JavaScript応用実習

02. Web API と Ajax

株式会社ジードライブ

今回学ぶこと

- Promise: 同期的な処理・非同期的な処理
- Web APIとAjax
- JavaScriptによるAjax通信
  - Fetch, Axios

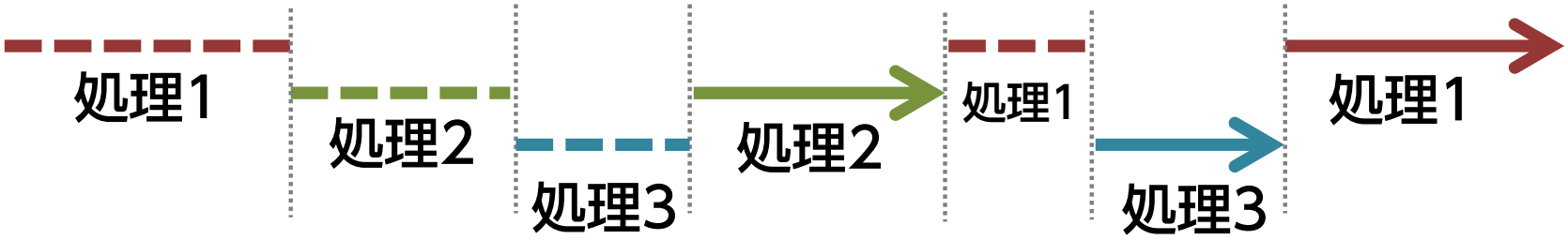
同期的/非同期的な処理

同期的な処理：

- ある処理が完了するまで、次の処理は開始しない

非同期的な処理：

- ある処理が開始した後、その完了を待たずに、次の処理を進める



setTimeout

- setTimeoutを含む関数は、非同期的に処理される

```
function printMessage(message, delay) {
    setTimeout(() => console.log(message), delay);
}

printMessage("メッセージ01", 2000);
printMessage("メッセージ02", 1000);
printMessage("メッセージ03", 1500);
```

実行結果

メッセージ02 *1秒後*
メッセージ03 *1.5秒後*
メッセージ01 *2秒後*

Promise

- Promiseオブジェクトを返す関数は同期的に実行できる
  - then()を使用することで、処理の完了を待って順番に実行できる

```
// Promiseを返す関数
function printMessage(message, delay) {
    return new Promise((resolve) => {
        setTimeout(() => {
            console.log(message);
            resolve();
        }, delay);
    });
}

printMessage("メッセージ01", 2000)
.then(() => {
   printMessage("メッセージ02", 1000)
   .then(() => printMessage("メッセージ03", 1500));
);
```

Promiseコンストラクタの引数は関数
⇒ この中に実行したい処理を記述する
resolveは処理が完了したことを示す関数

実行結果

メッセージ01 *2秒後*
メッセージ02 *3秒後*
メッセージ03 *4.5秒後*

async / await

- Promiseを返す関数はthen()の代わりにasync/await構文でも同期的に実行することができる

```
// Promiseを返す関数
function printMessage(message, delay) {
    return new Promise( …前頁と同様の記述… );
}

// async/await構文
async function printMessages() {
    await printMessage("メッセージ01", 2000);
    await printMessage("メッセージ02", 1000);
    await printMessage("メッセージ03", 1500);
}

printMessages();
```

asyncキーワードのついた関数内では、awaitを使い、Promiseを返す関数を同期的に実行できる

実行結果

メッセージ01 *2秒後*
メッセージ02 *3秒後*
メッセージ03 *4.5秒後*

Promiseと戻り値

- 最終的な戻り値はresolve()の引数として設定する
  - この戻り値は、then()メソッド内で利用することができる

```
// Promiseを返す関数
function getMessage(message, delay) {
    return new Promise((resolve) => {
        setTimeout(() => resolve(message), delay);
    });
}

const msg = getMessage("Hello", 1000);
console.log(msg);

getMessage("こんにちは", 1000)
.then((result) => console.log(result + "山田さん"));
```

最終的な戻り値

Promiseが返ってくる

最終的な戻り値
⇒then()の中で利用可能

実行結果

```
Promise { <pending> }
こんにちは山田さん
```

Promiseと戻り値

- 最終的な戻り値はawait()を使うことでも取得できる

```
// Promiseを返す関数
function getMessage(message, delay) {
    return new Promise( …前頁と同様の記述… );
}

// async/await構文
async function printMessages() {
    const m1 = await getMessage("メッセージ01", 2000);
    const m2 = await getMessage("メッセージ02", 1000);
    console.log(m1, m2);
    console.log(await getMessage("メッセージ03", 1500));
}

printMessages();
```

awaitがないとPromiseが入る

実行結果

```
メッセージ01 メッセージ02    3秒後
メッセージ03                 4.5秒後
```

Promiseとエラー処理

- Promiseのコンストラクタでは、reject関数を使い、エラーをスローすることができる

```
function getMessage(message, delay) {
    return new Promise((resolve, reject) => {
        if(delay >= 0) {
            setTimeout(() => resolve(message), delay);
        } else {
            reject("時間指定が不正");
        }
    });
}

getMessage("こんにちは", -1000)
.then((result) => console.log(result + "山田さん"))
.catch((error) => console.log(error));
```

エラーをスロー
⇒ catchメソッドで捕捉できる

実行結果

```
時間指定が不正
```

Promiseとエラー処理

- async関数内では、try/catch構文でエラーを捕捉できる

```
function getMessage(message, delay) { …前頁と同じ… }

async function printMessages() {
    try {
        const m1 = await getMessage("メッセージ01", 2000);
        console.log(m1);
        const m2 = await getMessage("メッセージ02", -1000);
        console.log(m2);
        const m3 = await getMessage("メッセージ03", 1500);
        console.log(m3);
    } catch (error) {
        console.log(error);
    }
}

printMessages();
```

エラーが発生
⇒ catchブロックに移る

実行結果

メッセージ01
時間指定が不正

練習

- 練習02-1

Web APIとは

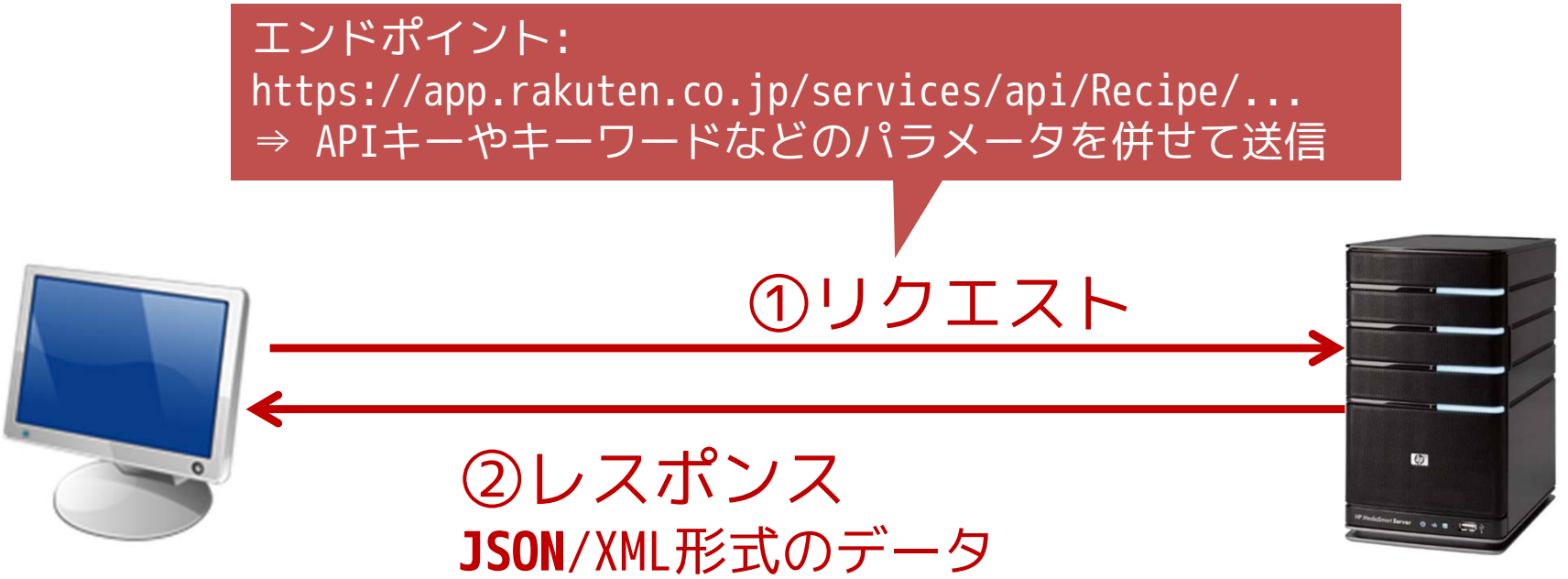
- 自社のもつデータや機能を、Webを通じて提供するサービス、またそのための窓口
  - 無償で提供されているサービスも存在する
  - 利用には登録が必要なことが多い
    ⇒ Web API利用に必要な「APIキー」を取得する

Web APIの例

企業	提供しているAPI
楽天	楽天市場、トラベル、レシピといったサービスで利用されているデータを提供
Google	Google Map、Gmail、YouTubeといったサービスで利用されているデータや機能を提供
Amazon	取り扱っている商品のデータを提供
Navitime	ルート検索などの機能を提供

Web APIとは

- エンドポイントと呼ばれるURLにリクエストを送ることで、データや機能にアクセスすることができる
  - JSON形式やXML形式のデータがレスポンスとして返ってくる



JSONとは

- JavaScript Object Notation
- JavaScriptのオブジェクトの表記方法がベースになっているデータ交換フォーマット
- JavaScriptと異なり、プロパティ名をダブルクォートで囲む必要がある(シングルクォートは不可)

JSのオブジェクト表記方法

```
{
    name: "taro",
    age: 33,
    address: "東京"
}
```

JSON

```
{
    "name" : "taro",
    "age" : 33,
    "address" : "東京"
}
```

JSONデータへのアクセス

- JavaScriptのオブジェクトとJSONデータ(JSON文字列)は相互変換が可能

JavaScriptオブジェクト ⇒ JSON文字列の変換

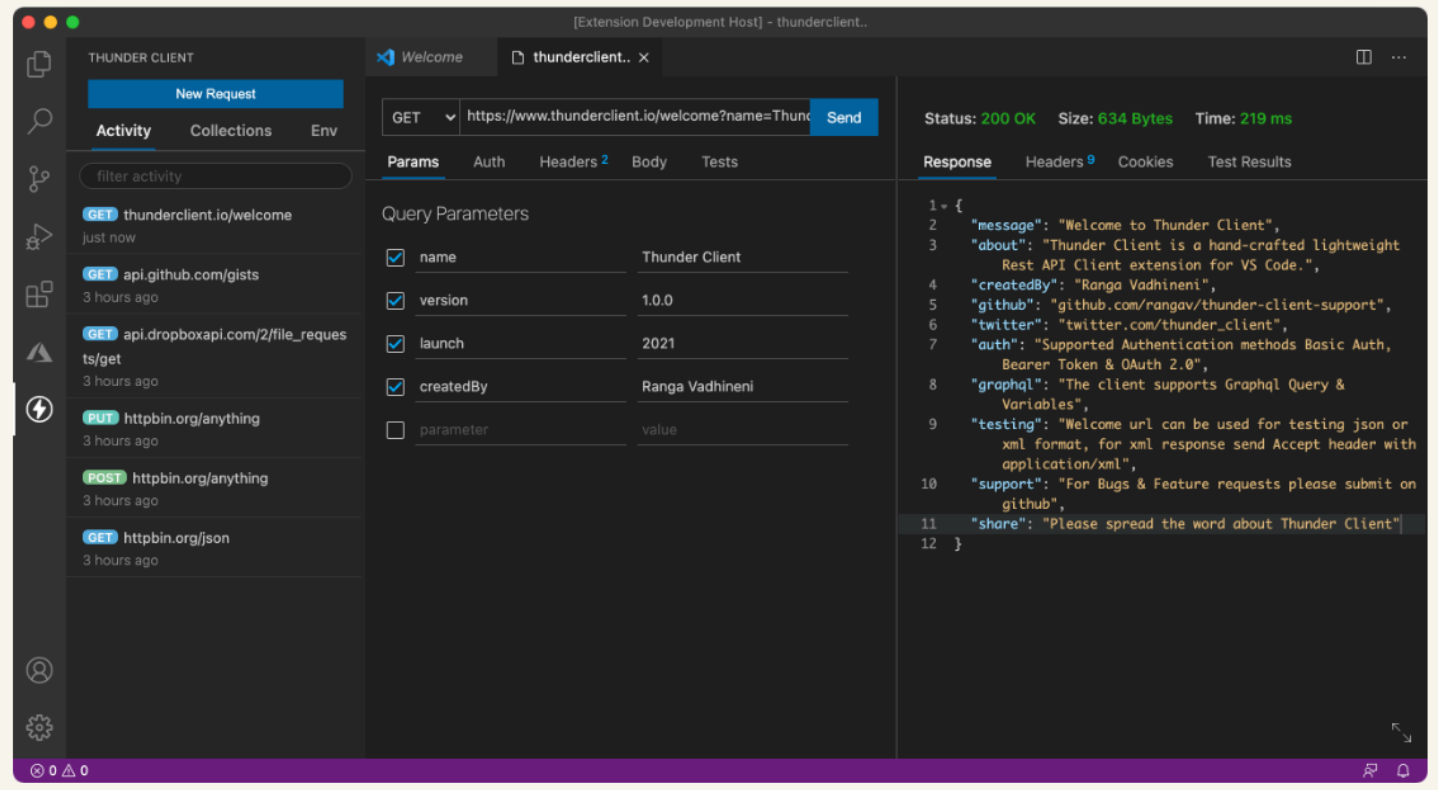
```
const user = {name: "山田太郎", age: 25};
const data = JSON.stringify(user);
```

JSON文字列 ⇒ JavaScriptオブジェクトへの変換

```
const data = '{"name": "山田太郎", "age": 25}';
const user = JSON.parse(data);
```

Thunder Client

- VS Codeの拡張機能で、開発時に使用するAPIテストツール
- JSONやXML形式でのデータのやり取りをシミュレーションすることができる

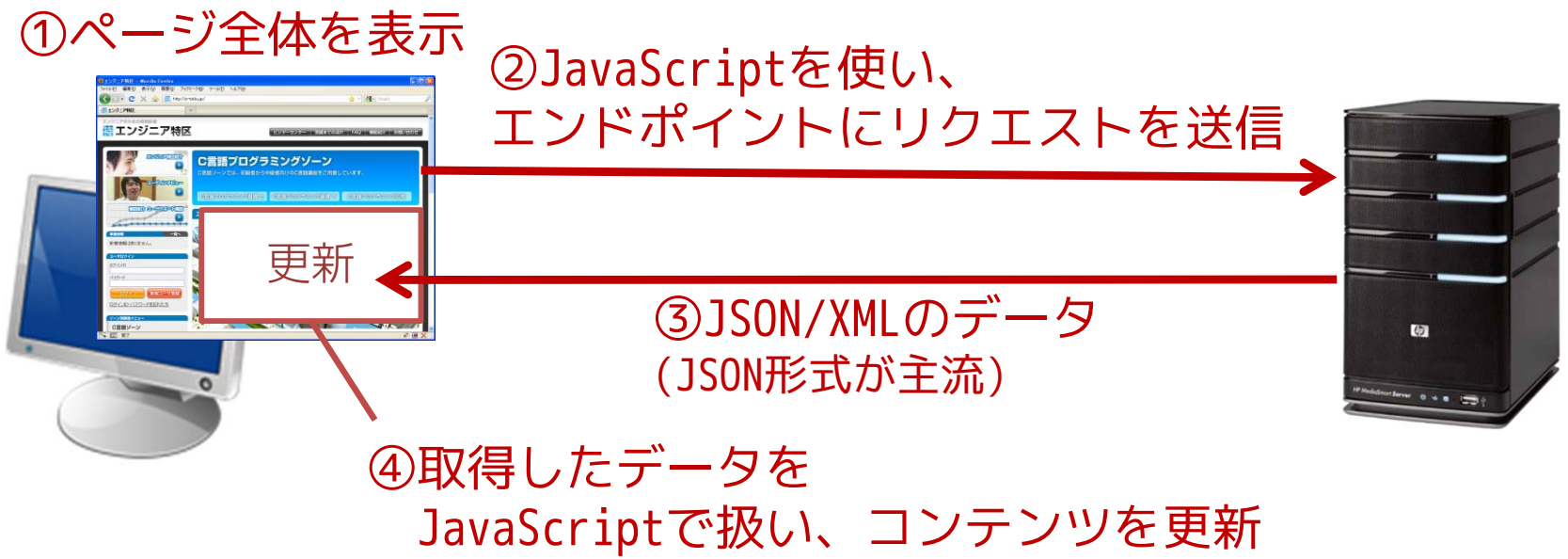


練習

- 練習02-2

Ajaxとは

- **A**synchronous **J**avaScript **a**nd **X**ML
  - JavaScriptを利用した非同期通信
- JavaScriptによるブラウザとWebサーバーとの通信により、ページ全体を読み込み直すことなく、動的にページの一部を更新するための仕組み



クロスドメインAjaxとは

- ページを配信しているサイトとは別のドメインに対してAjax通信を行うこと

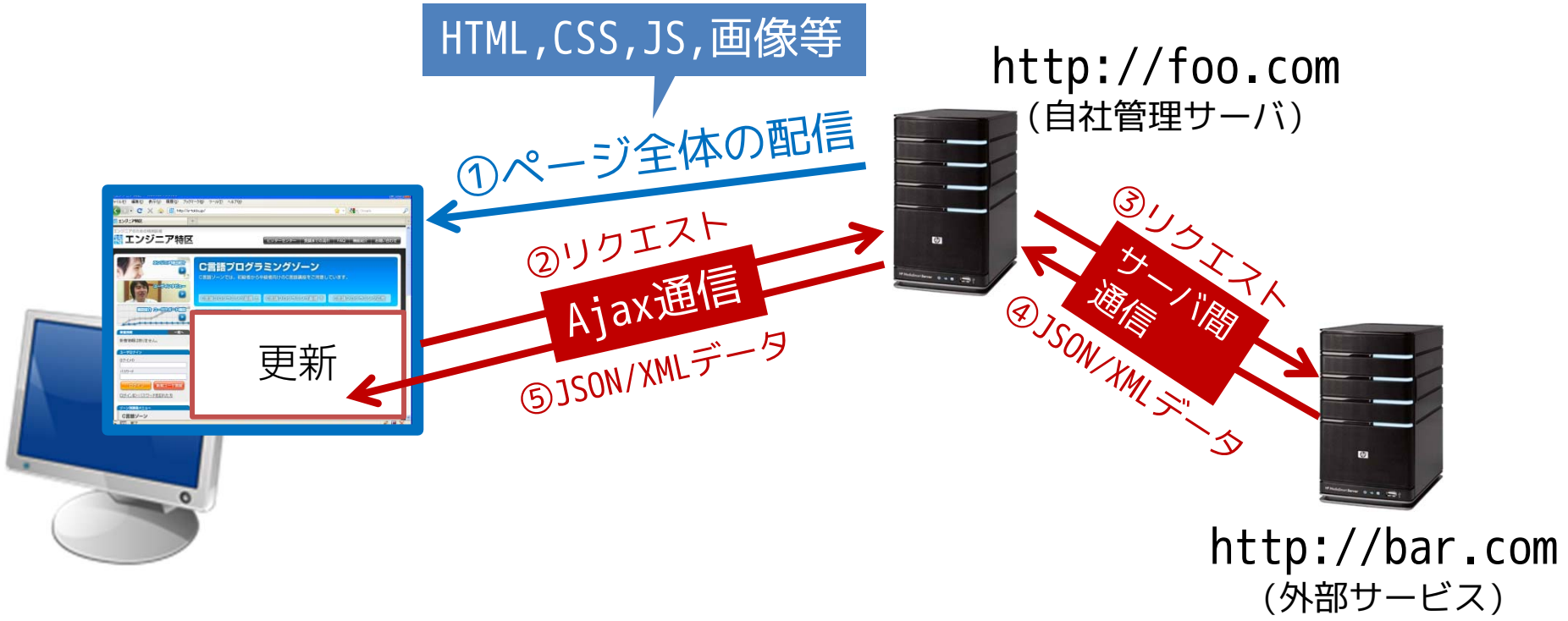
クロスドメインAjax

- ブラウザの制約(**同一オリジンポリシー**)により、通常はクロスドメインのAjax通信はできない
  - オリジン ＝ スキーム ＋ ホスト ＋ ポート

- サーバー側でこの制約が解除されている(CORS: Cross-Origin Resource Sharingの許可が設定がされている)場合、もしくはJSONPという形式でデータが配信されている場合は問題はない
  - これらが行われていない場合は、自社管理サーバにプロキシを設置する必要がある

自社管理サーバにプロキシを設置する

- サーバ間通信により目的のサイトからJSONを取得し、自サイトのページへ送信するプログラムを自社管理サーバに設置する



Ajax通信の方法

- JavaScriptを使用したAjax通信には以下の方法が考えられる

1. XMLHttpRequest(XHR)を利用する
2. jQuery(ライブラリ)の$.ajax()メソッドを利用する
3. fetchを利用する
4. Axios(ライブラリ)を利用する

この講座では、3・4番の方法を紹介する

fetchとは

- 標準で組み込まれているHTTP通信用関数
  - https://developer.mozilla.org/ja/docs/Web/API/fetch
  - 別途ライブラリを組み込む必要がないのが利点

- Promiseを返す関数で、then()メソッドによる利用方法とasync/await構文による利用方法がある

fetchの使用方法①

- データ取得後の処理はthen()メソッド内に定義する
  - Ajax通信に失敗した場合は、catch()メソッド内に記述する

```
fetch('APIのURL')
.then((response) => {
  // レスポンスボディをJavaScriptオブジェクトへ変換
  return response.json();
}).then((data) => {
  // データの処理(DOMへの追加など)
  console.log(data);
}).catch((error) => {
  // エラー発生時の処理
  console.log(error);
});
```

APIからの応答

変換されたデータ

エラー情報

fetchの使用方法②

- async/await構文を使う方が簡単に記述できる

```
async function getDataFromApi() {
  try {
    const response = await fetch('APIのURL');
    // レスポンスボディをJavaScriptオブジェクトへ変換
    const data = await response.json();
    // データの処理(DOMへの追加など)
    console.log(data);
  } catch(error) {
    // エラー発生時の処理
    console.log(error);
  }
}

getDataFromApi();
```

APIからの応答

変換されたデータ

エラー情報

練習

- 練習02-3

Axiosの使用方法①

- AxiosもfetchとPromiseを返す
  - fetchと比べ、then()の記述を一つ省略できる
  - 標準機能ではないので、ライブラリの読み込みが必要
    https://www.jsdelivr.com/package/npm/axios

```
axios.get('APIのURL')
.then((response) => {
  // データの処理(DOMへの追加など)
  console.log(response.data);
}).catch((error) => {
  // エラー発生時の処理
  console.log(error);
});
```

APIからの応答

エラー情報

Axiosの使用方法②

- async/await構文を使い、記述することも可能

```
async function getDataFromApi() {
  try {
    const response = await axios.get('APIのURL');
    // データの処理(DOMへの追加など)
    console.log(response.data);
  } catch(error) {
    // エラー発生時の処理
    console.log(error);
  }
}
```

APIからの応答

エラー情報

比較: fetch と Axios

- response(青地部分)の内容は同じだが、型が異なる
  - fetchの場合:
    https://developer.mozilla.org/ja/docs/Web/API/Response
  - Axiosの場合:
    https://axios-http.com/ja/docs/res_schema
- data(緑字部分)の中身は同じものになる
  - responseからの取り出し方が異なる

fetch

```
const response = await fetch('APIのURL');
const data = await response.json();
```

Axios

```
const response = await axios.get('APIのURL');
const data = response.data;
```

練習

- 練習02-4

パラメータの送信

- エンドポイントのURLや送信するパラメータは、各サービスごとに決められている
  - ドキュメントを読む必要がある

例）OpenWeatherAPIの送信パラメータ：https://openweathermap.org/current

パラメータ名	説明
lat	緯度(latitude)
lon	経度(longitude)
appid	APIキー
mode	JSON形式以外のデータを利用する場合に指定
units	単位の設定
lang	言語の設定

必須項目

パラメータの送信方法(fetch)

方法① クエリストリングとして送信する

```
const url = 'https://api.openweathermap.org/data/2.5/weather' +
            '?lat=35&lon=139&appid=123ABC';
const respons = await fetch(url);
```

クエリストリング

方法② データオブジェクトをクエリストリングに変換して送信する

⇒ **new URLSearchParams()** で変換する

```
const parameter = {
    lat: 35, lon: 139, appid: '123ABC'
};
const queryParameter = new URLSearchParams(parameter);
const url = 'https://api.openweathermap.org/data/2.5/weather'
          + '?' + queryParameter;
const respons = await fetch(url);
```

パラメータの送信方法(axios)

方法① クエリストリングとして送信する

```
const url = 'https://api.openweathermap.org/data/2.5/weather' +
            '?lat=35&lon=139&appid=123ABC';
const respons = await axios.get(url);
```

方法② データオブジェクトとして送信する

⇒ **params**プロパティにセットする

```
const parameter = {
    lat: 35, lon: 139, appid: '123ABC'
};
const respons = await axios.get(url, {params: parameter});
```

データの取得

- APIを通じて、どのようなデータが返ってくるかは、各サービスごとに決められている
  - ドキュメントを読む必要がある

例）OpenWeatherAPI：https://openweathermap.org/current#current_JSON

キー1	キー2	説明
weather（配列）	main	天気
	description	天気の説明
	icon	天気の画像アイコン
wind	speed	風速
	deg	風向き
main	temp	気温
	humidity	湿度
name	-	地名

データの取得・利用例

- OpenWeatherAPIから取得したデータを表示する

```
const API_URL = 'https://api.openweathermap.org/data/2.5/weather';

// 緯度、経度を元に地名と天気を表示する関数
async function getData(lat, lon) {
  const parameter = {lat, lon, appid: '123ABC'};
  const queryParameter = new URLSearchParams(parameter);
  const res = await fetch(API_URL+ '?' + queryParameter);
  const data = await res.json();
  // 地名と天気をh1,h2要素として表示
  document.querySelector('h1').textContent = data.name;
  document.querySelector('h2').textContent = data.weather[0].main;
}

getData(39.8865, -83.4483); // Londonの天気を表示
```

データの取得・利用の注意点

- async関数やthen()の戻り値はPromiseオブジェクトになる

```
// async関数
async function getData() {
  const response = await fetch('APIのURL');
  const data = await response.json(); // データの取得
  console.log(data); // APIから取得したデータが表示される
  return data; // APIから取得したデータを戻り値にしている
}

const data = getData();
console.log(data); // APIから取得したデータは表示されない
```

Promiseオブジェクトが
出力される

```
▼Promise {<pending>} i
  ▶[[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
  ▶[[PromiseResult]]: Array(10)
```

データの取得・利用の注意点

- APIから取得したデータを戻り値にする場合、それを利用する際には、then() または async/awaitを使用する必要がある

```
async function getData() {
  const response = await fetch('APIのURL');
  const data = await response.json(); // データの取得
  return data; // APIから取得したデータを戻り値にしている
}

getData().then((data) => console.log(data));

async function showData() {
   const data = await getData();
   console.log(data);
}

showData();
```

練習

- 練習02-5
- 練習02-6