

オープンデータを活用しよう

～

初めてのウェブアプリ

アントワン トルヴェ (<http://trouve.sakura.ne.jp>)

後藤 孝行

2015/11/12、BODIKワークショップ、SRPビル

本日の材料はBODIKのウェブサイトにあります

①

本資料のPDF

技術紹介など

②

演習指示ページ

演習でやって欲しいこと（答えつき）

③

演習ファイル

演習に必要なファイル（HTMLやライブラリー）

全ては<http://www.bodik.jp/ws3>にあります



今日は Webページ を作ります

...ので、少し背景を説明させてください・・・

技術的背景

プロトコル

IPアドレス

HTTP

DNS

聞いたことがありますか？

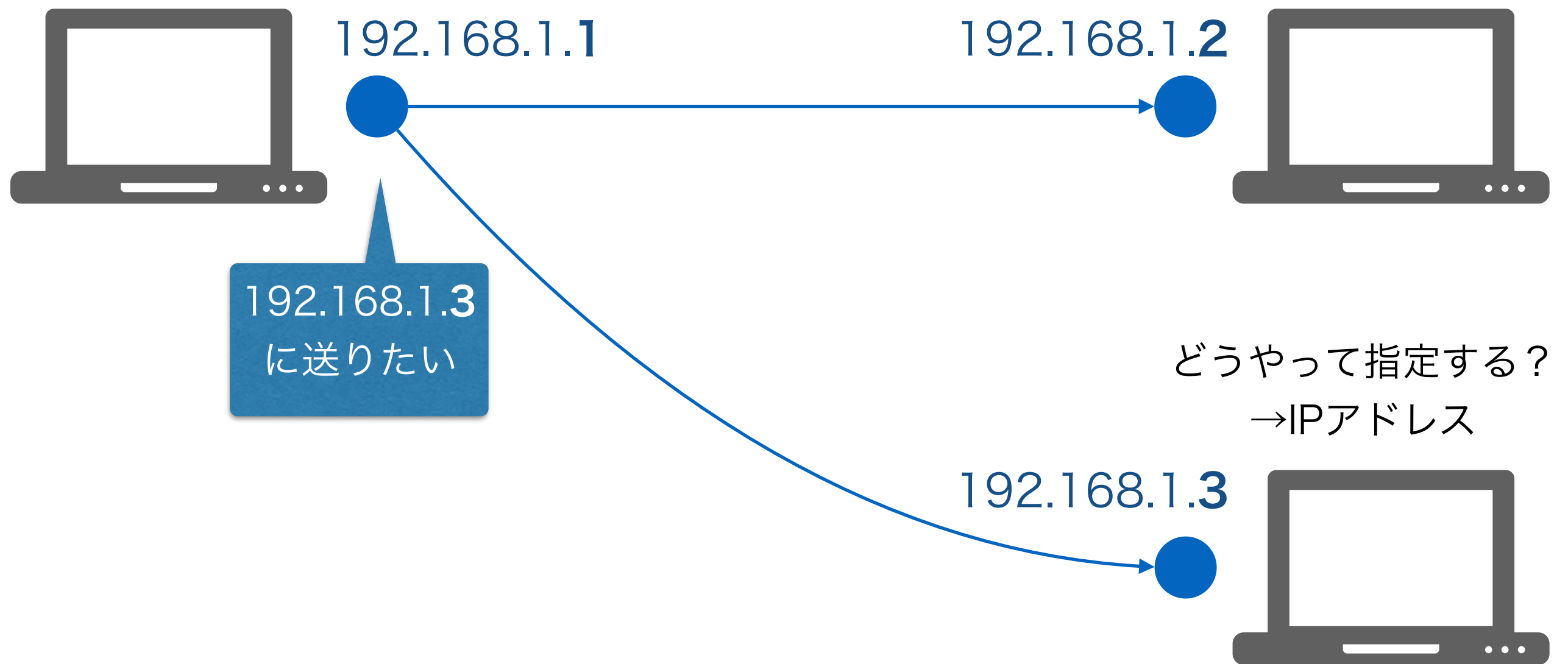
AJAX

ポート番号

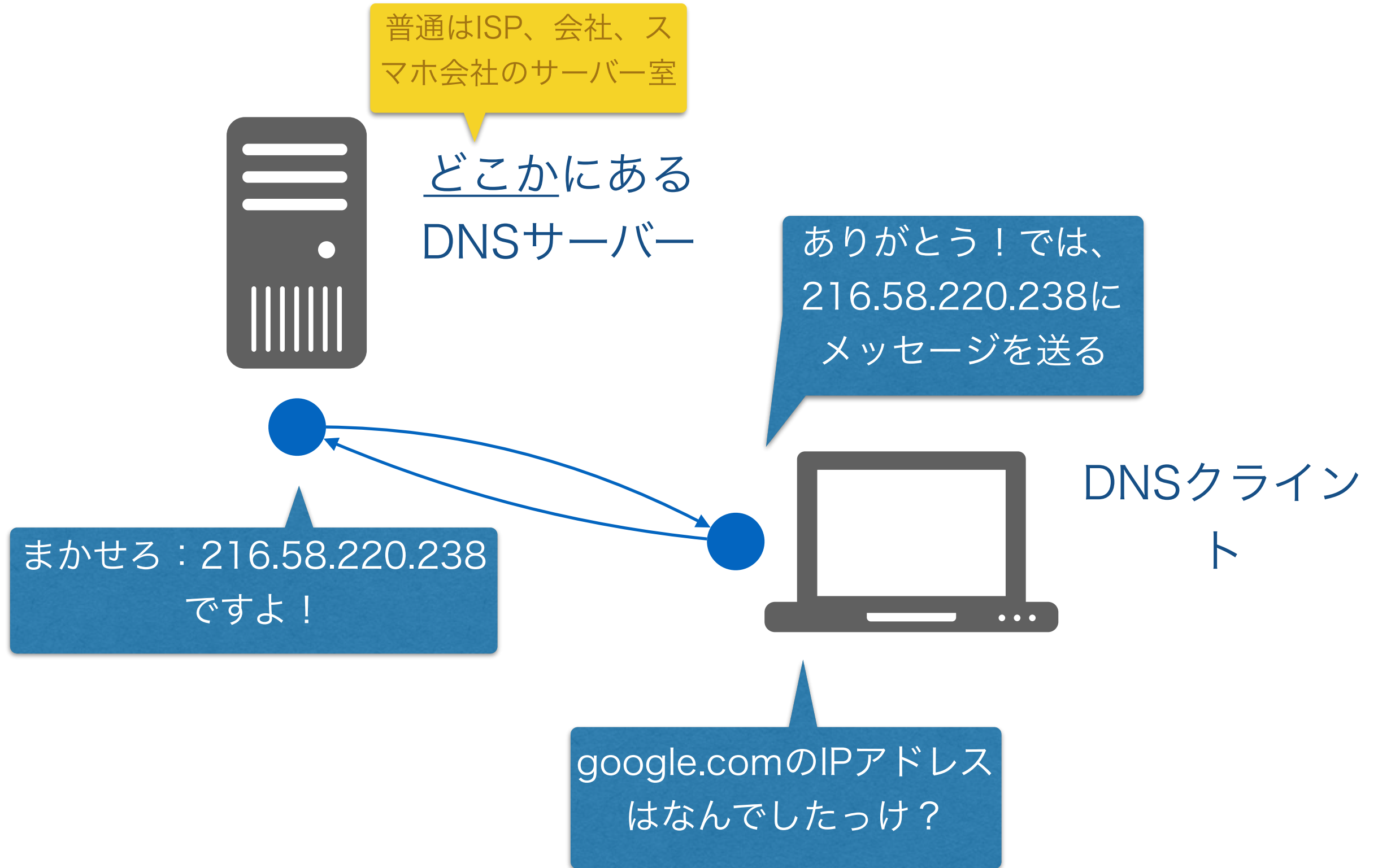
HTML

非同期プログラミング

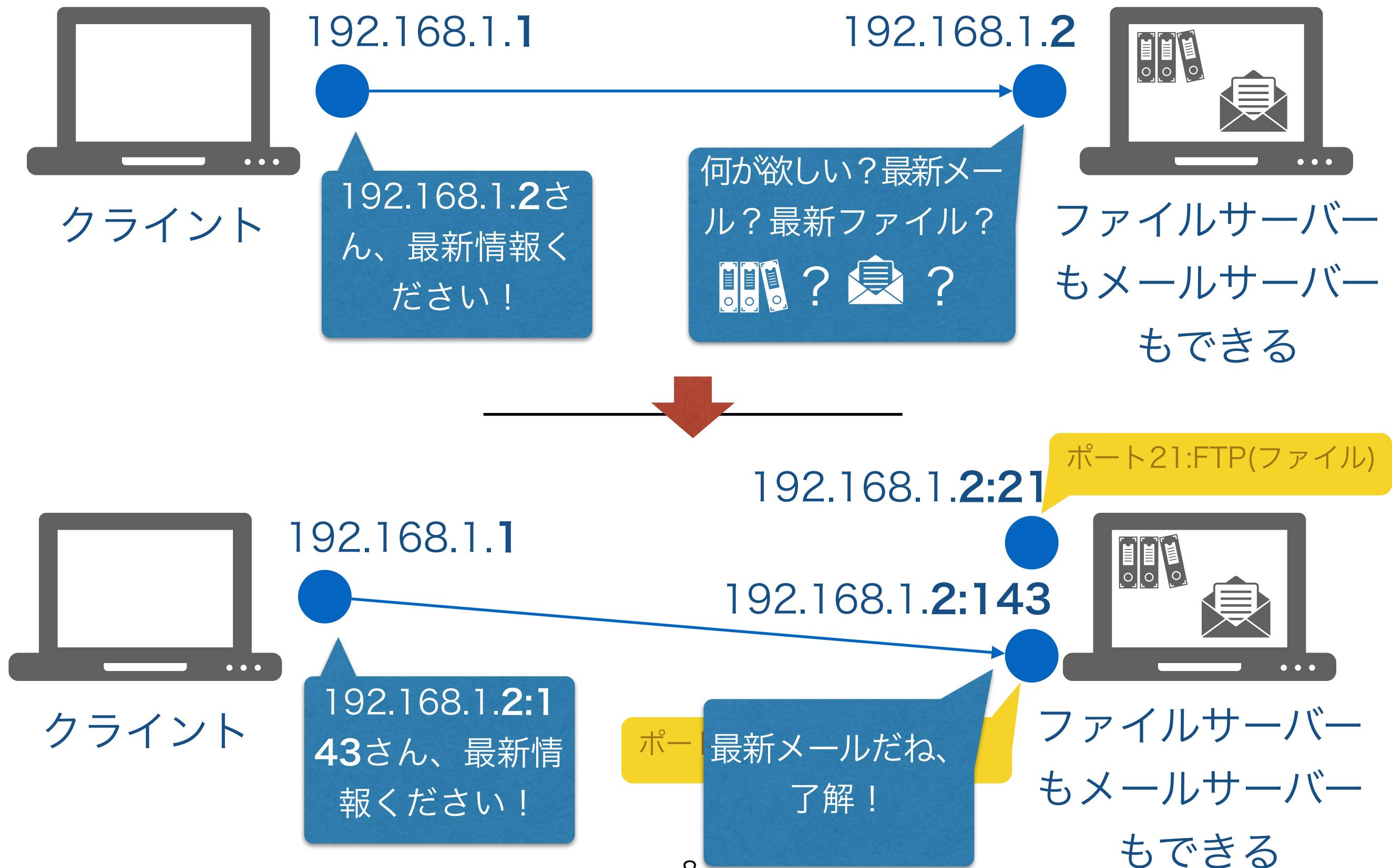
IPアドレスで遠隔パソコンを指定



DNSでIPアドレスを覚える必要ない



ポート番号でサービスを指定できる



HTTPはウェブで情報を交換できる



- ・ ウェブページが欲しいので、**HTTP**を使うと既に知っている
- ・ HTTPポート番号はIANAによると**80**です
- ・ がどうやって欲しい情報を指定できる？ → **HTTPプロトコル**

HTTPプロトコルはウェブで情報交換 するためにパソコンが使う言語です

クライアントはこの内容（テキスト）を
192.168.1.2のポート80に送信する

このウェブページが欲しい：
「192.168.1.2:80/perso/index.html」

GET /perso/index.html HTTP/1.1
Host: 192.168.1.2
Accept: text/html



クライアント

192.168.1.1

192.168.1.2:80



HTTPサーバー

サーバはこの内容（テキスト）を192.168.1.1に返す

HTTP/1.1 200 OK
Content-Type: text/html
<html> ... <html>

あった！送るね～

HTTPメッセージ内容について

- ・ HTTPメッセージは2つのパートに分けてある：
 - ・ **ヘッダー**：メッセージについて（メタデータ）
 - ・ **ボディ**：内容
- ・ メッセージは2種ある：
 - ・ **リクエスト**：クライアントとからサーバーへ
 - ・ **レスポンス**：サーバーからクライアントへ（リクエストに対する返事）

リクエストの事例

GET:やりたいアクション（他にPOST、PUT、DELETEなどがあります）

```
GET /perso/index.html HTTP/1.1  
Host: 192.168.1.2  
Accept: text/html
```

このリクエスはボディがない（基本的にGETリクエストはボディがない）

レスポンスの事例

STATUSコード。200は「OK」と意味するが、他に沢山あります（404など）

```
HTTP/1.1 200 OK  
Content-Type: text/html  
  
<html> ... </html>
```

ヘッダーとボディは空行で区切る

このレスポンスの内容はHTMLページです

HTTPヘッダーについて

リクエスト

```
GET /perso/index.html HTTP/1.1
Host: 192.168.1.2
Accept: text/html
```

ヘッダーパラメータもIANA
が管理する

レスポンス

```
HTTP/1.1 200 OK
Content-Type: text/html

<html> ... </html>
```

MIMEタイプもIANAが管理する

1 行目：基本情報

- ・ HTTPバージョン
- ・ レスポンスコード
- ・ リクエストの種類

2 行目～：パラメータ

キー：値

- ・ 主なヘッダーパラメータ：
 - ・ **Content-Type**：ボディのMIMEタイプ
 - ・ **Accept-Type**：欲しいMIMEタイプ（リクエストのみ）
 - ・ **Cookie**：クッキーの内容

URLはインターネット上のパソコンのアドレス

Uniform **R**esource **L**ocator

サンプル

http://www.bodic.jp:80/event/workshop/

プロトコル

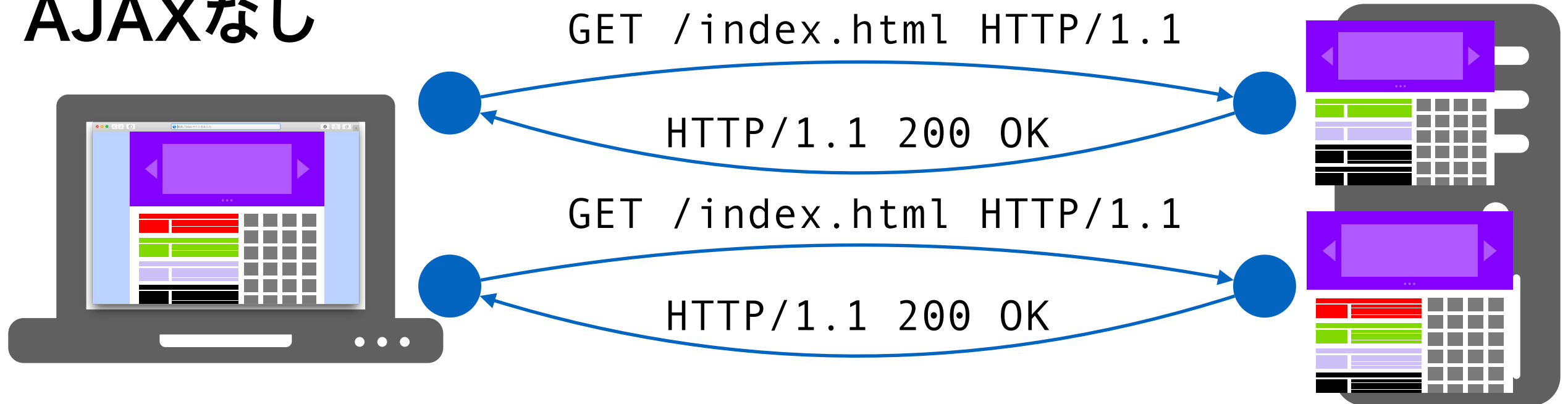
サーバー名（DNS
でIPアドレスに変換
する）

ポート番号。標準ポー
トだと、省略できる

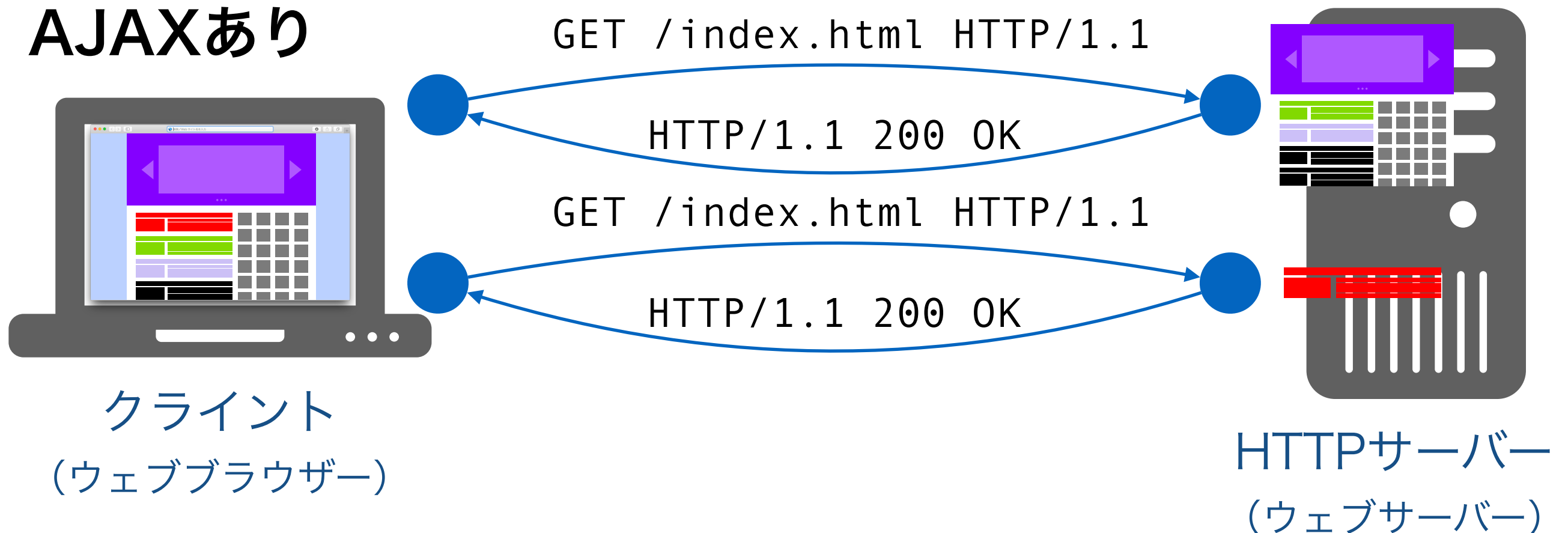
サーバー上
のパス

AJAXで動的に新しいコンテンツを取得できる

AJAXなし



AJAXあり



AJAXで動的に新しいコンテンツを取得できる

AJAXなし

GET /index.html HTTP/1.1

HTTP/1.1 200 OK

GET /index.html HTTP/1.1

HTTP/1.1 200 OK

メリット

- ・新しいコンテンツを得るためにページを再読み込みする必要はない

AJAXあり

GET /index.html HTTP/1.1

HTTP/1.1 200 OK

GET /index.html HTTP/1.1

HTTP/1.1 200 OK

- ・クライアント側に複数の箇所から情報を集めてページを生成できる (e.g. ミューズアグリゲータサイト)

クライアント

(ウェブブラウザ)

HTTPサーバー

(ウェブサーバー)

HTML/CSS/Javascriptでウェブページ内容を表現する

- ウェブサーバーがHTTPレスポンスのボディに格納する
- 現在バージョンはHTML5/CSS3

HTML5 を指定する	[	<pre><!DOCTYPE html> <html></pre>
headはペー ジのメタデー タを指定	[	<pre><head> <meta charset="UTF-8"> <title>Title</title> <style> /* some styling rules */ </style> <link rel="stylesheet" href="some_file.css"> <script> /* some javascript code */ </script> <script src="some_file.js"></script> </head></pre>
bodyには HTMLページ 自身を格納	[	<pre><body> <!-- Some HTML code --> </body> </html></pre>

HTML/CSS/Javascriptでウェブページ内容を表現する

- ・ウェブサーバーがHTTPレスポンスのボディに格納する
- ・現在バージョンはHTML5/CSS3

```
<!DOCTYPE html>  
<html>
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <title>Title</title>
```

①

```
  <style> /* some styling rules */ </style>
```

②

```
  <link rel="stylesheet" href="some_file.css">
```

①

```
  <script> /* some javascript code */ </script>
```

②

```
  <script src="some_file.js"></script>
```

CSS指定

JavaScript指定

```
<body>
```

```
<!-- Some HTML code -->
```

```
</body>
```

```
</html>
```

CSSとJavaScriptの指定方法が二つある

①直接HTMLファイルに格納

②単独ファイルをインポート

全ては<body>の中にも使える

それぞれの役割

- ・HTML：ページのコンテンツを静的に表現
- ・CSS：HTMLエレメントの見た目を静的に指定
- ・JavaScript：HTML/CSSで静的に宣言しているものを動的に変更する

HTMLページはDOMエレメントで出来ている

- ・ HTMLは内容を「箱」単位で構成されている
- ・ 各「箱」は「DOMエレメント」という

HTMLコード

```
<html>
<head>...</head>
<body>
<header>
  <h1>ページタイトル</h1>
</header>
<article>
  <h2>記事1</h2><p>ほげほげ</p>
</article>
<article>
  <h2>記事2</h2><p>ほげほげ</p>
</article>
</body>
</html>
```

ブラウザー画面



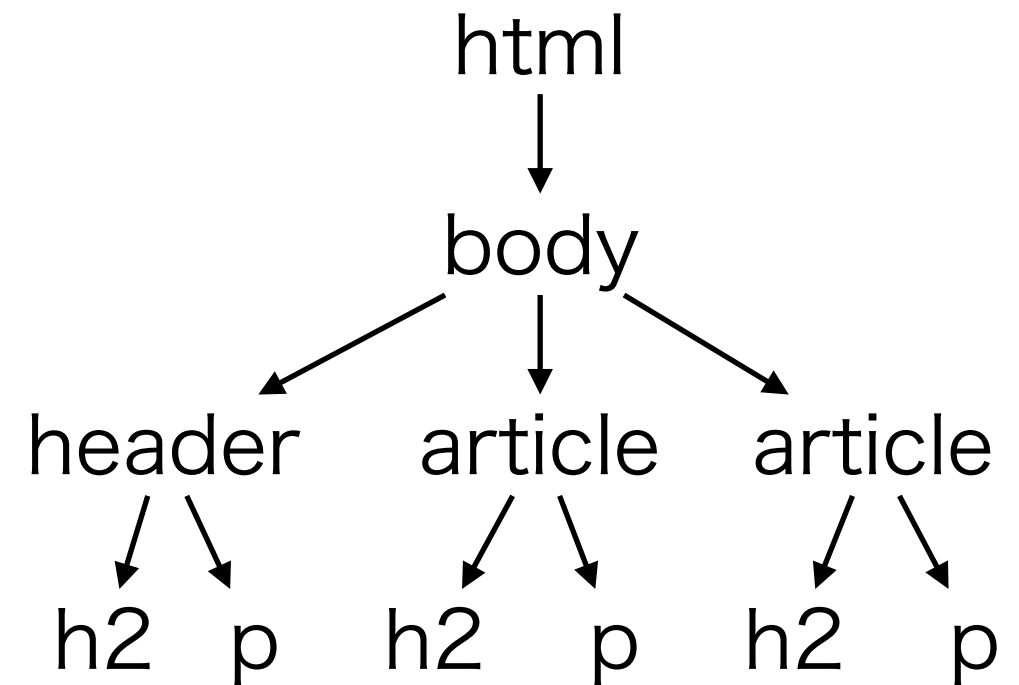
CSSルール
を省略

DOMエレメントについて

- ・ DOMエレメント親子関係があり、ツリー構造で表すことができる
- ・ 各DOMエレメントは**付加情報**を用いる
 - スタイル (CSSで定義)
 - アトリビュート (属性)

主なアトリビュート：

- ・ 「style」 オンラインCSSルール
- ・ 「class」 CSSクラス (CSSルールセット)
- ・ 「id」 ページ内のDOMエレメントのユニークID



サンプル

```
<div class="CSSルール" id="ユニークID" class="CSS  
ルールセット名">...</div>
```


セレクタでDOMエレメントを指定できる

セレクタはCSSやjQueryで使われている

①をセレクトする

```
「body header」
```

全ての②ををセレクトする

```
「body article h2」、「article h2」
```

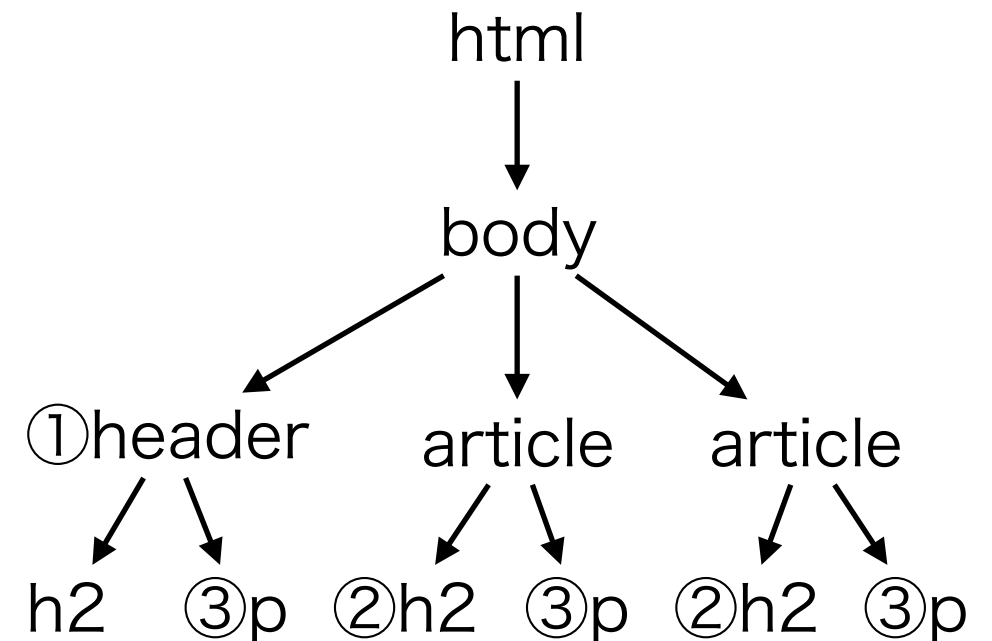
全ての③ををセレクトする

```
「body p」、「p」
```

IDやクラス名でも指定できる

```
p classname、div#idname
```

より複雑なセレクターもありますが、今回は省略する



jQueryでは簡単にDOMやAJAX操作できる

- ・ jQueryはJavaScriptライブラリです
- ・ ページを起動したあとにDOM操作やAJAXデータ取得ができる
- ・ jQueryがなくても直接Javascriptで同じことができるが、jQueryのほうが楽
- ・ jQuery機能はJavascriptの「\$」関数で利用可能

```
<head>  
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
  <title>title</title>  
  <script src="http://code.jquery.com/jquery-1.10.2.js"></script>  
</head>
```

HTMLの<head>に参照し、使えるようになる

jQueryでAJAXデータ取得について

例 1 : 「http://www.bodic.org」 のHTMLコードを取得する

```
> var process_result = function(response_body, status) {}  
> $.get('http://www.bodic.org', process_result)
```

- ・ 今回がGETリクエストを投げる。他に「post」などあるます
- ・ パラメータは2つあります：サーバーのURLと結果処理関数
- ・ 結果処理関数はサーバーからレスポンスが来てから実行されます（非同期という）

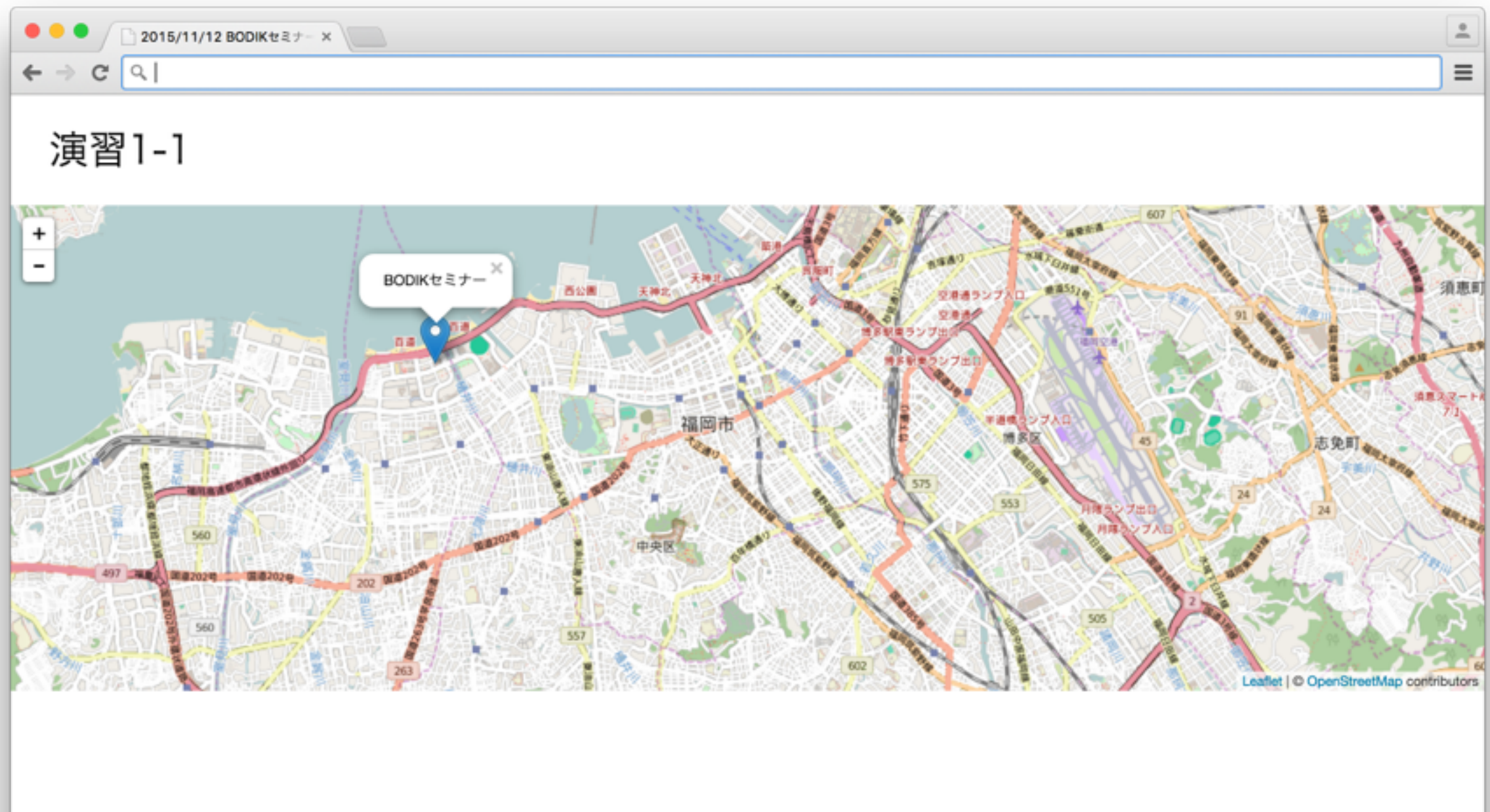
例 2 : 「http://www.bodic.org」 のHTMLコードを全ての
<pre>にエリメントに挿入する

```
> $.get('http://www.bodic.org', function(data, status){  
    $('pre').text(data)  
})
```


演習1. 地図でSPARQLクエリー結果を可視化する



演習1-1:地図初期化、マーカー、ポップアップ



```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <title>2015/11/12 BODIKセミ
```

```
  <!-- Leaflet(地図ライブラリ)>
```

```
  <script type="text/javascript">
```

```
  <link rel="stylesheet" href="../../leaflet/leaflet.css">
```

```
  <style>
```

```
    body { padding: 0; margin: 0; }
```

```
    h1 { margin-left: 1em; font-family: sans-serif; font-weight: lighter; }
```

```
    #map { width: 100%; height: 400px; padding: 0; margin: 0; }
```

```
  </style>
```

```
</head>
```

```
<body>
```

```
<h1>演習1-1</h1>
```

```
<div id="map"></div>
```

```
<script type="text/javascript">
```

```
var map = L.map('map').setView([51.505, -0.09], 13);
```

```
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {  
  attribution: '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors'  
}).addTo(map);
```

```
// 問題: SRPの座標(33.5932094, 130.3558809)にマーカーを追加してください
```

```
</script>
```

```
</body>
```

```
</html>
```

演習1-1ファイル: ex11/ex11.html

Leafletという地図ライブラリをインポート

- ・「.js」: JavaScriptで書かれたロジック
- ・「.css」: 地図エレメントを正しく表示するためのCSSルール

ページ表示のためのカスタムCSSルール

DOMエレメント(2つだけ!)

今回編集する箇所

地図可視化ロジック
(JavaScriptで書く)

今回編集する箇所

Leafletについて

- ・ 地図の表示・操作を簡単にできるライブラリー
 - ・ 表示だけ：地図データはGoogleなどから取得する必要があります
- ・ Leaflet.jsをインポートすると「L」関数を使えるようになる
 - ・ Leafletの全ての機能は「L」から使う
- ・ Leafletでは地図が複数のレイヤーで構成される
 - ・ 地図レイヤー、マーカーレイヤー、操作レイヤーなど
 - ・ 重ねることによって、複雑な地図をつくれます
- ・ 地図を作成すると、動的に（JavaScriptを利用）必要なDOMエレメントが追加される

```
<div id="map"></div>
```

```
> var map = L.map('map')
```

このmapはDOMエレメントのID。新しいDOMエレメントはこのエレメントにいれる。

Leafletタイルレヤーについて

Leafletは地図データをタイル単位で管理する
→Leaflet独自ではなく、標準的なやり方です
タイルは地図の一部
→緯度・経度・拡大度・種類で指定する

タイルレヤーの作成

```
> var tile_layer = L.tileLayer(...)
```

パラメータは地図タイルのURLのテンプレート（今回はOpenStreetMapを利用）と表示したいコピーライト情報

タイルレヤーを地図に追加する

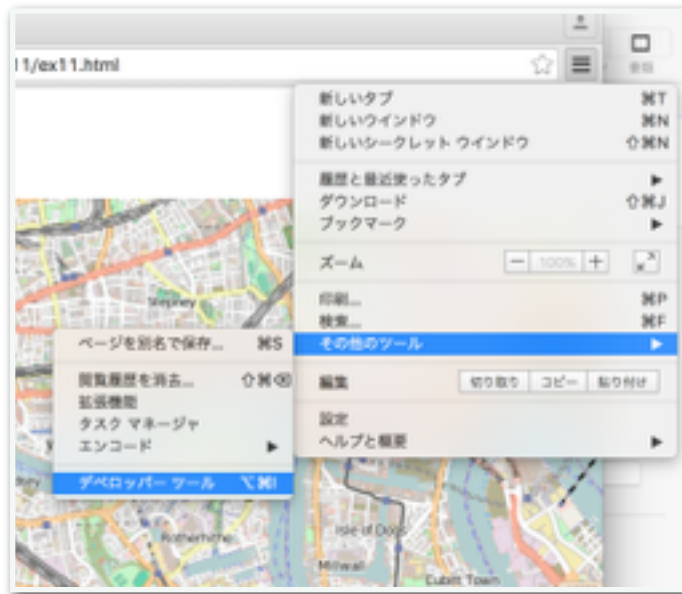
```
> tile_layer.addTo(map)
```

ex11.htmlでは両方一緒にしている

ブラウザー開発機能で

Leafletが生成するDOMを確認しましょう

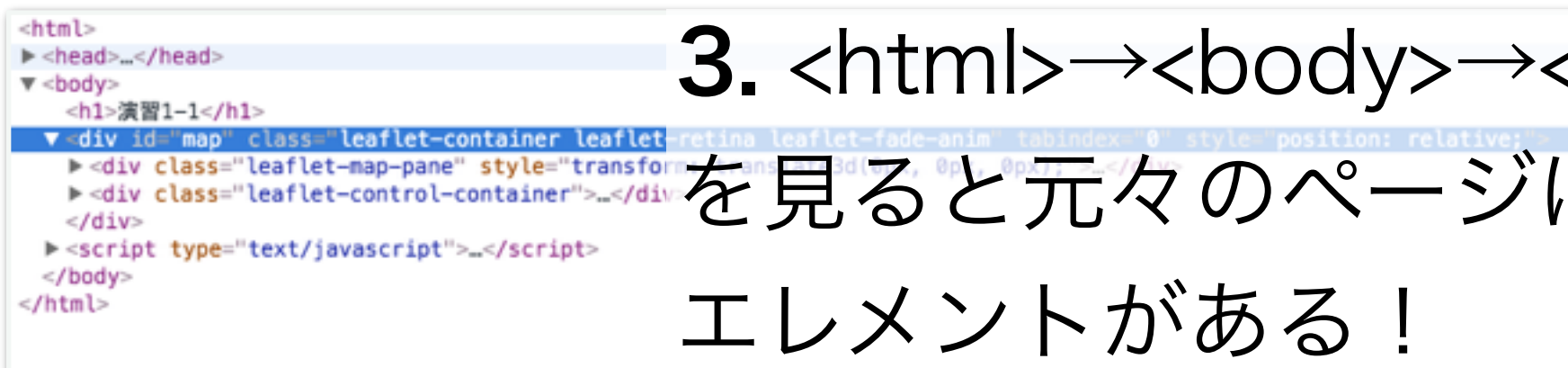
1. Chromeでは「メニュー」
→「その他のツール」→「デベ
ロッパー」



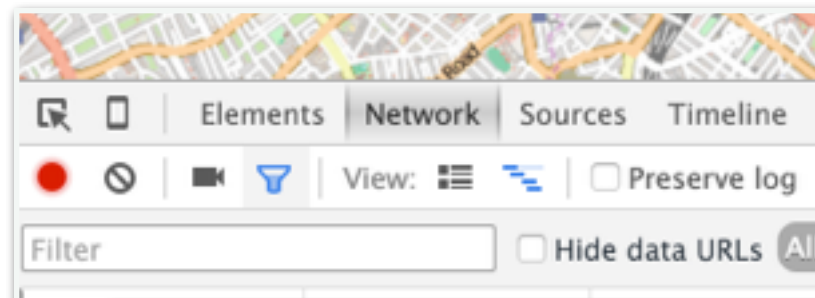
2. 下に表すGUIに「Elements」をク
リックするとDOMが表示される



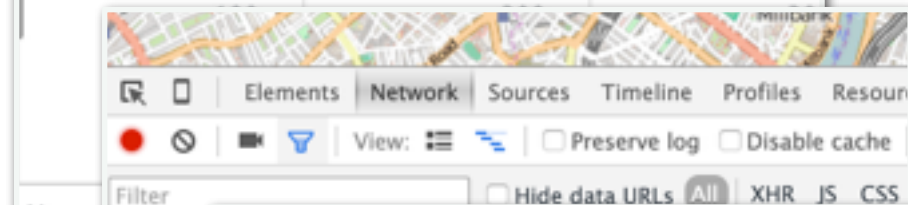
3. `<html>→<body>→<div id="map">`
を見ると元々のページになかったDOM
エレメントがある！



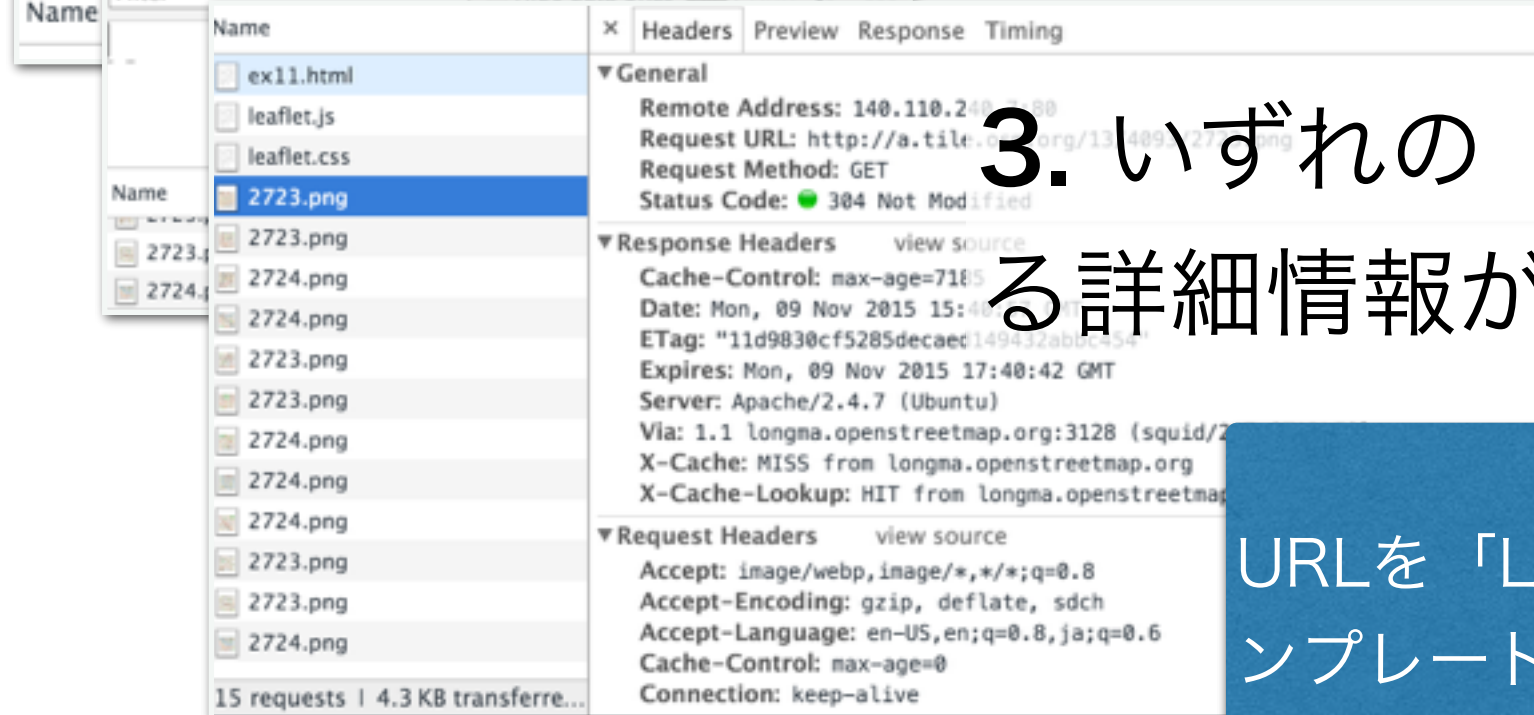
ブラウザー開発機能で地図タイルを見ましょう



1. 開発GUIの「Network」タブを開く



2. 再読み込みするデータを見れる



3. いずれの「xxxx.png」をクリックする詳細情報が表される

URLを「L.tileLayer()」に渡したURLテンプレートと比較してください。

演習問題1-1の1:地図で福岡を中心に置く

下記の行は地図を作り、MapViewで中央座標や拡大度を設定する

これはロンドンの緯度・経度

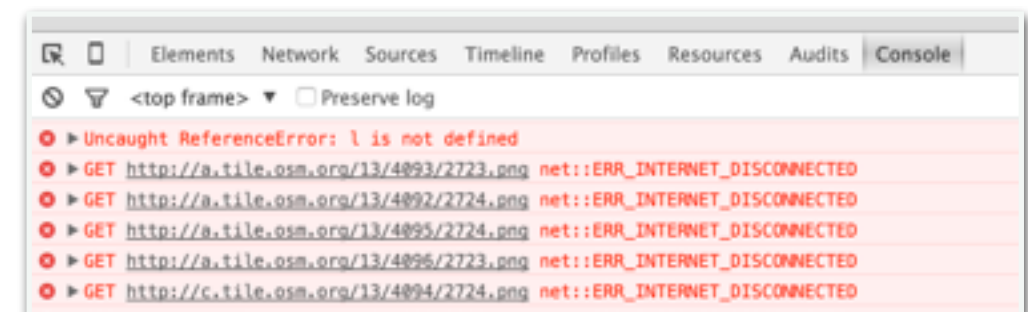
拡大度は適当に13にしているが、触る必要ない

```
> var map = L.map('map').setView([51.505, -0.09], 13);
```

問題：まず、ロンドンの座標を**福岡の座標**に変えて下さい

福岡の座標はググってみよう

便利機能：何も表示されなかったら、開発GUIのConsoleをご確認下さい



演習問題1-1の2: マーカーをSRPの位置に地図上に追加する

緯度 : 33.5932094
経度 : 130.3558809

マーカーオブジェクト作成

```
> var marker = L.marker([緯度, 経度])
```

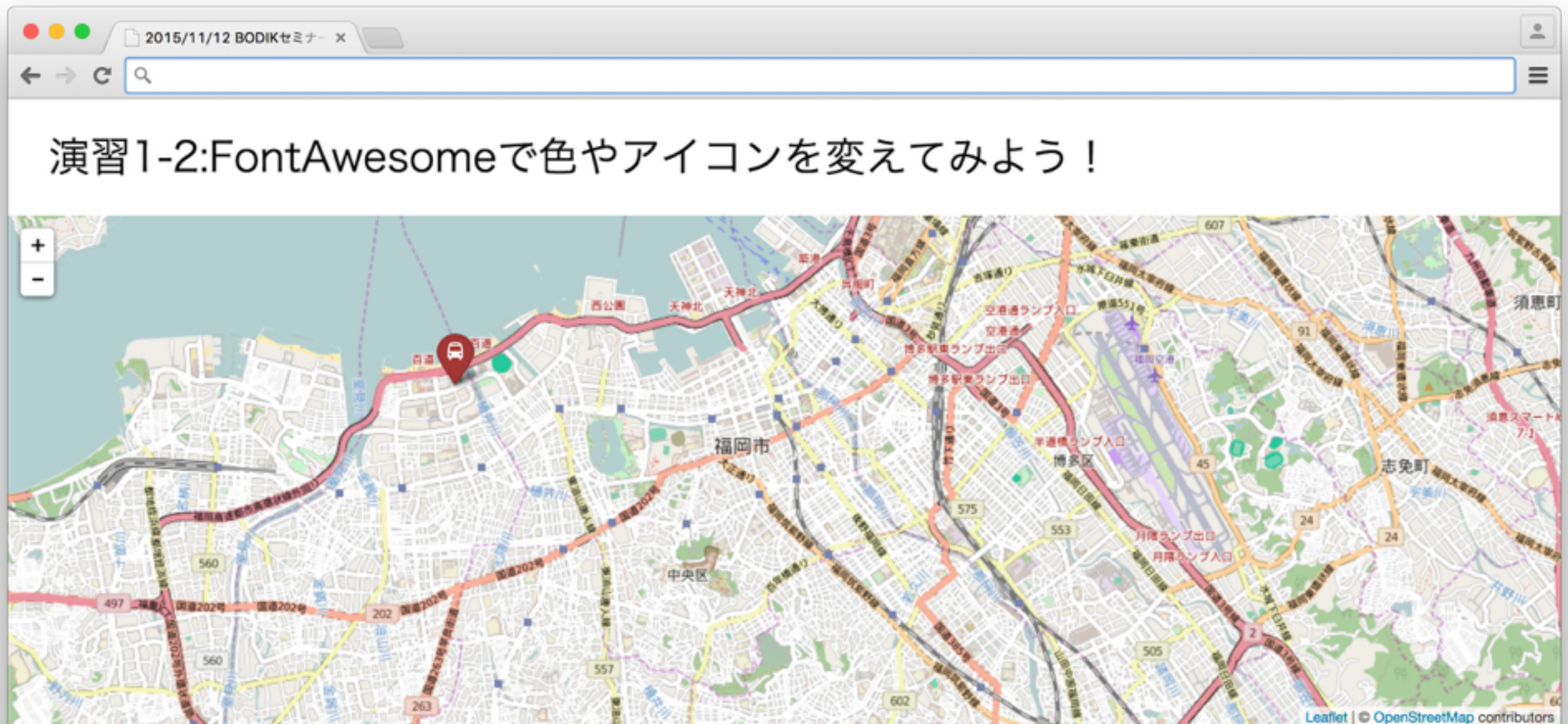
マーカーを地図に追加する

```
> marker.addTo(map)
```

既存マーカーにポップアップを付ける

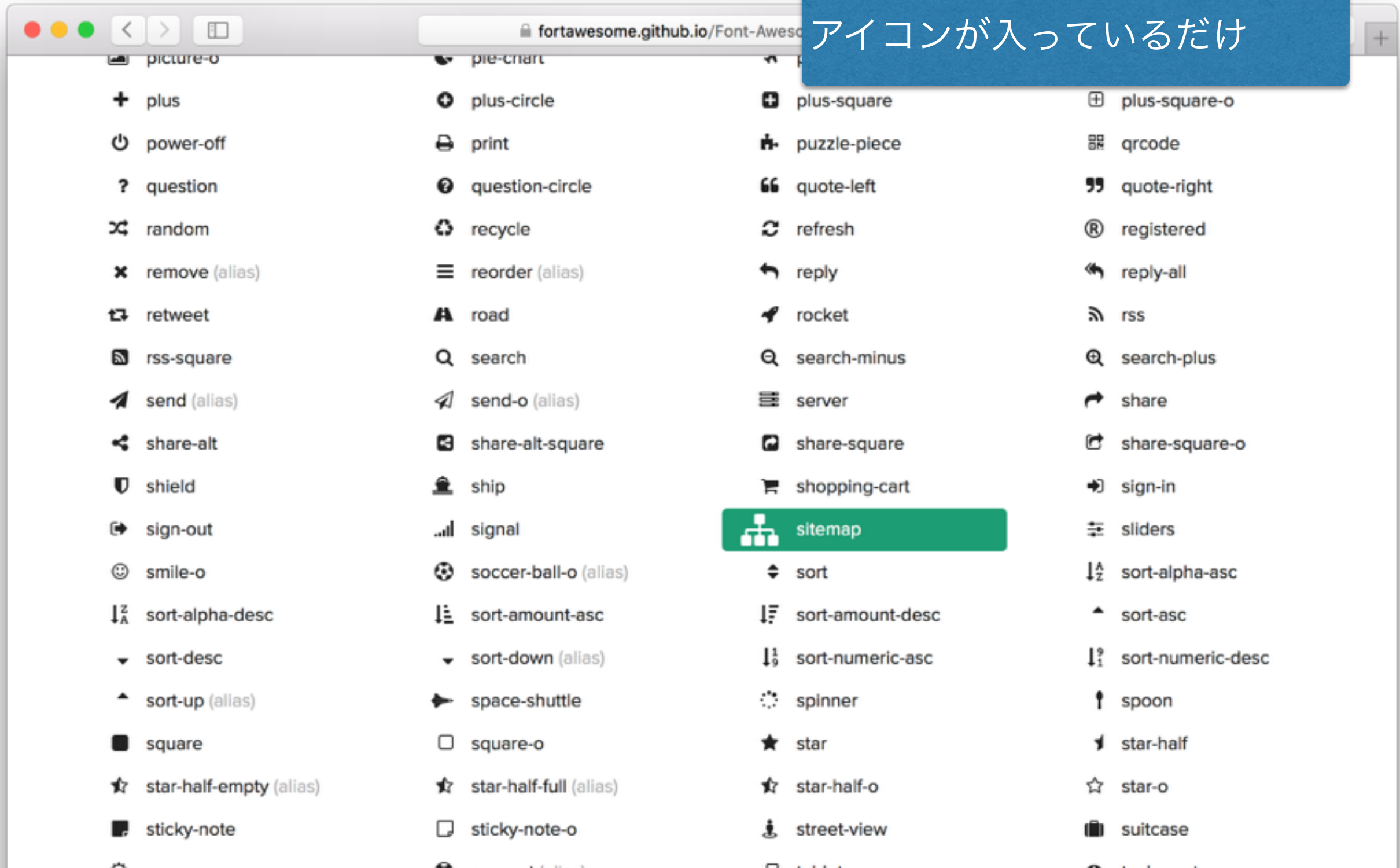
```
> marker.bindPopup('BODIKセミナー')
```


演習1-2: マーカーの色とアイコン コングで遊ぶ



FontAwesomeは諸々なアイコンを手軽 に使えるようにするフォント

フォントです：文字の代わりに
アイコンが入っているだけ



FontAwesomeは使い易い！

FontAwesomeはCDNが定常しているCSSだけで使える

この行は「ex12/ex12.html」の<head>にある

```
<link href='http://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.4.0/css/font-awesome.min.css' rel='stylesheet'>
```

この演習では「CloudFlare」というContentDeliveryNetworkが提供してるFontAwesome4.4のCSSを利用

HTMLページでは、下記のコードで各アイコンを使える

```
<i class="fa fa-car"></i>
```

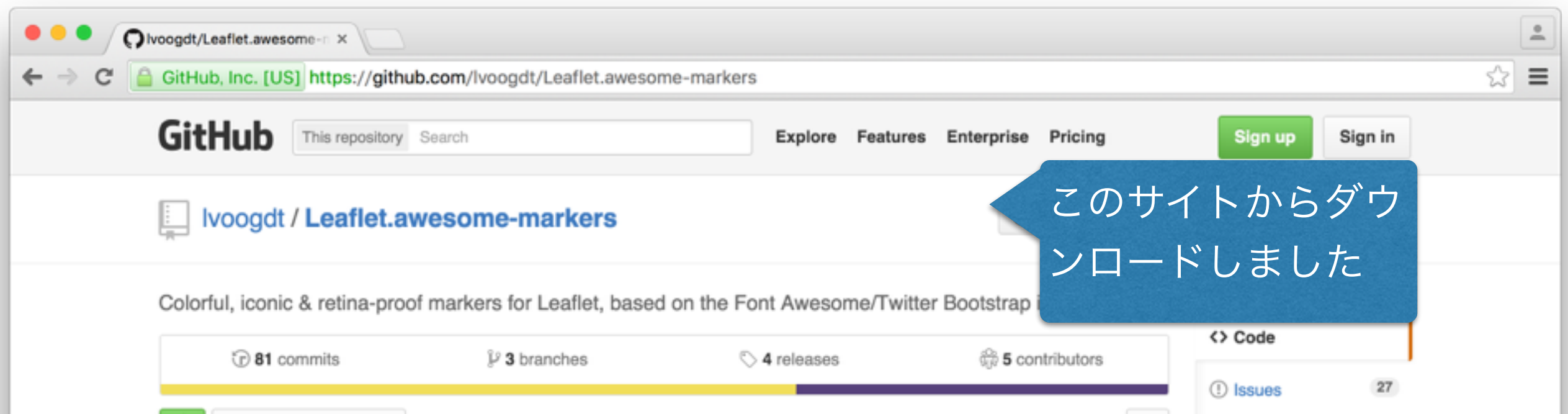
アイコン「car」の場合。利用可能アイコンのリストは「<https://fontawesome.github.io/Font-Awesome/icons/>」をご参考

Leaflet-Awesome-MarkerはFontAwesomeのアイコンをLeafletのマーカーに使えるようにするライブラリ

この行は「ex12/ex12.html」の<head>にある

```
<link rel="stylesheet" href="../../leaflet.awesome-markers/dist/leaflet.awesome-markers.css">
...
<script type="text/javascript" src="../../leaflet.awesome-markers/dist/leaflet.awesome-markers.js"></script>
```

Font-Awesome-MarkerはCSSもJavascriptもファイルを2つが必要です



Leaflet-Awesome-Markerの使い方

```
var restaurantMarker =  
L.AwesomeMarkers.icon({  
  icon: 'coffee',  
  markerColor: 'darkred',  
  prefix: 'fa',  
  iconColor: 'white'  
});
```

ステップ1：マーカースタイル作成

- ・「icon」：Font-Awseomeのアイコン名
- ・markerColor:マーカーの色
- ・iconColor:アイコンの色

```
L.marker([33.5932094, 130.3558809],  
{ icon: restaurantMarker })  
  .addTo(map)  
  .bindPopup('BODIKセミナー')
```

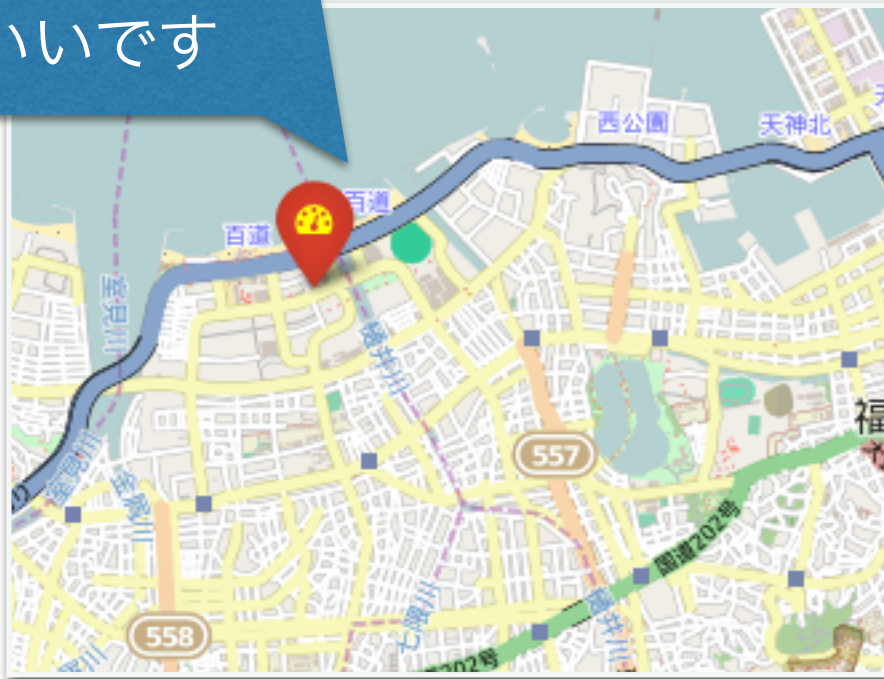
ステップ2：スタイルを指定

マーカー作成関数に追加パラメーターに
ステップ1 に作ったスタイルを指定

このコードは既にex12/ex12.htmlにあります！

演習 1-2 : 色やアイコンを 変えてみよう

色やアイコンはなんでもいいです



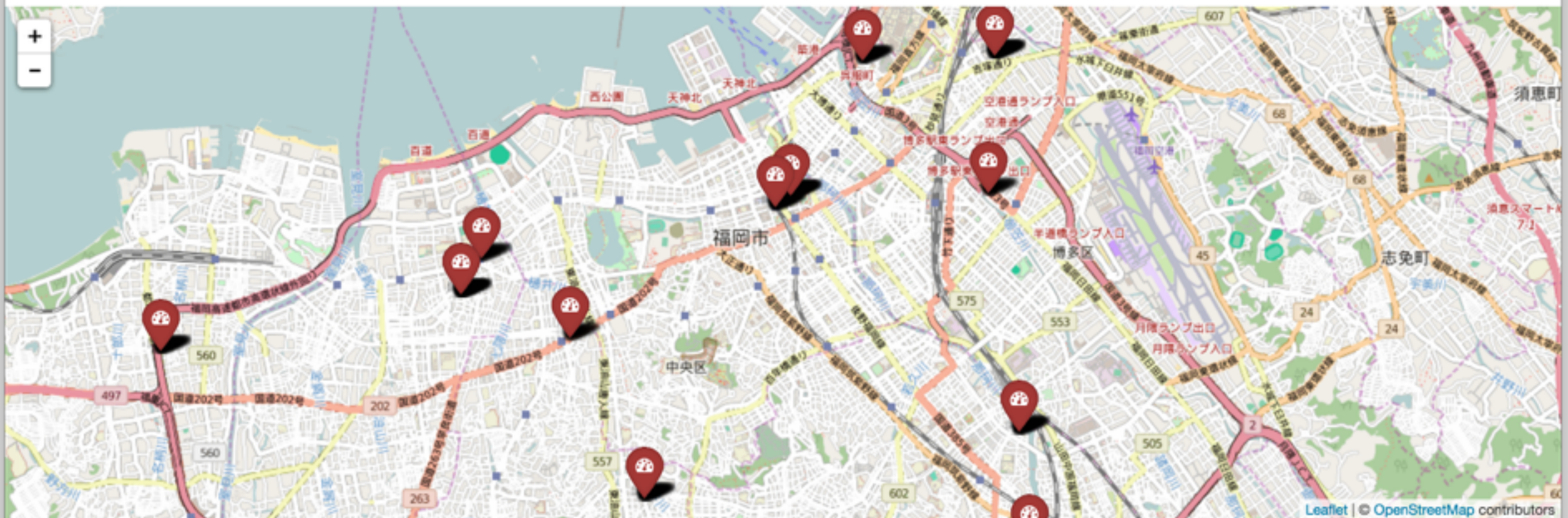
を変えてみよう！🚗



興味のある方はFontAwesomeのアイコンをHTMLで使ってみて下さい

演習1-3:SPARQLクエリの結果を地図上に可視化する

演習1-3:SPARQL検索結果を地図で可視化しよう！



HTTP/1.1 200 OK
pragma: no-cache
Content-Type: application/sparql-results+json; charset=utf-8
cache-control: must-revalidate,no-cache,no-store

```
{  
  "head": {  
    "vars": [ "s", "lon", "lat", "sensor type" ]  
  },  
  "results": {  
    "bindings": [  
      {  
        "s": "http://www.example.org/sensor/1",  
        "lon": 130.4, "lat": 38.3, "sensor type": "Temperature"  
      },  
      {  
        "s": "http://www.example.org/sensor/2",  
        "lon": 130.5, "lat": 38.4, "sensor type": "Humidity"  
      },  
      {  
        "s": "http://www.example.org/sensor/3",  
        "lon": 130.6, "lat": 38.5, "sensor type": "Air Quality"  
      },  
      {  
        "s": "http://www.example.org/sensor/4",  
        "lon": 130.7, "lat": 38.6, "sensor type": "Noise Level"  
      },  
      {  
        "s": "http://www.example.org/sensor/5",  
        "lon": 130.8, "lat": 38.7, "sensor type": "Light Intensity"  
      },  
      {  
        "s": "http://www.example.org/sensor/6",  
        "lon": 130.9, "lat": 38.8, "sensor type": "Vibration"  
      },  
      {  
        "s": "http://www.example.org/sensor/7",  
        "lon": 131.0, "lat": 38.9, "sensor type": "Air Pressure"  
      },  
      {  
        "s": "http://www.example.org/sensor/8",  
        "lon": 131.1, "lat": 39.0, "sensor type": "Soil Moisture"  
      },  
      {  
        "s": "http://www.example.org/sensor/9",  
        "lon": 131.2, "lat": 39.1, "sensor type": "Water Level"  
      },  
      {  
        "s": "http://www.example.org/sensor/10",  
        "lon": 131.3, "lat": 39.2, "sensor type": "Wind Speed"  
      }  
    ]  
  }  
}
```


「...」で数箇所を省略

ex13/ex13.htmlの全体構成

```
<!DOCTYPE html>
<html>
<head>
...
</head>
<body>
<h1>演習1-3: SPARQL 検索結果を地図で可視化しよう！</h1>
<div id="map"></div>
<pre id="sandbox">
</pre>
<script>
$.post(
  'https://www.bodic.org/sparql',
  {
    query: ...
  },
  function(data, status, jqXHR) {
    $("#sandbox").text("HTTP/1.1 " + jqXHR.status + " " + jqXHR.statusText + "\n" + jqXHR.getAllResponseHeaders() + "\n" + jqXHR.responseText)
    ...
  }
).fail(function(jqXHR) {
  $("#sandbox").text("HTTP/1.1 " + jqXHR.status + " " + jqXHR.statusText + "\n" + jqXHR.responseText)
})
</script>
</body>
</html>
```

今回はjQueryを使う：

- ・ AJAXデータ取得
- ・ <pre>DOMエレメントにAJAXから得たレスポンス挿入

AJAXコールGETではなくPOSTリクエストを投げる

BODICのSPARQLエンドポイントのアドレス

AJAXレスポンスが来た時の処理。“text”間せうで、<pre>エレメントの内容を書き換える

AJAXコールが失敗した時の処理。エラーは<pre>エレメントに挿入

AJAXデータ取得

このHTTPリクエストが「www.bodic.org」に送られる

```
$.post(
  'https://www.bodic.org/sparql',
  {
    query: [
      'prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>',
      'prefix bodic:<'+bodic_voc+'>',
      'select ?s ?lon ?lat ?stype ?sensor_type',
      'graph <'+dataset+'>',
      '?lat_hiri a bodic:RawPosition;',
      '?lon_hiri a bodic:RawPosition;',
      '?stype_hiri a bodic:RawPosition;',
      '?s ?lon_hiri ?lon ; ?lat_hiri ?lat ; ?stype_hiri ?sensor_type',
      '}',
      'FILTER(?sensor_type != "GPS")',
      '}]'.join(" ")
    ],
    function(data, status, jqXHR) {
      $("#sandbox").text("HTTP/1.1 200 OK");
      jqXHR.getAllResponseHeaders() + "Content-Type: application/sparql-results+json; charset=utf-8";
      var bindings = data.results.bindings;
      for(var i=0; i<bindings.length; i++) {
        var b = bindings[i];
        var lon = parseFloat(b.lon_hiri);
        var lat = parseFloat(b.lat_hiri);
        var sensor_type = b.sensor_type;
        // 問題:ここにマーカー導入コード
      }
    }
  )
);
```

POST /sparql HTTP/1.1

...
query=prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>

このHTTPリレスポンスが「www.bodic.org」から戻ってくる（うまく行く場合）

HTTP/1.1 200 OK

Content-Type: application/sparql-results+json;
charset=utf-8

...

```
{
  "heads": { vars: [...] },
  "results": { "bindings": [...] }
}
```

HTTPだけど、ボディはHTMLではない

今回のSPARQLクエリーの説明

- ・「GRAPH」キーワードでは検索を指定したデータセットに限定できる
- ・データセットのIRIの取得はhttps://www.bodic.org/search_datasets.htmlに検索できる。今回は「福岡市の大気環境測定結果（直近48時間）」というデータセットを利用

```
prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>,
prefix bodic:<http://www.bodic.org/common_vocabulary#>,
select ?s ?lon ?lat ?sensor type {
  graph <http://www.bodic.org/datasets/U44c89acdc50078f5/
G2a57ee9d4b0c2dbddeb0c4b017752880> {
    ?lat_hiri a bodic:RawProperty ; rdfs:label "緯度"@ja .
    ?lon_hiri a bodic:RawProperty ; rdfs:label "経度"@ja .
    ?stype_hiri a bodic:RawProperty ; rdfs:label "測定項目名称"@ja .
    ?s ?lon_hiri ?lon ; ?lat_hiri ?lat ; ?stype_hiri ?sensor_type .
  } .
  FILTER(?sensor_type != "")
}
```

今回のSPARQLクエリーの説明

```
prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>,
prefix bodic:<http://www.bodic.org/common_vocabulary#>,
select ?s ?lon ?lat ?sensor_type {
  graph <http://www.bodic.org/datasets/U44c89acdc50078f5/
G2a57ee9d4b0c2dbddeb0c4b017752880> {
    ?lat_hiri a bodic:RawProperty ; rdfs:label "緯度"@ja .
    ?lon_hiri a bodic:RawProperty ; rdfs:label "経度"@ja .
    ?stype_hiri a bodic:RawProperty ; rdfs:label "測定項目名称"@ja .
    ?s ?lon_hiri ?lon ; ?lat_hiri ?lat ; ?stype_hiri ?sensor_type .
  } .
  FILTER(?sensor_type != "")
}
```

BODIC.orgのサイトで可視化できるヘッダー
名からヘッダーのIRIを取得

今回のSPARQLクエリーの説明

```
prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>,
prefix bodic:<http://www.bodic.org/common_vocabulary#>,
select ?s ?lon ?lat ?sensor_type {
  graph <http://www.bodic.org/datasets/U44c89acdc50078f5/
G2a57ee9d4b0c2dbddeb0c4b017752880> {
    ?lat_hiri a bodic:RawProperty ; rdfs:label "緯度"@ja .
    ?lon_hiri a bodic:RawProperty ; rdfs:label "経度"@ja .
    ?stype_hiri a bodic:RawProperty ; rdfs:label "測定項目名称"@ja .
    ?s ?lon_hiri ?lon ; ?lat_hiri ?lat ; ?stype_hiri ?sensor_type
  } .
  FILTER(?sensor_type != "")
}
```

今回のデータセットは大気センサーです：

- ・ センサーの座標(変数?lonと?latに格納)
- ・ 計測データ(今回は合致しない)
- ・ センサーの種類(変数?sensor_typeに格納)

今回のSPARQLクエリーの説明

```
prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>,
prefix bodic:<http://www.bodic.org/common_vocabulary#>,
select ?s ?lon ?lat ?sensor_type {
  graph <http://www.bodic.org/datasets/U44c89acdc50078f5/
G2a57ee9d4b0c2dbddeb0c4b017752880> {
    ?lat_hiri a bodic:RawProperty ; rdfs:label "緯度"@ja .
    ?lon_hiri a bodic:RawProperty ; rdfs:label "経度"@ja .
    ?stype_hiri a bodic:RawProperty ; rdfs:label "測定項目名称"@ja .
    ?s ?lon_hiri ?lon ; ?lat_hiri ?lat ; ?stype_hiri ?sensor_type .
  }
  FILTER(?sensor_type != "")
}
```

空文字列を無視するよ
うなフィルター

今回のSPARQLクエリーの説明

```
prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>,
prefix bodic:<http://www.bodic.org/common_vocabulary#>,
select ?s ?lon ?lat ?sensor_type {
  graph <http://www.bodic.org/datasets/U44c89acdc50078f5/
G2a57ee9d4b0c2dbddeb0c4b017752880> {
    ?lat_hiri a bodic:RawProperty ; rdfs:label "緯度"@ja .
    ?lon_hiri a bodic:RawProperty ; rdfs:label "経度"@ja .
    ?stype_hiri a bodic:RawProperty ; rdfs:label "測定項目名称"@ja .
    ?s ?lon_hiri ?lon ; ?lat_hiri ?lat ; ?stype_hiri ?sensor_type .
  } .
  FILTER(?sensor_type != "")
}
```

クエリーの意味:

「福岡市の大気環境測定結果（直近48時間）」という
データセットの中から、
全てのセンサーの座標や種類を取得する

SPARQL結果形式について

application/sparql-results+json;

```
select ?s ?lon ?lat ?sensor_type { ... }
```

```
{
  "head": {
    "vars": [ "s" , "lon" , "lat" , "sensor_type" ]
  },
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://www.bodic.org/datasets/U44c89acdc50078f5/
G2a57ee9d4b0c2dbddeb0c4b017752880/raw_line_246" } ,
        "lon": { "type": "literal" , "value": "130.437768" } ,
        "lat": { "type": "literal" , "value": "33.672388" } ,
        "sensor_type": { "type": "literal" , "value": "総炭化水素" }
      },
      ...
    ]
  }
}
```

heads.vars[]クエリーの変数名を格納する配列

results.bindingsは合致した結果を格納する配列です。各結果はクエリーの変数の値をオブジェクトを使って格納する。

- ・ JSON形式ですので、直接JavaScriptオブジェクトとして扱える
- ・ JSONの代わりにXMLなども戻すことができますが、 bodic.org は標準でJSONを返す

AJAX結果処理関数の説明

```
<pre id="sandbox"></pre>
```

パラメータについて

- ・ data: HTTPレスポンスのボディ (JavaScriptオブジェクトとして)
- ・ status: HTTP Status (200:成功)
- ・ jqXHR: 無視していいです

```
function(data, status, jqXHR) {  
    $("#sandbox").text("HTTP/1.1 " + jqXHR.status + " "  
jqXHR.statusText + "\n" + jqXHR.getAllResponseHeaders()  
"\n" + jqXHR.responseText)  
    var bindings = data.results.bindings;  
    for(var i=0; i<bindings.length; i++){  
        var b = bindings[i]  
        var lon = parseFloat(b.lon.value)  
        var lat = parseFloat(b.lat.value)  
        var sensor_type = b.sensor_type.value  
        // 問題:ここにマーカー導入コードを入れましょう  
    }  
}
```

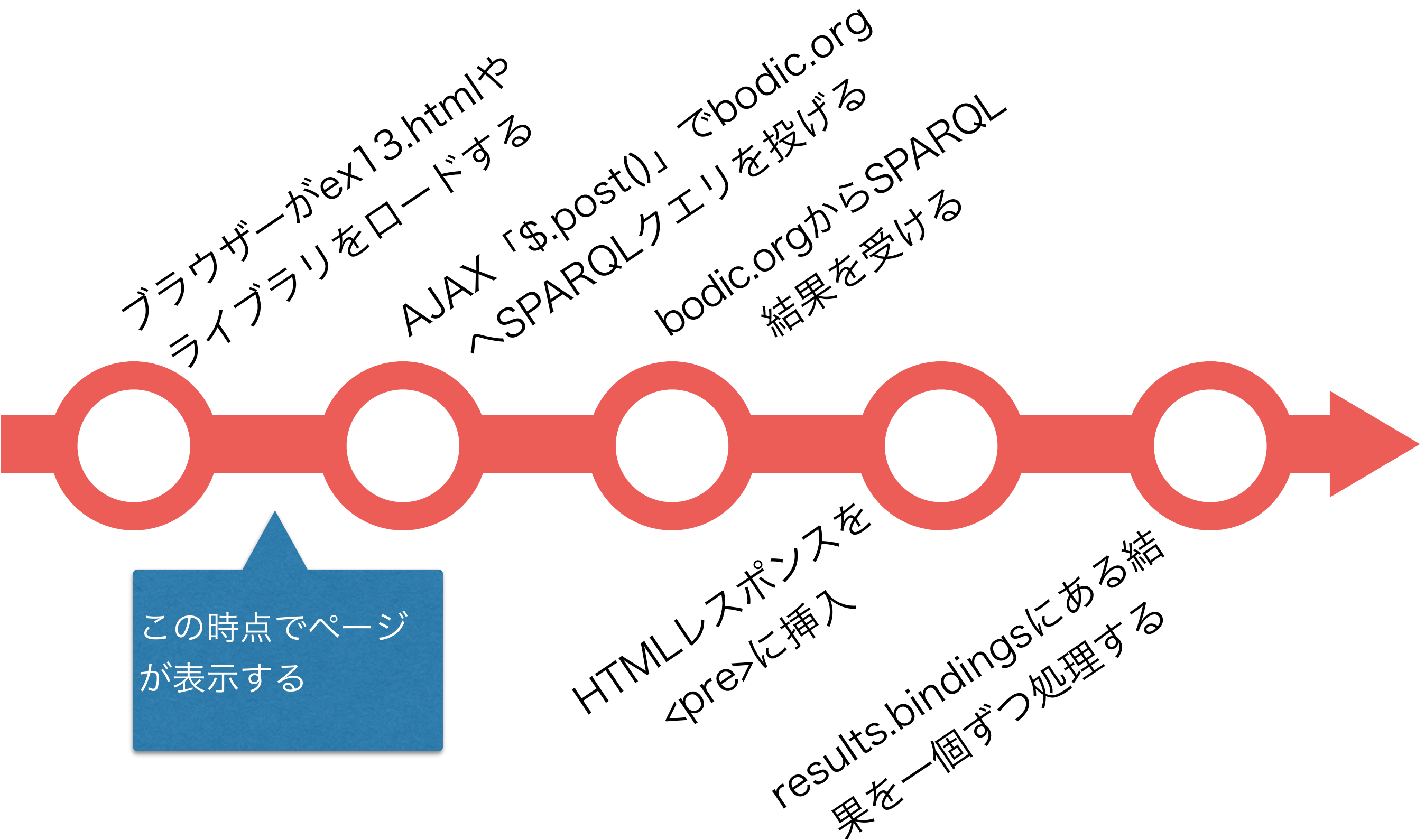
<pre>内容を挿入

細かいところは無視していいです

SPARQL結果処理部

forループを使って、結果を1
個ずつみて、緯度と経度を取
得し、文字列から数値へ変換
する (parseFloatで)

ex13/ex13.htmlの実行フロー



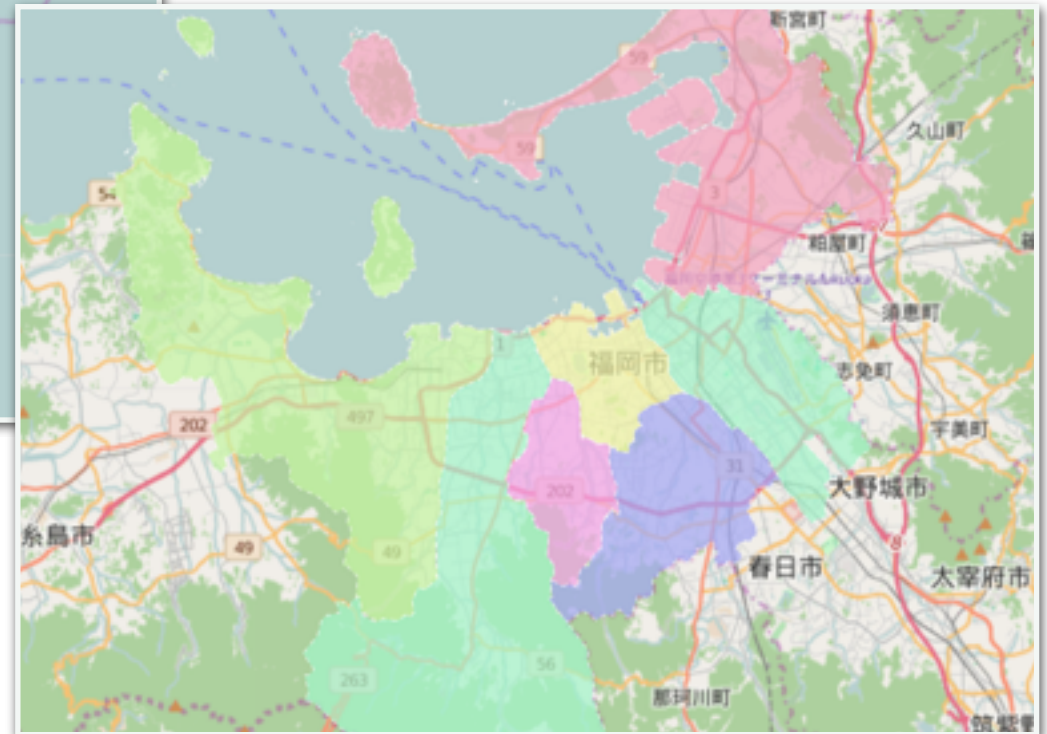
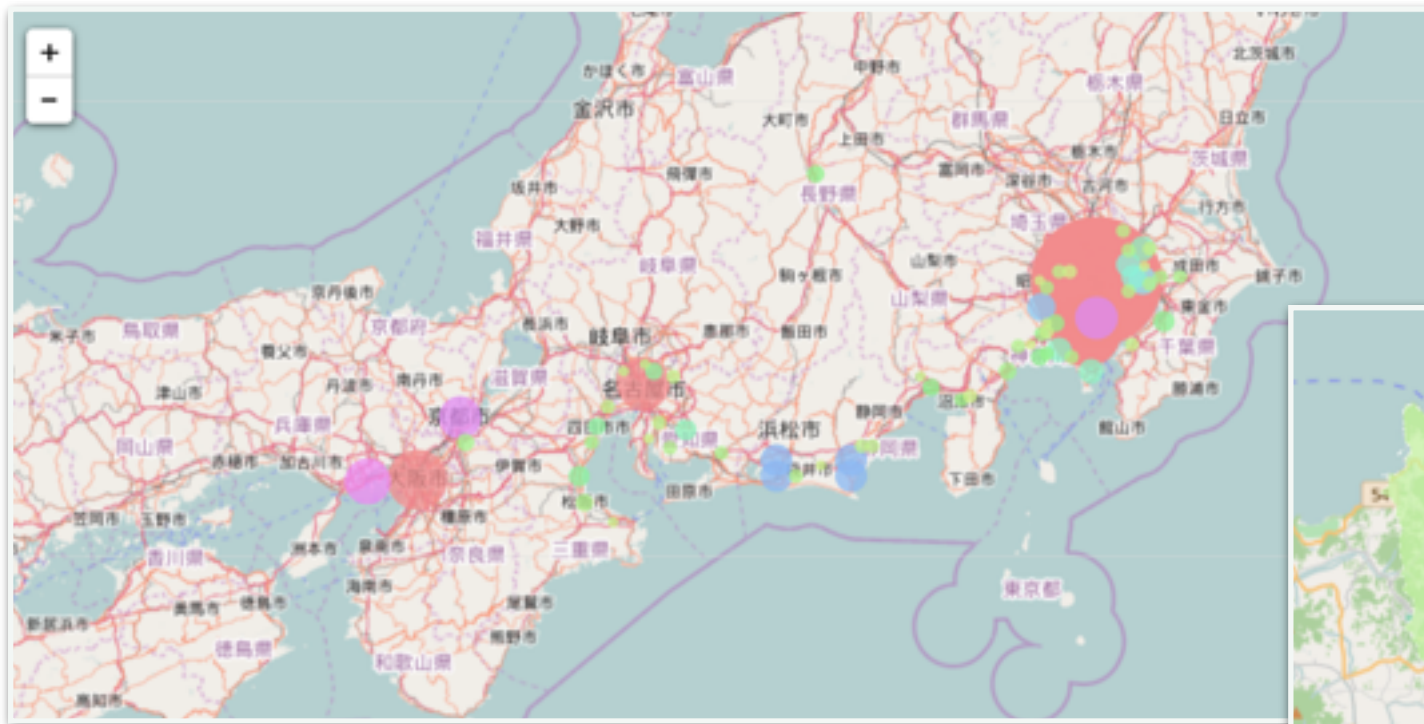
演習1-3：結果毎にマーカーを作りたい

Forループ
(iの値を変えて、{}の中を
繰り返した実行)

```
function(data, status, jqXHR) {  
    $("#sandbox").text("HTTP/1.1 " + jqXHR.status + " " +  
jqXHR.statusText + "\n" + jqXHR.getAllResponseHeaders() +  
"\n" + jqXHR.responseText)  
    var bindings = data.results.bindings;  
    for(var i=0; i<bindings.length; i++){  
        var b = bindings[i]  
        var lon = parseFloat(b.lon.value)  
        var lat = parseFloat(b.lat.value)  
        var sensor_type = b.sensor_type.value  
        // 問題:ここにマーカー導入コードを入れよう  
    }  
}
```

ここ! 演習1-1か1-2で勉強したやり方でマーカーと作る。
Forループ内ですので、SPARQL結果毎に実行する

演習2. 地図上にデータを を表示する



演習2-1:日本の主な都市の人口を円形で表して、地図上で可視化しましょう！



ex21/ex21.htmlの説明

ファイル構成はex21と同じですが、

- ・ 違うSPARQLエンドポイントを利用する (bodic.orgの代わりにdbpediaを利用)
- ・ SPARQLクエリーが違う ↓ ↓ ↓

```
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
prefix dbpedia:<http://dbpedia.org/resource/>
prefix dbpedia-p:<http://dbpedia.org/property/>
prefix dbpedia-o:<http://dbpedia.org/ontology/>
prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>
select ?name ?pop ?lon ?lat {
  ?s a dbpedia-o:City ;
  dbpedia-o:timeZone dbpedia:Japan_Standard_Time ;
  rdfs:label ?name ;
  dbpedia-p:populationTotal ?pop ;
  geo:long ?lon ;
  geo:lat ?lat .
  FILTER(lang(?name)="ja" && ?pop>100000 )
} ORDER BY DESC(?pop)
```

日本の全ての都市の名前と
人口を取得する

細かいところはダークマジック

人口が10万を超えてる都
市に限定し、順序を超える

Leafletで円形を表示する

Leafletでマーカーだけでなく、形も書ける。円形の作り方は下記の通り：

マーカーのパラメータに加えて、半径も指定する（単位：メートル）

```
> var circle = L.circle([lat, lon], circle_radius, {  
  fillColor: circle_fill_color,  
  color: false,  
  fillOpacity: 0.75  
})  
> circle.addTo(map)
```

マーカーと同様なので、細かい説明を省略する

マーカーのように円形にポップアップを付加できる

```
> circle.bindPopup("hoge")
```

SPARQL結果処理コードの説明

\$.post()のパラメーター関数

Forループ

```
for(var i=0; i<bindings.length; i++){
  var b = bindings[i]
  var name = b.name.value
  var pop = parseInt(b.pop.value)
  var lon = parseFloat(b.lon.value)
  var lat = parseFloat(b.lat.value)

  // 問題:円形の面積を人口に比例するようにしてください
  var circle_radius = 10000
  // 問題:色も人口に応じて変動してください
  var circle_fill_color = 'red'

  var circle = L.circle([lat, lon], circle_radius, {
    fillColor: circle_fill_color,
    color: false,
    fillOpacity: 0.75
  }).addTo(map)
  .bindPopup("<h3>" + name + "</h3>人口: " + pop + "人");
}
```

クエリー結果変数
値取得

緯度経度は数値に変換

円形や色を宣言
編集箇所

円形作成と
ポップアップ付加

HTML/CSS色指定について

方法1：色の名前から

```
> var color = 'red'
```

利用可能色は「http://www.w3schools.com/html/html_colors.asp」をご参考

方法2：rgba値から

```
> var color = 'rgba(red,green,blue,alpha)'
```

red,green,blueは0～255
alphaは0～1（透明度）

方法2：hsl値から

```
> var color = 'hsla(hue,saturation,lightness,alpha)'
```

Hue：色相

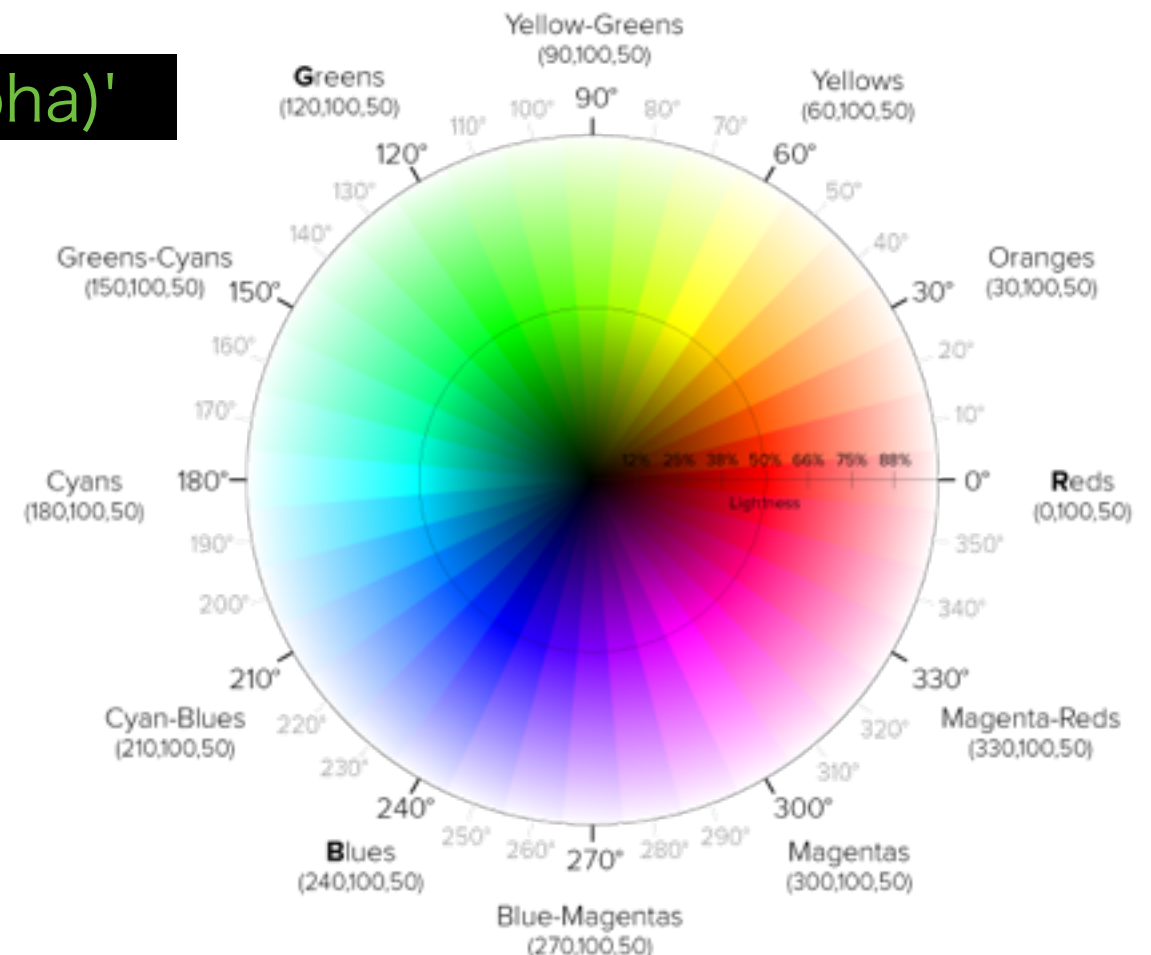
0～360度

Saturation：彩度

0%～100%

Lightness：バールール

0%～100%



JavaScriptで色指定文字列の作り方について

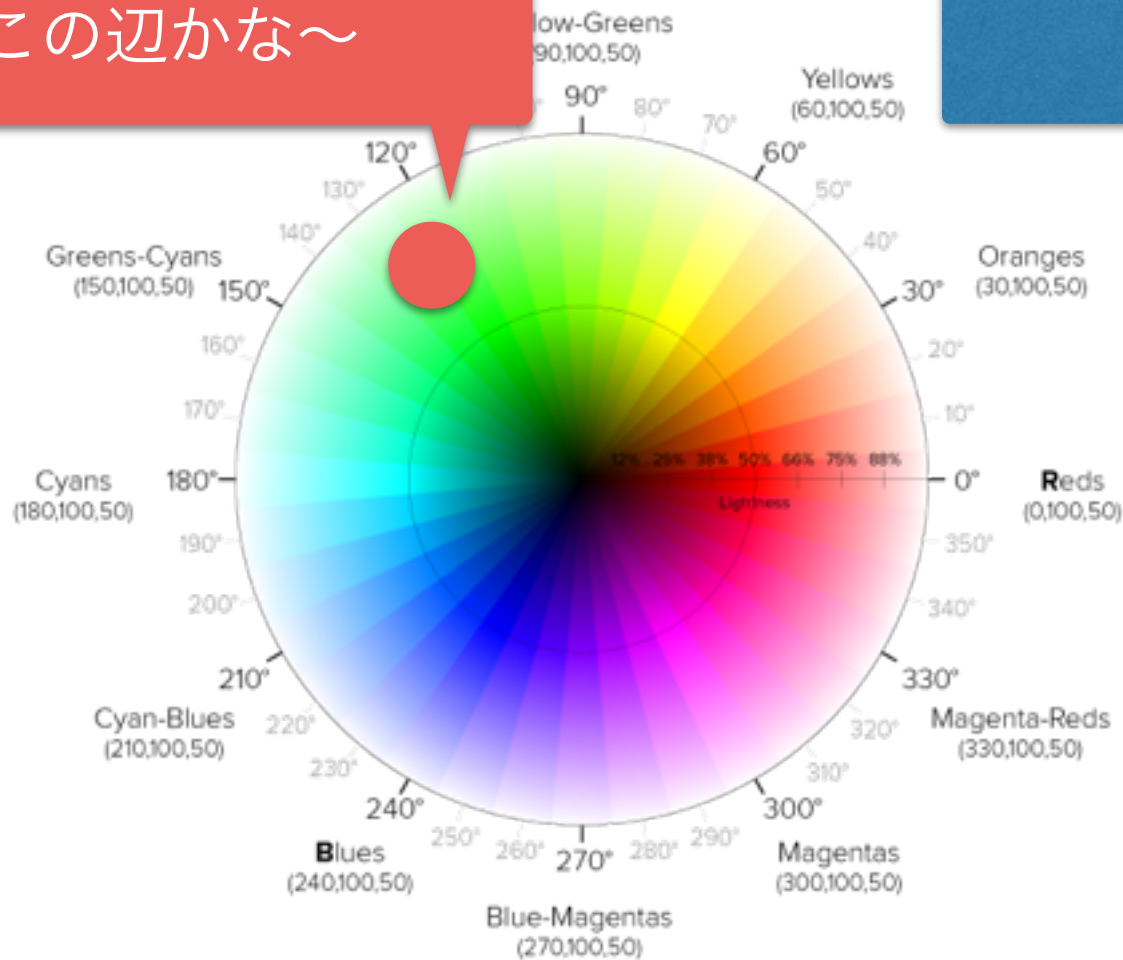
JavaScriptで変数から文字列を作れる

colorの値は「hsla(123,50%,80%,0.8)」

```
> var hue=123, saturation=50  
> var color = 'hsla('+hue+', '+saturation+'%,80%,0.8)'
```

この辺かな～

文字列を足し算をかけると
連結できる



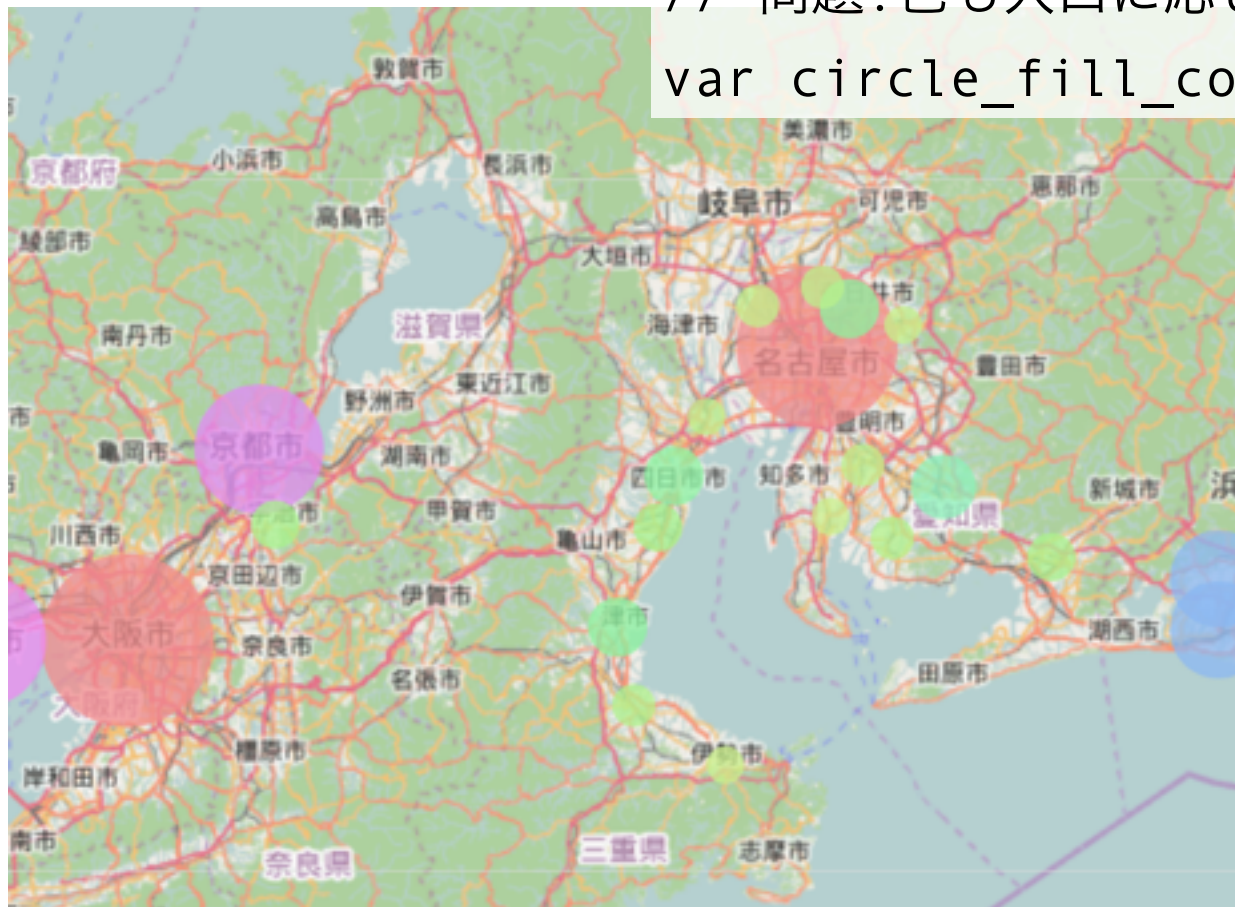
演習2-1：SPARQL結果毎に都市の位置に半径が人口に比例する円形を表示する

// 問題:円形の面積を人口に比例するようにしてください

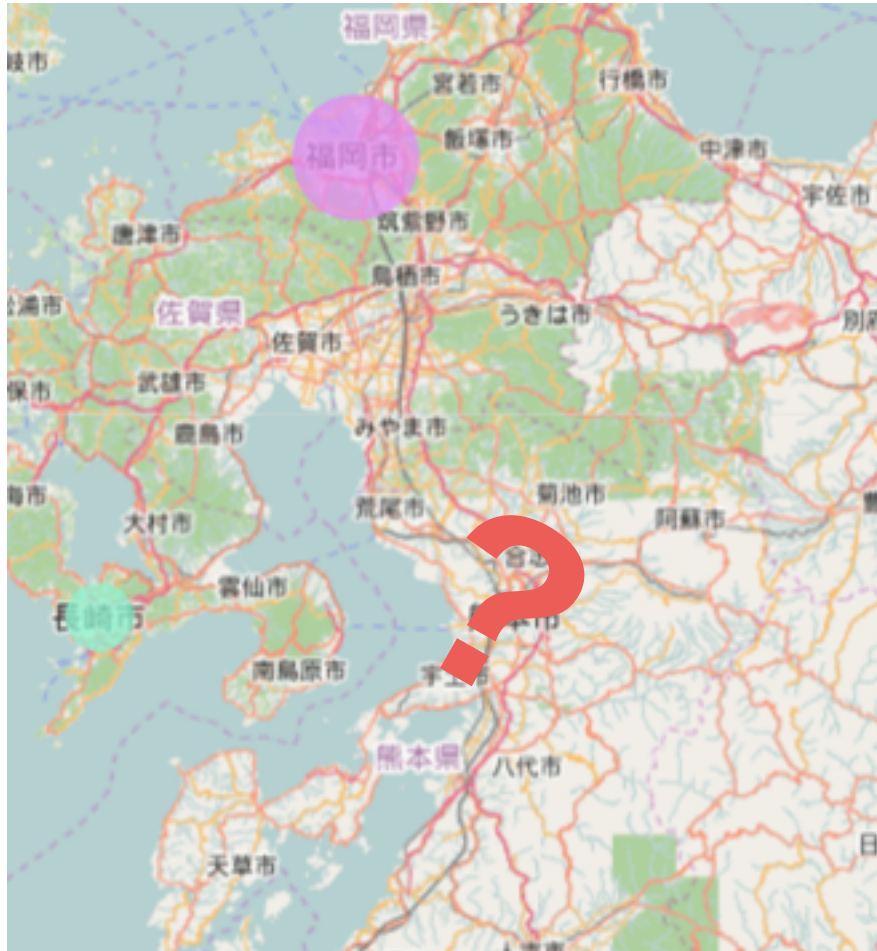
```
var circle_radius = 10000
```

// 問題:色も人口に応じて変動してください

```
var circle_fill_color = 'red'
```



ちなみに



いない！

dbpediaでもなかなか語彙が揃っていないので、
熊本の人口は他の都市と違う語彙をつかっている

演習2-2:ヒートマップを作しましょう！



ex22/ex22.htmlの構成説明

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
...
```

```
</head>
```

```
<body>
```

```
<h1>演習2-2: ヒートマップを作しましょう！</h1>
```

```
<div id="map"></div>
```

```
<script type="text/javascript">
```

```
var map = L.map('map').setView([33
```

```
// 問題: 前の演習に基づいて、地図を表示する
```

```
var fukuoka_wards_population = { ... }
```

```
function fukuoka_wards_style(feature){ ... }
```

```
function add_popup_on_ward_layer(feature, area_layer) { ... }
```

```
$.getJSON(..., function(japanData){  
  L.geoJson(...).addTo(map);  
})
```

```
</script>
```

```
</body>
```

```
</html>
```

今回はSPARQLを使わずに直接データを
JavaScript配列として記入

区のエリア毎に実行す
る関数（後で説明）

区のエリア毎に実行す
る関数（後で説明）

AJAXで区のエリア
データを取得

GeoJSON形式で地理データ（エリア） を記述できる

featuresはエリアの配列

features[i].propertiesは各
フィーチャーのメタデータ

区の名前は「N03_004」
というプロパティ

features[i].geometryは各
フィーチャーのエリアデータ
（緯度・経度で表現）

フィーチャー1

フィーチャー2

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": { "N03_001": "福岡県", "N03_002": null, "N03_003": "福岡市", "N03_004": "西区",
"N03_007": "40135" },
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [130.03217153900005, 33.864072038000074],
            [130.03311353800007, 33.86517603800007],
            [130.0331045380001, 33.865209038000046],
            ...
          ]
        ]
      }
    },
    {
      "type": "Feature",
      "properties": { "N03_001": "福岡県", "N03_002": null, "N03_003": "福岡市", "N03_004": "東区",
"N03_007": "40131" },
      "geometry": {
        "type": "Polygon",
        "coordinates": [ ... ]
      }
    },
    ...
  ]
}
```


GeoJSON処理について

③GeoJSONファイルにあるフィーチャー毎に

「L.geoJson」が下記の関数を呼ぶ：

- ・ fukuoka_wards_population：エリアの見た目を計算する
- ・ add_popup_on_ward：その他の処理（今回はポップアップを付加加する）

```
var fukuoka_wards_population = {
```

```
function fukuoka_wards_style(feature) {
  var ward_name = feature.properties.name;
  var ward_population = fukuoka_wards_population[ward_name];
  var ratio = (ward_population - 150000) / 150000;
  var color = 'hsl(' + 360 * ratio + ', 80%, 80%)';
  console.log(color);

  return { fillColor: color, weight: 0.5, ... };
}
```

```
function add_popup_on_ward_layer(feature, area_layer) {
  area_layer.bindPopup("ほげ");
}
```

①まず、AJAX経由で
GeoJSONを取得する

```
$.getJSON('https://raw.githubusercontent.com/niiyz/JapanCityGeoJson/master/geojson/福岡県/福岡市.json', function(japanData){
```

```
  L.geoJson(japanData, { style: fukuoka_wards_style, onEachFeature:
    add_popup_on_ward_layer }).addTo(map);
})
```

②AJAXレスポンスが来次第、LeafletのGeoJSON処理関すを呼び出す

Leafletエリアのスタイル設定について

- StyleはJavaScriptスタイルオブジェクトかJavaScriptスタイルオブジェクトを返す関数
- スタイルオブジェクトの利用可能メンバーはSVGプロパティの全て

```
L.geoJson(japanData, {  
  style: fukuoka_wards_style,  
  onEachFeature: add_popup_on_ward_layer  
})
```

スタイル関数のパラメータはGeoJSONのフィーチャーのみ (JavaScriptオブジェクトとして)

```
function fukuoka_wards_style(feature){  
  var ward_name = feature.properties['N03_004']  
  var ward_population = fukuoka_wards_population[ward_name]  
  var ratio = (ward_population-150000) / 150000  
  var color = 'hsl('+360*ratio+',80%,80%)'  
  console.log(color)  
  
  return { fillColor: color, weight: 0.5, ... };  
}
```

```
"features": [  
  {  
    "type": "Feature",  
    "properties": { ... },  
    "geometry": {  
      "type": "Polygon",  
      "coordinates": [ ... ]  
    }  
  },  
]
```

GeoJSON
ファイル

演習2-2：地図タイル表示とポップアップの内容変更

問題1：演習1-1などのコードに基づいて、地図タイルを表示するようにする

```
var map = L.map('map').setView([33.5833, 130.4000], 11);  
// 問題: 前の演習に基づいて、地図を表示するようにしてください
```

ここ！map変数の定義のあと

問題2：ポップアップの内容を「ほげ」からフィチャのデータにする

```
function add_popup_on_ward_layer(feature, area_layer) {  
  // 問題: 「fukuoka_wards_style」に基づいて、...  
  area_layer.bindPopup("ほげ")  
}
```

ここ！ほげをどうにかする

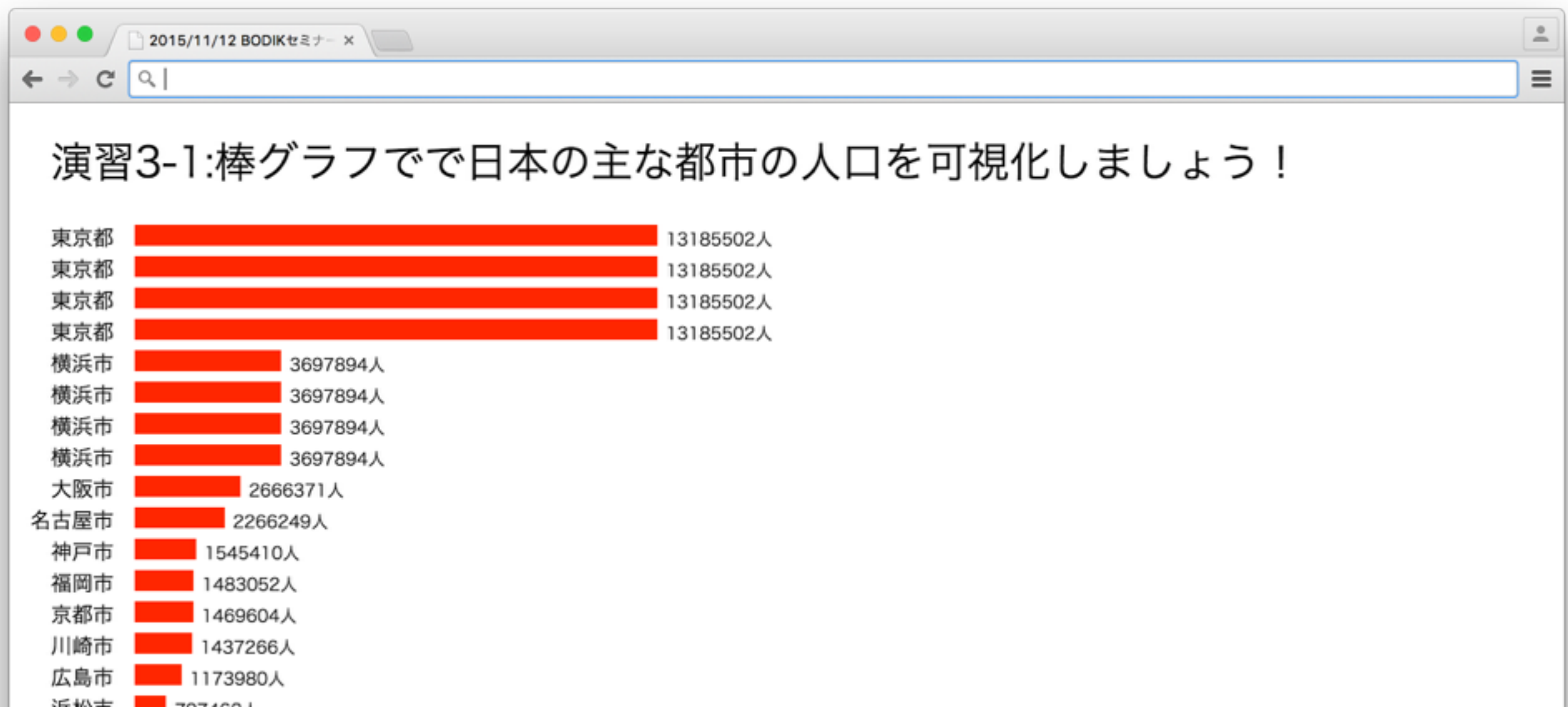
サンプル：



遊び：色などを変更して下さい



演習3. D3.jsによるグラフ作成の基本



ex31/ex31.htmlの構成説明

ベースは演習2-1の
ファイルです

```
<!--
<h1>
<h2>
...
<style>
...
#graph { width: 700px; padding: 0; margin: 0; }
...
</style>
</head>
```

グラフ用のCSSルール
を追加

```
<body>

<h1>演習3-1: 棒グラフで日本の主な都市の人口を可視化しましょう！</h1>

<div id="graph"></div>
<pre id="sandbox"></pre>
```

```
<script type="text/javascript">
```

```
var sparql_query = [...].join("\n")
```

```
$.post(
  'http://dbpedia.org/sparql?format=json',
  { query: sparql_query },
  function(data, status, jqXHR) { ... }
).fail(function(jqXHR) {
  $("#sandbox").text("HTTP/1.1 " + jqXHR.status + " " + jqXHR.statusText + "\n" +
jqXHR.getAllResponseHeaders() + "\n" + jqXHR.responseText)
})
```

AJAXコールが成功した時の処理関
数：グラフ作成

```
</script>
```

```
</body>
```

```
</html>
```


DOMエレメントでグラフ作成の基本

アイデア：HTMLの通常<div>エレメントの幅をデータに基づいて
変えることによって棒グラフを作る

静的サンプルHTMLコード

```
<!DOCTYPE html>

<html>
<head>
  <style>
    .bar { height: 20px; background-color: red; margin: 10px; }
  </style>
</head>

<body>

  <div class="bar" style="width:300px"></div>
  <div class="bar" style="width:250px"></div>
  <div class="bar" style="width:100px"></div>

</body>

</html>
```

このwidth(幅)をJavaScriptで
データに基づいて設定すればいい



D3.jsでデータを元にDOMオブジェクトのプロパティを手軽に編集できる



D3.jsがやってくれること

データの数に合わせてDOMエレメントの生成・削除

便利なスケール関数（サイズ、色など）

軸の生成

様々な便利関数（データ集計など）

ex31/ex31.htmlのグラフ作成コード

```
function(data, status, jqXHR) {  
    $("#sandbox").text("HTTP/1.1 " + jqXHR.status + " " + jqXHR.statusText + "\n" +  
    jqXHR.getAllResponseHeaders() + "\n" + jqXHR.responseText)
```

演習1-3と同様：HTTPレスポンスを<pre>に挿入

```
var bindings = data.results.bindings;  
var data = []
```

```
for(var i=0; i<bindings.length; i++) {  
    var b = bindings[i]  
    var pop = parseInt(b.pop.value)  
    var name = b.name.value  
    data.push({pop: pop, name: name})  
}
```

データをsparql+json形式からより簡単な形式に変換し、「data」という配列に格納

```
var max_population = d3.max($.map(data, function(d) {return d.pop} ))
```

```
var scale_x = d3.scale.linear()  
    .domain([0, max_population])  
    .range([0, 400]);
```

横軸のスケールの定義。スケールは人口データからピクセルへ変換するための関数：

```
var div = d3.select("#graph")  
    .selectAll("div")  
    .data(data)  
    .enter()  
    .append("div")
```

domain：入力幅（人口データ）

range：出力幅（ピクセル）

```
div.append("p")  
    .attr("class", "label")  
    .text(function(d) { return d.name } )  
div.append("div")  
    .attr("class", "bar")  
    .style("width", function(d) { return scale_x(d.pop) + "px" } )
```

グラフ生成箇所。次のスライドに説明する・・・

```
// 問題：人口ラベルをバーの後ろに追加してください(クラスはnumeric-labelをご利用ください)
```

```
}
```


ex31/ex31.htmlのグラフ作成コード

```
var div = d3.select("#graph")
    .selectAll("div")
    .data(data)
    .enter()
    .append("div")

div.append("p")
    .attr("class", "label")
    .text(function(d) { return d.name })
div.append("div")
    .attr("class", "bar")
    .style("width", function(d) {
        return scale_x(d.pop) + "px" })
```

データ項目毎に<div>エレメントを生成

- ・ enter()はD3.jsの中心的な関数：data()関数に渡した配列の項目に合わせて<div>、selectAll()でセレクトしたエレメントの数を調節する

- ・ append("div")はデータ項目毎に実行される

追加した<div>エレメントの中身を埋める

- ・ <p>は都市の名前

- ・ <div>は棒グラフの棒

data配列

```
[
  {pop: 123456, name: "都市1"},
  {pop: 654321, name: "都市2"},
  ...
]
```

DOM

<div>

<p>

都市1

<div>

<div>

...

演習3-1 : 棒グラフにラベルを付加する



ヒント :

```
div.append("p")
    .attr("class", "label")
    .text(function(d) { return d.name } )
div.append("div")
    .attr("class", "bar")
    .style("width", function(d) {
        return scale_x(d.pop) + "px" } )
div.append(...).text(...)
```

演習はここまで
お疲れさまでした

D3.jsで他にできること

d3.jsのサンプリページをご参考

HTML DOMエレメントの代わりにSVGエレメントを使うとより複雑なグラフを作れる

The image shows a screenshot of the D3.js Visual Index gallery page. The page is titled "Visual Index" and displays a grid of 16 different chart types, each with a small thumbnail image and a title. The charts include: Box Plots, Bubble Chart, Bullet Charts, Calendar View, Non-contiguous Cartogram, Chord Diagram, Dendrogram, Force-Directed Graph, Circle Packing, Population Pyramid 2000, Stacked Bars, and Streamgraph. On the right side of the page, there is a sidebar with links to various resources, including "API 中文手册", "API Reference", "API Reference (русскоязычная версия)", "Api 参考", "Arrays", "Behaviors", "Bundle Layout", "Chord Layout", "Cluster Layout", "CN Home", "Colors", and "Core". At the bottom of the sidebar, there is a link to "Show 70 more pages...". The browser's address bar shows the URL "https://github.com/mbostock/d3/wiki/Gallery".

Visual Index

- Box Plots
- Bubble Chart
- Bullet Charts
- Calendar View
- Non-contiguous Cartogram
- Chord Diagram
- Dendrogram
- Force-Directed Graph
- Circle Packing
- Population Pyramid 2000
- Stacked Bars
- Streamgraph

API 中文手册
API Reference
API Reference (русскоязычная версия)
Api 参考
Arrays
Behaviors
Bundle Layout
Chord Layout
Cluster Layout
CN Home
Colors
Core
Show 70 more pages...

Clone this wiki locally
<https://github.com/mbostock/d3/wiki/Gallery>