API

アプリケーション・プログラミング・インターフェース（API: Application Programming Interface）とは、あるソフトウェア・アプリケーションをプログラムで利用するための命令およびそれに関する決め事の集まりです。API を使用することで、API が提供するサービスを利用した製品の設計ができます。HTML5 には新しい API がいくつも追加されています。例えば、以下の API です。

- グラフや視覚的な表現を行う画像をレンダリングするために新規 canvas 要素で使用する 2 次元描画用 API
- オフライン Web アプリケーションをサポートするキャッシング・メカニズム用 API
- 新しい video 要素と audio 要素で使われる動画や音声を再生するための API
- 閲覧履歴にアクセスできるようにして、そこにページを追加できるようにするための履歴用 API
- `draggable` 属性とともに使用するドラッグ・アンド・ドロップ API
- `contenteditable` 属性とともに使用する編集用 API
- キーと値のペアおよび組み込み SQL データベースを扱うための、クライアント・サイドのストレージの JavaScript API

ここでは新しく追加された API のうち、上記には挙げていない Geolocation API に焦点を絞ります。API の Geolocation（地図表示と GPS）について解説した後、API を組み込んだページを作成します。

Geolocation API が機能する仕組み

Geolocation API のベースは、グローバル `navigator` オブジェクトに新しく追加されたプロパティーである `navigator.geolocation` です。JavaScript の `navigator` オブジェクトは、訪問者のブラウザーとシステムに関する有用な情報を提供します。Geolocation API は、IP アドレス、Web ベースのデータベース、ワイヤレス・ネットワーク接続、そして三角測量または GPS 技術を使用して、緯度と経度を判断することができます。Geolocation API によって提供される情報の正確さは、その情報を取得する手段によって異なることに注意してください。場合によっては、明確なジオロケーション測定値を取得できない位置、あるいはデータをまったく取得できない位置もあります。
スクリプトでは、`navigator.geolocation` オブジェクトを利用してユーザーのホスト機器に関する位置情報を取得することができます。位置情報が取得されると Position オブジェクトが作成され、このオブジェクトにデータが取り込まれます。
`navigator.geolocation` オブジェクトには、以下の 3 つのメソッドがあります。

- `getCurrentPosition()`
- `watchPosition()`
- `clearWatch()`

getCurrentPosition() メソッド

getCurrentPosition() は、ユーザーの現在位置を取得するメソッドです。
（ただし 1 度しか試行されません）
スクリプトによって呼び出されると、このメソッドは非同期でホスト機器の現在位置を取得しようとします。
非同期通信とは、送信側と受信側が同時に通信に携わらないことを意味します。
したがって、非同期通信を使用すればブラウザーが他のアクティビティーを続行できることから、
受信側エンティティーからのレスポンスを待機する必要がなくなります。
getCurrentPosition() メソッドには、以下の 3 つの引数を指定することができます。

- **geolocationSuccess**: 現在位置が返されるコールバック（必須）。
- **geolocationError**: エラーが発生した場合のコールバック（オプション）。
- **geolocationOptions**: ジオロケーションのオプション（オプション）。

navigator.geolocation.getCurrentPositon() メソッドは、Position オブジェクトを
パラメーターとして、ホスト機器の現在位置を geolocationSuccess コールバックに返します。
エラーが発生した場合は、PositionError オブジェクトを使用して geolocationError
コールバックが呼び出されます。geolocationOptions に設定できるプロパティーには、
enableHighAccuracy、timeout、および maximumAge の 3 つがあります。
これらのオプション・プロパティーによって、高い精度を指定することや（機器がサポートしている場合）、
現在位置が返されるまでの時間制限を指定すること、そしてキャッシュされた位置情報の最大有効時間を
指定することができます。

getCurrentPosition() メソッドは、以下のようにして呼び出されます。

```
void navigator.geolocation.getCurrentPosition(
        geolocationSuccess, geolocationError, geolocationOptions);
```

演習２：API の実装：Google Map API を使う例

前ページで学習した内容を実際にWEBページに実装し、住所検索によるMoveMapの機能を使えるようにしてみましょう。

Google MapのAPIは、下記アドレスから取得します。

http://maps.google.com/maps/api/js?sensor=true&language=ja

実装時の記述は、Scriptとして head 内に記述し、WEB上からリンクさせます。

```
<script src="http://maps.google.com/maps/api/js?sensor=true&language=ja"></script>
```

最初のロケーションはとりあえず渋谷にしておきます。

●もしも初期設定の値になる緯度経度を知りたい場合は、Google Mapで検索表示し、
その位置の選択地区の住所を右クリックして、『この場所について』を選択すると表示されます。

```
<div id="page">

<h1>- Google MAP API WEBページに実装する -</h1>
<!—住所やキーワードの検索ボックスの記述 ：テーブルの中に、textボックスを設置する。-->
        <table  width="95%" cellspacing="8">
                <tr valign="middle">
                <td width="30%"><h1>HTML5 Google map：API</h1></td>
                <td width="70%" align="right">住所検索：
                        <input id="address" type="text" size="45" value="" />
                        <input type="button" value="Search" onClick="moveMap();" />
                </td></tr>
        </table>

<!—Mapのサイズや、背景などのスタイルシート -->
        <style>
                #map {
                        width:100%;    height:80%;    border:thick solid #CCCCCC;
                         }
         </style>
```

```
<!—Map表示部分のエリア決めとgeolocationの記述 -->

<div id="map"></div>
    <script>
            var latlng = new google.maps.LatLng(35.66, 139.69);
            /*経度・緯度を使った渋谷区の表示（初期画面）
            新横浜(35.507865,139.617514)などに変更してみましょう。
            任意の緯度および経度は、GoogleMapの『この場所について』で
            チェックできます。*/
            var opts = {
                    zoom: 15,       center: latlng,
                    mapTypeId: google.maps.MapTypeId.ROADMAP
                    }
            var map = new google.maps.Map(document.getElementById('map'), opts);

            function moveMap() {
            var geocoder = new google.maps.Geocoder();
            geocoder.geocode({
                    'address' : document. getElementById('address').value
        }, function(result, status) {
                    if (status==google.maps.GeocoderStatus.OK) {
                    map.panTo(result[0].geometry.location);
                                var marker = new google.maps.Marker({
                                        position: result[0].geometry.location,
                                        map: map
                                });

        } else {
                    alert("エラーです。");
                    }
         });
            }

    </script>
    </div>
```

最後に、前記の地図の初期画面に、マーカーと吹き出しを入れてみましょう。

まず、渋谷の地図を単なる渋谷周辺から、渋谷駅を指している経度と緯度に変更します。

```
var latlng = new google.maps.LatLng(35.659327,139.700962);//JR 渋谷駅
```

次にマーカーと吹き出しの設定をする JS を記述します。

```
var obj={
        position:new google.maps.LatLng(35.659327,139.700962), map:map
        };    //マーカーの位置(検索地点と同じ経度と緯度)

        var marker=new google.maps.Marker(obj);    //マーカーの表示

        var info=new google.maps.InfoWindow({content:'
        <img src="img/XXXX.jpeg" width="180" alt="渋谷まちなみ">
        <p>渋谷駅周辺</p>'});    //吹き出しの設定と内容(画像を挿入した場合)

        info.open(map,marker); //マーカークリックで吹き出しを表示

        google.maps.event.addListener(marker,'click',function(){
        info.open(map,marker);
        });  //クリックして、吹き出しを表示させる関数
```

上記で Map の API 実装は完了です。