TypeScript を用いたスクレイピングとデータ整形について

5431　藤本 祥太（指導教員：才田 聡子）

Scraping and Data Formatting Using TypeScript

概要： 電離圏研究のために実施されている HFD 観測のデータはオープンデータとして公開されており，自由な使用が認められている．現状，この観測データの利活用のためにいくつかの web アプリケーションが開発されているが，その多くで課題点を抱えてしまっている．そのため，主な利用者である研究者たちの要望も取り入れつつ，よりよい web アプリケーションの提案，開発する．本研究では，その中の web スクレイピングを用いてデータを取得する処理および取得したデータを扱いやすくする処理に焦点を当てる．

キーワード　TypeScript，スクレイピング

1.　はじめに

地震等に伴う電離圏の変動を観測するため，HF 帯電波を用いた HFD 観測という手法が取られている．この HFD(短波ドップラー) 観測のデータは電離圏研究のためにオープンデータとして公開されており，自由な使用が認められている．[1)]

現状，データ活用を促進するために数件の web アプリケーションが開発されている．しかし，設計時点で複数の言語が使用されている，ページの読み込みに時間がかかるなど，その多くが開発運用の困難性やユーザーエクスペリエンス等の観点で問題を抱えている．例として，図 1 に HFD のオープンデータが公開されている web サイトの一部を示す．[2)]

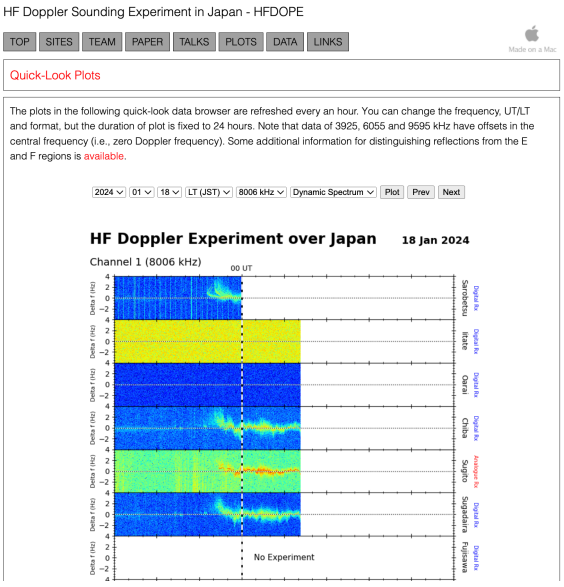

図 1　オープンデータが公開されている web サイト

これは，公開されているオープンデータを元にプロットされたグラフを表示しているページである．図に上部にあるタブから日付やチャンネルなどを変更し，プロットされるグラフのデータを変更することができる．しかし，観測年，月,，観測方法，チャンネルの周波数などを 1 つずつ変更する必要がある上，タブ自体が小さく押しづらいとかなり不便になってしまっている．また，同時に 1 つまでしかグラフを表示することができず日付ごとの比較などが不便になってしまっている，上部のボタンが背景と文字のコントラストが小さく視認性が悪い，文字が小さく読みづらい等，UI，UX 上の観点で様々な問題を抱えている．

また，web アプリケーションの主なユーザーとなる電離圏の研究者から，web アプリケーションの仕様についていくつかの要望が出されている．それらを一部抜粋して，以下表 1 に示す．

表 1　web アプリケーションに対する要望の抜粋

- 周波数固定で全ての局を見せる
- 局固定で全ての周波数を見せる
- 局，周波数を選んでカスタムプロットしたい
- ダイナミックスペクトルを見たい
- Doppler の時系列に Amplitude で色を付けたい
- Es などのイベントを抽出したい（機械学習）
- ランニングアベレージを取りたい
- PDF などの形に出力できると良い
- パンとズームができると良い

これら既存の web アプリケーションにおける課題点の解消，及び研究者からの要望を反映した web アプリケーションを提供するため，web アプリケーションの新規設計を行う．

全体のシステム構成図を他のメンバーとかぶってもいいから示す

既存の web アプリケーションにおける課題点を解消するため，ページレンダリング手法を考慮して開発を進めていく．本研究では Next.js と呼ばれる JavaScript のフレームワークを用いて実装を行う．JavaScript の UI ライブラリの一つである React.js がベースとなっており，サーバ機能も有しているため効率的な web 開発が可能である．また，サーバーサイドレンダリング (SSR) や静的サイト生成 (SSG) といった機能を持つ web サイトの作成を得意とする．そのため，今回開発する web アプリケーションの要件に合致していると判断した．[3)]

本開発でのシステム構成に関して，同研究室の中嶋中嶋くんの名前は出した方が良いでしょうかがインクリメンタルスタティックリジェネレーション (ISR) を用いた手法と，サーバーサイドレンダリング (SSR) とスタティックジェネレーション (SG) を用いた手法について，実行にかかる時間の計測・比較を行った．その結果，ISR を用いた手法では初回実行時に約 30 分の待機

時間が発生すること，それに対し SSR と SG を用いた手法ではデータのページに遷移する際に約 1 秒のスクレイピング時間を必要とすることがわかった．SSR と SG を用いた手法ではページ遷移の度に約 1 秒の待機時間が発生してしまうが，ISR を用いた手法の場合で初回ビルド時に発生してしまう約 30 分のオーバーヘッドの問題を踏まえれば許容範囲であると考えた．そのため，本開発では SSR と SG を用いた手法で web アプリケーション開発を行なっていく．[4)]

本研究ではこの web アプリケーションにおける web スクレイピングを用いてオープンデータが公開されている web サイトからデータを取得する処理，および，取得したデータを整形してフロントエンドで扱いやすく整形する処理に焦点を当てる．また，使用する言語を web アプリケーション開発全体で統一して管理コストの低減を図るため，web スクレイピングに関する処理は TypeScript を用いて行う．

2. Web スクレイピング

web スクレイピングとは，web サイトから情報を抽出するコンピュータソフトウェア技術のことを指す．HTML フォーマットからテキストを抽出してスプレッドシートや json ファイル等の構造化データへの変換を行い，web 上にあるデータを取得して扱うことを目的とする．今回のように，目的のデータが web サイト上にオープンデータとして公開されている場合はデータ取得の手段として適している．取得したデータはそのままでは扱いにくいことがあるため，その場合は適宜扱いやすいように整形しておく必要がある．
web スクレイピングの手法としては，正規表現や DOM 解析，HTML パーサ等を用いたものがある．本研究では，ISR を用いた web アプリケーションを設計するため，スクレイピングの処理に速度を求める必要がない．そのため，本研究では軽量でサーバへの負担が小さい cheerio を HTML パーサとして用いる．

3. 使用技術

本研究で使用した主な TypeScript ライブラリについて以下に示す．

3.1 TypeScript

TypeScript は JavaScript に静的な型付けやクラスベースなオブジェクト指向を加えたスーパーセットである．JavaScript と比べ型定義を静的に行えるという特徴がある．あらかじめ関数の引数等などの型を定義しておくことで，コードの可読性を向上させたり、実行時のエラーを削減したりできるという利点がある．また，本開発ではアプリケーションフレームワークに Next.js を採用する．上記の利点に加え，web アプリケーション開発全体で使用する言語を統一することで管理コストの低減を図ることができるため TypeScript が本研究でのデータ整形，出力処理に適していると判断し採用した．

3.2 Next.js

Next.js は JavaScript のフレームワークの１つであり，UI ライブラリである React.js がベースとなっている．その特徴としてページレンダリングのサポートが充実していることやサーバ機能を有しているため効率的な web 開発を行うことができることなどが挙げられる．そのため，アプリケーションの目的に応じて最適なレンダリング手法を選択することができることを強みとする．
Next.js のレンダリングオプションには一般的なクライアントサイドレンダリング (CSR) の他に，本開発で用いるサーバーサイドレンダリング (SSR)，スタティックジェネレーション (SG)，インクリメンタル静的再生成 (ISR) を含む．

3.3 Superagent

HTML リクエストを送信することのできるモジュール．GET 通信及び POST 通信を行うことができる．非同期処理に適しており，同期処理に不向きな代わりに動作が早く，負荷が軽いという特徴がある，そのため，本研究でのスクレイピング処理に適していると判断し採用した．本研究では HFD 観測データが公開されている web ページから HTML レスポンスとしてテキストデータを取得する際に用いる．[5)]

3.4 Cheerio

HTML パーサに jQuery のサブセットを実装したもの．HTML ドキュメントのタグやクラスを指定して，目的のテキストを抜き出すことができる．利点として，jQuery の関数を用いてスクレイピングの処理を行うことができる．jQuery ライクに記述することがでる，動作が早く，負荷が軽い，ほぼすべての HTML ドキュメントをパースできるという点が挙げられる．そのため，本研究では本研究では HTML レスポンスからテキストデータだけを抜き出す際に用いる．[6)]

4. 実装

4.1 目標

本開発では，Next.js を用いて開発運用及びユーザーエクスペリエンスの観点で従来のものよりも優れており，なおかつ主なユーザーである研究者の要望を取り入れた Web アプリケーションを新規に作成することを目標とする．本研究ではその中の，web スクレイピングを用いてオープンデータが公開されているサイトからデータを取得する処理及び，それをフロントエンドで扱いやすいように整形する処理を実装することまでを目標とする．
本研究ではスクレイピングおよび，データ処理を行う関数を実装し，フロントエンドから求められているデータ様式の要件を以下表 2 に示す．

4.2 今回のデータ整形処理について

このオープンデータは観測地点によってチャンネルの数および周波数が異なり，それに応じて処理の内容と出力する json ファイルのプロパティ名を変更する必

表 2　データ様式の要件

- json 形式にまとめられている
- ”hour”，”minute”，”second”という 3 つの
  プロパティを含んでいる”time”プロパティを持つ
- チャンネルの周波数ごとに”freq”と”amp”が
  まとめられたプロパティ，”channel”を持つ

要がある．
データ整形処理の実装当初，この観測地点ごとのチャンネルの違いについてあらかじめ json の型定義に全ての周波数分のプロパティを用意しておき，必要に応じて利用するという方法を試みた．しかし，その方法では出力したデータファイルに無駄が多く，データファイルを閲覧する際の視認性が悪く，データファイルを保存する際に容量を圧迫してしまうため好ましくない．そのため，元データのラベル部分をあらかじめデータ整形用の配列とは別に保存しておき，json 生成の際に利用することで観測地点ごとにチャンネルの数と周波数ごとに適切な処理を行う方法をとった．
また，当初は json の構成でデータ部分を”channel”プロパティにまとめずそのまま周波数ごとに記述していた．しかし，観測地点ごとにチャンネルの数および周波数が違う今回の場合ではその方法ではフロントエンドで扱いづらく，また可読性等の観点で好ましいデータ様式ではないため上記表 2 のように”time”プロパティと”channel”プロパティを分けることとした．

4.3　スクレイピング処理

本開発では SSR と SG を併用した手法で開発を行っていく．データが公開されている web サイトの検索ページに存在するパスを静的に生成し，各ページでリクエストが発生した際に動的に波形データを返すという処理を行う．今回実装したスクレイピング処理およびデータ整形処理を行う関数は，各ページでリクエストが発生した際にサーバーサイドで実行されフロントエンドにスクレイピング・整形後のデータを渡すという処理で使用する．

Superagent の GET 関数を用いて，HF ドップラーのデータが公開されている web サイト [2] に HTML リクエストを送り，web サイトの HTML リソースを取得する．例として，2020/11/20Sarobetsu のデータを示す．図 3 は Superagent を用いて取得した web サイトの HTML リソースの一部である．

この HTML リソースから，cheerio を用いて目的のテキストデータのみを取り出す．今回は body すべてを取り出して，そのテキストデータをフロントエンドから求められているデータ様式の要件に沿って整形していく．
図 4 に HTML リソースから取り出したデータの一部を示す．

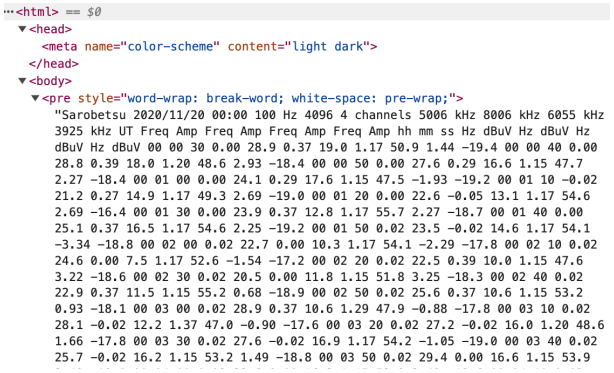

図 2　サイトの HTML リソース

```
data: {
  waveData: 'Sarobetsu  2020/11/20  00:00    100 Hz  4096\n' +
    '4 channels 5006 kHz      8006 kHz      6055 kHz      3925 kHz\n' +
    'UT         Freq    Amp  Freq    Amp  Freq    Amp  Freq    Amp\n' +
    'hh mm ss     Hz   dBuV    Hz  dBuV    Hz   dBuV    Hz   dBuV\n' +
    '00 00 30   0.00   28.9  0.37  19.0  1.17   50.9  1.44  -19.4\n' +
    '00 00 40   0.00   28.8  0.39  18.0  1.20   48.6  2.93  -18.4\n' +
    '00 00 50   0.00   27.6  0.29  16.6  1.15   47.7  2.27  -18.4\n' +
    '00 01 00   0.00   24.1  0.29  17.6  1.15   47.5 -1.93  -19.2\n' +
    '00 01 10  -0.02   21.2  0.27  14.9  1.17   49.3  2.69  -19.0\n' +
    '00 01 20   0.00   22.6 -0.05  13.1  1.17   54.6  2.69  -16.4\n' +
    '00 01 30   0.00   23.9  0.37  12.8  1.17   55.7  2.27  -18.7\n' +
    '00 01 40   0.00   25.1  0.37  16.5  1.17   54.6  2.25  -19.2\n' +
    '00 01 50   0.02   23.5 -0.02  14.6  1.17   54.1 -3.34  -18.8\n' +
    '00 02 00   0.02   22.7  0.00  10.3  1.17   54.1 -2.29  -17.8\n' +
    '00 02 10   0.02   24.6  0.00   7.5  1.17   52.6 -1.54  -17.2\n' +
    '00 02 20   0.02   22.5  0.39  10.0  1.15   47.6  3.22  -18.6\n' +
    '00 02 30   0.02   20.5  0.00  11.8  1.15   51.8  3.25  -18.3\n' +
    '00 02 40   0.02   22.9  0.37  11.5  1.15   55.2  0.68  -18.9\n' +
    '00 02 50   0.02   25.6  0.37  10.6  1.15   53.2  0.93  -18.1\n' +
    '00 03 00   0.02   28.9  0.37  10.6  1.29   47.9 -0.88  -17.8\n' +
    '00 03 10   0.02   28.1 -0.02  12.2  1.37   47.0 -0.90  -17.6\n' +
    '00 03 20   0.02   27.2 -0.02  16.0  1.20   48.6  1.66  -17.8\n' +
    '00 03 30   0.02   27.6 -0.02  16.9  1.17   54.2 -1.05  -19.0\n' +
    '00 03 40   0.02   25.7 -0.02  16.2  1.15   53.2  1.49  -18.8\n' +
    '00 03 50   0.02   29.4  0.00  16.6  1.15   53.9  3.49  -19.0\n' +
    '00 04 00   0.02   32.6  0.00  16.3  1.15   52.9  3.49  -19.6\n' +
    '00 04 10   0.02   31.1  0.00  16.6  1.15   51.7 -0.76   -0.2\n' +
    '00 04 20   0.05   26.0  0.00  17.9  1.20   53.3 -1.20    7.3\n' +
    '00 04 30   0.00   27.5  0.00  15.8  1.17   51.8  0.00    5.1\n' +
    '00 04 40   0.00   29.4  0.00  11.6  1.22   51.0  0.00   -7.7\n' +
    '00 04 50   0.00   29.3  0.02  10.6  1.22   50.0 -1.05  -18.6\n' +
    '00 05 00   0.05   26.3  0.02   5.2  1.15   46.4  0.85  -18.2\n' +
    '00 05 10   0.05   23.0 -0.05  10.1  1.17   50.9 -0.83  -19.6\n' +
    '00 05 20   0.05   19.5 -0.05  12.4  1.17   53.9  2.00  -18.5\n' +
    '00 05 30   0.02   18.4 -0.05  10.0  1.17   52.7 -0.61  -19.4\n' +
    '00 05 40   0.00   21.0 -0.02   8.1  1.17   53.0 -0.37  -19.9\n' +
```

図 3　抜き出したテキストデータ

4.4　データ整形処理

4.4.1　テキストデータを改行ごとに区切る

TypeScript の split 関数を用いてテキストデータを改行ごとに区切り，splitLineData という配列に格納する．split 関数はテキストに対して使用することができ，引数に渡した文字ごとにテキストを区切って配列にすることができる．今回は”\n”を引数に渡すことで，改行ごとにテキストデータを区切り配列に格納する．
図 5 に splitLineData の一部を示す．

4.4.2　データのラベルを取り出し，テキストデータのうちラベルにあたる最初の 4 行を削除する

TypeScript の slice 関数を用いて，ラベルが記述されている splitLineData の 2 行目を取り出して labelItem という配列に格納する．slice 関数は元の配列に影響を及ぼさないため，このような場合には適している．[8]
このデータは観測地点によってチャンネル数が異なっているため，チャンネル数に応じて処理を変える必要がある．それぞれテキストデータの 2 行 1 文字目にチャンネル数が記述されているため，labelItem の 0 番目の要素を数値に変換して number 型の定数 N に格納する．ラベルから必要なデータを抜き出したため，必要の無

```
[
  'Sarobetsu  2020/11/20  00:00    100 Hz  4096',
  '4 channels 5006 kHz     8006 kHz     6055 kHz     3925 kHz',
  'UT          Freq    Amp  Freq    Amp  Freq    Amp  Freq    Amp',
  'hh mm ss      Hz   dBuV    Hz   dBuV    Hz   dBuV    Hz   dBuV',
  '00 00 30    0.00   28.9  0.37   19.0  1.17   50.9  1.44  -19.4',
  '00 00 40    0.00   28.8  0.39   18.0  1.20   48.6  2.93  -18.4',
  '00 00 50    0.00   27.6  0.29   16.6  1.15   47.7  2.27  -18.4',
  '00 01 00    0.00   24.1  0.29   17.6  1.15   47.5 -1.93  -19.2',
  '00 01 10   -0.02   21.2  0.27   14.9  1.17   49.3  2.69  -19.0',
  '00 01 20    0.00   22.6 -0.05   13.1  1.17   54.6  2.69  -16.4',
  '00 01 30    0.00   23.9  0.37   12.8  1.17   55.7  2.27  -18.7',
  '00 01 40    0.00   25.1  0.37   16.5  1.17   54.6  2.25  -19.2',
  '00 01 50    0.02   23.5 -0.02   14.6  1.17   54.1 -3.34  -18.8',
  '00 02 00    0.02   22.7  0.00   10.3  1.17   54.1 -2.29  -17.8',
  '00 02 10    0.02   24.6  0.00    7.5  1.17   52.6 -1.54  -17.2',
  '00 02 20    0.02   22.5  0.39   10.0  1.15   47.6  3.22  -18.6',
  '00 02 30    0.02   20.5  0.00   11.8  1.15   51.8  3.25  -18.3',
  '00 02 40    0.02   22.9  0.37   11.5  1.15   55.2  0.68  -18.9',
  '00 02 50    0.02   25.6  0.37   10.6  1.15   53.2  0.93  -18.1',
  '00 03 00    0.02   28.9  0.37   10.6  1.29   47.9 -0.88  -17.8',
  '00 03 10    0.02   28.1 -0.02   12.2  1.37   47.0 -0.90  -17.6',
  '00 03 20    0.02   27.2 -0.02   16.0  1.20   48.6  1.66  -17.8',
  '00 03 30    0.02   27.6 -0.02   16.9  1.17   54.2 -1.05  -19.0',
  '00 03 40    0.02   25.7 -0.02   16.2  1.15   53.2  1.49  -18.8',
  '00 03 50    0.02   29.4  0.00   16.6  1.15   53.9  3.49  -19.0',
  '00 04 00    0.02   32.6  0.00   16.3  1.15   52.9  3.49  -19.6',
  '00 04 10    0.02   31.1  0.00   16.6  1.15   51.7 -0.76   -0.2',
  '00 04 20    0.05   26.0  0.00   17.9  1.20   53.3 -1.20    7.3',
  '00 04 30    0.00   27.5  0.00   15.8  1.17   51.8  0.00    5.1',
  '00 04 40    0.00   29.4  0.00   11.6  1.22   51.0  0.00   -7.7',
  '00 04 50    0.00   29.3  0.02   10.6  1.22   50.0 -1.05  -18.6',
  '00 05 00    0.05   26.3  0.02    5.2  1.15   46.4  0.85  -18.2',
  '00 05 10    0.05   23.0 -0.05   10.1  1.17   50.9 -0.83  -19.6',
  '00 05 20    0.05   19.5 -0.05   12.4  1.17   53.9  2.00  -18.5',
  '00 05 30    0.02   18.4 -0.05   10.0  1.17   52.7 -0.61  -19.4',
  '00 05 40    0.00   21.0 -0.02    8.1  1.17   53.0 -0.37  -19.9',
```

図 4　改行ごとに区切ったデータ

くなったテキストデータのラベル 4 行を splice 関数を用いて削除し，spliceData 関数に格納する．
図 6 に spliceData の一部を示す．

```
[
  '00 00 30    0.00   28.9  0.37   19.0  1.17   50.9  1.44  -19.4',
  '00 00 40    0.00   28.8  0.39   18.0  1.20   48.6  2.93  -18.4',
  '00 00 50    0.00   27.6  0.29   16.6  1.15   47.7  2.27  -18.4',
  '00 01 00    0.00   24.1  0.29   17.6  1.15   47.5 -1.93  -19.2',
  '00 01 10   -0.02   21.2  0.27   14.9  1.17   49.3  2.69  -19.0',
  '00 01 20    0.00   22.6 -0.05   13.1  1.17   54.6  2.69  -16.4',
  '00 01 30    0.00   23.9  0.37   12.8  1.17   55.7  2.27  -18.7',
  '00 01 40    0.00   25.1  0.37   16.5  1.17   54.6  2.25  -19.2',
  '00 01 50    0.02   23.5 -0.02   14.6  1.17   54.1 -3.34  -18.8',
  '00 02 00    0.02   22.7  0.00   10.3  1.17   54.1 -2.29  -17.8',
  '00 02 10    0.02   24.6  0.00    7.5  1.17   52.6 -1.54  -17.2',
  '00 02 20    0.02   22.5  0.39   10.0  1.15   47.6  3.22  -18.6',
  '00 02 30    0.02   20.5  0.00   11.8  1.15   51.8  3.25  -18.3',
  '00 02 40    0.02   22.9  0.37   11.5  1.15   55.2  0.68  -18.9',
  '00 02 50    0.02   25.6  0.37   10.6  1.15   53.2  0.93  -18.1',
  '00 03 00    0.02   28.9  0.37   10.6  1.29   47.9 -0.88  -17.8',
  '00 03 10    0.02   28.1 -0.02   12.2  1.37   47.0 -0.90  -17.6',
  '00 03 20    0.02   27.2 -0.02   16.0  1.20   48.6  1.66  -17.8',
  '00 03 30    0.02   27.6 -0.02   16.9  1.17   54.2 -1.05  -19.0',
  '00 03 40    0.02   25.7 -0.02   16.2  1.15   53.2  1.49  -18.8',
  '00 03 50    0.02   29.4  0.00   16.6  1.15   53.9  3.49  -19.0',
  '00 04 00    0.02   32.6  0.00   16.3  1.15   52.9  3.49  -19.6',
  '00 04 10    0.02   31.1  0.00   16.6  1.15   51.7 -0.76   -0.2',
  '00 04 20    0.05   26.0  0.00   17.9  1.20   53.3 -1.20    7.3',
  '00 04 30    0.00   27.5  0.00   15.8  1.17   51.8  0.00    5.1',
  '00 04 40    0.00   29.4  0.00   11.6  1.22   51.0  0.00   -7.7',
  '00 04 50    0.00   29.3  0.02   10.6  1.22   50.0 -1.05  -18.6',
  '00 05 00    0.05   26.3  0.02    5.2  1.15   46.4  0.85  -18.2',
  '00 05 10    0.05   23.0 -0.05   10.1  1.17   50.9 -0.83  -19.6',
  '00 05 20    0.05   19.5 -0.05   12.4  1.17   53.9  2.00  -18.5',
  '00 05 30    0.02   18.4 -0.05   10.0  1.17   52.7 -0.61  -19.4',
  '00 05 40    0.00   21.0 -0.02    8.1  1.17   53.0 -0.37  -19.9',
```

図 5　ラベル部分を削除したデータ

4.4.3　データを空白ごとに区切り，配列に格納する

TypeScript の map 関数を用いて，spliceData の要素ひとつひとつに対してそれぞれ split 関数で処理を行う．splitData 関数の要素，つまり改行で区切ったテキストデータは空白で区切って記述されている．そのため，split 関数を用いて空白ごとに区切り，二重配列 splitSpaceData に格納する．このデータは空白 1 つで区切られている場合と空白 2 つ場合があり，そのまま格納してしまうと空白の要素が生まれてしまう．そのため，filter 関数を用いて空白のみの要素を取り除く．filter 関数は配列に格納する際に用い，指定された要件に合った要素以外を排除することができる．今回は，filter 関数に空白 (" ") を渡し，さらに要素の長さが 1 に満たない要素を排除することで，空白のみの要素を取り除いている．図 7 に splitSpaceData の一部を示す．

```
[
  [
    '00',    '00',
    '30',    '0.00',
    '28.9',  '0.37',
    '19.0',  '1.17',
    '50.9',  '1.44',
    '-19.4'
  ],
  [
    '00',    '00',
    '40',    '0.00',
    '28.8',  '0.39',
    '18.0',  '1.20',
    '48.6',  '2.93',
    '-18.4'
  ],
  [
    '00',    '00',
    '50',    '0.00',
    '27.6',  '0.29',
    '16.6',  '1.15',
    '47.7',  '2.27',
    '-18.4'
  ],
  [
    '00',    '01',
    '00',    '0.00',
    '24.1',  '0.29',
    '17.6',  '1.15',
    '47.5',  '-1.93',
    '-19.2'
  ],
```

図 6　空白ごとに区切ったデータ

4.4.4　データを数値に変換する

データをフロントエンドでグラフのプロット等に使用できるようにするために，データの型を数値型にしておく必要がある．splitSpaceData 配列に格納されている要素全てを string 型から number 型に変換する．splitSpaceData 配列は二重配列であるため，二重に map 関数をかけることで要素全てを number 型に変換し，NumberData 配列に格納する．TypeScript では Number(データ) と返すことでデータを number 型に変換することができる．

4.4.5　データを json ファイルに出力

NumberData を，フロントエンドから求められているデータ様式の要件に沿うように json ファイルに出力する．今回は，Number 配列を forEach 関数を用いて処理する．forEach 関数は map 関数と同じコールバック関数で，json を扱うこともできる．forEach 関数は内部に value と index という変数を持つことができ，value が配列や json の要素のうち今扱っているものを，index が今扱っている要素が配列や json のうち何番目かをそれぞれ表している．今回の場合，value は Number 配列の中の配列になるため，テキストデータの改行 1 つ分を空白で区切った配列となる．　Number 配列は，二重配列の中の要素が，0 番目の要素から順に"hour"，"minute"，"sec-

ond”，”freq”，”amp”，”freq”，”amp”… となる配列になっている．観測地点ごとにチャンネルの数および周波数が異なるため，”freq”と”amp”の組み合わせがいくつあるかはデータによって異なる．そのため，value[0]〜value[2] をそれぞれ json ファイルの time プロパティの”hour”，”minute”，”second”に格納する．value[3] からは，”freq”と”amp”の 2 つごとに labelItem の偶数番目に格納されているチャンネルの情報をもとに json に格納する．

全ての要素に対して上記の処理を行ったのち，json をファイルに出力する．
生成した json の一部を図 8 に示す．


```
{
  "data": [
    {
      "time": { "hour": 0, "minute": 0, "second": 30 },
      "channel": {
        "3925": { "freq": 1.44, "amp": -19.4 },
        "5006": { "freq": 0, "amp": 28.9 },
        "6055": { "freq": 1.17, "amp": 50.9 },
        "8006": { "freq": 0.37, "amp": 19 }
      }
    },
    {
      "time": { "hour": 0, "minute": 0, "second": 40 },
      "channel": {
        "3925": { "freq": 2.93, "amp": -18.4 },
        "5006": { "freq": 0, "amp": 28.8 },
        "6055": { "freq": 1.2, "amp": 48.6 },
        "8006": { "freq": 0.39, "amp": 18 }
      }
    },
    {
      "time": { "hour": 0, "minute": 0, "second": 50 },
      "channel": {
        "3925": { "freq": 2.27, "amp": -18.4 },
        "5006": { "freq": 0, "amp": 27.6 },
        "6055": { "freq": 1.15, "amp": 47.7 },
        "8006": { "freq": 0.29, "amp": 16.6 }
      }
    },
    {
      "time": { "hour": 0, "minute": 1, "second": 0 },
      "channel": {
        "3925": { "freq": -1.93, "amp": -19.2 },
        "5006": { "freq": 0, "amp": 24.1 },
        "6055": { "freq": 1.15, "amp": 47.5 },
        "8006": { "freq": 0.29, "amp": 17.6 }
      }
    },
```


図 7　生成した json

4.5　プロットしたグラフ

実際にフロントエンドでプロットしたグラフを図 9 に示す．データの通りにプロットができており，データ整形とデータの受け渡しが上手くいっていることがわかる．

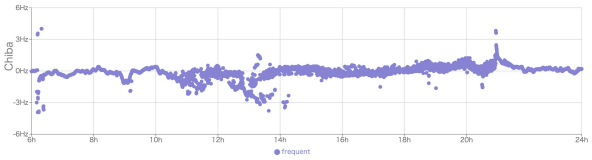

図 8　プロットしたグラフ

5.　おわりに

今回は電離圏の研究を目的とする HDF の観測データの利活用を促進するための新規 web アプリケーション作成に際し，オープンデータからスクレイピングを用いて観測データを取得，web アプリケーションのフロントエンドで扱いやすいよう整形するという処理の検討・実装を行った．データの観測地点ごとのチャンネル情報の差異や，フロントエンドで扱う際の都合を考慮して適切なスクレイピング・データ整形処理を行えたのではないかと考える．
必要とするデータが web サイト上にオープンデータとして公開されている場合は，データを取得する手段として web スクレイピングが有力な手法となる．また，web アプリケーション開発等でデータを扱う際はあらかじめ適切な形に整形しておくことが重要である．

参考文献

1) 吉川晃平，HF ドップラーにより観測された地震に伴う電離圏変動の中性待機波導数値シミュレーションによる定量的評価，電気学会論文誌 A Vol.136 No.5 pp.259 264
2) HF Doppler Sounding Experiment in Japan - HFDOPE `http://gwave.cei.uec.ac.jp/~hfd/pre.html` 2024/1/13
3) Next.js by Vercel - The React Framework `https://nextjs.org/` 2024/1/13
4) 中嶋柊，HF ドップラー観測データの利活用を目的とする Web アプリケーションのレンダリング設計，令和 5 年度 北九州工業高等専門学校 生産デザイン工学科 情報システムコース 卒業研究論文集
5) Superagent-npm `https://www.npmjs.com/package/superagent` 2023/01/25
6) cheeriojs/cheerio: Fast, flexible, and lean implemen- https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/GlobalObjects/Array/mapt ation of core jQuery designed specifically for the server. `https://github.com/cheeriojs/cheerio` 2023/01/25
7) Array.prototype.splice() - JavaScript — MDN `https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects/Array/splice` 2024/1/13
8) Array.prototype.slice() - JavaScript — MDN `https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects/Array/slice` 2024/1/13
9) Array.prototype.map() - JavaScript — MDN `https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects/Array/map` 2024/1/13
10) Array.prototype.filter() - JavaScript — MDN `https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects/Array/filter` 2024/1/13
11) Array.prototype.forEach() - JavaScript — MDN `https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach` 2024/1/13