HTML5 Professional Certification

HTML5プロフェッショナル認定 レベル1 技術解説無料セミナー

2024/03/21 開催

主題	1.5.4 通信系API概要
副題	XMLHttpRequest(XHR) / fetch API

本日の講師

FUJITSU
株式会社富士通ラーニングメディア
吉田 隼人

LPI-JAPAN

自己紹介

- 吉田隼人
  - 所属企業：　株式会社富士通ラーニングメディア
  - 担当業務：　研修講座の運営、登壇
  - 趣味：　旅行、ゴルフ、英会話


Batu Caves, Malaysia

- これまでに登壇した研修講座の分野
  - C言語、C++、Java、C#、VB
  - 組み込みソフトウェア開発（6502マイコン互換機の機械語・アセンブリ言語）
  - **HTML、CSS、JavaScript**
  - ソフトウェアテスト技術
  - オブジェクト指向デザインパターン
  - データベース論理設計、オブジェクト指向データモデリング
  - サーバー仮想化ソフトウェア、クラウドコンピューティング
- 登壇以外の業務
  - クラウド人材の育成企画立案および推進
  - クラウドサービスの運用支援、マニュアル開発（DITA、XSLTスタイルシート）
  - 業務マニュアル開発のプロジェクトマネジメント
  - 無線LAN環境の構築（RADIUSサーバーの構築）
  - **社内教育ポータルサイト**の機能開発（PHP、jQuery、PostgreSQL）

所属企業のご紹介

株式会社富士通ラーニングメディア

- 事業内容：法人向け人材育成・研修サービス
  - **人材育成**に関するコンサルティング、人材力診断／適性診断等の提供
  - **研修講座**（コース、カリキュラム）の企画、開発、実施、運営および運営支援
  - コース教材／**マニュアル**等の開発、制作、**翻訳**、**出版**および販売
  - 人材／研修講座の運営／マニュアル制作の管理に関連する**ソフトウェア**の開発および販売

- 事業所・関連施設
  - ✓ 本社（川崎市）
  - ✓ 札幌事業所
  - ✓ 仙台事業所
  - ✓ 福島事業所
  - ✓ 大宮事業所
  - ✓ 品川サテライトオフィス
  - ✓ 神奈川事業所
  - ✓ 石川事業所
  - ✓ 中部事業所
  - ✓ 関西事業所
  - ✓ 広島事業所
  - ✓ 九州事業所
  - ✓ 札幌ラーニングセンター
  - ✓ 品川ラーニングセンター
  - ✓ 名古屋ラーニングセンター
  - ✓ 関西ラーニングセンター
  - ✓ 沖縄ラーニングセンター

所属企業のご紹介

関連講座

JavaScript基礎ステップアップAPI編 ～ストレージ・通信・デバイスの利用

￥99,000

Webページのオフライン動作やサーバとの通信処理、ハードウェアアクセスなどを実現するためのWeb StorageやWebSocket、Geolocation APIなどのJavaScriptの**API**を講義と実習によって学習します。

集合開催

東京
2024/07/04-05

Web技術者のためのREST API開発（JS編） ～構築から利用まで～

￥99,000

JavaScriptとNode.js、expressを使用した**REST APIの実装方法**および、jQueryを使用したREST APIとの連携方法を、説明と実習を通して学習します。実習では、Node.jsを使用したWebサーバの起動からWebブラウザへのデータ送信方法までを体験します。

集合開催

東京
2024/06/10-11
2024/09/26-27

大阪
2024/09/19-20

Web開催

2024/07/08-09
2024/08/15-16

体験！初めてのReact.js ～SPA開発編～

￥55,000

React.jsでWebアプリケーション開発をするための基本を学習します。コンポーネントなどの概念や、JSXといった基本構文を学習しながら、簡単なSPA（シングルページアプリケーション）を実装します。

集合開催

東京
2024/07/24

名古屋
2024/09/11

Web開催

2024/08/19
2024/09/02

■HTML5プロフェッショナル認定とは

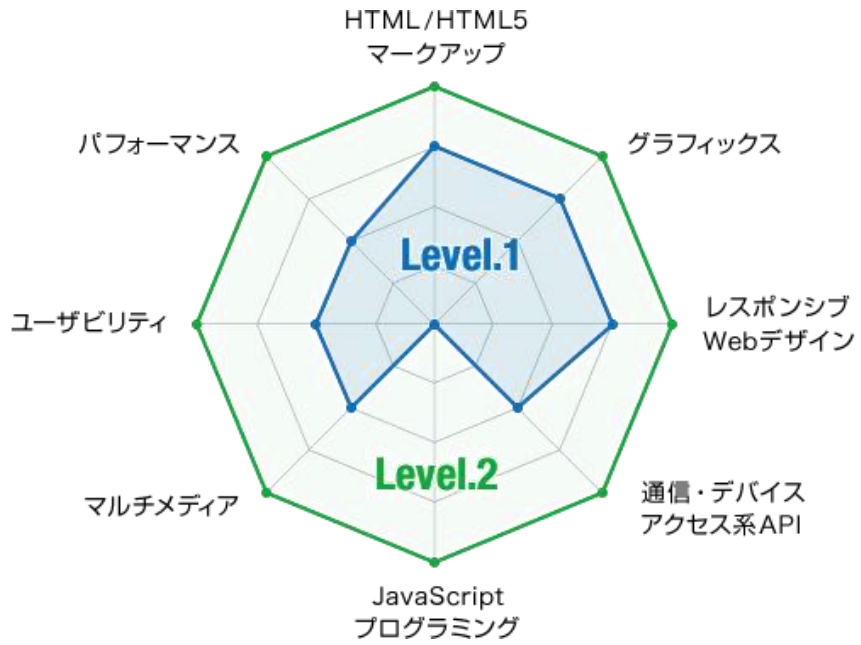
WEBサイトやWEBアプリケーションを開発する上で必須である HTML5／CSS JavaScriptなどについての技術力を証明する認定です。基礎から網羅的に学ぶことは、効率的に開発を行う上できっと役立つことでしょう。

✓レベル１はHTMLとCSS

HTMLの基本的な部分からレスポンシブデザインが中心で、サイト制作のためのスキルの証明

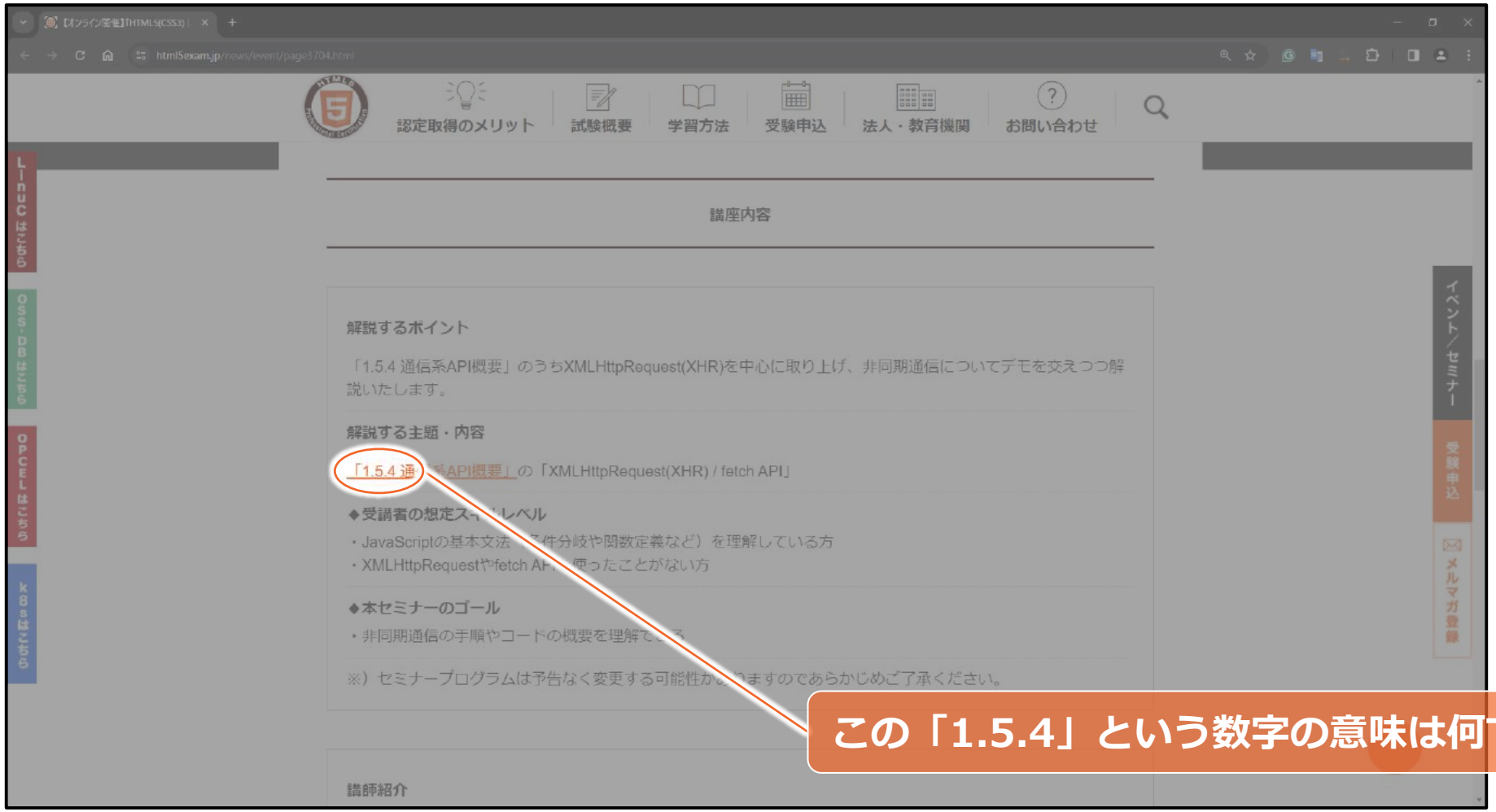
✓レベル２ではJavaScript

JavaScriptを使ってWEBアプリケーションを構築できるだけのスキルの証明



今回のテーマ

■ 今回のセミナーの開催案内画面



今回のテーマ

- **HTML5プロフェッショナル認定試験 レベル1 出題範囲の案内ページ**

今回のテーマ

「1.5.4 通信系API概要」の詳細

1.5.4 通信系API概要

重要度

★★★ 3

出題種別

- 知識問題
- 記述問題

説明（望まれるスキル）

JavaScriptからさまざまな通信プロトコルを使ってクラウドと通信する仕組みと特性を理解し、利用シーンに応じて適切なAPI選択ができる。

主要な知識範囲

- Ajax
- 双方向データリアルタイム通信
- サーバープッシュ

重要な技術要素

- XMLHttpRequest(XHR) / fetch API
- WebSocket API
- Server-Sent Events
- WebRTC

この辺りが今回のテーマです。

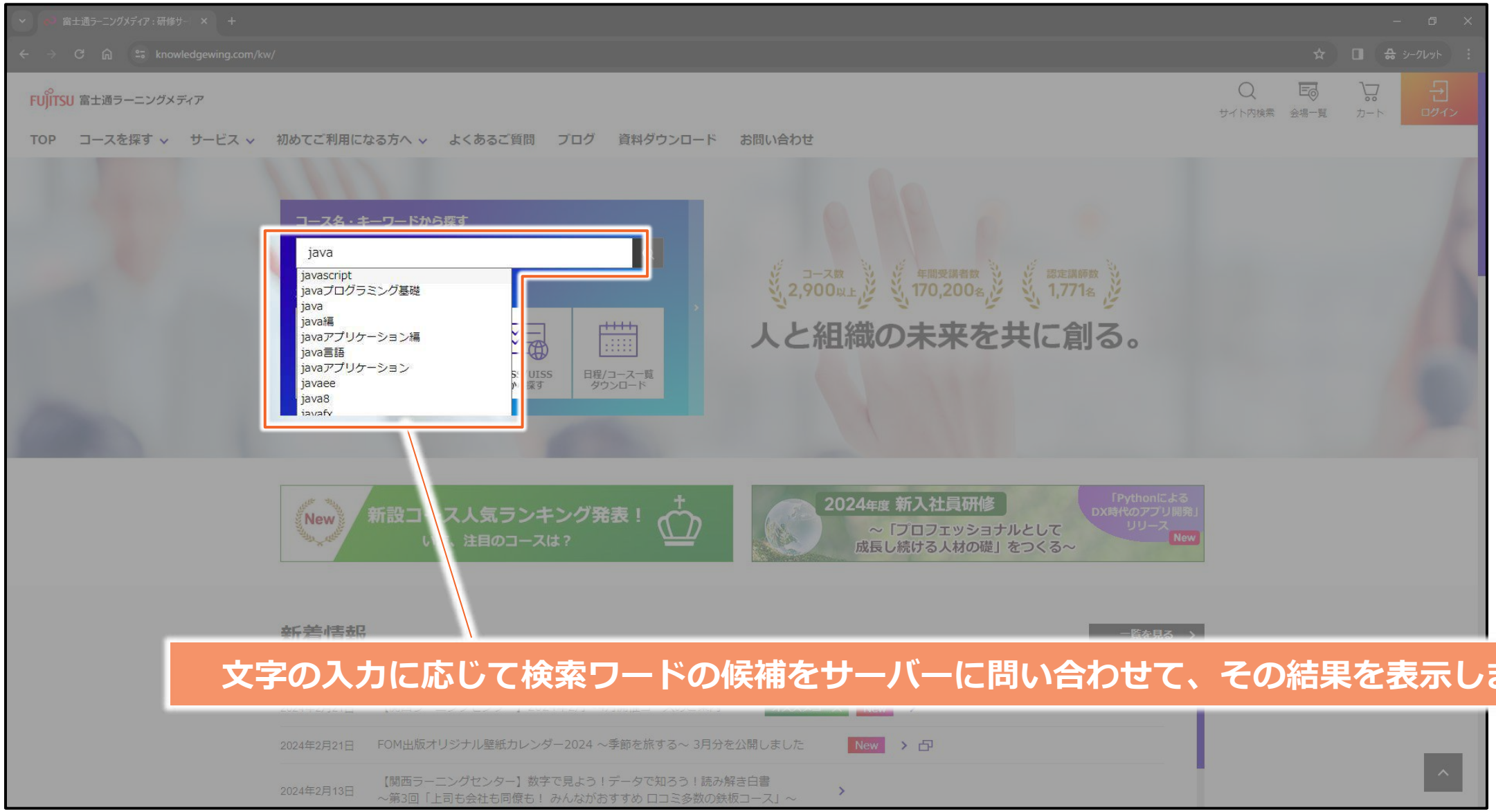
本日の目次

1. 用語の確認
   - ① Ajax
   - ② XML
   - ③ JSON
2. XMLHttpRequest(XHR)
3. Fetch API

1. 用語の確認

Ajax, XML, JSON

Ajaxの例



文字の入力に応じて検索ワードの候補をサーバーに問い合わせて、その結果を表示します

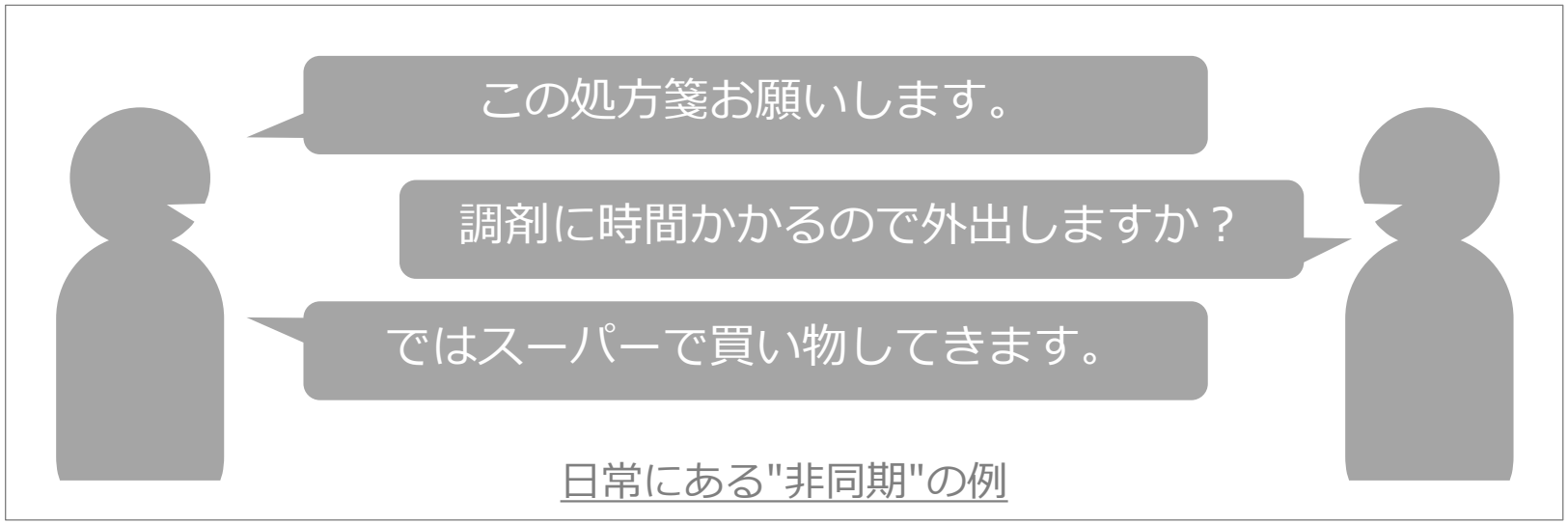
Ajaxとは

Ajax（Asynchronous JavaScript and XML）

- 非同期通信によってクライアントとサーバーがデータをやり取りし、**動的にページの一部を書き換えて**いく処理方式。

非同期通信

- Webブラウザー上の処理と**並行して**Webサーバーとのやり取りを進める通信方式。
  - 通信の完了を待つことなく、レスポンスがある前でもユーザーによる操作を受け付け可能です。



日常にある"非同期"の例

Ajaxの利点

- Ajaxを活用することで操作性の高いWebページを実現可能です。
  - 動的にページの一部を書き換えられなかった頃は、地図サイトなどの操作性は悪かったです。

移動や拡大/縮小のために、画面上のボタン等を操作する必要がある

XMLとは

- XML（eXtensible Markup Language）
  - 文書やデータの意味や構造を記述するための拡張可能なマークアップ言語。
    - HTMLと同様にタグを使いますが、タグ名や属性名が規格化されてはいません。

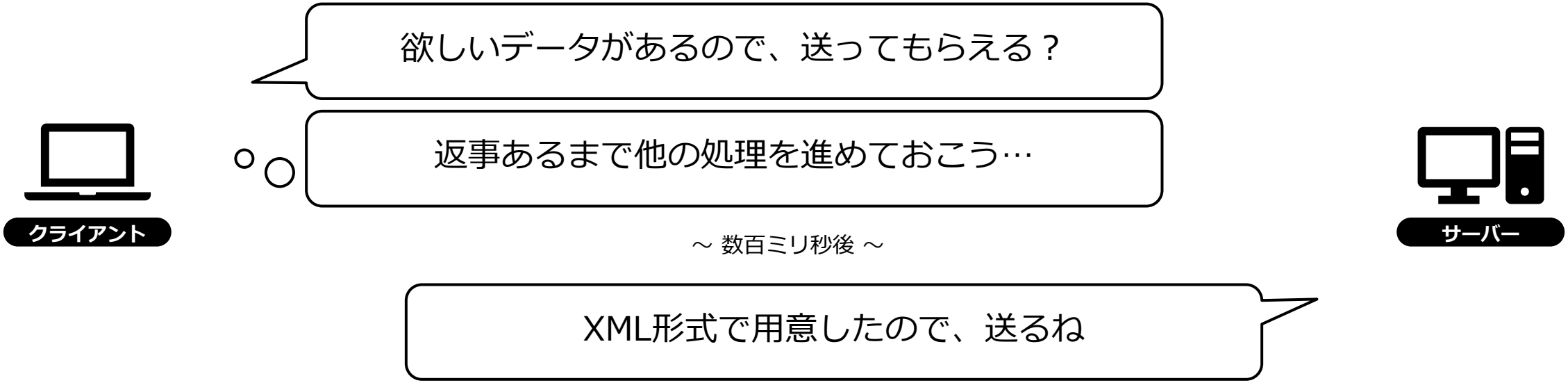
例

```
<suggestion>
        <keywords>
            <candidate>javascript</candidate>
            <candidate>javaプログラミング基礎</candidate>
            <candidate>java</candidate>
            <candidate>java編</candidate>
            <candidate>javaアプリケーション編</candidate>
            <candidate>java言語</candidate>
            <candidate>javaアプリケーション</candidate>
            <candidate>javaee</candidate>
            <candidate>java8</candidate>
            <candidate>javafx</candidate>
        </keywords>
        <keyword>java</keyword>
        <request_id>4 </request_id>
</suggestion>
```

（ここで改めて）Ajaxとは

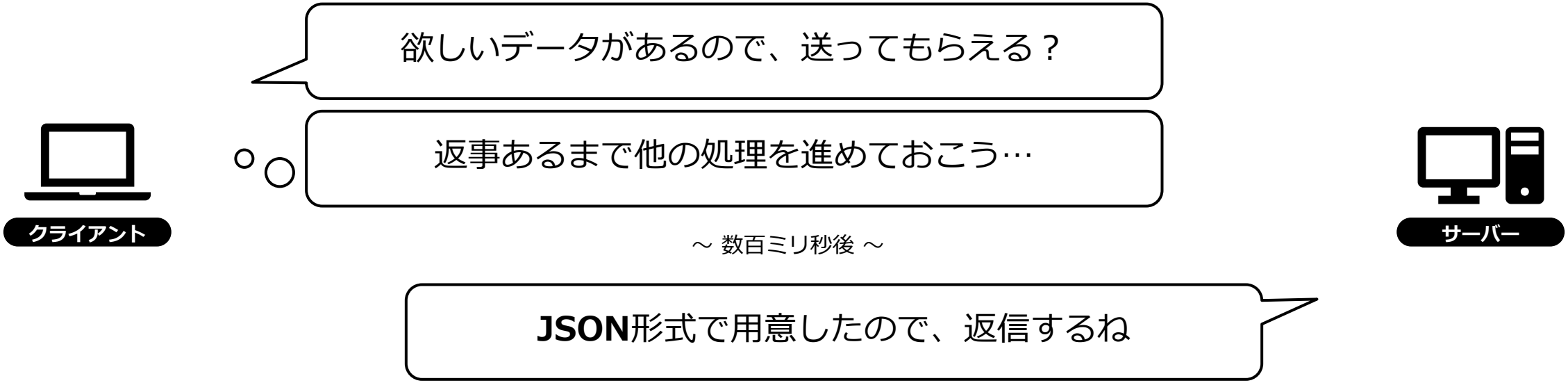
- つまり、クライアント側においてJavaScriptを用いて非同期通信を発生*させ、その結果をXML形式で受け取る方式がAjaxです。

  ＊非同期通信を発生させるきっかけは、ユーザーによる何らかの操作や、一定の時間経過などです。



しかし実際のところ…

- Ajaxと言いつつ、XML形式の代わりにJSON形式を使うことも多いです。



XML vs JSON

- 一番分かりやすい違いは、データサイズです。
  - 先ほどの例の場合、下記のとおり約2倍の違いがあります。

XML

```
<suggestion>
    <keywords>
        <candidate>javascript</candidate>
        <candidate>javaプログラミング基礎</candidate>
        <candidate>java</candidate>
        <candidate>java編</candidate>
        <candidate>javaアプリケーション編</candidate>
        <candidate>java言語</candidate>
        <candidate>javaアプリケーション</candidate>
        <candidate>javaee</candidate>
        <candidate>java8</candidate>
        <candidate>javafx</candidate>
    </keywords>
    <keyword>java</keyword>
    <request_id>4 </request_id>
</suggestion>
```

446文字
※空白文字含む

JSON

```
{
    "keywords": [
        "javascript",
        "javaプログラミング基礎",
        "java",
        "java編",
        "javaアプリケーション編",
        "java言語",
        "javaアプリケーション",
        "javaee",
        "java8",
        "javafx"
    ],
    "keyword": "java",
    "request_id": "4"
}
```

215文字
※空白文字含む

- おすすめ記事：Amazon Web Services社のサイトに詳細な比較情報が掲載されています。
  - https://aws.amazon.com/jp/compare/the-difference-between-json-xml/

JSONとは

- JSON（JavaScript Object Notation）
  - JavaScriptのオブジェクトの構文と同様の形式によるデータ表記法。
    - 文字列（キー）と値のペアを記述します。値の部分には配列なども指定可能です。
    - JavaScriptのコード内でJSON.parseメソッドやJSON.stringifyメソッドを使用することでJSON形式文字列とオブジェクトを相互に変換できます（次ページ掲載）。

例

```
{
    "keywords": [
        "javascript",
        "javaプログラミング基礎",
        "java",
        "java編",
        "javaアプリケーション編",
        "java言語",
        "javaアプリケーション",
        "javaee",
        "java8",
        "javafx"
    ],
    "keyword": "java",
    "request_id": "4"
}
```

JSON形式文字列とオブジェクトの相互変換

- JavaScriptの標準組み込みオブジェクトである「JSON」を用いての変換が可能です。

```
// 1. オブジェクトの作成
let jsonObject = {"keywords": ["javascript","javaプログラミング基礎"],"keyword": "java","request_id": "4"};

// 2. 「オブジェクト→文字列」の変換
let jsonText = JSON.stringify(jsonObject);

// 3. 「（JSONの表記法に則っている）文字列→オブジェクト」の変換
let jsonObject2 = JSON.parse(jsonText);
```

（スペースの関係上、配列の要素を前ページまでの例から少し削りました）

変数「jsonText」が扱うデータは文字列なので、下記のように比較してみると、結果はtrueとなります。

```
jsonText == "{¥"keywords¥":[¥"javascript¥",¥"javaプログラミング基礎¥"],¥"keyword¥":¥"java¥",¥"request_id¥":¥"4¥"}"
```

また、下記のようにlengthで文字数を確認できます。

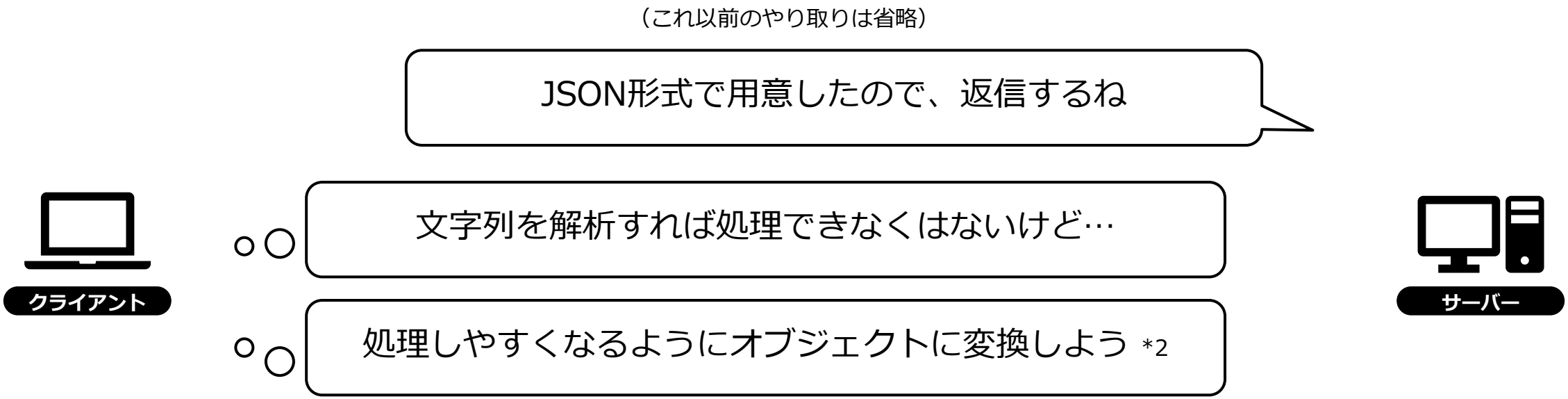
```
jsonText.length
```

変数「jsonObject」や「jsonObject2」が扱うデータはオブジェクトなので、下記のようにプロパティ名を指定できます。

```
alert(jsonObject2.keyword);
alert(jsonObject2.keywords[0]);
```

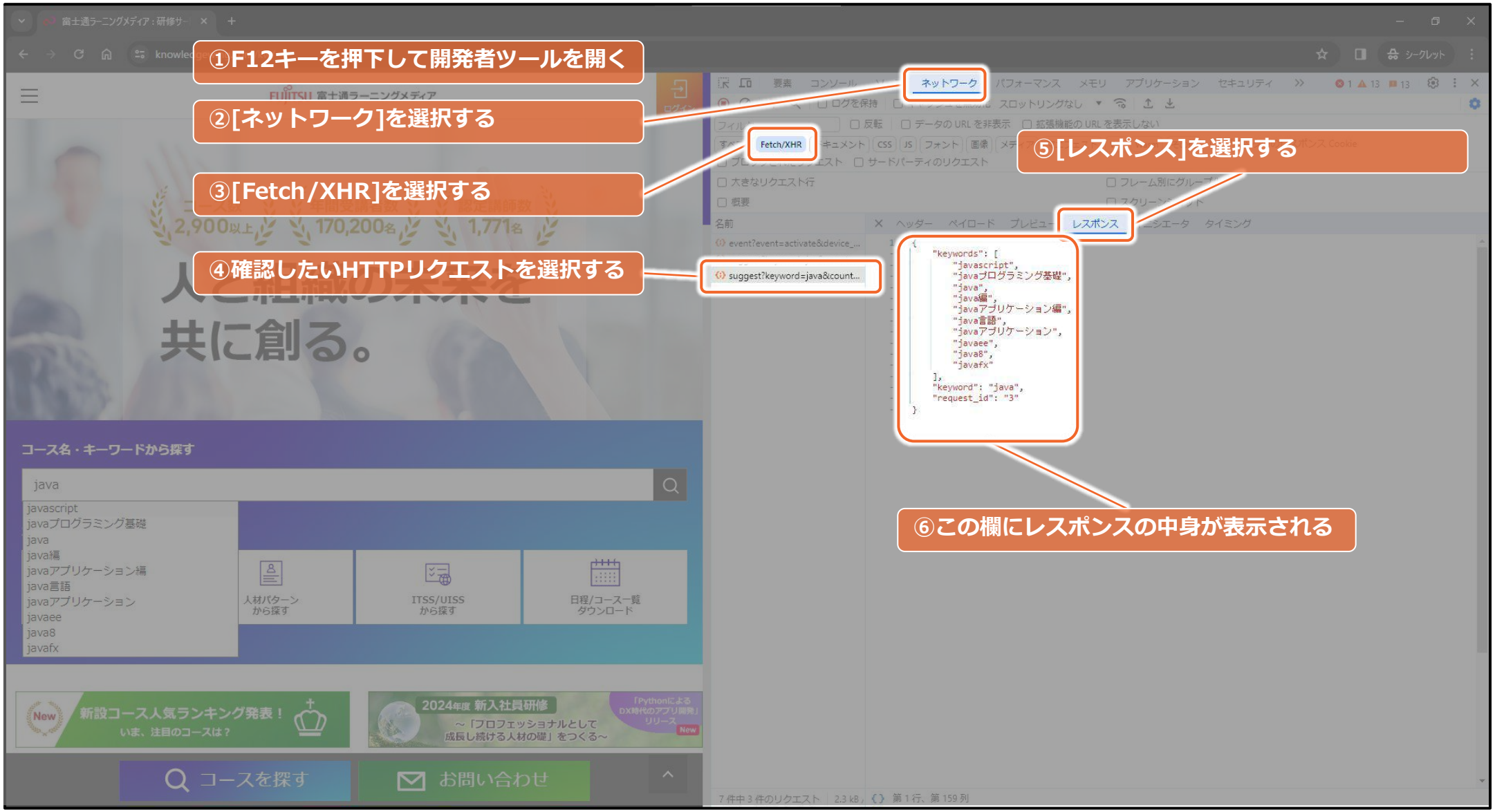
変換が必要な場面の例

- サーバーから返信されるデータはJSON形式の**文字列**[*1]であるため、クライアント側では一般的に「**文字列→オブジェクト**」の変換を行います。



*1 Webブラウジング時にHTMLがテキストデータとして通信されるのと同様に、サーバーから返信されるJSON形式の情報もテキストデータ（＝文字列）です。
*2 変換した方が処理しやすいという点は、XML形式にも同じことが言えます。

サーバーからの返信データを一応確認しておきましょう



この章のまとめ

- **ここまでAjaxに関して確認してきました。ポイントは下記のとおりです。**
  - クライアント側（Webブラウザ）においてJavaScriptを用いて非同期通信を発生させ、サーバーからの応答が返ってくるまでの間にユーザーによるWebブラウザの操作を受け付けることができる。
  - サーバーからの応答として、XML形式やJSON形式（の文字列）が返ってくる。
  - JSONの場合は、下記のメソッドでオブジェクトと文字列を相互に変換できる。
    - 文字列用変数 = **JSON.stringify**(オブジェクト);
    - オブジェクト用変数 = **JSON.parse**(文字列);
  - サーバーからの応答結果を用いて、動的にページの一部を書き換えることができる。

- **次の章以降で、JavaScriptを用いて非同期通信を発生させる方法を確認します。**

XMLHttpRequestとFetch APIの話の前に…

ここからの説明で使うAPI

- ここからは、株式会社アイビス様が提供している郵便番号データ配信サービスである「zipcloud」の郵便番号検索APIを使って、XMLHttpRequestとFetch APIのコード例を紹介します。

例

クライアント

https://zipcloud.ibsnet.co.jp/api/search?**zipcode=9000015**

サーバー

```
{
    "message": null,
    "results": [
        {
            "address1": "沖縄県",
            "address2": "那覇市",
            "address3": "久茂地",
            "kana1": "ｵｷﾅﾜｹﾝ",
            "kana2": "ﾅﾊｼ",
            "kana3": "ｸﾓｼﾞ",
            "prefcode": "47",
            "zipcode": "9000015"
        }
    ],
    "status": 200
}
```

このJSON形式文字列がオブジェクトに変換された場合、例えば「沖縄県」の部分は下記の記述で読み取れます。
オブジェクト名.result[0]["address1"]

2. XMLHttpRequest

(XHR)

XMLHttpRequestとは

- 非同期のHTTP通信を開始し、結果を受け取ることができるオブジェクトです。
  - 細かく説明すると、、、
    - JavaScriptを使用した非同期のHTTP通信に関する仕様である「XMLHttpRequest API」がまず規定されています。
    - XMLHttpRequestオブジェクトはこの仕様に基づいており、同オブジェクトが持つメソッドやプロパティを利用することで、非同期のHTTP通信を行うことができます。
  - Webブラウザによって多少の差異はありますが、 XMLHttpRequestオブジェクトは組み込みオブジェクトとして利用可能です。

XMLHttpRequestのコード例（古いやり方）

```
<html>
    <head></head>
    <body>
        <script>
            // XMLHttpRequestのコード例（古いやり方）

            // 非同期通信でリクエストを送信する準備
            let url = "https://zipcloud.ibsnet.co.jp/api/search?zipcode=9000015";
            let xhr = new XMLHttpRequest();
            xhr.open("GET", url);

            // 非同期通信の状態が変化したときのイベント処理
            xhr.onreadystatechange = function() {
                // 操作完了状態であり、かつHTTPレスポンスステータスが200である場合
                if(xhr.readyState == 4 && xhr.status === 200) {
                    // JSON形式文字列からオブジェクトへの変換
                    let parsedResult = JSON.parse(xhr.responseText);

                    // 変換後のオブジェクトを読み取って表示
                    alert(parsedResult.results[0]["address1"] + parsedResult.results[0]["address2"] + parsedResult.results[0]["address3"]);
                }
            };

            // リクエスト送信
            xhr.send();
        </script>
    </body>
</html>
```

動的にページの一部を書き換えるAjaxの例ではなく、
XMLHttpRequestオブジェクトを用いて通信するだけの例です

このあたりの書き方が古いです
- ✓ 通信結果を利用できるようになるタイミングを端的に表現できない
- ✓ JSON形式文字列からオブジェクトへの変換を明示的に行っている

XMLHttpRequestのコード例（新しいやり方）

```
<html>
    <head></head>
    <body>
        <script>
            // XMLHttpRequestのコード例（新しいやり方）

            // 非同期通信でリクエストを送信する準備
            let url = "https://zipcloud.ibsnet.co.jp/api/search?zipcode=9000015";
            let xhr = new XMLHttpRequest();
            xhr.open("GET", url);

            // JSONデータを受け取る準備
            xhr.responseType = "json";

            // レスポンスのロード完了時のイベント処理
            xhr.onload = function() {
                // HTTPレスポンスステータスが200である場合
                if(xhr.status === 200) {
                    // レスポンス内容を元に自動生成されたオブジェクトを読み取って表示
                    alert(xhr.response.results[0]["address1"] + xhr.response.results[0]["address2"] + xhr.response.results[0]["address3"]);
                }
            };

            // リクエスト送信
            xhr.send();
        </script>
    </body>
</html>
```

このあたりの書き方が変わりました
- ✓ 通信結果を利用できるようになるタイミングが端的に表現できる
- ✓ JSON形式文字列からオブジェクトへの変換が自動的に行われる

２つのコード例の比較

旧

```
<html>
    <head></head>
    <body>
        <script>
            // XMLHttpRequestのコード例（古いやり方）

            // 非同期通信でリクエストを送信する準備
            let url = "https://zipcloud.ibsnet.co.jp/api/search?zip（略）
            let xhr = new XMLHttpRequest();
            xhr.open("GET", url);

            // 非同期通信の状態が変化したときのイベント処理
            xhr.onreadystatechange = function() {
                // 操作完了状態であり、かつHTTPレスポンスステータスが200（略）
                if(xhr.readyState == 4 && xhr.status === 200) {
                    // JSON形式文字列からオブジェクトへの変換
                    let parsedResult = JSON.parse(xhr.responseText);

                    // 変換後のオブジェクトを読み取って表示
                    alert(parsedResult.results[0]["address1"] + （略）
                }
            };

            // リクエスト送信
            xhr.send();
```

- ✓ 通信結果を利用できるようになるタイミングを端的に表現できない
- ✓ JSON形式文字列からオブジェクトへの変換を明示的に行っている

新

```
<html>
    <head></head>
    <body>
        <script>
            // XMLHttpRequestのコード例（新しいやり方）

            // 非同期通信でリクエストを送信する準備
            let url = "https://zipcloud.ibsnet.co.jp/api/search?zip（略）
            let xhr = new XMLHttpRequest();
            xhr.open("GET", url);

            // JSONデータを受け取る準備
            xhr.responseType = "json";

            // レスポンスのロード完了時のイベント処理
            xhr.onload = function() {
                // HTTPレスポンスステータスが200である場合
                if(xhr.status === 200) {
                    // レスポンス内容を元に自動生成されたオブジェクトを（略）
                    alert(xhr.response.results[0]["address1"] + （略）
                }
            };

            // リクエスト送信
            xhr.send();
```

- ✓ 通信結果を利用できるようになるタイミングが端的に表現できる
- ✓ JSON形式文字列からオブジェクトへの変換が自動的に行われる

XMLHttpRequestを用いたAjaxのコード例

```
<html>
    <head>
        <script>
            // 郵便番号入力用テキストボックスのkeyupイベントハンドラ
            function search(e){
                // テキストボックスの入力文字数が7文字である場合
                if(e.value.length == 7){
                    let url = "https://zipcloud.ibsnet.co.jp/api/search?zipcode=" + e.value;
                    let xhr = new XMLHttpRequest();
                    xhr.open("GET", url);
                    xhr.responseType = "json";

                    xhr.onload = function() {
                        if(xhr.status === 200) {
                            document.querySelector("#address").value = xhr.response.results[0]["address1"] + xhr.response.results[0]["address2"]
                                                                     + xhr.response.results[0]["address3"];
                        }
                    };

                    xhr.send();
                } else {
                    document.querySelector("#address").value = "";
                }
            }
        </script>
    </head>
    <body>
        <input type="text" id="zipcode" onkeyup="search(this)">
        <input type="text" id="address" style="border: black solid 1px">
    </body>
</html>
```

通信結果を利用して画面内の要素を変更する

※スペースの関係上、コメントはほとんど省略しています。

3. Fetch API

Fetch APIとは

- 非同期のHTTP通信を行うための仕組みです。
  - XMLHttpRequestと同じことを実現できますが、Fetch APIはプロミス(*)ベースであるため比較的シンプルなソースコードを記述できます。

＊非同期処理に関して「この処理が正常に完了したときに実行したい処理」や「失敗したときに実行したい処理」といったコードを書くことができる仕組みです。

Fetch APIを用いたコード例

```
<html>
    <head>
        <script>
            // 郵便番号入力用テキストボックスのkeyupイベントハンドラ
            function search(e){
                // テキストボックスの入力文字数が7文字である場合
                if(e.value.length == 7){
                    // 非同期通信でリクエストを送信する準備
                    let url = "https://zipcloud.ibsnet.co.jp/api/search?zipcode=" + e.value;

                    // リクエスト送信と、通信成功時の処理の定義
                    fetch(url)
                    .then(response => response.json())
                    .then(jsonObject => document.querySelector("#address").value = jsonObject.results[0]["address1"]
                                                                                 + jsonObject.results[0]["address2"]
                                                                                 + jsonObject.results[0]["address3"]);

                } else {
                    document.querySelector("#address").value = "";
                }
            }
        </script>
    </head>
    <body>
        <input type="text" id="zipcode" onkeyup="search(this)">
        <input type="text" id="address" style="border: black solid 1px">
    </body>
</html>
```

Fetch APIはプロミスを使用しているため、イベント名や通信完了を判定する条件式を**意識せず**にコードを書ける。

- ✓ 1つめのthenは非同期通信が正常終了したときの処理
- ✓ 2つめのthenはJSON文字列からの変換が正常終了したときの処理

ところで、本当に"非同期"に実行しているの？

```
<html>
    <head>
        <script>
            // 郵便番号入力用テキストボックスのkeyupイベントハンドラ
            function search(e){
                // テキストボックスの入力文字数が7文字である場合
                if(e.value.length == 7){
                    // 非同期通信でリクエストを送信する準備
                    let url = "https://zipcloud.ibsnet.co.jp/api/search?zipcode=" + e.value;

                    // リクエスト送信と、通信成功時の処理の定義
                    fetch(url)
                    .then(response => response.json())
                    .then(jsonObject => document.querySelector("#address").value = jsonObject.results[0]["address1"]
                                                                                  + jsonObject.results[0]["address2"]
                                                                                  + jsonObject.results[0]["address3"]);

                    // 非同期通信の開始から終了までの間は「処理中」と表示させておく
                    document.querySelector("#address").value = "処理中";
                } else {
                    document.querySelector("#address").value = "";
                }
            }
        </script>
    </head>
    <body>
        <input type="text" id="zipcode" onkeyup="search(this)">
        <input type="text" id="address" style="border: black solid 1px">
    </body>
</html>
```

Fetch APIの一連の処理が完了する前にこの処理がきちんと実行されます

参考サイト

- **XMLHttpRequestの仕様は、下記のサイトに掲載されています。**
  - https://developer.mozilla.org/ja/docs/Web/API/XMLHttpRequest
    - 日本語なので分かりやすいです。
  - https://xhr.spec.whatwg.org/#interface-xmlhttprequest
    - 仕様が掲載されている本家です。

- **Fetch APIの仕様は、同様に下記の各サイトに掲載されています。**
  - https://developer.mozilla.org/ja/docs/Web/API/Fetch_API
  - https://fetch.spec.whatwg.org/#fetch-method

4. おわりに

まとめ

- **本日のポイントは下記のとおりです。**
  - **Ajax関連**
    - クライアント側（Webブラウザ）においてJavaScriptを用いて非同期通信を発生させ、サーバーからの応答が返ってくるまでの間にユーザーによるWebブラウザの操作を受け付けることができる。
    - サーバーからの応答として、XML形式やJSON形式（の文字列）が返ってくる。
    - JSONの場合は、下記のメソッドでオブジェクトと文字列を相互に変換できる。
      - 文字列用変数 = JSON.stringify(オブジェクト);
      - オブジェクト用変数 = JSON.parse(文字列);
  - XMLHttpRequestとFetch APIのどちらを利用しても非同期通信を実現できる。

ご清聴ありがとうございました。

以降、質問タイムです。