プロトタイプ型オブジェクト指向 JavaScriptの利点を Webアプリケーションに活かす

群馬大学医学部附属病院
システム統合センター
鳥飼　幸太

目次

https://jsfiddle.net/5fh3o06s/45/

なぜ「今」JavaScriptなのか

1995年にWebブラウザであったネットスケープ内のスクリプト言語として誕生
1997年の標準化開始から、近年言語としての品質が急速に高まった

西暦	出来事
1990	WWW誕生
1994	Mosaic Communication創業
1995	JavaScriptが搭載される
1998	Google創業
2004	Facebook創業
2009	Node.jsが開発される(JSONサポート)
2010	OracleがSun Microsystemsを買収
2015	ECMAScript2015が制定 （クラス・モジュール）

名前は同じだが、その内容が急速に変化している、5年前に制定された新しい技術

プロトタイプ型オブジェクト指向（ECMA2015で策定）とは

・「クラスの管理者がいなくなった後のコード開発問題」に解決法を与えるもの

クラスベース (Java) とプロトタイプベース (JavaScript) のオブジェクトシステムの比較	
クラスベース (Java)	プロトタイプベース (JavaScript)
クラスとインスタンスは異なる実体です。	すべてのオブジェクトは別のオブジェクトを継承できます。
クラス定義を用いてクラスを定義します。また、コンストラクターメソッドを用いてクラスをインスタンス化します。	コンストラクター関数を用いて一連のオブジェクトを定義および作成します。
new 演算子を用いて単一のオブジェクトを生成します。	同様です。
既存のクラスのサブクラスを定義するクラス定義を用いて、オブジェクト階層を構築します。	コンストラクター関数に結びつけられたプロトタイプとしてオブジェクトを代入することで、オブジェクト階層を構築します。
クラスチェーンに従ってプロパティを継承します。	プロトタイプチェーンに従ってプロパティを継承します。
クラス定義が、クラスから作られた全インスタンスすべてのプロパティを定義します。実行時に動的にプロパティを追加することはできません。	コンストラクター関数またはプロトタイプによって、一連の初期化されたプロパティが指定されます。個々のオブジェクトやオブジェクトのセット全体へ動的にプロパティを追加したり、それらからプロパティを削除したりできます。

https://developer.mozilla.org/ja/docs/Web/JavaScript/Guide/Details_of_the_Object_Model

M言語(IRIS)との共通性

JSON リテラル・コンストラクタの使用

ダイナミック・エンティティは、%DynamicObject または %DynamicArray のインスタンスです。これらのクラスは、JSON のデータ操作を Caché にシームレスに統合するように設計されています。標準の %New() メソッドを使用してこれらのクラスのインスタンスを作成できますが、ダイナミック・エンティティは、これよりはるかに柔軟で直感的な一連のコンストラクタをサポートしています。*JSON リテラル・コンストラクタ*を使用すると、JSON 文字列を変数に直接代入することでダイナミック・エンティティを作成できます。例えば、以下のコードは %DynamicObject と %DynamicArray の空のインスタンスを作成します。

```
set dynamicObject = {}
set dynamicArray = []
write dynamicObject,!,dynamicArray

3@%Library.DynamicObject
1@%Library.DynamicArray
```

%New() コンストラクタとは異なり、リテラル・コンストラクタの {} と [] は JSON フォーマットの文字列を引数として受け取ることができます。例えば、以下のコードは *prop1* という名前のプロパティを持つダイナミック・オブジェクトを作成します。

```
set dynamicObject = {"prop1":"a string value"}
write dynamicObject.prop1

a string value
```

実際には、JSON リテラル・コンストラクタの {} と [] を使用して、任意の有効な JSON 配列構造またはオブジェクト構造を指定できます。簡単に言うと、有効な JSON リテラル文字列はすべて、評価結果がダイナミック・エンティティとなる有効な Caché 式でもあります。

・IRISのDynamicObjectは
JSONのプロトタイプ型オブジェクト指向そのものである

オブジェクトとプロパティ

JavaScript のオブジェクトは、自身に関連付けられたプロパティを持ちます。オブジェクトのプロパティは、オブジェクトに関連付けられている変数と捉えることができます。オブジェクトのプロパティは、オブジェクトに属するものという点を除けば、基本的に通常の JavaScript 変数と同じようなものです。オブジェクトのプロパティは、オブジェクトの特性を定義します。オブジェクトのプロパティには、単純なドット表記でアクセスします。

```
objectName.propertyName
```

すべての JavaScript の変数と同じく、オブジェクト名 (通常の変数にもなります) とプロパティ名では、大文字と小文字は厳密に区別されます。プロパティに値を代入することでプロパティを定義する事ができます。以下のようにして、myCar という名前のオブジェクトを作成し、make、model、year という名前のプロパティを付与することができます

```
var myCar = new Object();
myCar.make = 'Ford';
myCar.model = 'Mustang';
myCar.year = 1969;
```

上記の例は、**オブジェクト初期化子**を使用して作成することもできます。オブジェクト初期化子は、中括弧 ({}) で囲まれたオブジェクトのプロパティ名と関連する値の 0 個以上のペアをカンマで区切ったリストです。

```
var myCar = {
    make: 'Ford',
    model: 'Mustang',
    year: 1969
};
```

オブジェクトに割り当てられていないプロパティは undefined です (null ではありません)。

プログラミングの構成要素

https://skillhub.jp/blogs/183

ほぼ全てのプログラムが備えている機能：

変数名を付けた変数（数字や文字を格納）

関数（特定の処理を関数名として呼ぶ）

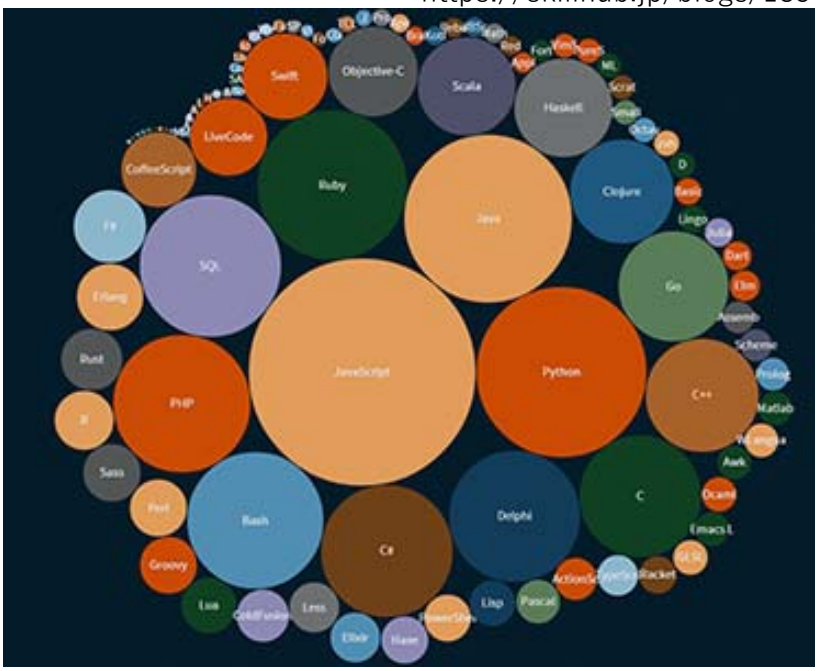
繰り返し（1億回でも繰り返せる）

条件分岐（場合分けの処理を記述する）

入出力（画面、ファイルシステム、デバイス）

外部呼出し（APIと呼ばれる）

<u>素朴な疑問：多言語は真に必須なのか？</u>



プログラム名とその比率

Web開発の問題点

最小構成のプログラムでも「Webサーバ」を意識しなければならない
- ・非同期通信
- ・ポートの概念
- ・URIの概念

ユーザーインターフェースにHTML、CSSを必要とする
- ・JavaScriptを含め文法が3つ存在する

サーバサイドJavaエンジニアにとってはこれまで（品質の高くない）言語であり、
習得コストも高かった

これらの特徴のため、RAD（Rapid Application Development）には向かなかった

Web開発の利点

世界中のサーバに置いてあるライブラリを利用できる

スクリプト言語と標準化がもたらす高い移植性

処理速度の遅さをハードウェアやメニーコアがカバーし、実用的に

クラス概念などの言語の洗練により、チームコーディングが容易に

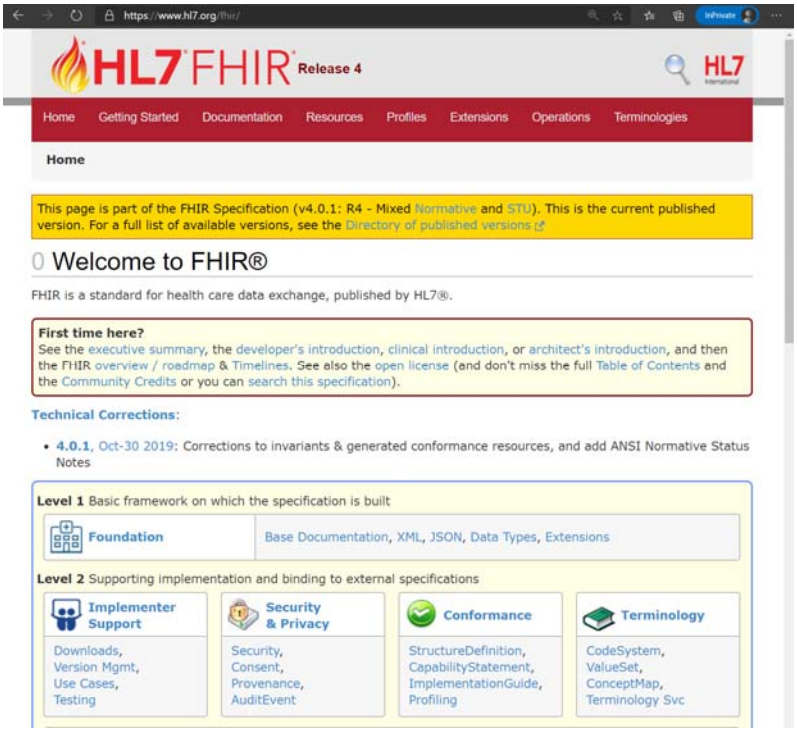
→エンジニアは「世界中で使われるjsライブラリの開発者」となる道と、
「これらライブラリを組み合わせて現場に必要なサービスアプリケーションを
迅速に提供する開発者」となる道とが生じてきた

FHIR-JSONを応用する直接的な利点

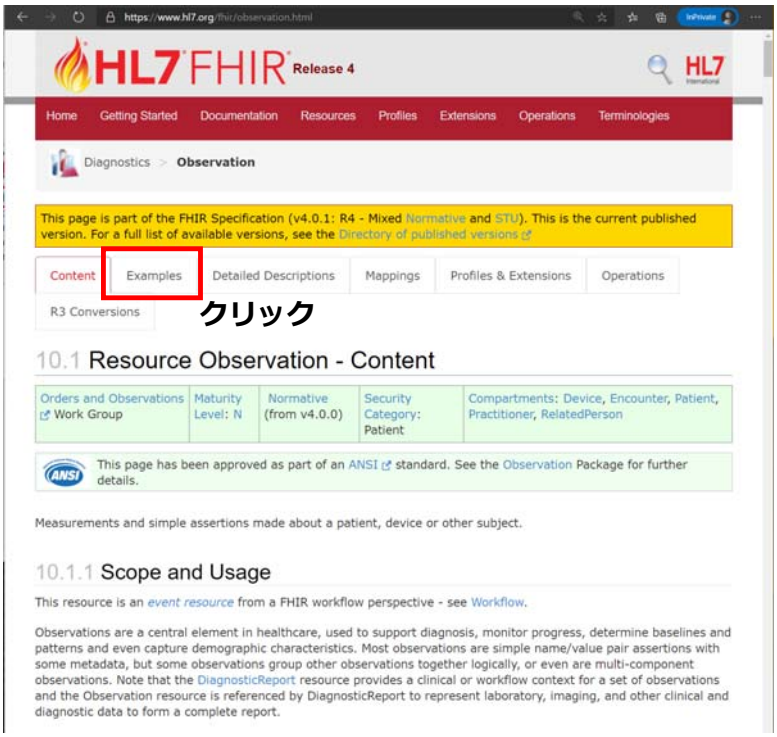
・Webページで公開されている例を直接プロトタイプとして使うことができる
→DICOM Conformance Statementのように、コードと「別に」仕様書を（エンジニア自身が）記述する仕事から解放される

→仮に仕様が合わなくても、現地レベルで（つまり、コード生成の後ろの段で）修正モジュールを記述して通せばよい→作業性の改善

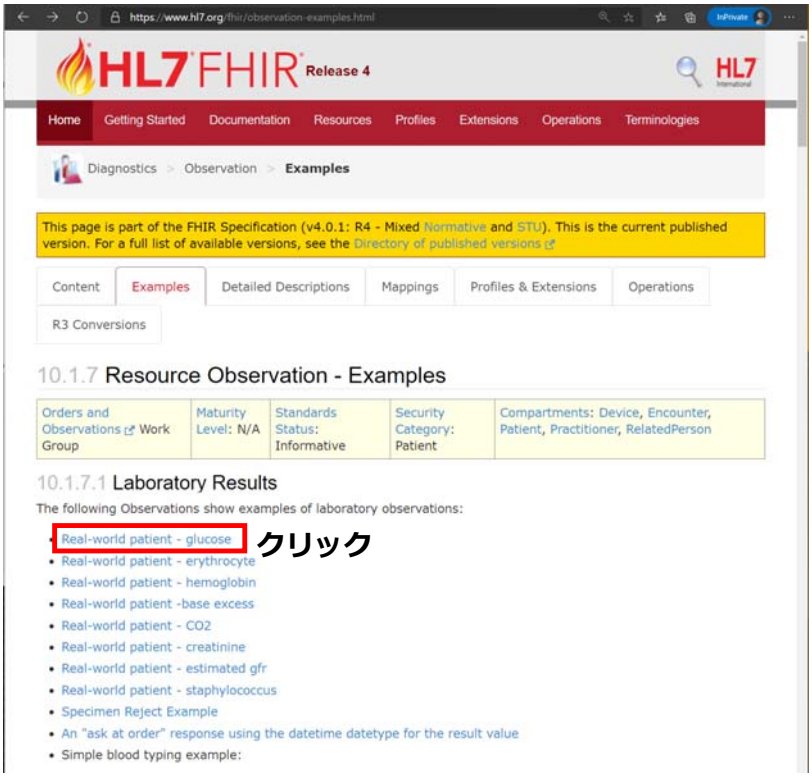
FHIRサイトでの一例



Exampleを開く

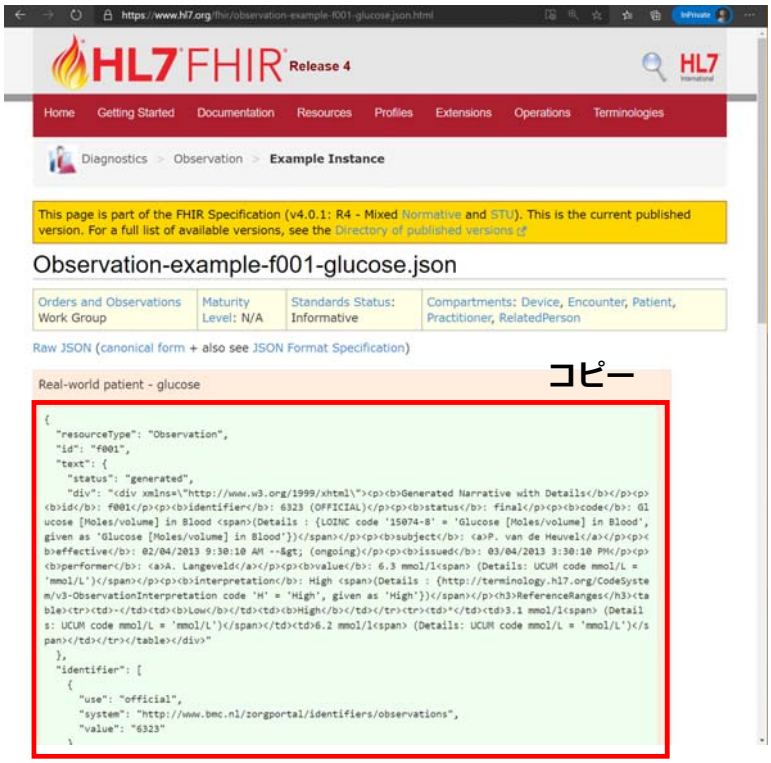


使いたい形式に近いExampleを選択

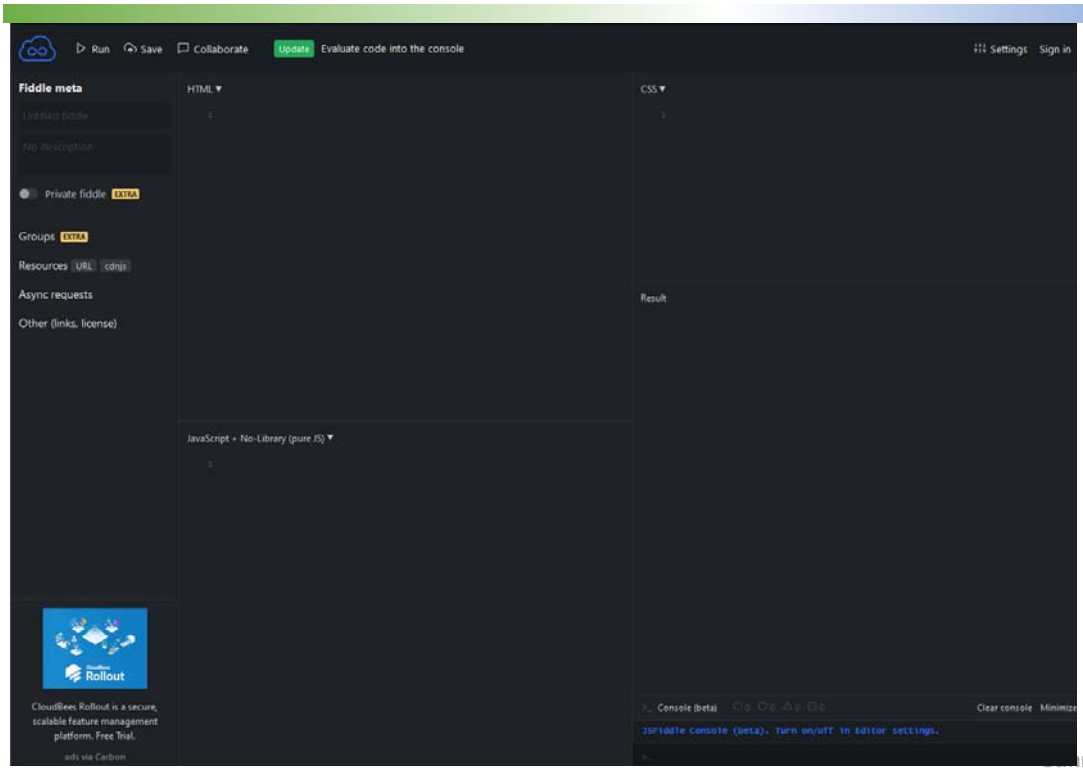


JSON形式を選択

JSON部分をコピーする

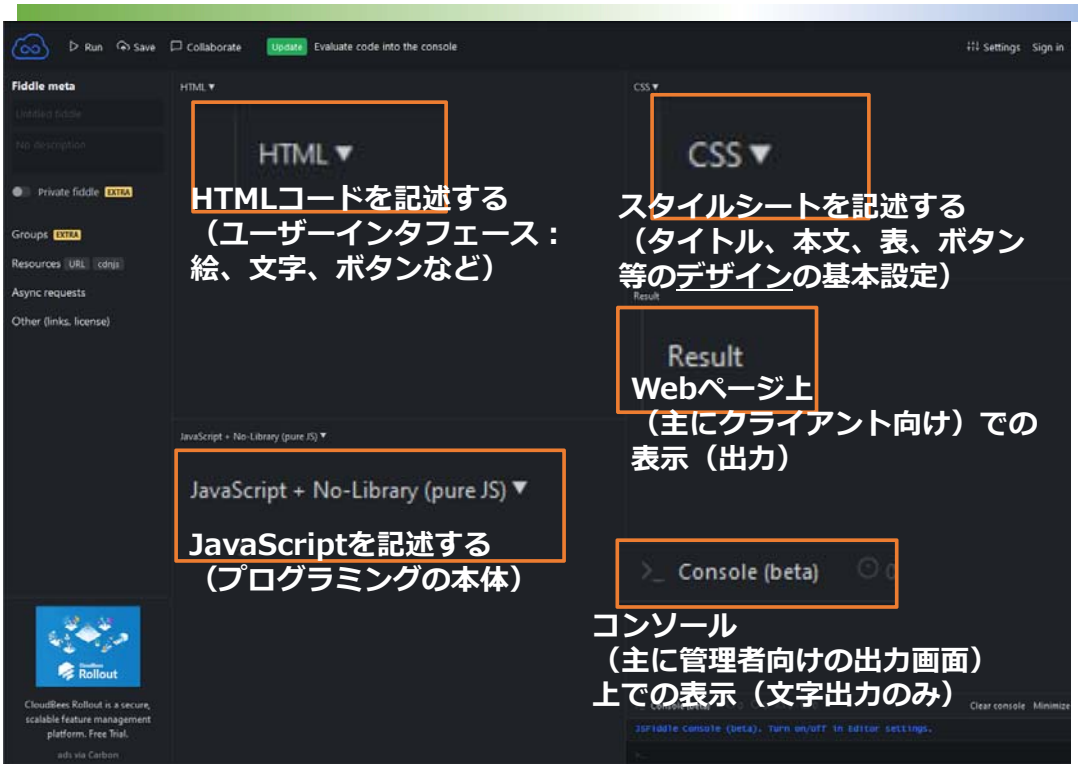


JSFiddleの起動



https://jsfiddle.net/
利用にあたり
アカウント作成や
ログインは不要

JSFiddle: 画面の使い方



HTML▼
HTMLコードを記述する
（ユーザーインタフェース：
絵、文字、ボタンなど）

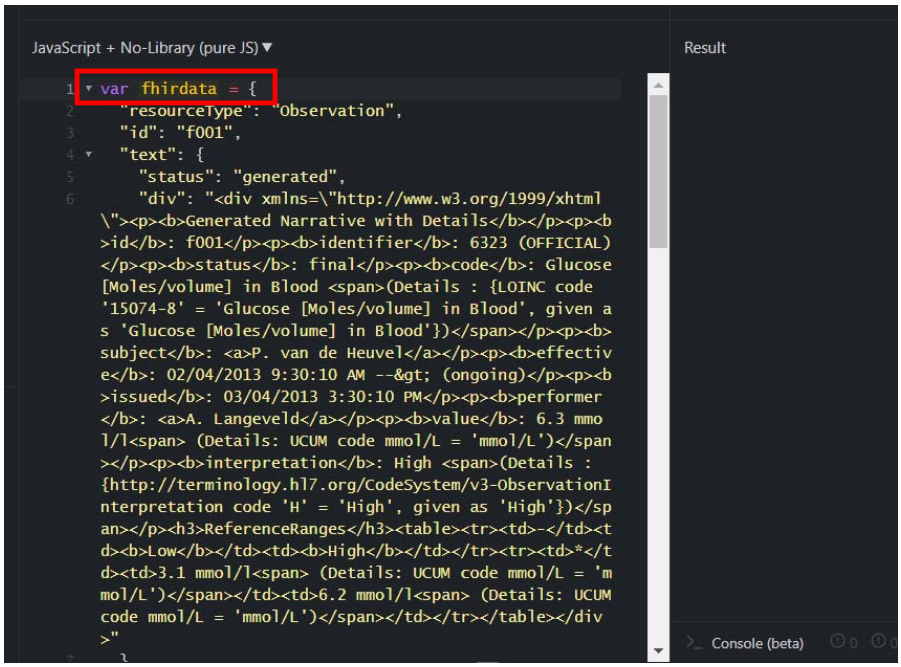
CSS▼
スタイルシートを記述する
（タイトル、本文、表、ボタン
等のデザインの基本設定）

Result
Webページ上
（主にクライアント向け）での
表示（出力）

JavaScript + No-Library (pure JS)▼
JavaScriptを記述する
（プログラミングの本体）

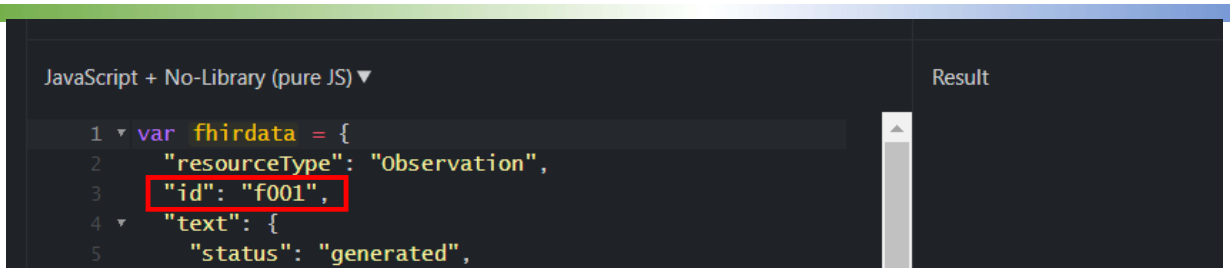
Console (beta)
コンソール
（主に管理者向けの出力画面）
上での表示（文字出力のみ）

JavaScript部にコピーしたデータを貼り付けて変数化



var fhirdata = の後にペーストし、最後の}の後に ； を記述すれば「ひな形」のデータが出来上がる

プロパティの表示



コピー

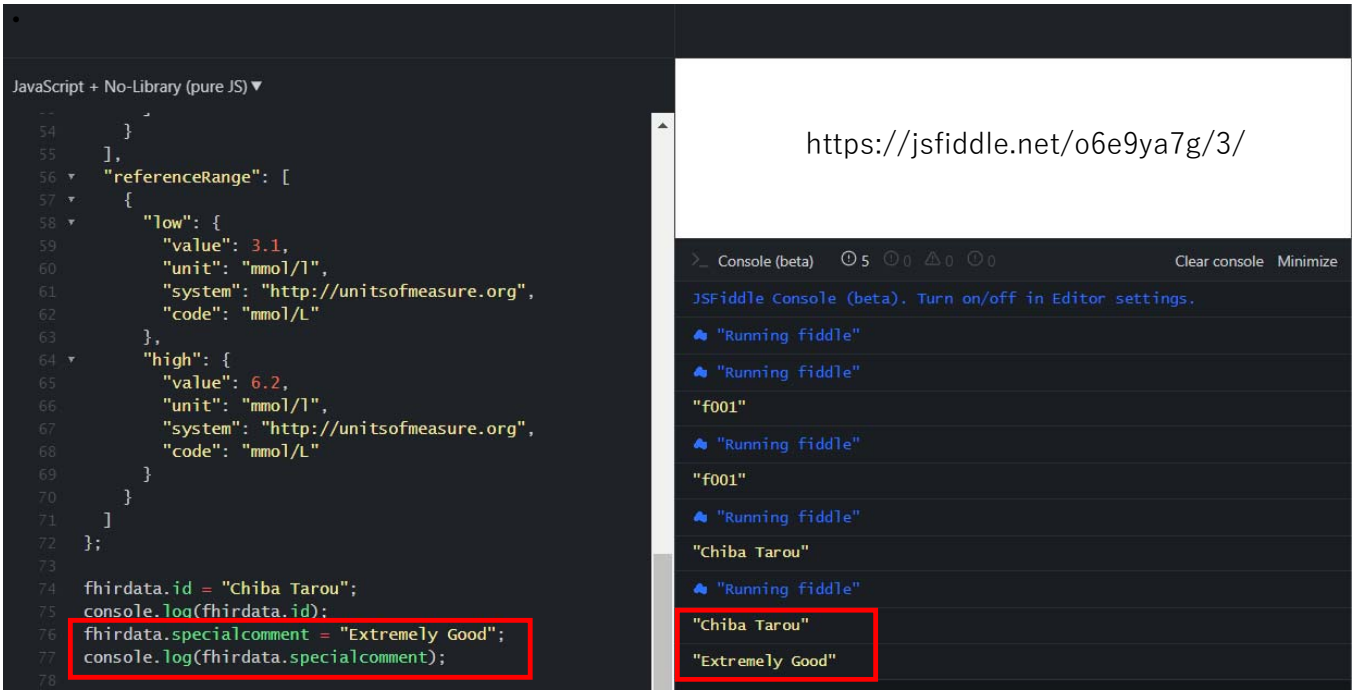
https://jsfiddle.net/o6e9ya7g/1/

コンソール出力関数は
console.log(…);
ここに、Exampleに入っているidを呼び出すと、f001が表示される

プロパティを書き換える

idが上書きされる

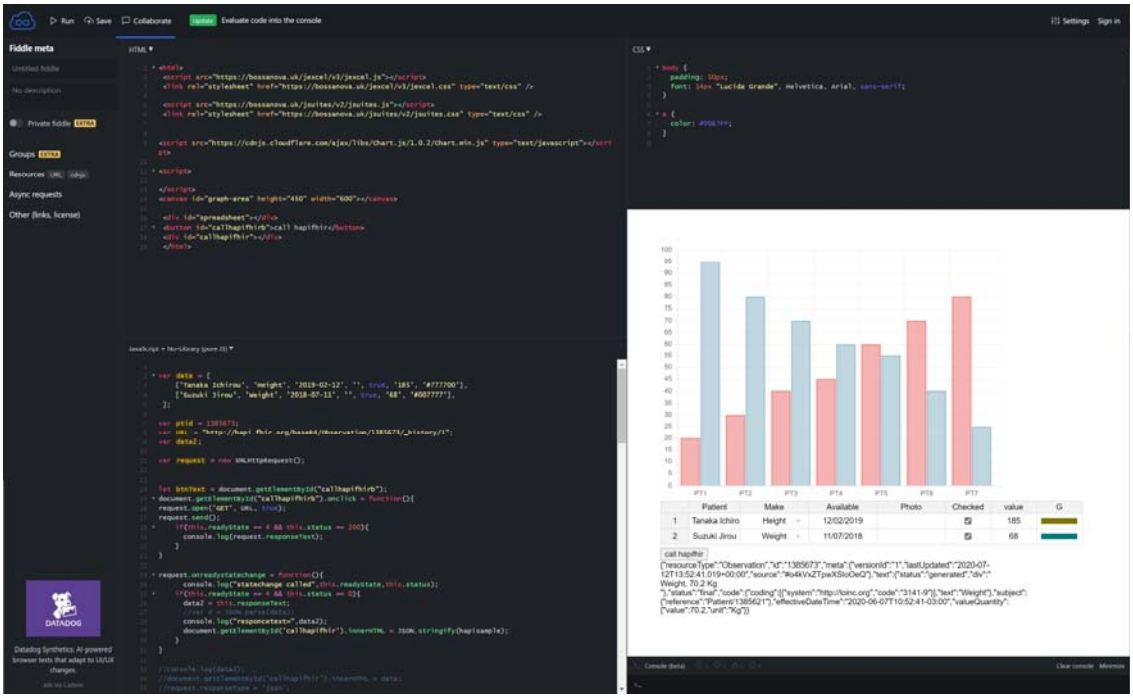
プロパティを動的に追加する（プロトタイプ型オブジェクトの特徴）



それまでに定義していなかったプロパティを動的に格納・使用できる

JSFiddleサンプル(Graph.js, jExcel)

https://jsfiddle.net/5fh3o06s/45/



Graph.jsのデータ設定

JavaScript + No-Library (pure JS)▼

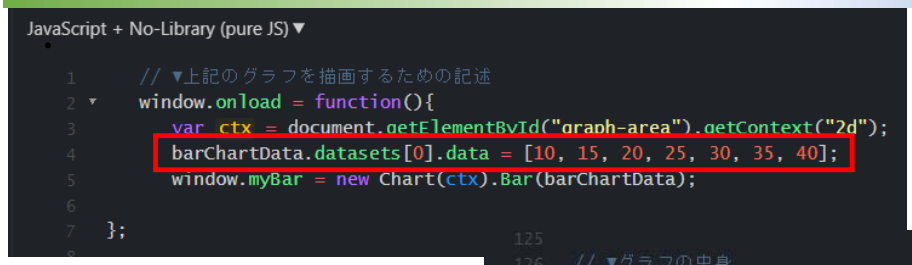
```
// ▼上記のグラフを描画するための記述
window.onload = function(){
    var ctx = document.getElementById("graph-area").getContext("2d");
    window.myBar = new Chart(ctx).Bar(barChartData);
}
```

```
// ▼グラフの中身
var barChartData = {
    labels : ["PT1","PT2","PT3","PT4","PT5","PT6","PT7"],
    datasets : [
        {
            fillColor : "rgba(240,128,128,0.6)",     // 塗りつぶし色
            strokeColor : "rgba(240,128,128,0.9)",   // 枠線の色
            highlightFill: "rgba(255,64,64,0.75)",   // マウスが載った際の塗りつぶし色
            highlightStroke: "rgba(255,64,64,1)",    // マウスが載った際の枠線の色
            data : [ 20, 30, 40, 45, 60, 70 , 80 ]     // 各棒の値(カンマ区切りで指定)
        },
        {
            fillColor : "rgba(151,187,205,0.6)",
            strokeColor : "rgba(151,187,205,0.9)",
            highlightFill : "rgba(64,96,255,0.75)",
            highlightStroke : "rgba(64,96,255,1)",
            data : [ 95, 80, 70, 60, 55, 40, 25 ]
        }
    ]

};
```

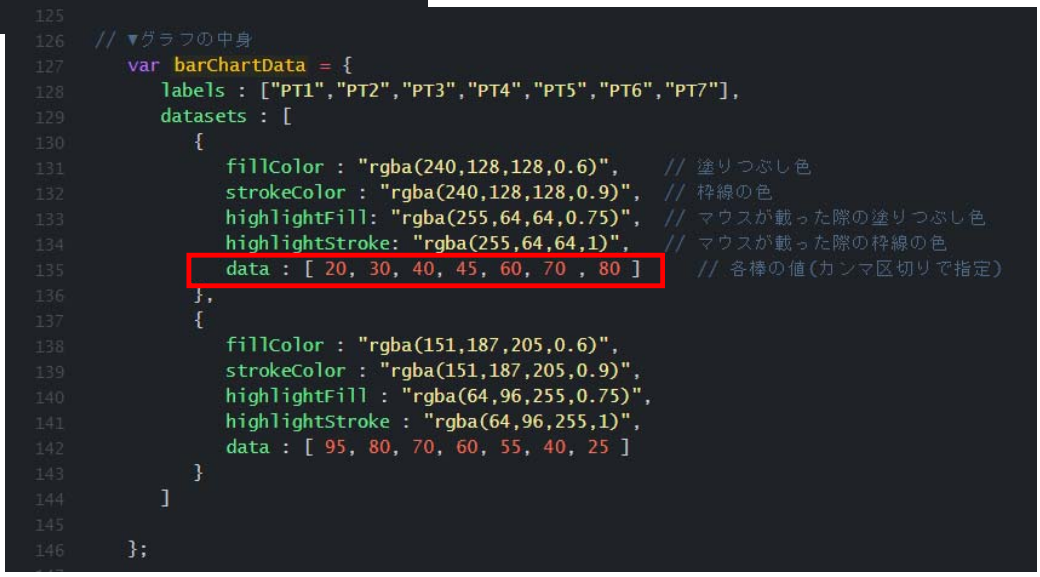
https://jsfiddle.net/j92vhk4y/

data:欄のデータを
代入で書き換えて表示を行う

Graph.jsのデータ設定



```
// ▼上記のグラフを描画するための記述
window.onload = function(){
    var ctx = document.getElementById("graph-area").getContext("2d");
    barChartData.datasets[0].data = [10, 15, 20, 25, 30, 35, 40];
    window.myBar = new Chart(ctx).Bar(barChartData);

};
```



```
// ▼グラフの中身
var barChartData = {
    labels : ["PT1","PT2","PT3","PT4","PT5","PT6","PT7"],
    datasets : [
        {
            fillColor : "rgba(240,128,128,0.6)",     // 塗りつぶし色
            strokeColor : "rgba(240,128,128,0.9)",   // 枠線の色
            highlightFill: "rgba(255,64,64,0.75)",   // マウスが載った際の塗りつぶし色
            highlightStroke: "rgba(255,64,64,1)",    // マウスが載った際の枠線の色
            data : [ 20, 30, 40, 45, 60, 70 , 80 ]     // 各棒の値(カンマ区切りで指定)
        },
        {
            fillColor : "rgba(151,187,205,0.6)",
            strokeColor : "rgba(151,187,205,0.9)",
            highlightFill : "rgba(64,96,255,0.75)",
            highlightStroke : "rgba(64,96,255,1)",
            data : [ 95, 80, 70, 60, 55, 40, 25 ]
        }
    ]

};
```

https://jsfiddle.net/j92vhk4y/1/

data:欄のデータを
代入で書き換えて表示を行う

jExcelのデータ設定

・

```
<html>
 <script src="https://bossanova.uk/jexcel/v3/jexcel.js"></script>
 <link rel="stylesheet" href="https://bossanova.uk/jexcel/v3/jexcel.css" type="text/css" />

 <script src="https://bossanova.uk/jsuites/v2/jsuites.js"></script>
 <link rel="stylesheet" href="https://bossanova.uk/jsuites/v2/jsuites.css" type="text/css" />


<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/1.0.2/Chart.min.js" type="text/javascript"></script>

<script>

</script>
<canvas id="graph-area" height="450" width="600"></canvas>

 <div id="spreadsheet"></div>
 <button id="callhapifhirb">call hapifhir</button>
 <div id="callhapifhir"></div>
 </html>
```

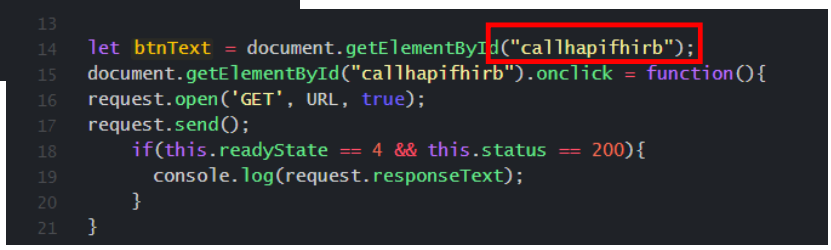
https://jsfiddle.net/j92vhk4y/2/

source:欄のデータを
代入で書き換えて表示を行う
この記述部がJSONファイルで読み込んでもよい

```
jexcel(document.getElementById('spreadsheet'), {
    data:data,
    columns: [
        {
            type: 'text',
            title:'Patient',
            width:90
        },
        {
            type: 'dropdown',
            title:'Make',
            width:120,
            source:[
                "Temperature",
                "Height",
                "Weight",
                "BloodType",
                "Age",
                "Name"
            ]
        },
        {
            type: 'calendar',
            title:'Available',
            width:120
        },
        {
            type: 'image',
            title:'Photo',
            width:120
        },
        {
            type: 'checkbox',
            title:'Checked',
            width:80
        },
        {
            type: 'numeric',
            title:'value',
            mask:'#.##,00',
            width:80,
            decimal:','
        },
        {
            type: 'color',
            width:80,
            render:'square',
        },
    ]
});
```

ボタンの配置

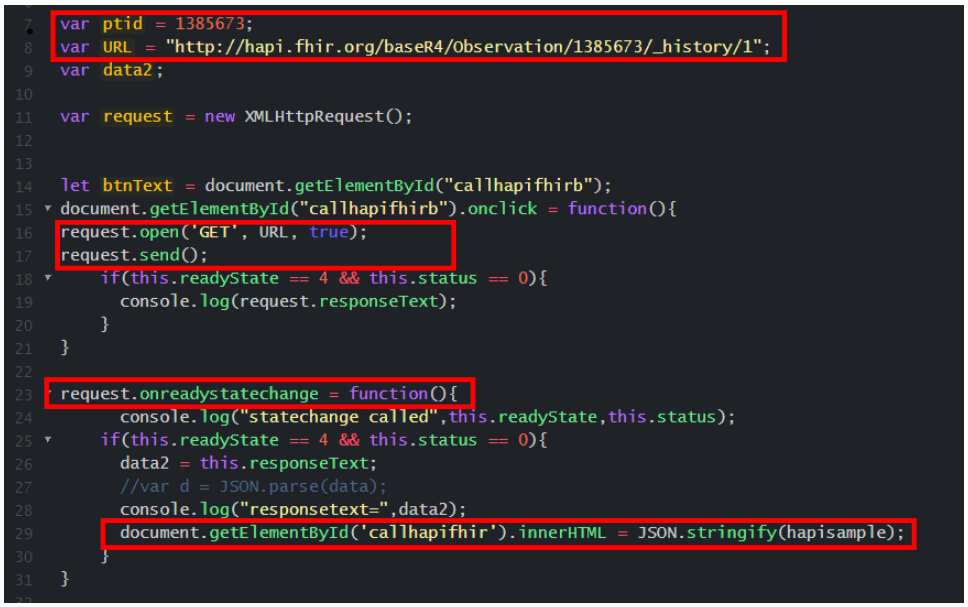
```
HTML ▼

<html>
 <script src="https://bossanova.uk/jexcel/v3/jexcel.js"></script>
 <link rel="stylesheet" href="https://bossanova.uk/jexcel/v3/jexcel.css" type="text/css" />

 <script src="https://bossanova.uk/jsuites/v2/jsuites.js"></script>
 <link rel="stylesheet" href="https://bossanova.uk/jsuites/v2/jsuites.css" type="text/css" />


<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/1.0.2/Chart.min.js" type="text/javascript"></script>

<script>

</script>
<canvas id="graph-area" height="450" width="600"></canvas>

 <div id="spreadsheet"></div>
 <button id="callhapifhirb">call hapifhir</button>
 <div id="callhapifhir"></div>
 </html>
```



```
let btnText = document.getElementById("callhapifhirb");
document.getElementById("callhapifhirb").onclick = function(){
request.open('GET', URL, true);
request.send();
    if(this.readyState == 4 && this.status == 200){
      console.log(request.responseText);
    }
}
```

<button>タグでidを置く
JavaScript中でdocument.getElementByIdでidをコール

FHIRサーバへの接続



```
var ptid = 1385673;
var URL = "http://hapi.fhir.org/baseR4/Observation/1385673/_history/1";
var data2;

var request = new XMLHttpRequest();


let btnText = document.getElementById("callhapifhirb");
document.getElementById("callhapifhirb").onclick = function(){
request.open('GET', URL, true);
request.send();
    if(this.readyState == 4 && this.status == 0){
      console.log(request.responseText);
    }
}

request.onreadystatechange = function(){
      console.log("statechange called",this.readyState,this.status);
    if(this.readyState == 4 && this.status == 0){
      data2 = this.responseText;
      //var d = JSON.parse(data);
      console.log("responsetext=",data2);
      document.getElementById('callhapifhir').innerHTML = JSON.stringify(hapisample);
    }
}
```

https://jsfiddle.net/5fh3o06s/45/

XMLHttpRequest()でURLをコールする
request.sendでHAPI serverにアクセスする

１：サーバアクセスの基本となるフレームHTMLを作成する

HTMLの構造

```
<html>↓
↓
<!-- ①HTMLのスタイル等設定箇所 -->↓
<head>↓
  <meta charset="UTF-8">↓
↓
  <title>FHIRTEST Step 1</title>↓
↓
  <style>body {↓
    background: #eeeeee;↓
  }</style>↓
↓
<!-- ②サーバに接続しFHIRデータを処理する箇所 -->↓
  <script type="text/javascript" language="javascript">↓
↓
        function onButtonClick() {↓
           var target = document.getElementById("output2");     // varは変数宣言，出力先を <div id="output2">に設定↓
           var txt = document.forms.id_form1.id_textBox2.value; // id_textBox2の呼び出し方↓
           var outtext = txt + "です";                          // 出力テキストの合成は + を用いる↓
           target.innerHTML = outtext;                          // ↓
        }↓
↓
↓
  </script>↓
↓
</head>↓
↓
<br>↓
<br>↓
<!-- ③画面出力を行う箇所 -->↓
<body>↓
  <form name="form1" id="id_form1" action="">↓
    <input name="textBox1" id="id_textBox2" type="text" value="" />↓
    <input type="button" value="Exec" onclick="onButtonClick();" />↓
  </form>↓
<br>↓
  <div id="output2"></div>↓
<br>↓
↓
</body>↓
</html>[EOF]
```

HTMLタグ：一番外側にくる構造

HEADタグ：スタイル、コードを格納

SCRIPTタグ：コード実体を記述

BODYタグ：コードにより出力されるHTMLのターゲット位置を指定

１：サーバアクセスの基本となるフレームHTMLを作成する

HTMLの構造


```
<html>↓
↓
<!-- ①HTMLのスタイル等設定箇所 -->↓
<head>↓
  <meta charset="UTF-8">↓
↓
  <title>FHIRTEST Step 1</title>↓
↓
  <style>body {↓
    background: #eeeeee;↓
  }</style>↓
↓
<!-- ②サーバに接続しFHIRデータを処理する箇所 -->↓
  <script type="text/javascript" language="javascript">↓
↓
        function onButtonClick() {↓
           var target = document.getElementById("output2");     // varは変数宣言，出力先を <div id="output2">に設定↓
           var txt = document.forms.id_form1.id_textBox2.value; // id_textBox2の呼び出し方↓
           var outtext = txt + "です";                          // 出力テキストの合成は + を用いる↓
           target.innerHTML = outtext;                          // ↓
        }↓
↓
↓
  </script>↓
↓
</head>↓
↓
<br>↓
<br>↓
<!-- ③画面出力を行う箇所 -->↓
<body>↓
  <form name="form1" id="id_form1" action="">↓
    <input name="textBox1" id="id_textBox2" type="text" value="" />↓
    <input type="button" value="Exec" onclick="onButtonClick();" />↓
  </form>↓
<br>↓
  <div id="output2"></div>↓
<br>↓
↓
</body>↓
</html>[EOF]
```

③関数onButtonClick()の動作を記述ここではtextbox2の値を読み込み

①画面から入力されたテキストボックスの内容はJavaScriptからid_textbox2として認識される

②ボタンクリックで、関数onButtonClick()をコール

④document.getElementId("output2").innerHTMLを設定

2 ：JavaScriptでサーバにアクセスする

サンプル

```
<html>↓
<head>↓
  <meta charset="UTF-8">↓
  <title>FHIRTEST Step 2</title>↓
  <style>body {↓
    background: #eeeeee;↓
  }</style>↓
↓
  <script type="text/javascript" language="javascript">↓
↓
↓
        function onButtonClick() {↓
          target = document.getElementById("output");↓

        var xhr = new XMLHttpRequest(); // HTTPサーバへの接続インスタンスを作成↓
        xhr.responseType = 'text';      // サーバからの応答をテキスト形式に指定↓
        xhr.open('GET',"http://iris4h-fhirapi.japaneast.cloudapp.azure.com:52773/csp/sqltofhir/fhir/observation/pulserate/1001",true); // RESTのGETコマンドで、リソースを指定する↓
        xhr.send();  // サーバに送る↓

        xhr.onreadystatechange = function(){ // サーバから応答があったときに実行される箇所↓
                if (xhr.readyState==4 && xhr.status==200) { // readyState=4はDONE（操作完了）かつstatus=200はOK（要求通りの処理がされた）↓
        var data = JSON.parse(xhr.responseText); // !!重要 JSON形式を構造体として解釈する これ以降はJSONタグをプロパティとして記述できる↓
        //追加した部分↓
        //変更した部分↓
        var outtext = xhr.responseText + '<br> patient id =' + data.id + '<br>'; // data.idと記述されている"id"はJSON内のidタグを指す↓
        //変更した部分↓

        target.innerHTML = outtext; //読み取ったタグの値をHTMLとして出力↓
                }↓
        };↓

        }↓

  </script>↓

</head>↓

<br>↓
<br>↓

<body>↓
  <form name="form1" id="id_form1" action="">↓
    <input name="textBox1" id="id_textBox1" type="text" value="" />↓
    <input type="button" value="Exec" onclick="onButtonClick();" />↓
  </form>↓
  <hr/>↓

<br>↓
  <div id="output"></div>↓
<br>↓
↓
</body>↓
</html>↓
```

①XMLHttpRequestのインスタンスを作成
②テキスト形式を指定
③GET/REST構文で欲しいリソースを指定
④構文をサーバに送信

読みだしたファイルの表示位置

3 ：データを取得しJSON形式でメモリに格納する

サンプル

```
<html>↓
<head>↓
  <meta charset="UTF-8">↓
  <title>FHIRTEST Step 2</title>↓
  <style>body {↓
    background: #eeeeee;↓
  }</style>↓
↓
  <script type="text/javascript" language="javascript">↓
↓
↓
        function onButtonClick() {↓
          target = document.getElementById("output");↓

        var xhr = new XMLHttpRequest(); // HTTPサーバへの接続インスタンスを作成↓
        xhr.responseType = 'text';      // サーバからの応答をテキスト形式に指定↓
        xhr.open('GET',"http://iris4h-fhirapi.japaneast.cloudapp.azure.com:52773/csp/sqltofhir/fhir/observation/pulserate/1001",true); // RESTのGETコマンドで、リソースを指定する↓
        xhr.send();  // サーバに送る↓

        xhr.onreadystatechange = function(){ // サーバから応答があったときに実行される箇所↓
                if (xhr.readyState==4 && xhr.status==200) { // readyState=4はDONE（操作完了）かつstatus=200はOK（要求通りの処理がされた）↓
        var data = JSON.parse(xhr.responseText); // !!重要 JSON形式を構造体として解釈する これ以降はJSONタグをプロパティとして記述できる↓
        //追加した部分↓
        //変更した部分↓
        var outtext = xhr.responseText + '<br> patient id =' + data.id + '<br>'; // data.idと記述されている"id"はJSON内のidタグを指す↓
        //変更した部分↓

        target.innerHTML = outtext; //読み取ったタグの値をHTMLとして出力↓
                }↓
        };↓

        }↓

  </script>↓

</head>↓

<br>↓
<br>↓

<body>↓
  <form name="form1" id="id_form1" action="">↓
    <input name="textBox1" id="id_textBox1" type="text" value="" />↓
    <input type="button" value="Exec" onclick="onButtonClick();" />↓
  </form>↓
  <hr/>↓

<br>↓
  <div id="output"></div>↓
<br>↓
↓
</body>↓
</html>↓
```

①XMLHttpRequestのインスタンスを作成
②テキスト形式を指定
③GET/REST構文で欲しいリソースを指定
④構文をサーバに送信

①応答が200(正常)であることを条件とし割り込み実行
②変数名dataに応答テキストをJSON形式でパース入力

①例えばJSONのラベル内に"id"が存在する場合、
これをdata.idのようにプロパティとして呼び出せる
→FHIRで医療情報のラベル名が固定されることで、
コードの再利用が非常に楽になることに注目する

読み出したファイルの表示位置

4：Graph.jsライブラリを参照する

Graph.jsの元コードでは、<body>内に<script>タグを置き、その中でChartインスタンスを生成しながらデータを割り当てている

```
<script>
var ctx = document.getElementById('myChart').getContext('2d');
var myChart = new Chart(ctx, {
  type: 'line',
  data: {
    labels: ['M', 'T', 'W', 'T', 'F', 'S', 'S'],
    datasets: [{
      label: 'apples',
      data: [12, 19, 3, 17, 6, 3, 7],
      backgroundColor: "rgba(153,255,51,0.4)"
    }, {
      label: 'oranges',
      data: [2, 29, 5, 5, 2, 3, 10],
      backgroundColor: "rgba(255,153,0,0.4)"
    }]
  }
});
</script>
```

データの内容はJSON形式で記述されていることがわかる

→何らかの構造体データを作り、それをJSON化するか、またはJSONフォーマット自体を生成して自動化

4：Graph.jsライブラリを参照する

```
function onShowChart(){
var chartheader = document.createElement("script");
chartheader.innerHTML = "var ctx = document.getElementById('myChart').getContext('2d');" +
                        'var myChart = new Chart(ctx, {' +
                        "type: 'line',"+
                        'data: {' +
                        "labels: ['M', 'T', 'W', 'T', 'F', 'S', 'S']," +
                        "datasets: [{" +
                        "label: 'apples',"+
                        'data: [12, 19, 3, 17, 6, 3, 7],'+
                        'backgroundColor: "rgba(153,255,51,0.4)"' +
                        '}, {' +
                        "label: 'oranges'," +
                        'data: [2, 29, 5, 5, 2, 3, 10],' +
                        'backgroundColor: "rgba(255,153,0,0.4)"' +
                        '}]' +
                        '}' +
                        '});';
  target = document.getElementById("outchartscript");
  target.appendChild(chartheader);
}
</script>
</head>
<br>
<br>
<body>
  <form name="form1" id="id_form1" action="">
    <input name="textBox1" id="id_textBox1" type="text" value="" />
    <input type="button" value="Exec" onclick="onButtonClick();" />
  </form>
  <form name="form2" id="id_form2" action="">
    <input name="textBox2" id="id_textBox2" type="text" value="" />
    <input type="button" value="Exec" onclick="onShowChart();" />
  </form>
<br>
  <div id="output"></div>
<br>
  <a href="/contact.html" class="btn">これはボタンです。</a>
<br>
</body>
  <body>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.1.4/Chart.min.js"></script>
    <canvas id="myChart"></canvas>
    <div id="outchartscript"></div>
  </body>
```

Chartの実体

ボタンと実装例

Graphタグ：この中にGraph.jsの実体を記述する

4 ：Graph.jsライブラリを参照する

サンプル

```
function onShowChart(){
var chartheader = document.createElement("script");
chartheader.innerHTML =  var ctx = document.getElementById('myChart').getContext('2d');" +
                    'var myChart = new Chart(ctx, {' +
                    "type: 'line'," +
                    'data: {' +
                    "labels: ['M', 'T', 'W', 'T', 'F', 'S', 'S']," +
                    "datasets: [{" +
                    "label: 'apples'," +
                    'data: [12, 19, 3, 17, 6, 3, 7],' +
                    'backgroundColor: "rgba(153,255,51,0.4)"' +
                    '}, {' +
                    "label: 'oranges'," +
                    'data: [2, 29, 5, 5, 2, 3, 10],' +
                    'backgroundColor: "rgba(255,153,0,0.4)"' +
                    '}]' +
                    '}' +
                    '});';
  target = document.getElementById("outchartscript");
  target.appendChild(chartheader);
}
```

出力自体がJavascriptとして動くことを指定

X-Yを指定した2次元グラフ

折れ線グラフ

X軸の要素ラベル

Y軸 1つ目のの数値列
表示色の指定（黄緑）

Y軸 2つ目のの数値列
表示色の指定（オレンジ）

出力自体がJavascriptとして記述されることを指定

5 ：条件式を用いて取り出す

JSON構造体を単純に読み出す関数（JSON配列の再帰呼び出しのアルゴリズムを理解する好例）

```
function plotres(response, prefix) {
        for (var key in response){
        if (typeof response[key] == "object") {
        if(Array.isArray(response[key])) {
                // 配列の場合は forEach で要素ごとにに再帰呼び出し
                response[key].forEach(function(item){
                plotres(item, prefix+" "+key) ;
        });
        } else {
                // 連想配列はそのまま再帰呼び出し
        plotres(response[key], prefix+" "+key) ;
        }
        } else {
                // 配列や連想配列でなければキーの値を表示
                console.log(prefix+" "+key+": "+response[key]);
                }
         }
}
```

6 ：Graph.jsのデータ入力形式に合わせてデータフォーマットを作成する

```
function onButtonClick() {

  var linktext5 = "http://hapi.fhir.org/baseDstu3/Observation?patient=1723865&_pretty=true";
  target = document.getElementById("output");

  var xhr = new XMLHttpRequest();
  xhr.responseType = 'text';
  xhr.open('GET',linktext5,true);
  xhr.send();

  let insertLabels = '';
  let insertDatas = '';

  xhr.onreadystatechange = function(){
      if (xhr.readyState==4 && xhr.status==200) {

  var data = JSON.parse(xhr.responseText);
  //要素の求め方 data['entry'].length

  var extractdata;
  extractdata += '<br>'

      for (  var i in data.entry  ) {
          if (data.entry[i].resource.code.text == 'Heart Rate'){
              insertLabels += "'" + data.entry[i].resource.effectiveDateTime + "'";
              insertDatas += "'" +  data.entry[i].resource.valueQuantity.value + "'";
              if(i < (data['entry'].length - 1)){
                  insertLabels += ',';
                  insertDatas += ',';
              }
          }
          extractdata +=  data.entry[i].resource.id + '-' + data.entry[i].resource.code.text + "-" + data.entry[i].resource.valueQuantity.value +  '<br>';  //['resource']['medicationCodeableConcept']['text'];
      }

  var outtext =  '<br> patient id =' + data.entry[0].resource.subject.reference + '<br> elements.No = ' + data['entry'].length + '<br>' +extractdata  + '<br>' + xhr.responseText;
  target.innerHTML = outtext;

  plotres(data, '');
  var insertLabel;
  var insertData;
  //日付
  insertLabel = "labels: [" + insertLabels + "],"; 
  //内容
  insertData = 'data: [' + insertDatas + '],'; 

  console.log(insertLabels);
  console.log(insertDatas);

  onShowChart2(insertLabel, insertData);
  onShowChart3(insertLabel, insertData);

      }
};

}
```

FHIRデータのcode.textから"Heart Rate"に該当する場合にその内容を読み込み
→他のデータに応用可能

7 ：Graph.jsの表示フォーマットを整える

表示項目を追加する方法（このフォーマット自体をライブラリ化してもよい）

```
var extractdata;
extractdata += '<br>'

    for (  var i in data.entry  ) {
        if (data.entry[i].resource.code.text == '体温'){
            insertLabels += "'" + data.entry[i].resource.effectiveDateTime + "'";
            insertDatas += "'" +  data.entry[i].resource.valueQuantity.value + "'";
            if(i < (data['entry'].length - 1)){
                insertLabels += ',';
                insertDatas += ',';
            }
        }
        if (data.entry[i].resource.code.text == 'Respiratory Rate'){
            insertLabels2 += "'" + data.entry[i].resource.effectiveDateTime + "'";
            insertDatas2 += "'" +  data.entry[i].resource.valueQuantity.value + "'";
            if(i < (data['entry'].length - 1)){
                insertLabels2 += ',';
                insertDatas2 += ',';
            }
        }
        extractdata +=  data.entry[i].resource.id + '-' + data.entry[i].resource.code.text + "-" + data.entry[i].resource.valueQuantity.value +  '<br>';  //['resource']['medicationCodeableConcept']['text'];
    }

var outtext =  '<br> patient id =' + data.entry[0].resource.subject.reference + '<br> elements.No = ' + data['entry'].length + '<br>' +extractdata  + '<br>' + xhr.responseText;
target.innerHTML = outtext;
```

体温と記述されたテキストを読み込み

Respiratory Rateと記述されたテキストを読み込み

7：Graph.jsの表示フォーマットを整える

異なるグラフを表示する方法

```
function onShowChart2(insertLabel, insertData){

  var chartheader = document.createElement("script");
  chartheader.innerHTML = "var ctx = document.getElementById('myChart').getContext('2d');" +
                          'var myChart = new Chart(ctx, {' +
                          "type: 'bar'," +
                          'data: {' +
                        //このラベル領域を差し替える
                          insertLabel +
                          "datasets: [{" +
                          "label: 'patient Heart Rate'," +
                        //このデータ領域を差し替える
                          insertData +
                          'backgroundColor: "rgba(153,255,51,0.4)"' +
                          '}]' +
                          ',{' +
                          '});';
  target = document.getElementById("outchartscript");
  target.appendChild(chartheader);
}

function onShowChart3(insertLabel, insertData){

  var chartheader = document.createElement("script");
  chartheader.innerHTML = "var ctx = document.getElementById('myChart2').getContext('2d');" +
                          'var myChart = new Chart(ctx, {' +
                          "type: 'line'," +
                          'data: {' +
                        //このラベル領域を差し替える
                          insertLabel +
                          "datasets: [{" +
                          "label: 'patient Respiratory Rate'," +
                        //このデータ領域を差し替える
                          insertData +
                          'backgroundColor: "rgba(255,153,0,0.4)"' +
                          '}]' +
                          ',{' +
                          '});';
  target = document.getElementById("outchartscript");
  target.appendChild(chartheader);
}
```

棒グラフの設定

折れ線グラフの設定

8：総合テスト

お疲れさまでした。
コードは動きましたか？

参考資料

サンプル