地図を表示する

ブラウザに GoogleMap を表示するには、HTML、CSS、そして JavaScript が記述されている必要がある。HTML は Web ページの枠組みを作るもの、CSS は Web ページの装飾をするもの、そして JavaScript は Web ページに動きを加えたり複雑な処理を行えるようにしたりするものである。今回の演習ではこれら 3 つがそれぞれ別のファイルに記述されているが、その内 HTML と CSS については時間の都合上説明を省き、JavaScript(拡張子が js のファイル)を中心として簡単に説明しながら書き換え、地図を表示させることを目指す。

本章では、指定した緯度経度を中心とする地図を表示させることが目標である。フォルダ「1-1」直下にある「main.js」というファイルには現状何も書かれていないはずであるので、まずはそのファイルに以下のように記述し、アップロードして地図を確認してみよう。

main.js

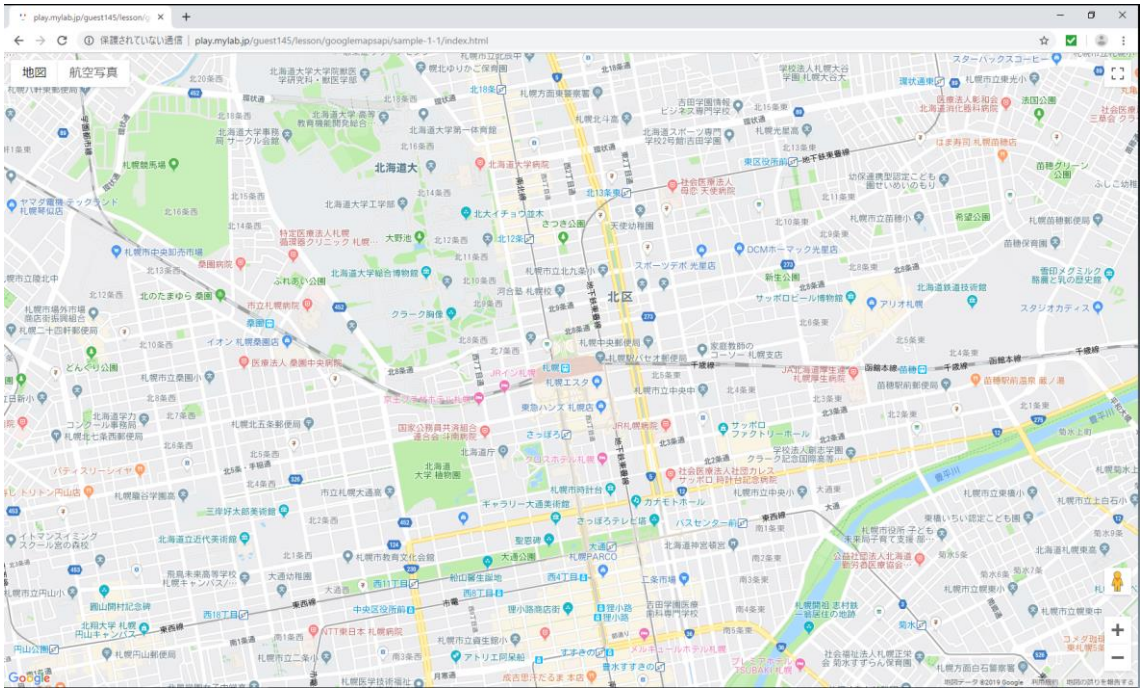
```
var map = null;

function initMap() {
    map = new google.maps.Map($('#map').get(0), {
        center: {lat: 43.068543, lng: 141.351128},
        zoom: 15
    });
}
```

上手くいけば、次ページのように JR 札幌駅を中心とした Google マップが表示されるはずである。このように、Google Maps API の google.maps.Map クラスを使うことで地図を表示させているが、特にここでは緯度経度とズームレベルの指定の仕方を見ていく。これらの指定は、オプションで行っている。

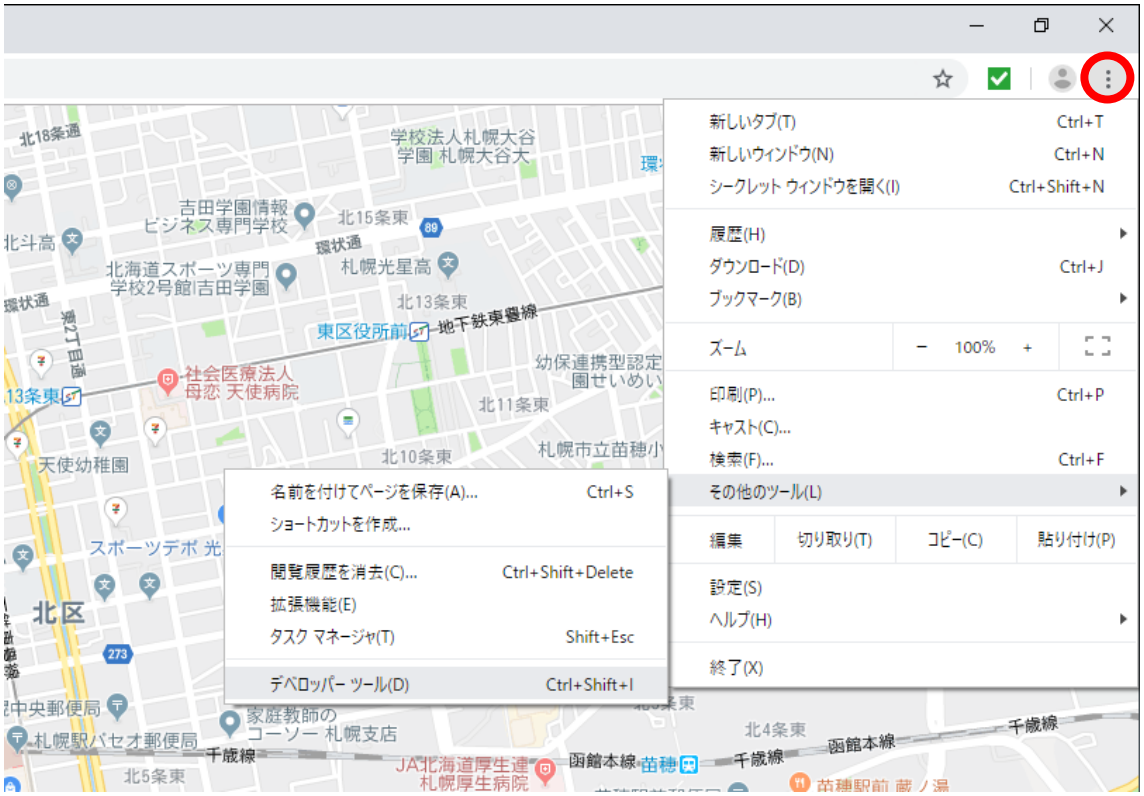
地図の中心の緯度経度は、プログラム 5 行目の「center」というオプションで設定しており、「lat」は緯度、「lng」は経度を意味する。単位は°(度)で、小数を使った表記(DEG 表記)を用いる。また、緯度が正の数の時は北緯、経度が正の数の時は東経を表す。すなわち、ここでは北緯 43.068543°、東経 141.351128°を中心にして Google マップを表示させている。

ズームレベルは、6 行目の「zoom」というオプションで設定しており、ここではズームレベル 15 を指定している。

以上のことを確認するには、緯度経度やズームレベル(0～21)を適当にいじってみるとよいだろう。



本章の課題が上手くいけば、上のような地図が表示されるはずである。もし上手くいかない場合は、Google Chrome のウィンドウ右上にある、縦に 3 つ並んだ点の部分(下図赤丸)をクリックし、「その他のツール」＞「デベロッパーツール」を開いて「Console」のタブをクリックすると、どの部分がエラーを起こしているか確認できる(大抵は単純な打ち間違いなので、よく確認してみよう)。



地図に図形を表示する

続いては、Google マップ上に図形を表示させる。そのためには、1-1 で学んだ Google マップを表示させるクラスに加えて、図形を表示させるクラスを用いる。今回は、Google マップ上に円を表示させるクラスを使って、指定した緯度経度に円を表示させてみる。フォルダ「1-2」直下の「main.js」には、以下のソースコードのうち黒字の部分(1-1 の「main.js」と同じ文)が記述されているので、その下の緑字の部分をその「main.js」に書き加えてアップロードし、地図の表示を確かめよう(今後、緑字の部分については「書き加え」を意味し、「+」より後の部分を書く。なお、「+」のみの行は空白行を加える)。

main.js

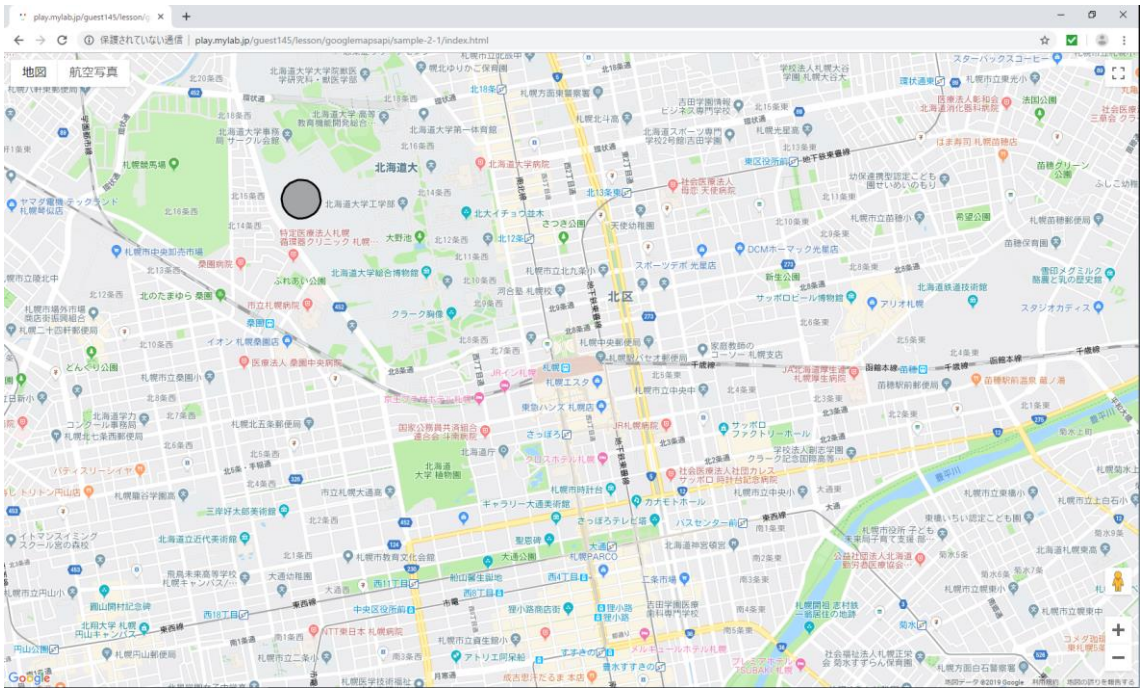
```
'strict'

var map = null;

function initMap() {
  map = new google.maps.Map($('#map').get(0), {
    center: {lat: 43.068543, lng: 141.351128},
    zoom: 15
    });
+
+  // circle
+  var circle_coordinate = {lat: 43.076490, lng: 141.333961};
+  var circle = new google.maps.Circle({
+    map: map,
+    center: circle_coordinate,
+    radius: 100
+  });
}
```

上手くいけば、地図上の北大敷地内に黒い円が描かれているはずである(次ページ図)。

地図上に円を描くには、google.maps.Circle クラスを用いる(プログラム 13 行目)。14 行目の「map」オプションで円を表示させる Google マップを指定し、15 行目の「center」オプションで円の中心を指定している。円の中心の緯度経度は、その前の 12 行目で「circle coordinate」という変数に格納しているので、15 行目では変数名「circle coordinate」で指定する。16 行目の「radius」オプションは円の半径を実際の距離で指定するもので、m 単位で指定できる。ここでは実際の大きさで半径 100m の円を描いていることになる。

時間があれば、円の場所や半径を変えてみよう。

なお、11 行目は円を描くことを明示しているコメントであり、プログラムの動作には関係ない(JavaScript では、1 行のコメントには先頭にスラッシュを 2 つつける)。


play.mylab.jp/guest145/lesson/g
保護されていない通信 | play.mylab.jp/guest145/lesson/googlemapsapi/sample-2-1/index.html
地図
航空写真
北海道大
北区
Google

地図に複数の図形を表示する

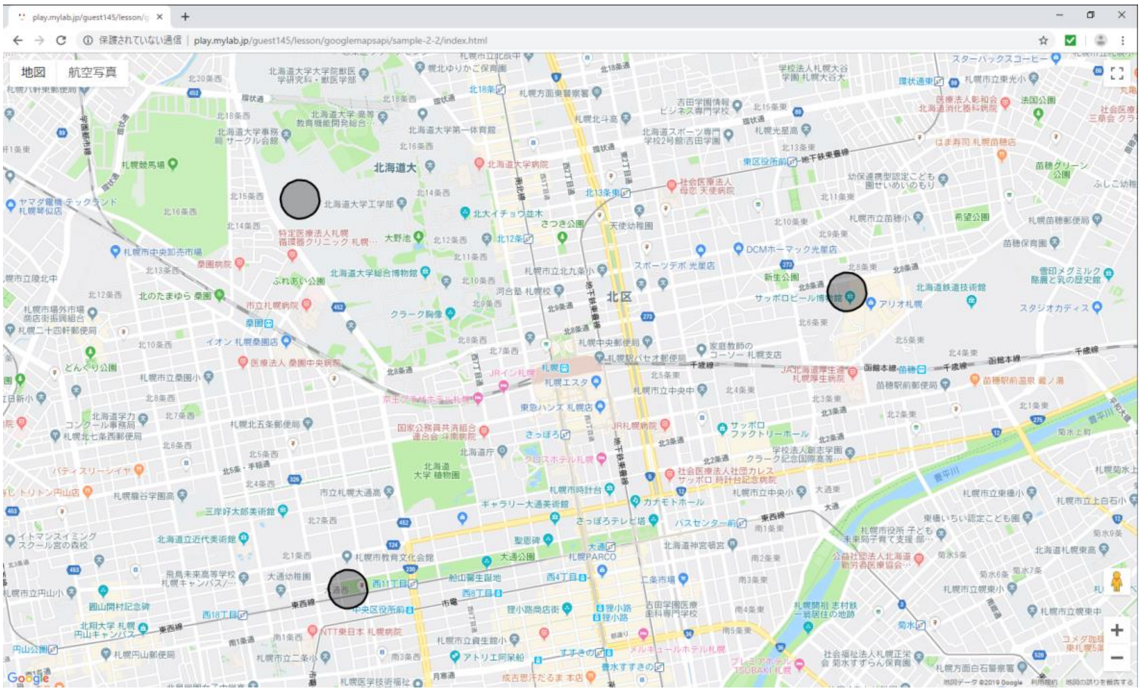
2-1 では 1 か所に円を描いたが、今度は複数個所に円を描いていく。フォルダ「2-2」直下の「main.js」について、以下の緑字の部分は 2-1 同様書き加え、赤字の部分は削除し(今後も同様、当然行頭の「-」は記述しない)、JavaScript を完成させる。

main.js

```
'strict'

var map = null;

function initMap() {
  map = new google.maps.Map($('#map').get(0), {
    center: {lat: 43.068543, lng: 141.351128},
    zoom: 15
    });

  // circle
-  var circle_coordinate = {lat: 43.076490, lng: 141.333961};
-  var circle = new google.maps.Circle({
-    map: map,
-    center: circle_coordinate,
-    radius: 100
+  var circle_coordinates = [
+    {lat: 43.076490, lng: 141.333961},
+    {lat: 43.072195, lng: 141.368723},
+    {lat: 43.058400, lng: 141.337008},
+  ];
+
+  circle_coordinates.forEach(function(coordinate){
+    var circle = new google.maps.Circle({
+      map: map,
+      center: coordinate,
+      radius: 100
+    });
  });
}
```

成功すると、3 か所に円が描かれた Google マップが表示されるはずである(次ページ図)。今回は指定する円の中心を 3 か所に増やし、それらの緯度経度を「circle coordinates」という配列に格納している(プログラム 12～16 行目)。

18 行目では、Google マップ上に円を描く動作を、配列「circle coordinates」の各緯度経度に対して行うように指示している。円を描く動作そのものは、2-1 と同様である。余裕があれば、円を描く緯度経度を変えたり円を増やしたりしてみてもよいだろう。



play.mylab.jp/guest145/lesson/
保護されていない通信
play.mylab.jp/guest145/lesson/googlemapsapi/sample-2-2/index.html
地図
航空写真
北海道大
北区

外部から図形情報を読み込む

2-1・2-2 では円の位置を JavaScript に直接書き込んだが、その方法では円を描く位置が多くなった場合、JavaScript が大変長くなってしまう。そこで、位置情報を記述したファイルを別に用意して読み込ませるという方法をとる。今回は、「data.json」というファイルに円の中心の緯度経度を記述している(JSON は Web アプリケーションでのデータのやり取りによく用いられ、人間も機械も読み取りやすい形式である)。おおよそ 2-2 と同じような配列になっているので、一応中身を確認してみよう。それでは、下のソースコードに従ってフォルダ「3-1」直下の「main.js」を書き換えてみよう。

main.js

```
'strict'

var map = null;

function initMap() {
  map = new google.maps.Map($('#map').get(0), {
    center: {lat: 43.068543, lng: 141.351128},
    zoom: 15
    });

-  // circle
-  var circle_coordinates = [
-    {lat: 43.076490, lng: 141.333961},
-    {lat: 43.072195, lng: 141.368723},
-    {lat: 43.058400, lng: 141.337008},
-  ];
+  $.ajax({
+    url: 'data.json',
+    type: 'GET',
+    dataType: 'json'
+  })
+  .done(function(data){
+    console.log(data);
+    render(data);
+  })
+  .fail(function(){
+    console.log('error');
+  });
+}

-  circle_coordinates.forEach(function(coordinate){
+function render(data){
+  data.circle_coordinates.forEach(function(coordinate){
    var circle = new google.maps.Circle({
      map: map,
      center: coordinate,
      radius: 100
    });
  });
}
```

ここでは、「Ajax」という処理方法を用いて、ページそのものを更新することなく、緯度経度のデータを別ファイルから取得している(プログラム 11～23 行目)。「url」にファイル名、「datatype」にファイル形式(この場合は JSON)を記述している。
上手くいくと、2-2 と同様、3 か所に円が描かれているはずである。時間があれば、「data.json」の緯度経度の値を変えて同じ場所にアップロードし、円の位置が変わるかどうか見てみよう。

MySQL から取得した緯度経度に円を描く

ここからは、MySQL を用いてデータベースにある位置情報を地図上にプロットしていく。ここでも 3-1 でやったように、Ajax を用いて外部からデータを取得することになるが、その際に同じフォルダの JSON ファイルではなく、PHP を通じてデータベースから SQL で取得したデータを用いる。
今回はファストフードチェーン・マクドナルドの店舗の位置情報を取得し、プロットしていく。ここではこれまで同様に 1 つ目のソースコードに従って「main.js」を書き換えると共に、「api.php」というまっさらなファイルがあるので、2 つ目のソースコードを「api.php」に書き込んでいこう。

main.js

```
'strict'

var map = null;

function initMap() {
  map = new google.maps.Map($('#map').get(0), {
    center: {lat: 43.068543, lng: 141.351128},
    zoom: 15
    });

  $.ajax({
-    url: 'data.json',
+    url: 'api.php',
    type: 'GET',
    dataType: 'json'
  })
  .done(function(data){
    console.log(data);
    render(data);
  })
  .fail(function(){
    console.log('error');
  });
}

function render(data){
-  data.circle_coordinates.forEach(function(coordinate){
+  data.forEach(function(geom){
    var circle = new google.maps.Circle({
      map: map,
-      center: coordinate,
+      center: {lat: geom.coordinates[1], lng: geom.coordinates[0]},
      radius: 100
    });
  });
}
```

api.php

```php
<?php

$db = mysqli_connect('localhost', 'guest', 'Test@1234', 'opendata');

if(!$db){
	return;
}

$sql = <<<EOS
SELECT ST_AsGeoJson((geom)) AS geojson
FROM mcdnald
EOS;

$stmt = $db->prepare($sql);

$stmt->execute();
$stmt->bind_result($geojson);

$json = [];
while(($row = $stmt->fetch()) != NULL){
	$json[] = json_decode($geojson);
}

echo json_encode($json);

exit;
```

上手くいけば、**図 1** のように札幌市中心部のマクドナルドの店舗位置がプロットされるはずである。試しに地図を動かせば他の場所にあるマクドナルドもプロットされているし、ズームを引くと日本全国のマクドナルドの店舗位置がプロットされているのが分かるだろう。デベロッパーツールの「Network」タブを開きながら Google マップを開き、「api.php」をクリックすると、全国約 3000 店舗のマクドナルドの位置情報を読み込んでいることが分かる(**図 2**)。

今回用いた PHP は、JavaScript とは異なり、サーバーサイドで実行される言語である。PHP については本筋から外れるので深くは触れないが、9～12 行目で MySQL を用い、データベースの「mcdnald」というテーブルからマクドナルドの位置情報を GeoJSON という形式で取得している(GeoJSON は、JSON を基にして地理情報を扱えるようにした形式である)。これらの作業をサーバー側で行うことにより、クライアント側ではあたかも PHP がマクドナルドの位置情報を記述した GeoJSON ファイルであるかのように見える。一方、JavaScript 側では PHP で取得したマクドナルド店舗の位置情報をプロットする作業を行っている。先ほどの Ajax では「url」が「data.json」であったが、今度は「api.php」になっている(プログラム 12 行目)。

なお、GeoJSON には「経度、緯度」の順番(x,y の順番)で格納されているので、円の中心を「center」オプションで指定する際は「緯度は配列の 2 要素目、経度は配列の 1 要素目」というように指定しなければならないが、JavaScript では配列の要素番号が[0]から始まるため、プログラム 29 行目のように記述する必要があるという点に注意が必要である。

ただし、今回のソースコードでは先ほど述べたように全国約 3000 店舗分のマクドナルドの店舗情

報を読み込んでいるので、データ量も大きくなり、表示にも時間がかかってしまう。次の 2-1 では、その問題を解決する。

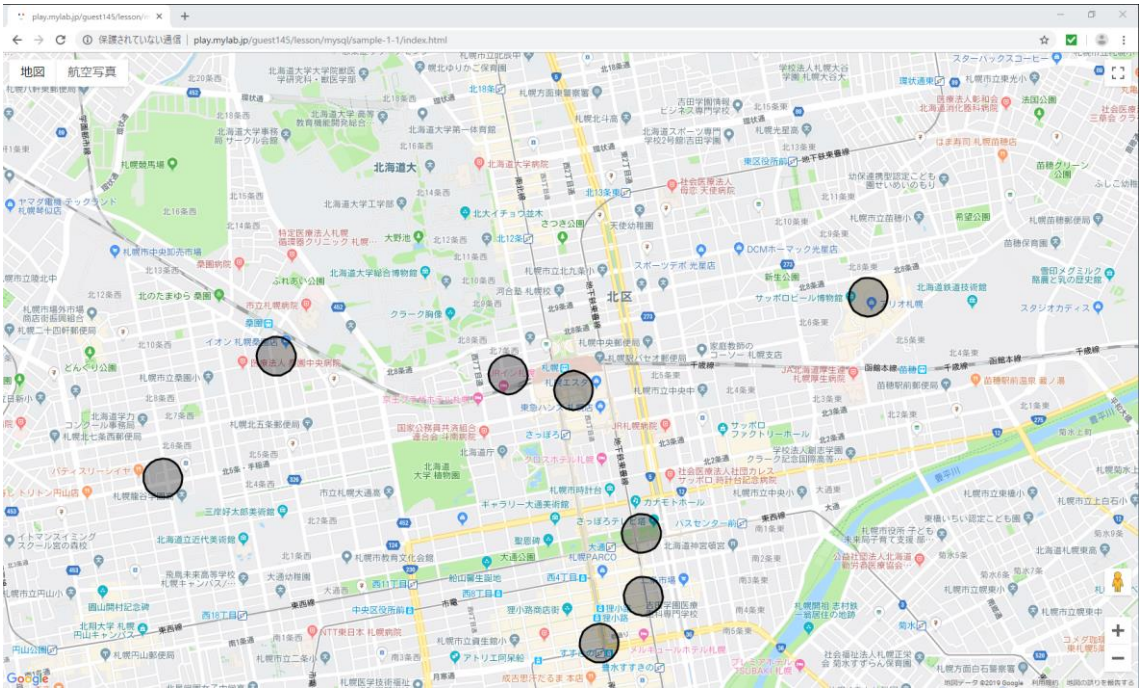


図 1. 札幌市中心部のマクドナルドの店舗位置をプロットした図。

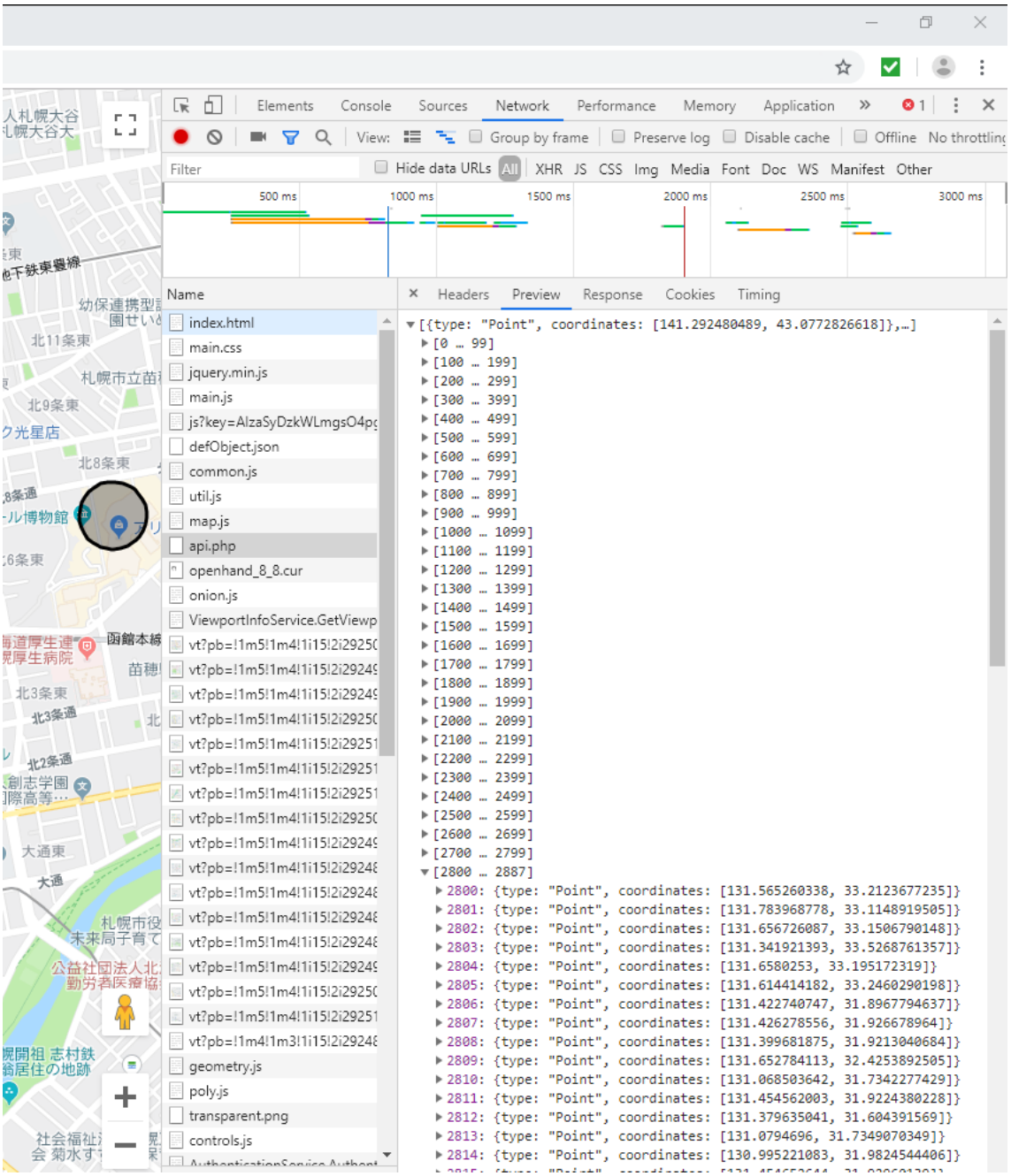


図 2.「api.php」の中身。全国 2887 店舗分の位置情報が読み込まれている。

表示範囲の経緯度のみを MySQL から取得

1-1 ではデータベースにあるマクドナルドの位置情報をプロットしたが、その際に取得したのは全国約 3000 店舗分のマクドナルドの位置情報である。したがってデータサイズが大きくなり、その分表示にも時間がかかってしまう。そこで、地図の表示範囲にある店舗の位置情報のみを取得してデータサイズを抑え、地図の表示時間を短縮する。

main.js

```
'strict'

var map = null;

function initMap() {
  map = new google.maps.Map($('#map').get(0), {
    center: {lat: 43.068543, lng: 141.351128},
    zoom: 15
    });

+  google.maps.event.addListener(map, 'idle', idleMaps);
+}
+
+function idleMaps()
+{
+  var zoom = map.getZoom();
+  var bounds = map.getBounds();
+  var ne = bounds.getNorthEast();
+  var sw = bounds.getSouthWest();
+   var params = 'zoom=' + zoom + '&n=' + ne.lat() + '&s=' + sw.lat() +
'&e=' + ne.lng() + '&w=' + sw.lng();
+
  $.ajax({
-    url: 'api.php',
+    url: 'api.php?' + params,
    type: 'GET',
    dataType: 'json'
  })
  .done(function(data){
    console.log(data);
    render(data);
  })
  .fail(function(){
    console.log('error');
  });
}

function render(data){
  data.forEach(function(geom){
    var circle = new google.maps.Circle({
      map: map,
      center: {lat: geom.coordinates[1], lng: geom.coordinates[0]},
      radius: 100
    });
```

```
  });
}
```

api.php

```
<?php

$db = mysqli_connect('localhost', 'guest' , 'Test@1234', 'opendata');

if(!$db){
        return;
}

+$north = $_GET['n'];
+$west = $_GET['w'];
+$east = $_GET['e'];
+$south = $_GET['s'];
+$zoom = $_GET['zoom'];
+
+$geom = 'POLYGON((';
+$geom .= $west . ' ' . $north . ',';
+$geom .= $west . ' ' . $south . ',';
+$geom .= $east . ' ' . $south . ',';
+$geom .= $east . ' ' . $north . ',';
+$geom .= $west . ' ' . $north;
+$geom .= '))';
+
$sql = <<<EOS
SELECT ST_AsGeoJson((geom)) AS geojson
FROM mcdnald
+WHERE  ST_Intersects(geom,  ST_GeomFromText(?,  4326,  'axis-order=long-
lat')) = True
EOS;

$stmt = $db->prepare($sql);
+$stmt->bind_param('s', $geom);
$stmt->execute();
$stmt->bind_result($geojson);

$json = [];
while(($row = $stmt->fetch()) != NULL){
        $json[] = json_decode($geojson);
}

echo json_encode($json);
```

地図の表示範囲にある店舗の位置情報のみを取得するには、おおよそ次のような流れで行う。

① JavaScript で表示領域の東西南北の緯度経度を取得し、パラメータとして PHP に渡す
② PHP で、渡された東西南北の緯度経度を用いて四角形のポリゴン(面)データを作成する
③ 生成された四角形に含まれる位置情報データのみを取得するよう SQL に条件追加(PHP)

まず①について、表示領域の東西南北の緯度経度を取得するには表示領域の四隅の座標が分かればよい。さらに言うと、対角に位置する 2 点の座標さえ分かれば四隅の座標も分かるので、東

西南北の座標を取得できる(図 1)。ここでは地図の表示領域を getBounds()で取得した後、北東(右上)の座標を getNorthEast()で、南西(左下)の座標を getSouthWest()で取得し、そこから北西(左上)と南東(右下)の座標を算出する、という処理を「main.js」で行っている(プログラム 17～19 行目)。こうして得られた表示領域の座標から東西南北の緯度経度をパラメータとして構築し、PHP に渡す。

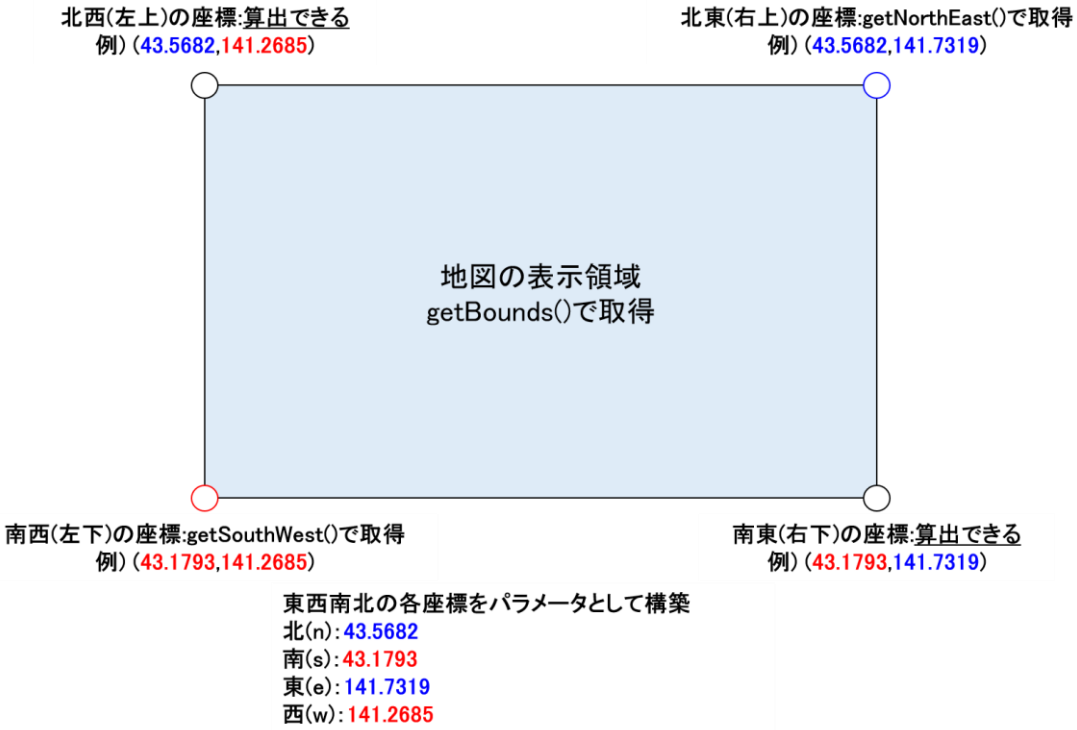


図 1. 地図の表示領域における東西南北の緯度経度を取得する方法のイメージ。

東西南北の緯度経度を渡された「api.php」では、②と③の処理を行うことになる。②については、「api.php」の 9～21 行目において、渡された東西南北の緯度経度を用いて 4 つの点を作り、四角形のポリゴンデータを生成する。ここでは、北西(左上)の点から反時計回りに 4 点をつなぎ、最後に北西の点に戻るという形で、四角形のポリゴンを作っている(図 2)。

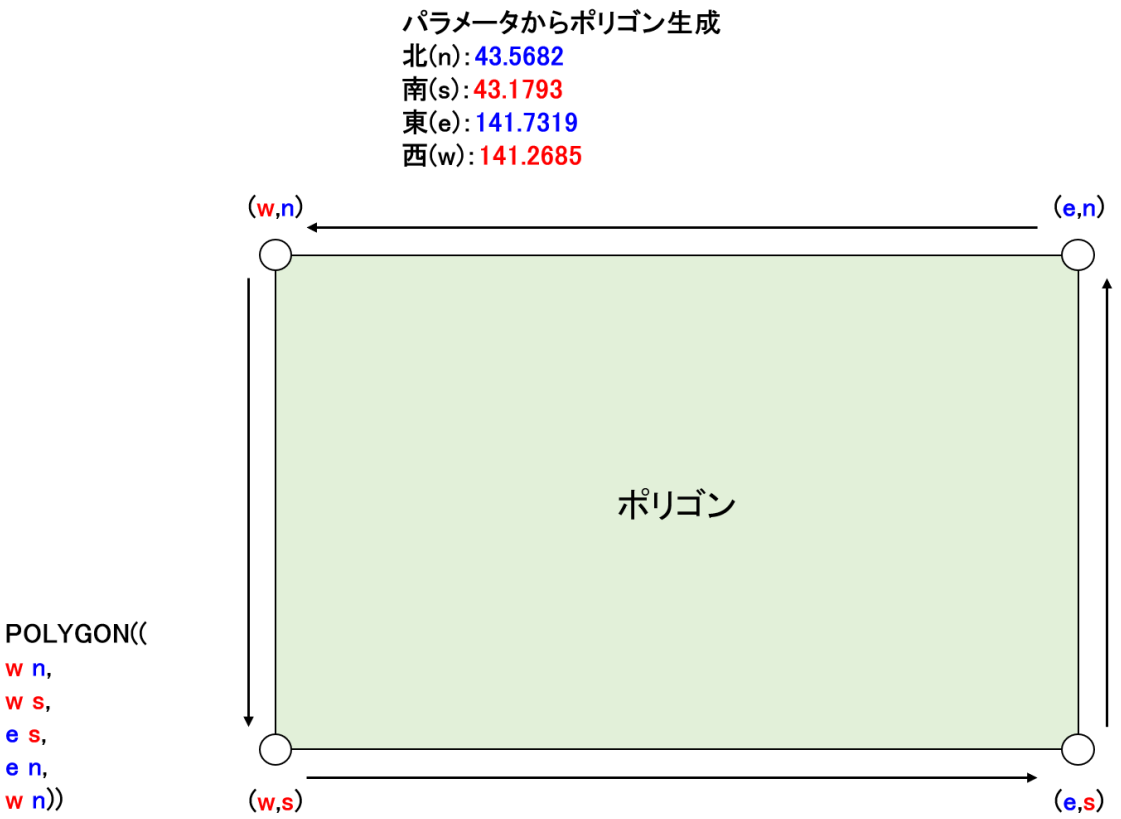
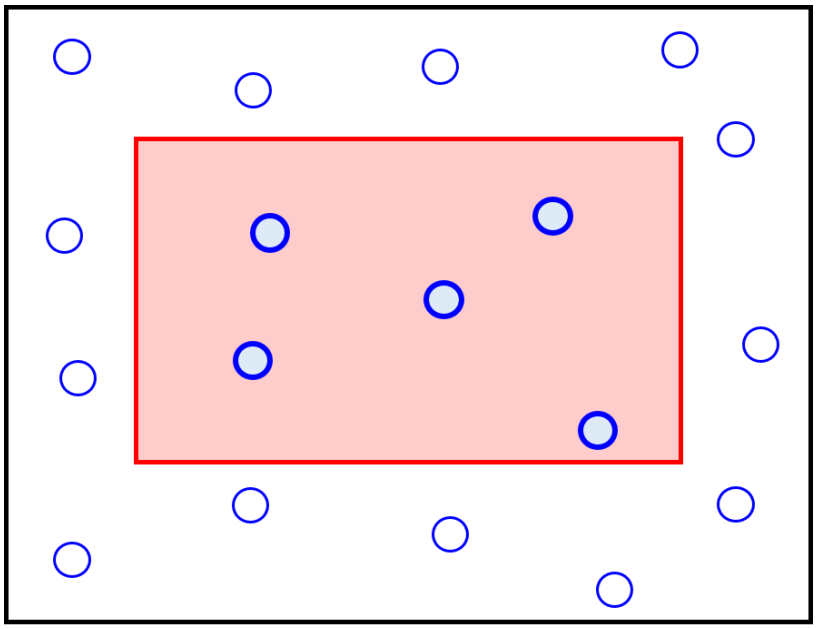


図 2. 東西南北の座標から四角形のポリゴンを作成するイメージ。

そして③では、生成した四角形に含まれるような店舗の位置情報を取得するよう SQL に条件を追加する。店舗が四角形の中に含まれるかを判断するためには、26 行目のように ST_Intersects を用いて記述する(**図 3**)。ST_Intersects では 2 つの図形に共通の部分があれば「True」を返すので、「api.php」26 行目のように「『ST_Intersects = True』となるマクドナルド店舗(を表す円の中心)の位置情報を取得する」という条件を与えれば、地図の表示範囲内のマクドナルド店舗の位置情報のみを取得することができるのである。



boolean ST_Intersects(geometry geomA, geometry geomB);
赤い四角形と交わる(True)のは太い円。

図 3. 店舗が生成した四角形の中に含まれるかを判断するイメージ。

以上のようにして、Google マップの表示領域内にあるマクドナルド店舗の位置情報のみを取得することができたので、後は再び JavaScript で 1-1 と同様に Ajax を使い、取得した位置情報をプロットする。

成功すれば、最初は 1-1 と同じ場所にマクドナルドの店舗位置がプロットされるだろう。しかし、デベロッパーツールの Network から api.php をみると、含まれているデータの数は数個しかないはずである(図 4)。地図を動かしたりズームレベルを変えたりすれば、その度に上記の処理が行われ、取得したマクドナルドの店舗の位置情報をプロットしてくれる。

表示時間の問題についてはこれで解決したが、もう一つ問題がある。それは、地図を動かしたりした際に円が上書きされ、どんどん濃くなることである(図 5)。それについては、次の 2-2 で解決する。

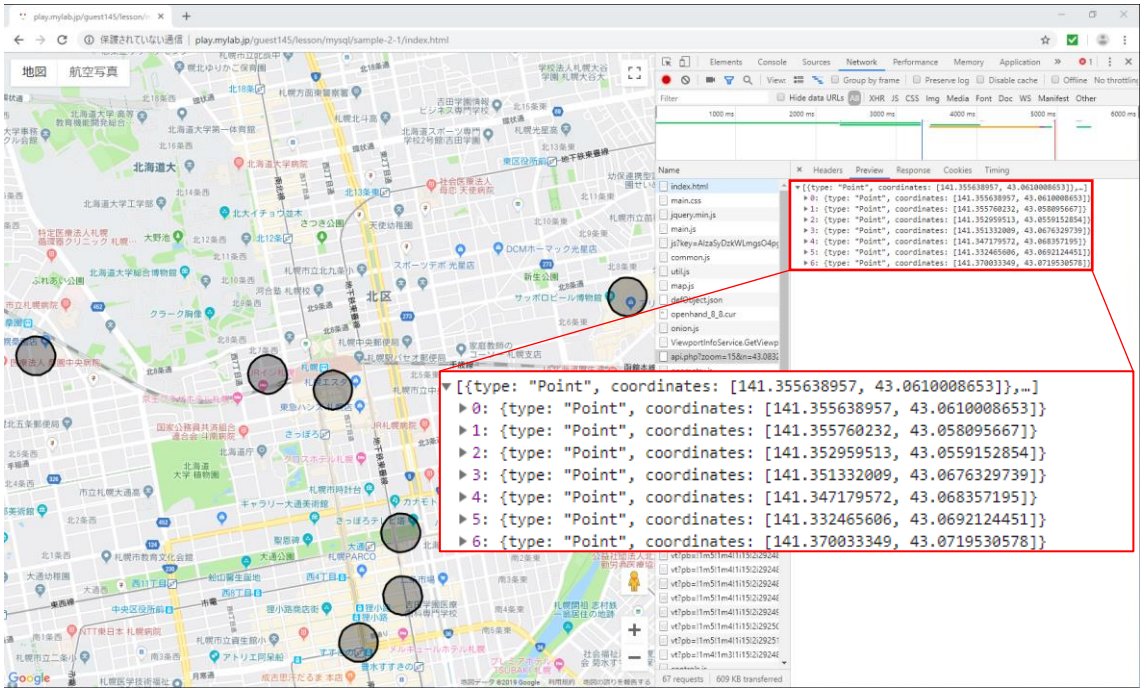


図 4. デベロッパーツールの「Network」タブから PHP を見たところ。地図に表示されている 7 店舗分の情報(赤枠部)のみ取得していることが分かる。

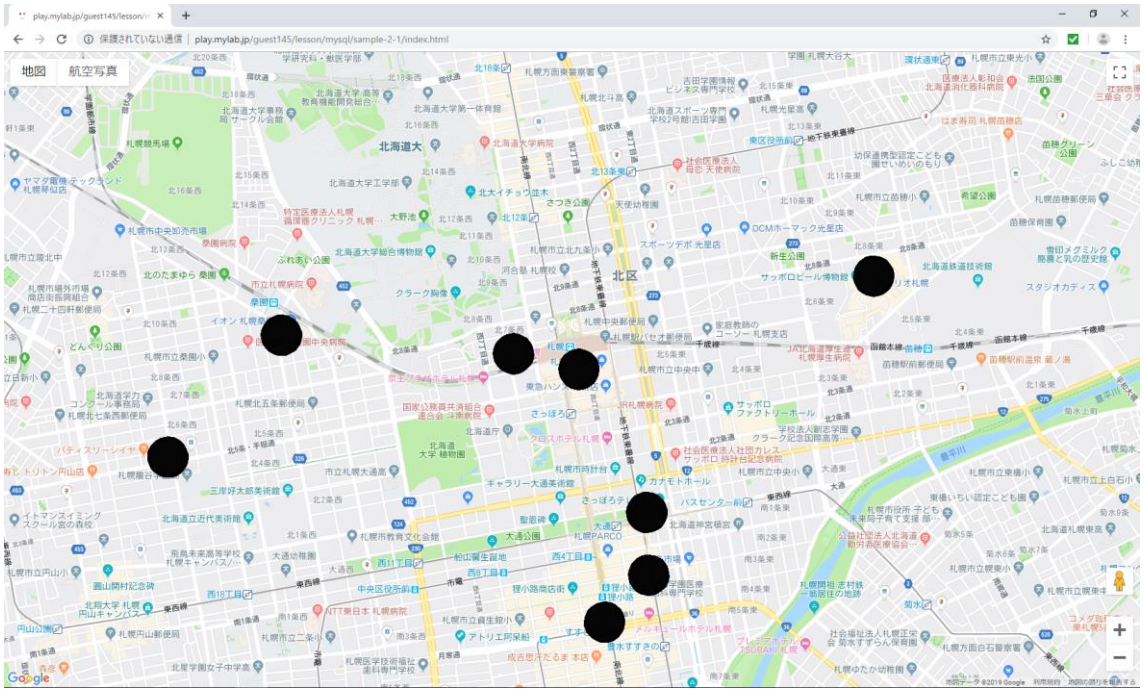


図 5. Google マップを数回動かしたときの図。円が何度も上書きされ、黒くなってしまっている。

円を上書きしないで再描画する

2-2 では、表示時間短縮やデータ量節減のために地図の表示領域内の店舗情報のみを取得して地図に表示させた。その結果、地図を動かすたびに処理が行われて円が上書きされるので、円がどんどん濃くなってしまうという問題が発生する。これを解決するために、円を描きなおす際に一度既に描いてある円を削除する処理を行う。今回は円を描く処理に関することなので、「main.js」のみを書き換える。

main.js

```
'strict'

var map = null;
+var objects = [];

function initMap() {
  map = new google.maps.Map($('#map').get(0), {
    center: {lat: 43.068543, lng: 141.351128},
    zoom: 15
    });

  google.maps.event.addListener(map, 'idle', idleMaps);
}

function idleMaps()
{
  var zoom = map.getZoom();
  var bounds = map.getBounds();
  var ne = bounds.getNorthEast();
  var sw = bounds.getSouthWest();
  var params = 'zoom=' + zoom + '&n=' + ne.lat() + '&s=' + sw.lat() + '&e='
+ ne.lng() + '&w=' + sw.lng();

  $.ajax({
    url: 'api.php?' + params,
    type: 'GET',
    dataType: 'json'
  })
  .done(function(data){
    console.log(data);
    render(data);
  })
  .fail(function(){
    console.log('error');
  });
}

function render(data){
+  objects.forEach(function(o){
+    o.setMap(null);
+  });
+
  data.forEach(function(geom){
```

```
    var circle = new google.maps.Circle({
      map: map,
      center: {lat: geom.coordinates[1], lng: geom.coordinates[0]},
      radius: 100
    });
+
+    objects.push(circle);
  });
}
```

ここでは、プログラム 39 行目で既に描いてある円を一度消去し、49 行目で再び円を描いている。

上手くいけば、何度地図を動かしたりズームレベルを変えたりしても円が濃くならなくなる。

追加演習

これまでの演習では、Google マップ上に自分で緯度経度を決めて円を描く方法と、外部から取得した緯度経度に円を描く方法を学んだ。円の半径(radius)については実際の距離を m 単位で設定できる。したがって、「〇〇駅から半径 500m の範囲を図示する」といったことも可能である。
そこで、札幌駅から半径 3km 圏内にあるマクドナルド店舗を知るため、追加演習として

「Google マップ上にマクドナルド店舗の位置情報をプロットし、なおかつ札幌駅から半径 3km の範囲を図示する」

ことにチャレンジしてみよう。これまでの演習と違って、正解のソースコード全文は書かずにヒントのみに留めておくので、ヒントを参考にしながら考えてみてほしい(もちろん質問したり調べたりするのは自由)。

ヒント

- 今回は外部データの取得方法を変えたりしないので、「main.js」のみ変更すればよい。変更する元の「main.js」は、MySQL・2-2 と同じものである。
- 札幌駅を中心とした円を描画するのは、GoogleMapsAPI・2-1 でやったように、地図そのものを描画するタイミングで行う。ちなみに、地図の中心は札幌駅である。
- 半径 3km の円がすっぽり入るよう、ズームレベルも変える。
- デフォルトの円の色は黒であるが、半径 3km の円とマクドナルドのプロットが同じ色では見づらい。そこで、「strokeColor」オプションで円のフチの色が、「fillColor」オプションで円の塗りつぶしの色が変更できるので、上手く色を変えてみてほしい。また、店舗のプロットであれば円の中の塗りつぶしは不透明でもよいだろう。塗りつぶしの透明度を変えるには、「fillOpacity」オプションを使う。
- マクドナルドの位置情報のプロットについては、Google マップでよく見るピンを使う方法もある。この場合、マクドナルドの位置情報を描画するクラスを「Circle」から「Marker」に変える。「Marker」クラスでは、地図を指定する「map」オプションとピンの置き場所を指定する「position」オプションを設定する。

今回のヒントで紹介したクラスやオプションについての詳細は、「Google Maps JavaScript API の使い方まとめ - SYNCER」(https://lab.syncer.jp/Web/API/Google_Maps/JavaScript/)を見るとよい。

参考資料

- FOSS4G Hokkaido 2019 ハンズオン「JavaScript と MySQL で GIS を作ってみよう」スライド資料
  http://bit.ly/foss4g-2019-mysql

- FOSS4G Hokkaido 2019 ハンズオン「JavaScript と MySQL で GIS を作ってみよう」演習資料
  http://bit.ly/git-web-gis

- 今さら聞けない！PHP とは【初心者向け】TechAcademy マガジン
  https://techacademy.jp/magazine/6618

- JSON の紹介
  https://www.json.org/json-ja.html

- GeoJSON ・GIS 実習オープン教材
  https://gis-oer.github.io/gitbook/book/materials/web_gis/GeoJSON/GeoJSON.html