプロトコルで、特殊なストリーミングサーバを必要としないのが特徴です。いわゆる生放送のようなイベントストリームの配信はもちろん、過去の生放送の録画[*1]等の VOD(Video on Demand) 配信も可能です。ユーザの回線速度に応じて品質の異なる動画を配信する**アダプティブストリーミング**にも対応しています。

Apple によって開発され、2017 年 10 月に **RFC8216** として仕様が公開されました。

本章では、HLS の概要を紹介していきますが、以下の資料も併せて参照することをおすすめします。

- HTTP ライブストリーミングの概要 (Apple - 日本語)
  - `https://developer.apple.com/jp/documentation/StreamingMediaGuide.pdf`
- RFC 8216 - HTTP Live Streaming (IETF - 英語)
  - `https://tools.ietf.org/pdf/rfc8216.pdf`

2.1 HLS の再生環境

HLS は HTTP ベースということもあってブラウザとの親和性が高く、一部のブラウザではデフォルトで HLS の再生に対応しています。

[*1] ニコニコ生放送で言うところのタイムシフトのような

2

Pwnable 超入門

kriw 著

Pwnable とは CTF(Capture The Flag) という情報セキュリティに関する競技の分野の一つです。Binary Exploit とか呼ばれていることもあります。わかりやすくいうとハッキングですね。（違法行為はダメ）一般的な Pwnable の問題では小さなバイナリファイルが渡されその脆弱性をついてプログラムの制御を奪うことを目的としておりコンピューターに関する深い知識が要求されるため CTF の中でも特に敷居の高い分野ではないかと思います。今回はそんな Pwnable の初歩を紹介したいと思います。

第 1 章

環境

途中、Python のスクリプトが登場しますがすべて Python2.7 で動かした結果です。また、今回の説明の実行環境は ubuntu16.04 の docker images 上で行っていて、練習問題も同じ ubuntu16.04 で動作しています。以下のコマンドで今回説明するコマンドやプログラムのソースコードが入った環境が手に入ります。

```
docker run ※※※※※※※※※※※※※※※※
```

1.1 練習問題の起動方法

練習問題は docker 上で動かすのでまず docker をインストールしておいて下さい。インストールしたら以下のコマンドで問題のコンテナをダウンロード&実行できます。

```
docker run ※※※※※※※※※※※※※※※※
```

docker の ip アドレスを調べます。まず、docker ps で起動中の docker のコンテナ ID を探します。このコンテナしか動いていない場合は以下のコマンドでコンテナ ID だけ取り出せます。

```
$ docker ps | tail -n1 | cut -d' ' -f1
3579cab05f3e
```

docker inspect コンテナ ID を実行して ip アドレスを探します。

```
$ docker inspect 3579cab05f3e | grep \"IPAddress
            "IPAddress": "172.17.0.2",
                    "IPAddress": "172.17.0.2",
```

試しに nc コマンド等を使って問題に接続してみます。

```
$ nc 172.17.0.2 1000
Do you want to login?
yes
You are not admin.
```

できました。

第 2 章

システムコールとシェルコード

CTF でも良くつかうシステムコール、シェルコードについて紹介します。

2.1 システムコール

システムコールは入出力の read, write や execve, fork といったカーネルに対して発行する命令です。シェルを取るためには例えば execve("/bin/sh") を実行する必要があります。

x86 では `int 0x80` 命令によってシステムコールが発行され

レジスタ	役割
eax	システムコール番号
ebx	第一引数
ecx	第二引数
edx	第三引数
esi	第四引数
edi	第五引数

x86_64 では

レジスタ	役割
rax	システムコール番号
rdi	第一引数
rsi	第二引数
rdx	第三引数
r10	第四引数
r8	第五引数
r9	第六引数

となっています。とりあえず CTF では以下のシステムコールを使うことが多いので覚えておくと良いです。

3

言われてみれば当たり前なタッチ入力の話

Namazu 著

こんにちは、Namazu です。皆さんスマホゲームは作っていますか？Unity などを使えばほとんど作れるようになりましたよね。今回は、そこで扱われるタッチ入力やそれを扱う UI について詳しく考えていきたいと思います。
尚、ここで紹介するフレームワークのテンプレートは GitHub にて公開しています。

- GitHub：`https://※※※※※※※※※※※※`
- README：`https://※※※※※※※※※※※※※※※※※※※※`

C#、MonoGame を用いた開発をしておりますが、今回は設計の話が主ですので環境が異なるという方も参考にしていただけるかと考えております。

第 1 章

タッチ入力とボタン入力の違い

Android 端末や iPhone、Windows Phone などはタッチで入力をします。一方、多くのゲームハード（PS、Wii、Switch 等）や PC ではボタン（キーボード）で[*1]入力をしますね。

タッチ入力とボタン入力では入力の処理の仕方が異なります。

1.1　ボタン入力

ボタン入力では、「どのボタンが押されたか」によって入力を判別します。押されたボタン毎に処理を書くだけで充分ですね。ただ、ボタン入力の種類にはただの押下だけでなく**ボタンを押し続ける**、**複数のボタンの同時入力**などがあります。これらについても処理を分ける必要がある場合があります。

1.2　タッチ入力

タッチ入力では、「どこが押されたか」によって入力を判別します。同じタップでもタッチされた位置によって処理が全く異なります。さらに、こちらの入力はタップ以外にも**ロングタップ**、**スワイプ**、**フリック**、**ピンチイン/アウト**など種類が多岐に渡るためボタン入力よりも複雑だということができるでしょう。

[*1] 便宜的にマウス入力はタッチ入力の一部だと考えます

第 3 章

タッチ入力の区別

突然ですが、問題です。

> ある距離の近い二点 A,B があるとして、以下の方法で二点に交互に入力が来た場合それらを区別できるでしょうか？ case1：AB 間を擦るようにスワイプした case2：点 A と点 B を日本の指で交互にタッチしたただし、入力はあるフレームでは点 A、次のフレームでは点 B、その次のフレームでは点 A…のように無駄な入力はなく且つ途切れることなく行われたとする

フレーム数	タッチされている点
1	A
2	B
3	A
4	B
5	A

これは、一度に入力されるタッチ数が高々一つであるマウス入力では考える必要がなかったものです。ゲームを作ったことのある方なら分かると思いますが、入力を連続的に取ることは出来ません。（だからこそこの問題が成立します）ですから、一見これらを区別することは不可能なように思えます。

4

そばやのワク☆ワク流体シミュレーション MPS 編

そばや著

以前から興味のあった流体をちょいと本読んでやってみました。
できあがったものはこちら https://※※※※※※※※※※
ソースはこちら https://※※※※※※※※※※

第 1 章

流体シミュレーションって？

流体シミュレーションとは、難しい方程式を解かないとわからないような流体の動きをプログラムで近似するものです。剛体シミュレーションで運動方程式

$$F = ma$$

を用いるところを、流体シミュレーションでは**ナビエ・ストークス方程式** (以下 NS 方程式) というものを使います。その昔、ナビエさんとストークスさんが発明したらしいです。wikipedia(`https://ja.wikipedia.org/wiki/`ナビエ–ストークス方程式) 曰く、その全貌は以下のようになっています。

$$\begin{aligned}\frac{Dv}{Dt} = &-\frac{1}{\rho}grad(p) + \frac{\mu}{\rho}\Delta v + \frac{\lambda+\mu}{\rho}grad(\Theta) + \frac{\Theta}{\rho}grad(\lambda+\mu) \\ &+ \frac{1}{\rho}grad(v\cdot grad(\mu)) + \frac{1}{\rho}rot(v\times grad(\mu)) - \frac{1}{\rho}v\Delta\mu + g\end{aligned}$$

わわわからぁぁぁぁぁん!!!! 何言ってんだこいつは！ ラグランジュ微分ってなんだ！ ふざけるな！！ ということで限定条件下で式を簡略化します。今回使う条件は「粘性率が一定の非圧縮性流れ」です。粘性率とは、まぁ粘り気です。非圧縮性とは、どんなに力を加えても流体の体積が一定に保たれるというものです。(現実にはありえません) すると使う式は

$$\frac{Dv}{Dt} = -\frac{1}{\rho}grad(p) + \nu\Delta v + g$$

となります。これならなんとか食らいつけそうです。1 項ずつ見ていきましょう。

$\frac{Dv}{Dt}$ $\frac{Dv}{Dt}$ はラグランジュ微分なのですが、まぁとりあえず普通に時間微分だと思って良いでしょう。v は速度なので、左辺はいわゆる加速度です。

$-\frac{1}{\rho}grad(p)$ 右辺第一項は圧力勾配項と呼ばれ、圧力の大きいところから小さいところにむけて加速度を作り出す作用のある項です。ρ は密度ですが、実質質量ですね。力を質量で割って加速度を得るようなかんじです。

第 2 章

流体シミュレーションの種類

流体シミュレーションの手法は大きく分けて 2 種類。

1 つは**格子法**です。格子法とは、計算する空間を適当な数の格子で分割し、その小領域それぞれに対し速度などのパラメータをあてていきます。つまり、「1 つの格子が流体の 1 単位です！」という主張なわけですね。実際、NS 方程式は速度**場**に対する加速度などを考えるものなので、これは結構自然な発想だと思います。

もう 1 つは**粒子法**です。今回テーマにしたいのはこちらですね。粒子法は、流体を粒子の集合として表現する方法です。粒子 1 つ 1 つに速度などが存在します。これもなんとなく納得できそうなかんじです。

どっちがいいの？

2 種類あると比較したくなるのが人情ってもんです。

まず格子法ですが、こちらは数学的な「場」の概念に則った自然なモデルです。ですから、(こういうとおかしいかもしれませんが) 物理法則が通用します。計算もとても安定しています。欠点としては

- 完全に流体の存在する領域を決定しないと使えない
- まともな見た目のものを作ると、計算負荷が高い
- 水面を作りづらい

などが考えられます。

一方粒子法です。粒子法について詳細の説明はこれからとなりますが、私の感想としては「粒子法は格子法に比べて、だいぶ無理がある」といったかんじです。若干こじつけじみたところがあり、そのせいで数値的安定性を欠く傾向があります。もちろんそのぶん得られるメリットもあります。メリットとしては

- 存在する場所を予め決めておかなくても良い
- 計算負荷の調整がある程度容易
- 水面や飛沫が表現できる

5

まだ間に合う！ Vue.js

とーふとふ著

初めまして、とーふとふといいます。今回は最近 Web フレームワークとしてかなりの勢いを持っている Vue.js についての記事を書きたいと思います。
この記事は今年の 11 月に私の所属サークルである東京工業大学デジタル創作同好会 traP の活動の一つチズケもくもく会 (チズケに集まってあるテーマに沿ったことを黙々と作業する会) で行った Vue.js 入門の資料に加筆・修正を加えたものです。

第 1 章

Vue.js とは？

図 1.1: Vue.js のロゴ

> Vue (発音は / v j u： / 、 view と同様) はユーザーインターフェイスを構築するためのプログレッシブフレームワークです。他の一枚板 (モノリシック: monolithic) なフレームワークとは異なり、Vue は初めから少しづつ適用していけるように設計されています。中核となるライブラリは view 層だけに焦点を当てているため、Vue.js を使い始めたり、他のライブラリや既存のプロジェクトに統合したりすることはとても簡単です。一方、モダンなツールやサポートライブラリと併せて利用することで、洗練されたシングルページアプリケーションを開発することも可能です。

つまり、導入は楽チンで取り入れやすく、その気になれば他のライブラリと連携させてイケイケなアプリを作れるすげーやつということです。

よく言われる利点としては

- 導入が楽である
- 学習コストが低い
- 公式ドキュメントが豊富
- 日本語情報が多い
- 勢いがある (利用者の数も開発も)

などです。

特に 2017 年は Web フレームワークの中では Vue.js の年と言っても過言ではないくらいで、Google の検索トレンドでは React.js を抜いたり、GitHub のスター数でも来年には抜かすのではないかというほどになっています。

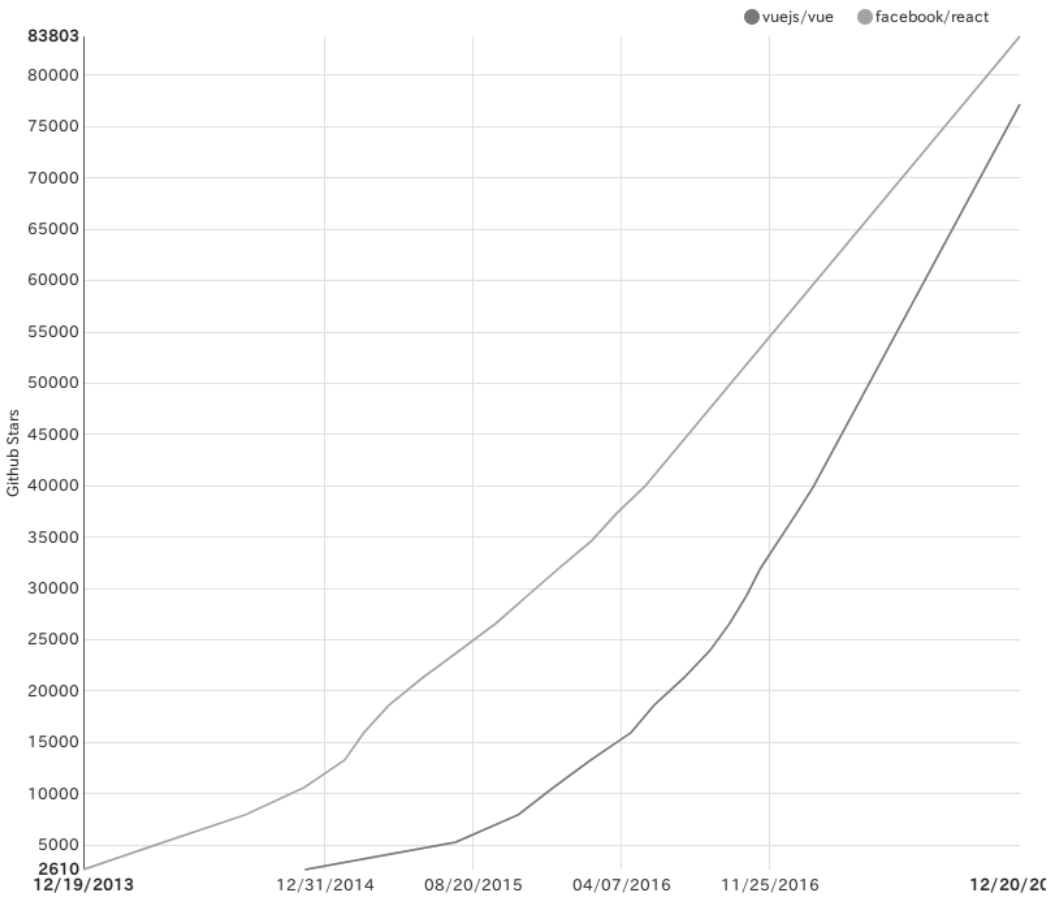

図 1.2: Star 数推移