



NAVITIME API チャレンジ ハンズオンセミナー

2019/7/3(水)
株式会社ナビタイムジャパン





14:00~ イベント概要 & 自己紹介

14:30~ 開発方法の説明 & 演習

15:30~ チーム毎の開発

17:00~ 懇親会

~18:00 解散予定



NAVITIME API チャレンジ

□ 目的

NAVITIME APIの利用を通じて交通に関する研究・開発に取り組んでいただき、業界全体の活性化を目指す

□ 賞金

★ 最優秀賞

¥1,000,000

★ 優秀賞

¥300,000

★ 特別賞

¥100,000

□ 今後の予定

10月 : 1次審査

12月10日 : 最終審査/

発表会・表彰式



APIを活用してモビリティ革命にチャレンジ！！

たとえば・・・

MaaSの研究開発



インバウンド観光
についての研究



場所と人の
最適なマッチング





自己紹介タイム



- 名前
 - (チームの場合) 各メンバーの名前
- 今日どこから来たか
- APIチャレンジを知ったきっかけ
- APIチャレンジの応募テーマ
(検討中の方は複数あげていただいてもOKです)
- ひとこと



Good & New

24時間以内にあった

“良かったこと” or “新しいこと”について
話しましょう





1. システム概要
2. ハンズオン概要
3. NAVITIME API
4. スポット検索
5. 地図表示
6. ルート線描画
7. デバッグ方法
8. 開発のアドバイス
9. (補足)
10. 開発タイム

ハンズオンの実装が終わったら、各チームで質問・相談があれば受け付けます。

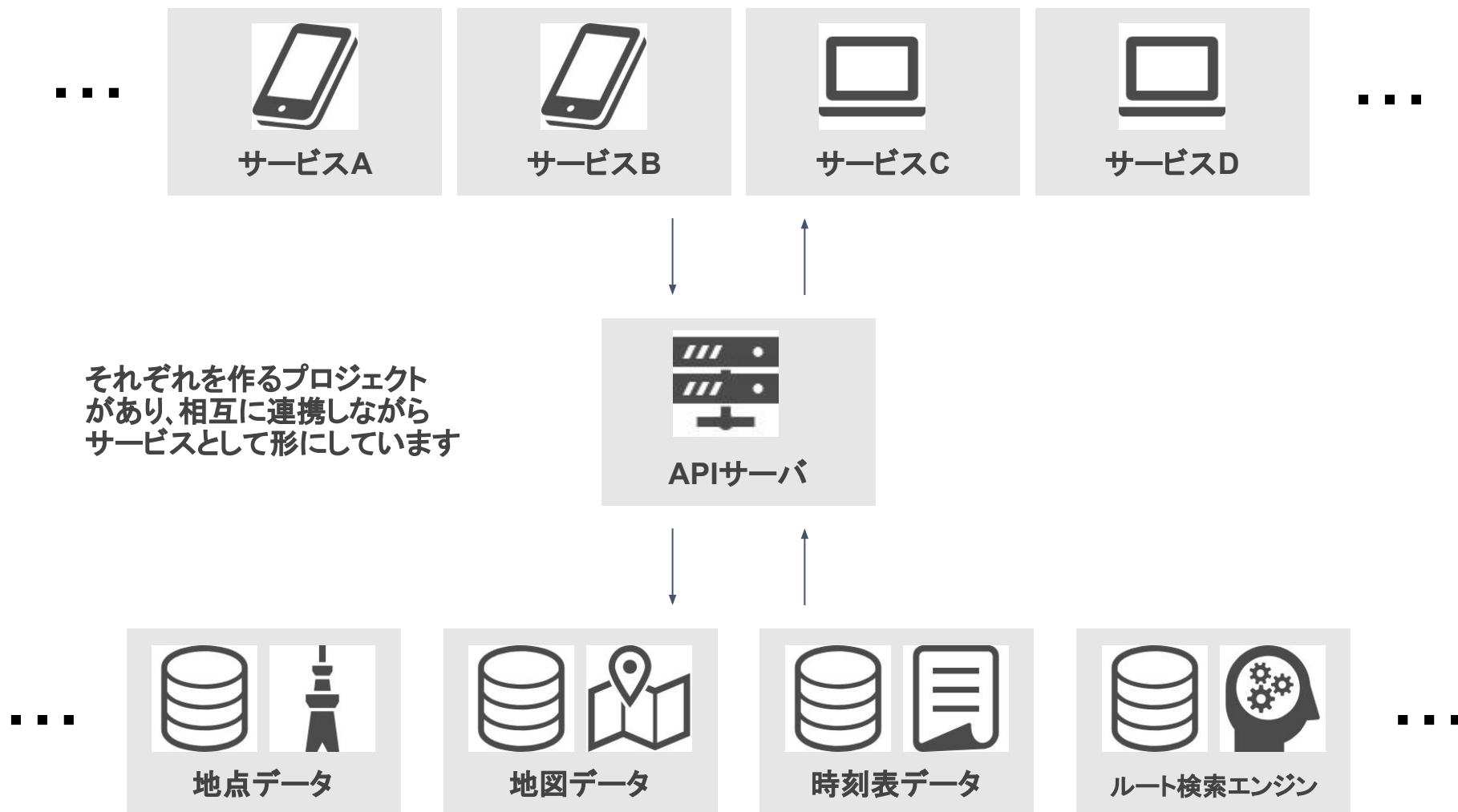


1. システム概要
2. ハンズオン概要
3. NAVITIME API
4. スポット検索
5. 地図表示
6. ルート線描画
7. デバッグ方法
8. 開発のアドバイス
9. (補足)
10. 開発タイム

ハンズオンの実装が終わったら、各チームで質問・相談があれば受け付けます。



各プロジェクトが連携しながらサービス開発





1. システム概要
2. ハンズオン概要
3. NAVITIME API
4. スポット検索
5. 地図表示
6. ルート線描画
7. デバッグ方法
8. 開発のアドバイス
9. (補足)
10. 開発タイム

ハンズオンの実装が終わったら、各チームで質問・相談があれば受け付けます。



NAVITIME API を利用して

経路検索サービスを開発してみましよう！



ハンズオン 開発イメージ

出発： 到着：





ハンズオン 開発イメージ

出発: ナビタイムジャパン 到着: 東京タワー

検索





ハンズオン 開発イメージ

出発： 到着：





ハンズオン 開発イメージ

出発：ナビタイムジャパン

到着：東京タワー

NAVITIME API を使って
地図上に **ルート線** を描く





ハンズオンが終了する頃には

- スポット検索
- 地図
- 経路検索

が、使えるようになっていきます！！！！

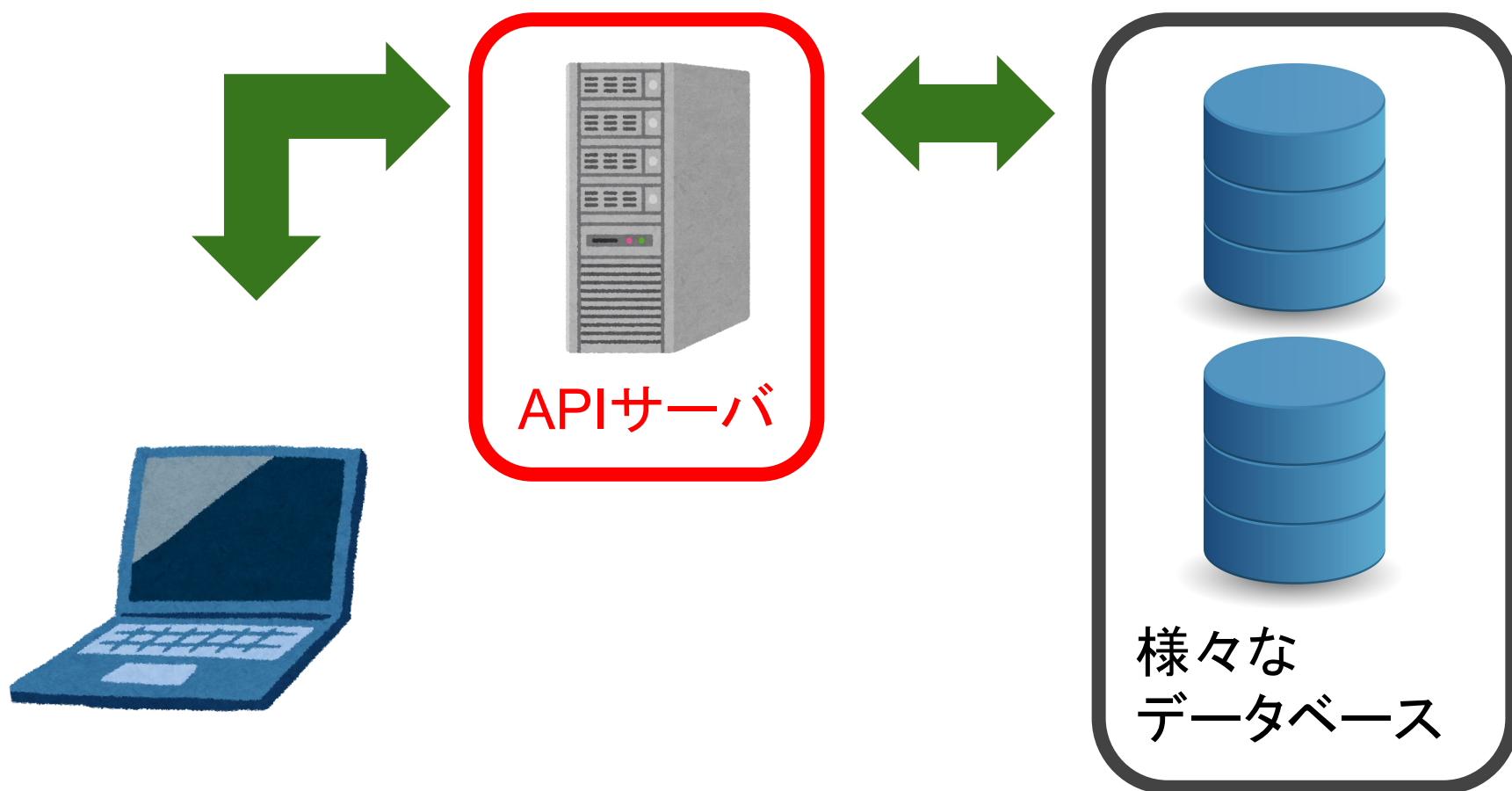


1. システム概要
2. ハンズオン概要
3. NAVITIME API
4. スポット検索
5. 地図表示
6. ルート線描画
7. デバッグ方法
8. 開発のアドバイス
9. (補足)
10. 開発タイム

ハンズオンの実装が終わったら、各チームで質問・相談があれば受け付けます。



- API(Web API)とは
 - インターネットを通じてデータを取得する方法



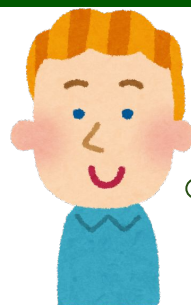


もう少し
わかりやすく...





例. レストラン



① オムライス食べたい！

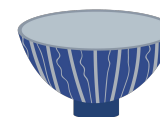
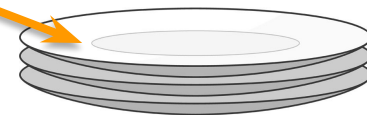
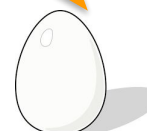
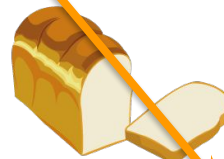


③ オムライスどうぞ！！



料理人 (= API)

② 器具と材料を使って調理





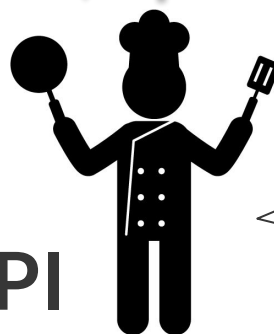
実際には...

表参道駅からナビタイムジャパン
までのルートが欲しい！

①リクエスト



③レスポンス



API

② 地点情報をデータからもらって
ルート検索エンジンで検索してもらって
...

駅データ

POIデータ

住所データ

ルート検索
エンジン

乗換検索
エンジン

時刻表計算
エンジン

タイル地図

地図注記



今回利用するのは...

- NAVITIME API
 - NAVITIME 独自の API
 - NAVITIMEの主要サービスを開発する上で必要な、スポットやルート情報を検索できる



NAVITIME API の リクエストとレスポンスについて 見ていきましょう。



- リクエストの作り方

- スポット検索を例に見てみましょう
- <https://api-challenge.navitime.biz/v1s/{SID}/spot/list?word=渋谷&datum=wgs84>

スポット検索を指定

どのようなスポットを
検索するか指定

- NAVITIME APIのURL

- 固定部 + API名 + パラメータ
- {SID}にはメールでお伝えしているSIDを設定してください



● レスポンス: 検索結果 (JSON形式)

```
- count: {  
  total: 45086,  
  offset: 0,  
  limit: 10  
},  
- items: [  
  - {  
    code: "02011-00003544",  
    - provider: {  
      id: "02011",  
      name: "ナビタイムジャパン"  
    },  
    name: "渋谷",  
    ruby: "しぶや",  
    address_name: "東京都渋谷区渋谷2丁目",  
    address_code: "13113012000",  
    - coord: {  
      lon: 139.702457,  
      lat: 35.659681  
    },  
  },  
]
```

取得できる情報

- **code**: スポットのID
- **provider**: 情報提供者
- **name**: スポット名
- **ruby**: スポット名の呼び名
- **address_name**: 住所
- **address_code**: 住所ID
- **coord**: 緯度経度



1. システム概要
2. ハンズオン概要
3. NAVITIME API
4. スポット検索
5. 地図表示
6. ルート線描画
7. デバッグ方法
8. 開発のアドバイス
9. (補足)
10. 開発タイム

ハンズオンの実装が終わったら、各チームで質問・相談があれば受け付けます。



スポット検索してみよう！





例) ボタンを押下するとスポットの情報を表示

例) 東京タワー

name(スポット名):
address(住所):
lat(緯度):
lon(経度):

イメージ

東京タワー

name(スポット名):
東京タワー
address(住所):
東京都港区芝公園4-2-8
lat(緯度):
35.658584
lon(経度):
139.745457



- mochaレスポンス内のスポット情報を表示

[mocha_spot_search.html](#)を見て、動作・書き方を確認してください。

```
script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

通信ライブラリ **axios** (アクシオス) を読み込み

```
<script>
```

```
let baseUrl = 'https://api-challenge.navitime.biz/v1s/19999993'
```

```
function search() {
```

```
  axios
```

```
    .get(baseUrl + '/spot/list?word=渋谷&datum=wgs84')
```

```
    .then(connectSuccess)
```

```
    .catch(connectFailure)
```

```
}
```

get() : リクエストしたいURL

then() : 通信成功時に実行する関数

catch() : エラー発生時に実行する関数

```
function connectSuccess(response) {
```

```
  let spot = response.data.items[0];
```

```
  $('#name').append(spot.name);
```

```
}
```

```
function connectFailure(error) {
```

```
  alert('failure');
```

```
}
```

```
</script>
```

最初のスポットを抽出し `spot` に格納
スポット名を画面に表示



詳しくは...

[NAVITIME API仕様書](#) を参考にしてください。

確認できるもの

- リクエストパラメータの指定方法
- レスポンスの出力形式

【補足】

リクエストパラメータには

datum=wgs84

を付けましょう。

(一部、付ける必要のないAPIもあります)

datum : 測地系

datum=tokyo : 日本測地系を指定



渋谷駅の緯度経度を指定した際に
測地系の影響でピンがずれることがあります。

datum=wgs84 : 世界測地系を指定



Proprietary & Confidential

Copyright (C) 2017 NAVITIME JAPAN Co., Ltd. All rights reserved.



1. システム概要
2. ハンズオン概要
3. NAVITIME API
4. スポット検索
5. 地図表示
6. ルート線描画
7. デバッグ方法
8. 開発のアドバイス
9. (補足)
10. 開発タイム

ハンズオンの実装が終わったら、各チームで質問・相談があれば受け付けます。

流れ

1. 地図を表示する領域を定義
2. mapscript を使用して地図を表示





1. 地図を表示する領域を定義

[sample_map.html](#) を見て、動作・書き方を確認してください。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <style>
      #map {
        width: 800px;
        height: 450px;
        background-color: black;
      }
    </style>
  </head>

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

`id="map"` の
高さ、幅、背景色を指定

地図を表示する領域を確保

2. mapscript を使用して地図を表示

sample_map.html を見て、動作・書き方を確認してください。

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

  <script
src="https://api-challenge.navitime.biz/v1s/19999993/mapscript?version=3.4&host=localhost">
</script>

  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <script>
    function main() {
      let center = new navitime.geo.LatLng('35.689614', '139.691634')
      let map = new navitime.geo.Map('map', center, 15)
    }
    window.onload = main
  </script>
</body>
```

mapscript を読みこみ

地図の中心緯度経度を定義

ズームレベル(地図の縮尺)

id="map" のdivタグに地図を表示

- 他にもさまざまな機能があります

- ピンを立てる
- 吹き出しを表示するなど



- 詳しくは[地図仕様書](#)をご確認ください



1. システム概要
2. ハンズオン概要
3. NAVITIME API
4. スポット検索
5. 地図表示
6. ルート線描画
7. デバッグ方法
8. 開発のアドバイス
9. (補足)
10. 開発タイム

ハンズオンの実装が終わったら、各チームで質問・相談があれば受け付けます。

流れ

- 1, NAVITIME API からルート線の情報を取得
- 2, NAVITIME API のレスポンスを利用して地図上に線を描画





NAVITIME API からルート線の情報を取得

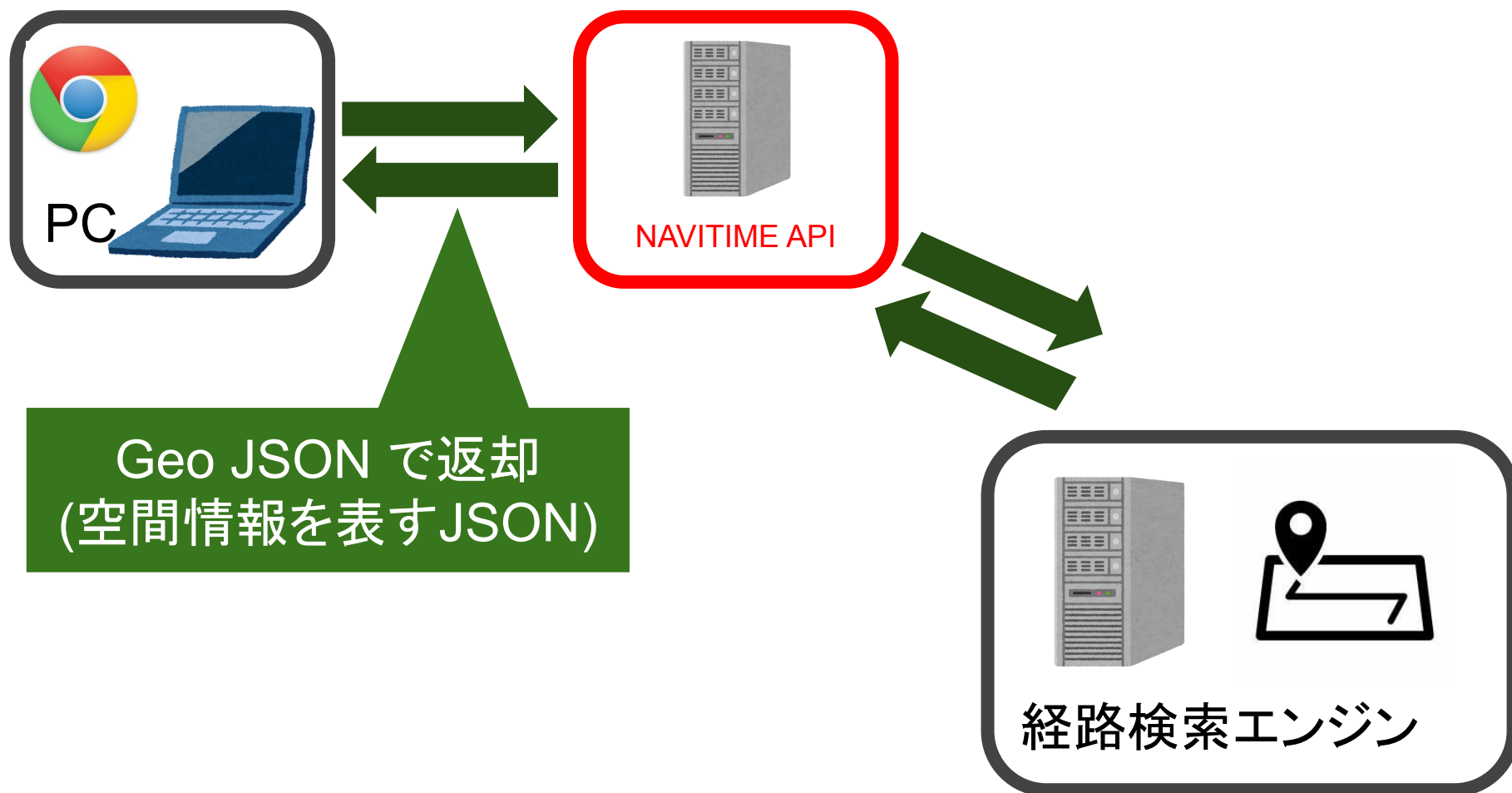
- [/route/shape](#) APIを利用します。

サンプルリクエスト

- パラメータ
 - start=35.658584,139.745457: 出発地=緯度経度
 - goal=35.667395,139.714896: 目的地=緯度経度
 - add=transport_shape: 追加情報=路線形状
 - shape-color=railway_line: 色分け情報=鉄道路線の色分け
 - datum=wgs84: 測地系=世界測地系



ルート線情報を取得する際は下記のような構成



[sample_route_shape.html](#) を見て、動作・書き方を確認してください。

```
function search() {
  let url = 'https://api-challenge.navitime.biz/v1s/19999993/route/shape?start=35.658584,139.745457
            &goal=35.667395,139.714896&add=transport_shape&shape-color=railway_line&datum=wgs84'
```

```
  axios
    .get(url)
    .then(connectSuccessRouteShape)
    .catch(connectFailure)
}
```

通信はスポット検索の時と同じです

```
function connectSuccessRouteShape(response) {
  route = response.data
```

```
  renderer = new navitime.geo.route.Renderer(route, {
    map: map,
    unit: 'degree',
    allRoute: true,
    arrow: true,
    originalColor: true,
  });
```

ルートを作成

第一引数に **NAVITIME API から取得したデータ** を指定

map : 地図表示 (p.38) で指定した map 要素

allRoute : すべてのルートを表示する場合は true

arrow : ルート線路上に進行方向の矢印を表示する場合は true

originalColor : 路線形状に指定された元々の色を利用する場合は true

```
  renderer.draw();
}
```

```
function connectFailure(error) {
  console.error(error);
}
```

ルートを描画



1. システム概要
2. ハンズオン概要
3. NAVITIME API
4. スポット検索
5. 地図表示
6. ルート線描画
7. **デバッグ方法**
8. 開発のアドバイス
9. (補足)
10. 開発タイム

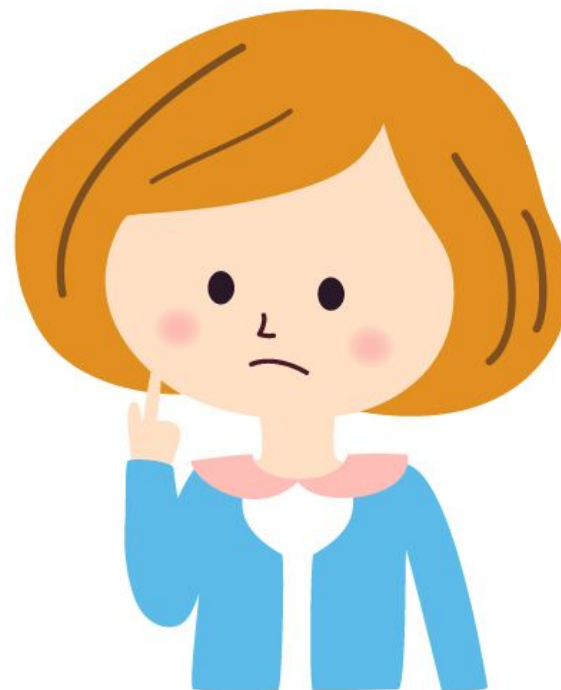
ハンズオンの実装が終わったら、各チームで質問・相談があれば受け付けます。



開発を始める前に
デバッグ方法について
学びましょう！！！！



なんのためにデバッグするの...？





デバッグで確認できること

- なんだか上手く動いてくれない
→ おかしな値が入っていないか、エラーが出ていないか確認できる
- APIにどんなリクエストを送っているかみたい
→ リクエストが上手くいったか、どのような情報が取れたかを確認できる
- 画面の見た目が崩れてしまう
→ デザインの設定がどのように効いているか確認できる

**目に見えないプログラムの挙動を
調査・確認することが出来ます！！！！**



debug_practice.html ファイルをブラウザ上で開いてください。

デバッグ練習

デバッグ練習の手順は下記の通りです。
わからない、話においてかれた場合は近くの社員に聞いてください。

1. 画面上のどこかで右クリック
2. 検証を選択
3. 出てきた検証画面内の**Sources**をクリック
4. 現在開いているファイルを指定
5. 33行目 `console.log("クリック回数：" + count);` に**ブレークポイント**をつくる
(行番号をクリックして数字が青く選択されればOK)
6. 下のボタンをクリックしてみましょう

クリックしたらポップアップ表示をします。
7. **Console**に `count` と入力し、Enterキーを押しましょう
8. 整数が表示されることを確認
9. 処理をひとつ進めましょう
10. **Console**に クリック回数: X と表示されていることを確認しましょう
11. 処理を最後まで進めましょう
12. ポップアップが出てくることを確認

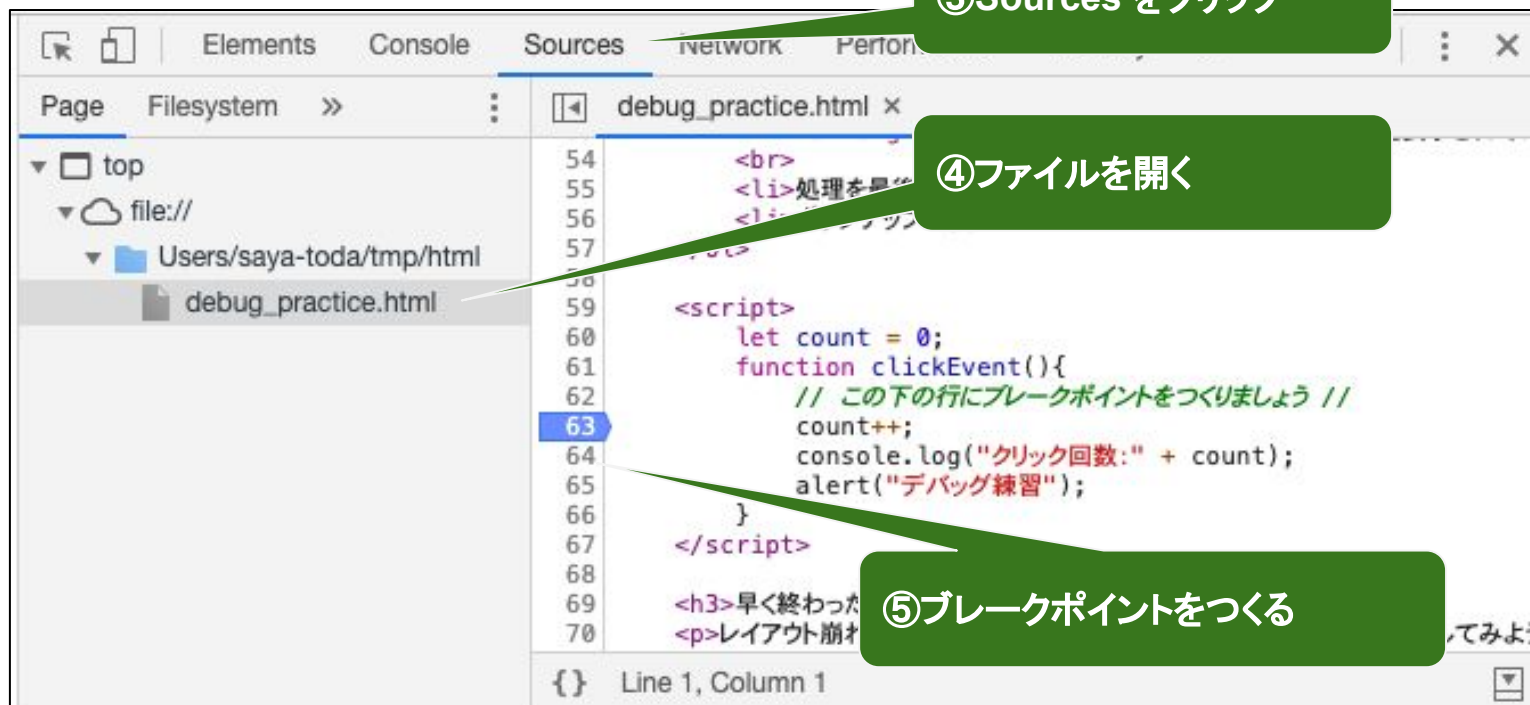


①ブラウザ上で 右クリック

②検証 を選択

③Sources をクリック

④ファイルを開く



⑤ブレークポイントをつくる

下のボタンをクリックしてみましょう

クリックしたらポップアップ表示をします。

⑥ボタンをクリック



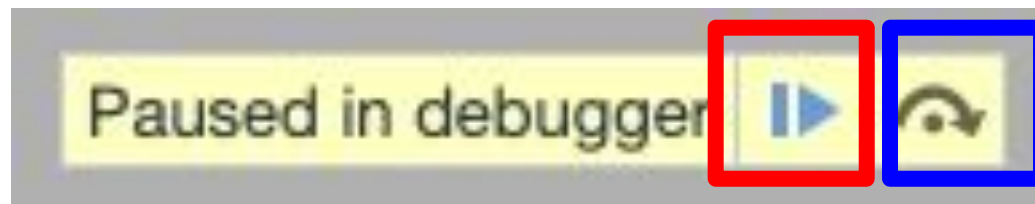
⑦count と入力してEnterキー

⑧変数の中の値が表示される

⑨処理をひとつ進める

⑩Consoleに表示

debug_practice.html:64



ひとつ下の行に進める

次のブレークポイントまで進める

※ブレークポイントが無ければ最後まで



レイアウトの確認をする



ここをクリック

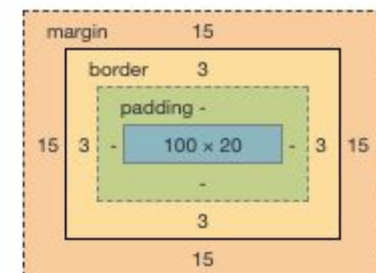
ブラウザ上のHTMLタグ要素の情報が見られる

要素をクリックするとstyleの設定なども見られる

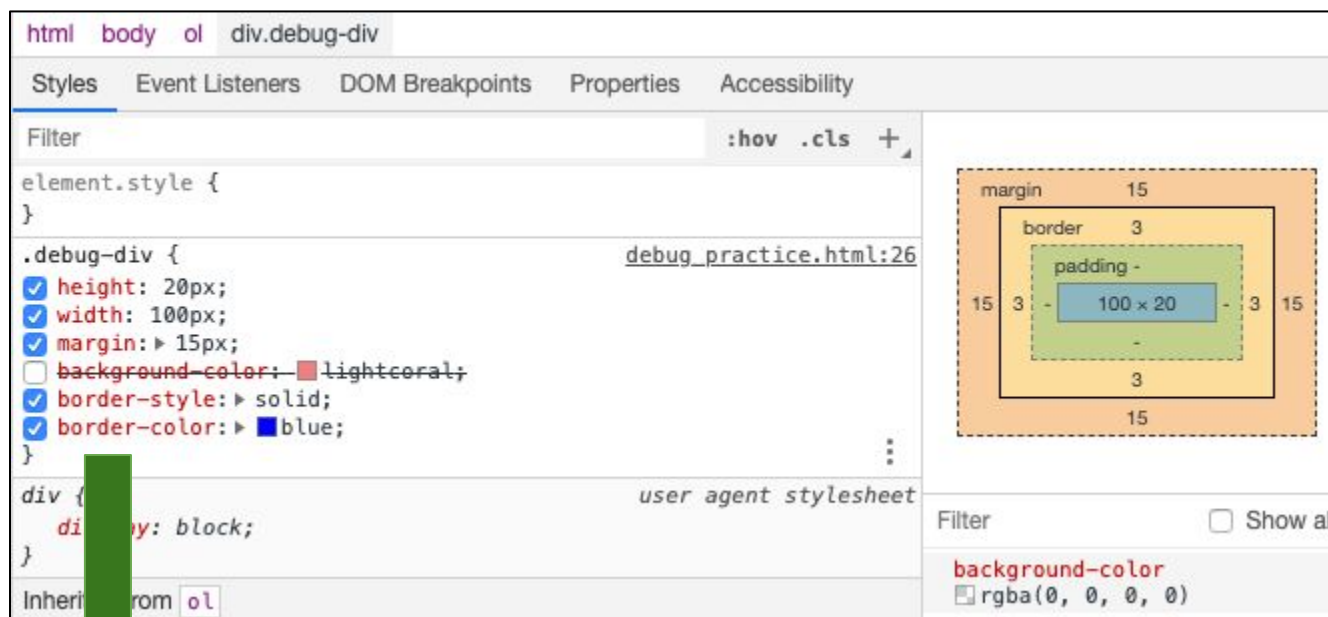
3. `div.debug-div` 106 x 26
4. Background ■ #F08080
5. Margin 15px

6. `.debug-div` の`{}`の中にcssの記載をしてみよう

```
element.style {  
  }  
  
.debug-div {  
  height: 20px;  
  width: 100px;  
  margin: 15px;  
  background-color: lightcoral;  
  border-style: solid;  
}  
  
div {  
  display: block;  
}  
  
Inherited from ol  
ol {  
  }
```



Filter ☐ Show all
▶ background-color
■ rgb(240, 128, 128)



3. 左上の矢印をクリック
4. その状態で各htmlタグ要素の上にカーソルを移動
5. その状態で下の箱をクリック



6. .debug-div の[]の中にcssの記載をし

border-color: blue; を追記

7. 枠線の色が変わったらok

styleの設定を追加/削除して
そのままブラウザ上で確認することも可能です。
※設定はファイルに保存されないので注意



1. システム概要
2. ハンズオン概要
3. NAVITIME API
4. スポット検索
5. 地図表示
6. ルート線描画
7. デバッグ方法
8. 開発のアドバイス
9. (補足)
10. 開発タイム

ハンズオンの実装が終わったら、各チームで質問・相談があれば受け付けます。



● 開発の流れ

- 地図を置く箱をつくる
- **APIを叩いて 地図** を表示
- 検索窓を作る
- APIを叩いて **スポット情報** を取ってくる
- 取ってきたスポット情報を利用して /route/shape のリクエストを作る
- APIを叩いて **ルート線** を描画

↑ ここまでを目標に頑張りましょう！！！！ ↑

※ ハンズオンでやったことをコピーして上手く組み合わせましょう



ハンズオン 開発イメージ

出発: ナビタイムジャパン 到着: 東京タワー

検索





地図描画用サンプルコード

<https://ntj.app.box.com/s/wdrwab9vvko3yd4enw39fxwzumgyxc2i/file/478815703676>

スポット検索用サンプルコード

<https://ntj.app.box.com/s/wdrwab9vvko3yd4enw39fxwzumgyxc2i/file/478811108973>

ルート線描画用サンプルコード

<https://ntj.app.box.com/s/wdrwab9vvko3yd4enw39fxwzumgyxc2i/file/478805770394>

デバッグ用サンプルコード

<https://ntj.app.box.com/s/wdrwab9vvko3yd4enw39fxwzumgyxc2i/file/478815217010>

開発イメージ用サンプルコード

<https://ntj.app.box.com/s/wdrwab9vvko3yd4enw39fxwzumgyxc2i/file/478815198433>



開発中





補足資料

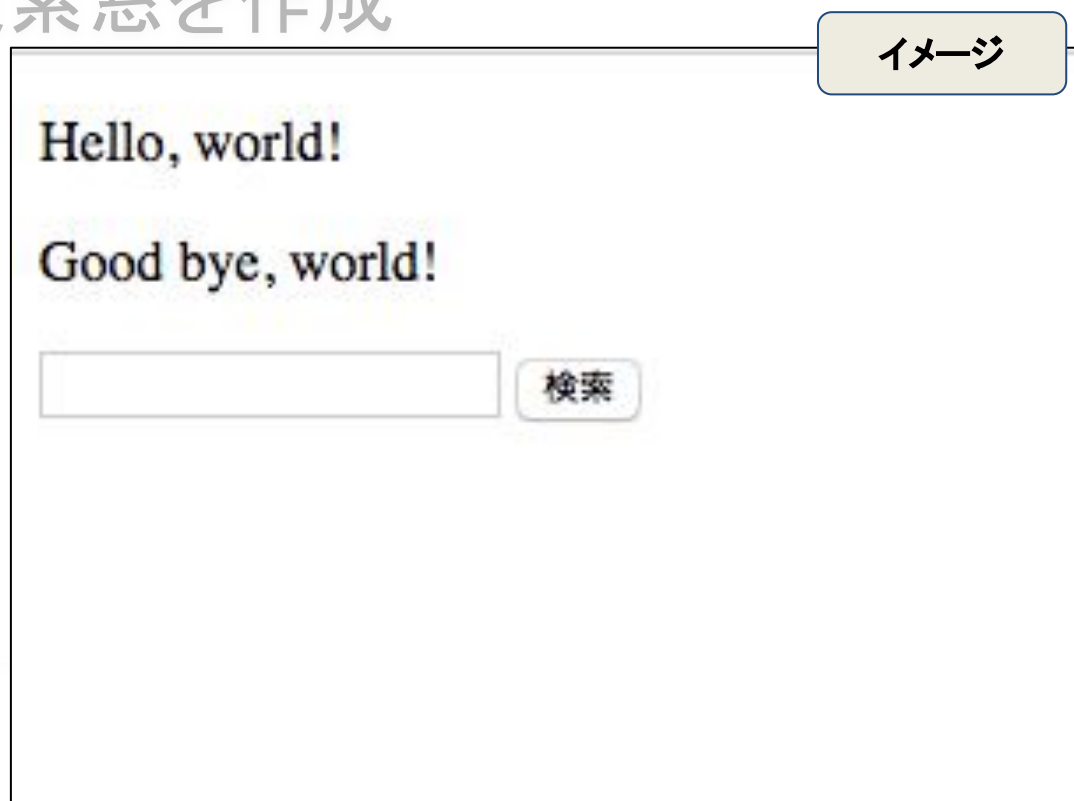


HTML (構造)



- HTML

- 段落を作成
- 検索窓を作成



Image

Hello, world!

Good bye, world!



● HTML

○ pタグを使うと段落 (paragraph)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <style>
    </style>
  </head>

  <body>
    <p>Hello, world!</p>
    <p>Good bye, world!</p>
    <script>
    </script>
  </body>
</html>
```

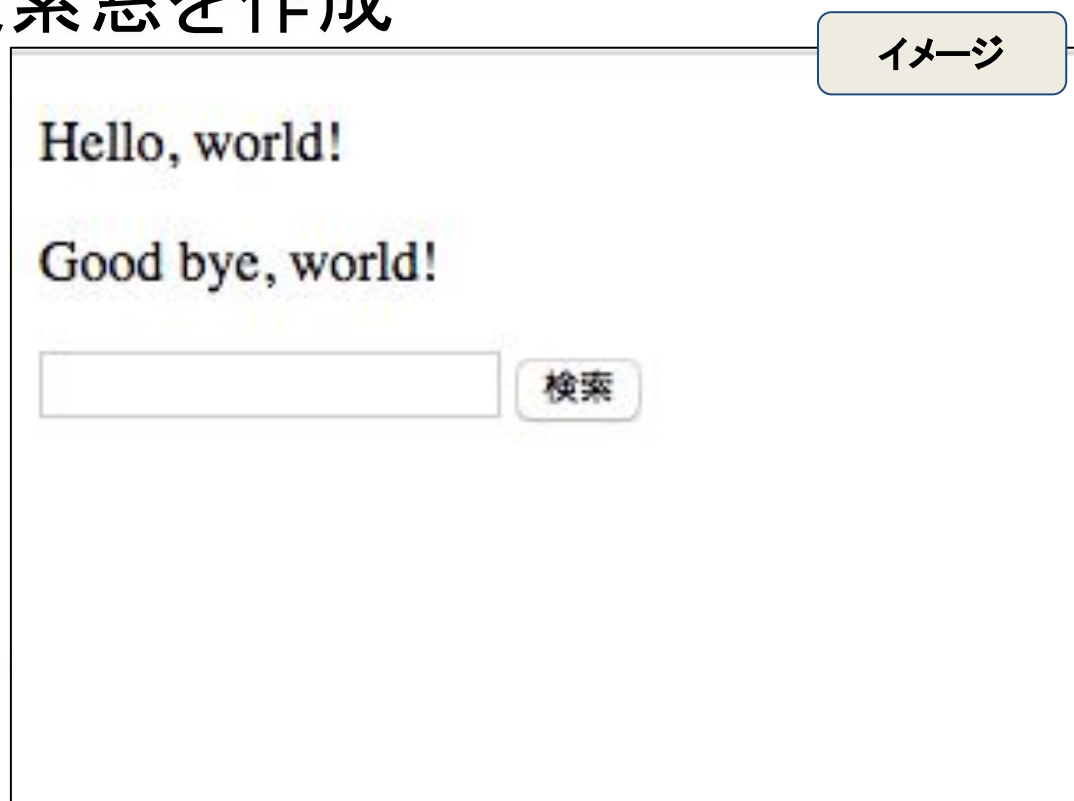
赤枠の外はおまじない

<p>内容</p> で段落になる



- HTML

- 段落を作成
- 検索窓を作成



イメージ

Hello, world!

Good bye, world!



● HTML

○ 検索窓を作るならform, input, button

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <style>
    </style>
  </head>
  <body>
    <p>Hello, world!</p>
    <p>Good bye, world!</p>
    <form name="searchForm">
      <input type="text" cols="30" id="searchWindow"/>
      <button type="button">検索</button>
    </form>
    <script>
    </script>
  </body>
</html>
```

<input type="text" /> でテキストフィールド

<button type="button"></button> でボタン



- HTML

Hello, world!

Good bye, world!

ナビタイム

検索

文字を入力できるスペースが出来る。

ボタンを配置しただけなので、
クリックしても何も起こりません。

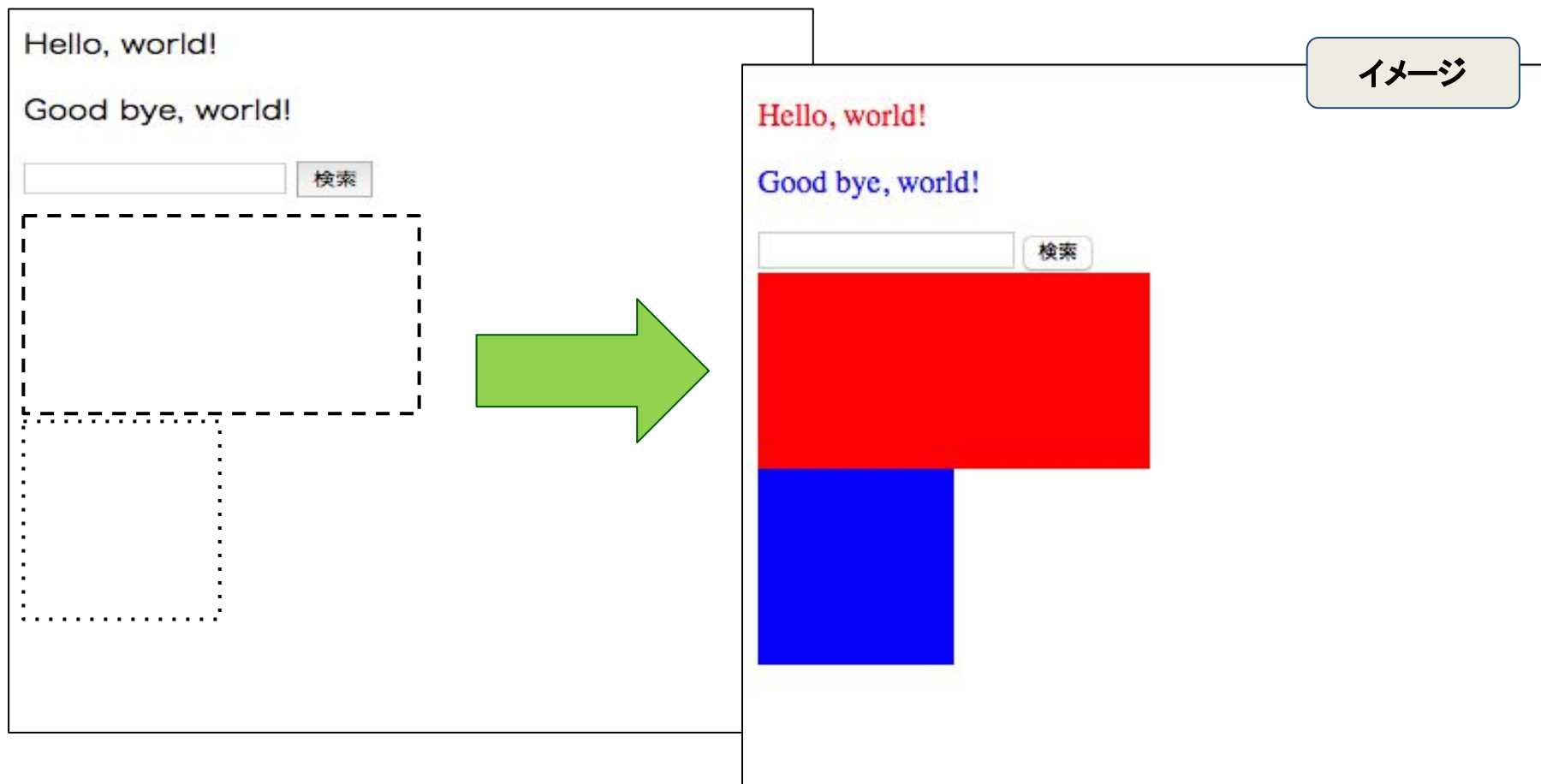


CSS (見た目)



- CSS

- 文字や長方形の領域の色を変更する





● CSS

○ 色を変える時はcolorを使う

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <style>
      .hello {
        color: red;
      }

      #goodBye {
        color: blue;
      }
    </style>
  </head>
```

CSSは **<style>** ~ **</style>** の中に書く

.hello {} のスタイルは
class="hello" の要素に適用される

#goodbye {} のスタイルは
id="goodbye" の要素に適用される
※原則として同じidの要素は複数存在してはいけない

```
<body>
  <p class="hello">Hello, world!</p>
  <p id="goodBye">Good bye, world!</p>
  ...
```



● CSS

- 幅と高さ: widthとheight(単位はpx)
- 背景色 : background-color

```
...  
<style>  
...  
#divTop {  
  width: 200px;  
  height: 100px;  
  background-color: red;  
}  
  
#divBottom {  
  width: 100px;  
  height: 100px;  
  background-color: blue;  
}  
</style>  
...  
<div id="divTop"></div>  
<div id="divBottom"></div>  
...
```

id="divTop" の要素は

幅は **200px**

高さは **100px**

背景色は **red**

※色の指定はカラーコードでも指定可能

[カラーコード一覧表](#)

id="divBottom" の要素は

幅は **100px**

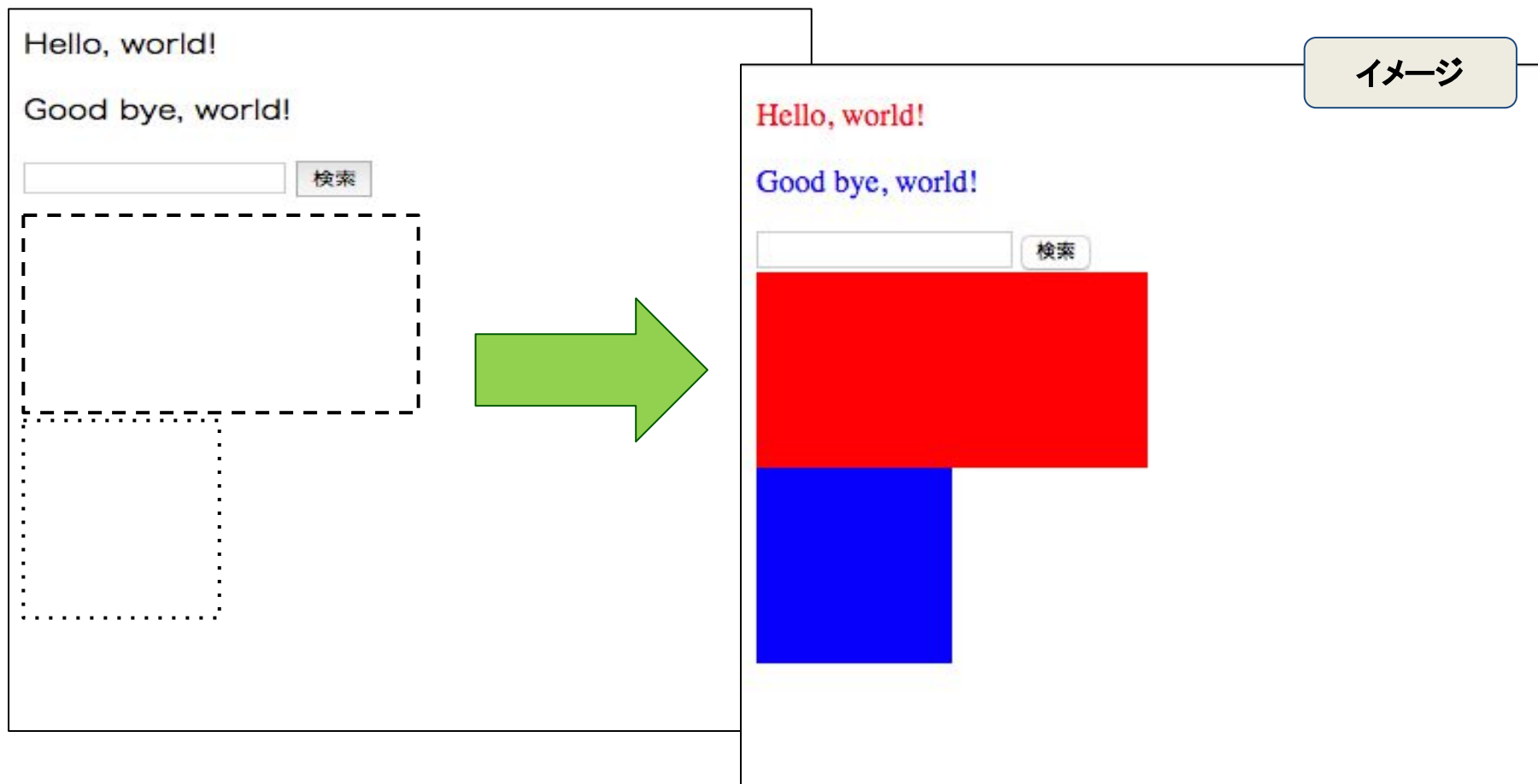
高さは **100px**

背景色は **blue**



- CSS

- 文字や長方形の領域の色を変更する





JavaScript (動き)



- JavaScript
 - ボタンが押されたら検索ワードをポップアップ表示する





● 最も単純なJavaScript

...

```
<body>
  <p class="hello">Hello, world!</p>
  <p id="goodBye">Good bye, world!</p>
  <form name="searchForm">
    <input type="text" cols="30" id="searchWindow"/>
    <button type="button" onclick="search()">検索</button>
  </form>
  <div id="divTop"></div>
  <div id="divBottom"></div>

  <script>

    function search() {
      let searchWord = document.searchForm.searchWindow.value
      alert(searchWord)
    }

  </script>
</body>
</html>
```

<button onclick="search()">

でボタンがクリックされた時に **search()** を実行

document.[form_name].[input_id].value

でテキストフィールドの中の文字列を取得
(新しい変数の宣言は型に関わらず **let** を使う)

alert() で検索ワードをポップアップ表示



jQuery



● jQuery

- JavaScriptを簡単に書ける(ライブラリ)
- jQueryを使用するための準備

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

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

```
<script src='https://code.jquery.com/jquery-1.12.0.min.js'></script>
```

```
</head>
```

```
<body>
```

```
<p class="text"></p>
```

```
<script>
```

```
$(document).ready(function() {
```

```
  $("#text").text("Hello World!");
```

```
});
```

jQueryを使用するおまじない

ページが読み込まれてから
動作するように記載

参考サイト: <http://semooh.jp/jquery/> (日本語サイト)



- jQuery
 - ボタンを押したら段落の文字列を変更する





● jQuery

○ text() で文字列を置換する

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <script src='https://code.jquery.com/jquery-1.12.0.min.js'></script>
  </head>
  <body>
    <p class="hello">Hello, world!</p>
    <p id="goodBye">Good bye, world!</p>
    <script>
      $(document).ready(function() {
        $('#btn').click(function(e) {
          $("#goodBye").text("Have A Nice Day!");
        });
      });
    </script>
  </body>
```

click() は
ボタン押下時のアクション

text() に指定した文字に置換する



- jQuery

- 入力した文字列をリストの要素に足す





● jQuery

○ append() でリストの要素を追加する

```
<body>
  <ul>
    <li>リスト</li>
  </ul>
  <form name="searchForm">
    <input type="text" cols="30" id="searchWindow"/>
    <button id="btn" type="button" >追加</button>
  </form>
  <script>
    $(document).ready(function() {
      // ボタンクリックで窓に入力された文字列をリストの要素に追加
      $('#btn').click(function(e) {
        let searchWord = document.searchForm.searchWindow.value
        $("ul").append("<li>" + searchWord + "</li>");
      });
    });
  </script>
```

**** で箇条書き、
**** は項目を表す

append() で要素を追加

****の中に****が入ります↓

<https://saruwakakun.com/html-css/basic/ul-ol-li>