# Webプログラミング実習II　授業用レジュメ①

## ■1.WebAPI(ウェブAPI)

### ■そもそも『API』とは

『**A**pplication **P**rogramming **I**nterface』。
**あるプログラムが持つ機能や情報を他のプログラムから利用するための仕組み、もしくはアプリケーションを便利に製作できるように外部であらかじめ用意された仕組みのこと。**
非常に範囲の広い言葉なので色々なものに対して使われる（例えば、Unityの標準機能やJavaの標準ライブラリも『API』と呼ばれているし、これから説明する『**WebAPI**』も概して『API』と呼ばれる場合が多い）。

### ■『WebAPI』って？

APIの一種。**Webサーバにあるプログラム(Webアプリケーション)をインターネット経由で実行させ、その実行結果を他のプログラムなどで利用するための仕組み。**

### ■WebAPIの身近な例

**●zipcloud『郵便番号検索API』**
郵便番号から住所を探すことができるWebAPI。このカリキュラム内で利用する。
http://zipcloud.ibsnet.co.jp/doc/api

**●twitter『twitter API』**
（twitterのログイン後画面から使用申請が必要なのでURLは割愛）
このAPIを利用して、「ユーザーが持つアカウントのツイート（つぶやき）を分析して言葉遣いの傾向を調べる」のような機能を持つWebアプリケーションが多数作られている。

**●google『GoogleMaps Platform』**
googleが収集した地図の情報を使って、目的地案内やルート探索などを行うことができる。

（アカウント登録が必要）

https://developers.google.com/maps/documentation/

Google Maps Platform | Google Developers • developers.google.com

## ■WebAPIからデータを取得してみよう

前述の『郵便番号検索API』では、

```
http://zipcloud.ibsnet.co.jp/api/search?zipcode=郵便番号
```

のように検索したい郵便番号をリクエストパラメータから指定することで住所を取得できる。

試しに

```
http://zipcloud.ibsnet.co.jp/api/search?zipcode=5300045
```

↑のURLにブラウザからアクセスしてみよう。

すると、レスポンスとして

```
{
        "message": null,
        "results": [
                {
                        "address1": "大阪府",
                        "address2": "大阪市北区",
                        "address3": "天神西町",
                        "kana1": "ｵｵｻｶﾌ",
                        "kana2": "ｵｵｻｶｼｷﾀｸ",
                        "kana3": "ﾃﾝｼﾞﾝﾆｼﾏﾁ",
                        "prefcode": "27",
                        "zipcode": "5300045"
                }
        ],
```

```
        "status": 200
}
```
のようなJSON形式の文字列がブラウザに表示されるはず。

このリクエスト→レスポンスのやりとりの間に、WebAPIの内部では

**①：リクエストパラメータで受け取った郵便番号を元にデータベースを検索し、該当するデータを抜き出し**

**②：データベースから抜き出したデータをJSON形式に直す**

**③：②のJSONをレスポンスとして返す**

という処理が行われている。

WebAPIはこのように、**「クライアントのリクエストに応じてWebアプリケーションが処理を行い、処理結果としてデータが返ってくる」**ような造りになっていることが多い。

# ■2.Jsonについて

## ■JSONとは

『**J**ava**S**cript **O**bject **N**otation』。CSVなどと同様に、**いろいろなデータの集まりを文字列で書き表せるようにしたもの。**

~~※「気味の悪い拡張子」などとメディアで話題になった事もありましたね~~

「JSON形式で記述された文字列を書き出す仕組み」「JSON形式で記述された文字列を読み込む仕組み」を各プログラミング言語に持たせることによって、言語をまたいだデータのやりとりを簡単に行うことができる。

## ■JSONで扱えるデータの形式

**●文字列**

『" "』で囲んで記述する。

```
"にゃ―ん"
```

**●数値**

整数や小数。そのまま書き含める。

```
25
-5
3.14
```

**●真偽値**

trueおよびfalse。

```
true
false
```

**●null**

「何もない」。

```
null
```

## ■JSONの構成要素

**●メンバ**

**「メンバ名を表す文字列(Key)」「メンバが持つ値(Value)」のひと組を『: (コロン)』で区切ったもの。**

【記述例】

```
"name":"ミナト"
"age":23
"isSuccess":false
```

**●オブジェクト**

**メンバの集まりを『{}』で囲み、『, (カンマ)』で区切ったもの。**

**この「オブジェクト」ひとつひとつがJavaやC#の「インスタンス」に対応している。**

基本的に、このカリキュラムでは

**・JSONオブジェクトをJavaやC#の指定した型のインスタンスに変換**

**・JavaやC#のインスタンスをJSONオブジェクトに変換**

という２つの操作を適宜行うことでJSONを利用していく。

【記述例】

```
  {"name":"ミナト","age":23}
```

※オブジェクトをメンバとして別のオブジェクトの中へ含めることもできる

```
{"name":"勇者","sword":{"name":"鋼の剣","attack":12}}
```

●配列

**データやオブジェクトが集まったもの。**

『[]』で囲み、データおよびオブジェクトの間は『,』で区切る。

基本的にはメンバとしてオブジェクトの中に含める形で使用する。

【記述例】

```
{"scoreList" : [10,20,30,54,128]}
```

※オブジェクトの配列をメンバにした場合

```
{"party":[{"name":"勇者","lv": 23},{"name":"魔法使い", "lv":27},
{"name":"僧侶", "lv":22}]}
```

※配列から始まる(配列をルートとした)JSONも存在する。

この形式のJSONをUnityのJsonUtilityで扱う場合は特別な処理が必要になる

```
[10,20,30,54,128]
[{"name":"勇者","lv": 23},{"name":"魔法使い", "lv":27},{"nam
e":"僧侶", "lv":22}]
```

# ■3.JSONの使い方

JSONをプログラムの中で読み込み・書き出しするためには、それぞれの言語に用意されたJSON用のライブラリを用いることが多い。

このカリキュラム内では以降、C#(Unity)では『**JsonUtility**』、Javaでは『**Jackson**』という仕組みを用いてJSONを扱っていく。

## ■共通のきまりごと

●変数の名前は統一しておく
JSONのメンバ名と各プログラムで用いる変数名が食い違うと変換に失敗してしまう。

【例：文字列型変数"name"と整数型変数"age"をJava～C#間で相互にやりとりしたい場合】

【C#側】
```
[System.Serializable]
public Class ExampleObject
{
  public string name ;
  public int age;
}
```

↑変換↓

【JSON】
```
{"name":"ミナト","age" : 23}
```

↑変換↓

【Java側】
```
public Class ExamObj //クラス名は異なってもいい
{
  private String name;
  private int age;
  //以下にgetter `setterを記述
}
```

## ■UnityでJSONを扱うしくみ『JsonUtility』
Unityの標準機能として用意されているJSON変換プログラム(パーサ)。
Unityさえあればすぐ使える上、導入の手間が要らない。

**■JsonUtilityを扱うために必要なもの**

**●JSONの変換元、変換先となるクラス**

『JsonUtility』を用いたJSONファイルの書き出し、JSONファイルからの読み込みに使うクラスは、おおまかに言って

**・条件①：クラスに『System.Serializable』属性が付いている**
**・条件②：JSONに書き出したいフィールド変数のアクセスレベルがpublicになっている**
**・条件③：static(静的メンバ)やconst(定数)ではない**

の3つの条件を満たす必要がある(条件を満たしていない変数は)。
(他にもJSONに変換可能となる記述方法はあるが、混乱を防ぐためにここでは取り上げない)

【例】

```
//usingは省略


//【利用条件①】『System.Serializable』属性の付いたクラスである
[System.Serializable]
public Class JsonSampleForUnity()
{
  //【利用条件②】Jsonに書き出し･Jsonから読み込みしたいフィールド変数のアクセスレベルがpublicである
  public int sampleNumber = 3;
  public string sampleString = "文字列も扱えます";
  //（publicでない変数は書き出し･読み込み時に無視される）
  int ignoreValue = 150;
}
```

【補足】クラス型のフィールド変数をJSONで読み書きしたい場合は、その変数の型となるクラスも上記の利用条件を満たしていなければならない。

【利用条件を満たしていないクラス】

```
//usingは省略
```

```
public Class NotSerializableClass()
{
  public string massage = "クラスに『System.Serializable』属性が付いていないので変換不可";
}
```

【利用条件を満たしているクラス】

```
//usingは省略
[System.Serializable]
public Class SerializableClass()
{
  public string massage = "属性も付いており `変数もpublicなので変換可能"
}
```

【上記2つのクラス型のフィールド変数を持ったクラス】

```
//usingは省略

//属性も付いており `変数もpublic °一見すべての変数が変換可能に見えるが……
[System.Serializable]
public Class JsonTest()
{
  //変数のデータ型となっているクラスが利用条件を満たしていないため `無視される
  public NotSerializableClass ns_Class = new NotSerializableClass();

  //こちらは利用条件を満たしたクラスをデータ型としているため `JSONオブジェクトに変換される
  public SerializableClass s_Class = new SerializableClass();

}
```

**■『JsonUtility.ToJson』メソッドによるJSONの書き出し（インスタンス→JSON）**
**【構文】**

```
string 変数名 = JsonUtility.ToJson(jsonにしたいインスタンス);
```

正常に変換が完了していれば、string型の変数に変換後のJSON文字列が入る。

**■『JsonUtility.FromJson』を使ったJSONの読み込み（JSON→インスタンス）**
**【構文】**

```
変換後の型 変数名 = JsonUtility.FromJson<変換後の型>(JSON文字列);
```

正常に変換が完了していれば、変数にJSONから変換されたインスタンスが入る。

**■【補足説明】『JsonUtility』の弱点**
JSONファイルが『[ ]』から始まっている(配列をルートとしている)場合、上手く変換することができない(例外を吐き、nullが返ってくる)。
なので、例えば以下のようなクラス・メソッドなどを使って補助を行う必要がある。

**【参考：JsonHelper(GitHubより引用)】**

https://gist.github.com/yuiyoichi/792af1cb87011e1f648956826c0d0aa0

https://gist.github.com/yuiyoichi/792af1cb87011e1f648956826c0d0aa0

**【コードの解説】**
配列から始まるJSONを『getJsonArray<配列のデータ型>』メソッドに渡すと、そのJSONをオブジェクトから始まる形に直した上でWrapper型のインスタンスに変換。変換後の配列を戻り値として返してくれる。

※出典先のコードは戻り値を**配列型(クラス名[])**で返すようになっています。**リスト型(List<クラス名>)**の戻り値を返すように改変したものを下に記述しておきます。
**【JsonHelper(List版)】**

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System;
```

```
/// <summary>
/// クラスの配列をJsonUtilityでデシリアライズするヘルパー
/// http://forum.unity3d.com/threads/how-to-load-an-array-with-jsonutility.375735/
/// </summary>
public class JsonHelper {
    /// <summary>
    /// 配列になっているJSONデータをデシリアライズする
    /// </summary>
    /// <typeparam name="T">配列になっているデータの型</typeparam>
    /// <param name="json">json文字列</param>
    /// <returns>FromJsonで生成されたオブジェクトの配列</returns>
    public static List<T> GetJsonArray<T>(string json)
    {
        string newJson = "{ \"array\": " + json + "}";
        Wrapper<T> wrapper = JsonUtility.FromJson<Wrapper<T>>(newJson);
        return wrapper.array;
    }

    [Serializable]
    private class Wrapper<T>
    {
//『#pragma ~』は `プログラムの動きとは直接関係ない記述
// (『この番号の警告を無視しろ』というコンパイラへの命令文)
#pragma warning disable 649
        public List<T> array;
    }
}
```

【呼び出す際の構文】

```
    List<クラス名> objects = JsonHelper.getJsonArray<クラス名> (jsonString);
```

## ■JavaでJSONを扱うしくみ『Jackson』

有志が提供するJava用のJSONパーサ。
外部で作られたプログラムであるため、ライブラリの導入が必要となる。

**■Jasksonを扱うために必要なもの**

**●Jacksonライブラリ**

- 『jackson-annotations』
- 『jackson-core』
- 『jackson-databind』

の3つのjarファイルをライブラリとして導入する必要がある。

**●『ObjectMapper』クラスのインポート、例外のthrowかcatch**

JacksonでのJSON変換は同ライブラリの『ObjectMapper』クラスを使って行うため、クラス内に下記のimport文が必要となる。

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
```

また、ObjectMapperのメソッドは『JsonProcessingException』というチェック例外を出す場合があるので、throws、またはtry-catch文を記述しないとコンパイルエラーが出る。

**【throws文を使う場合】（このカリキュラムではこちらがおすすめ）**

```
//import `クラス宣言は省略
アクセスレベル 戻り値 ObjectMapperを使いたいメソッド(引数) throws JsonProcessingException{
    //ObjectMapperを使った処理内容
}
```

**【try-catch文を使う場合】**

```
//import `クラス宣言は省略
アクセスレベル 戻り値 ObjectMapperを使いたいメソッド(引数){
    try{
        //ObjectMapperを使った処理内容
    }catch(JsonProcessiongExcaption e){
        e.printStackTrace();
    }
}
```

**●JSONの変換元、変換先となるクラス**
『Jackson』を用いたJSONファイルの書き出し、JSONファイルからの読み込みに使うクラスは、おおまかに
**①：フィールド変数全てがカプセル化されている**
**②：フィールド変数に対し、命名規則に沿ったgetter、setterが用意されている**
**③：フィールド変数がstatic(静的メンバ)ではない(static変数は変換時に無視される)**
の３つの条件を満たしている必要がある。

```
//usingは省略

public Class JsonSampleForJackson()
{
  //【利用条件①】フィールド変数がカプセル化されている
  private int sampleNumber = 3;
  private String sampleString = "文字列も扱えます";

  //【利用条件②】命名規則に沿ったgetter `setterが用意されている
  public int getSampleNumber(){
    return sampleNumber;
  }
  public void setSampleNumber(int sampleNumber){
```

```
    this.sampleNumber = sampleNumber;
  }
  public String getSampleString(){
    return sampleString;
  }
  public void setSampleString(int sampleString){
    this.sampleString = sampleString;
  }
}
```

**■『ObjectMapper.writeValueAsString』メソッドによるJSONの書き出し（インスタンス→JSON）**
【構文】

```
ObjectMapper mapper = new ObjectMapper();
string 変数名 = mapper.writeValueAsString(jsonにしたいインスタンス);
```

正常に変換が完了していれば、string型の変数に変換後のJSON文字列が入る。

**■『ObjectMapper.readValue』を使ったJSONの読み込み（JSON→インスタンス）**
【構文】

```
ObjectMapper mapper = new ObjectMapper();
変換後の型 変数名 = mapper.readValue(JSON文字列,変換後の型.class);
```

正常に変換が完了していれば、変数にJSONから変換されたインスタンスが入る。

# ■4.WebAPIを作ってみよう

## ■下準備

**■MAMPの起動**
以降、作成するプログラムを動作させる際は、必ず『MAMP』を起動しておくこと。

**■(MAMP)データベースの作成**
データベース名：zipcode20190218

**■dropboxにて配布するsqlファイルのインポート**

# ■Java側プログラムの制作（動的Webプロジェクト）

**■dropboxにて配布するライブラリの導入**

●『Java用ライブラリ』フォルダの中にある全てのJarファイルをWebContent>WEB-INF>libフォルダ内にコピー＆ペースト

**■Entity(『model』パッケージに作成)**
**【ZipcodeData】：データベースから取り出したデータを格納するためのエンティティクラス**
※「JacksonでJSONの読み書きをする条件」に沿っていることに注目してください

```
package model;


public class ZipcodeData {
	private String address1;
	private String address2;
	private String address3;
	private String kana1;
	private String kana2;
	private String kana3;
	private String zipcode;



	public ZipcodeData(String address1, String address2, S
tring address3, String kana1, String kana2, String kana3,
			String zipcode) {
		this.address1 = address1;
		this.address2 = address2;
		this.address3 = address3;
		this.kana1 = kana1;
		this.kana2 = kana2;
```

```
                this.kana3 = kana3;
                this.zipcode = zipcode;
        }

        public String getAddress1() {
                return address1;
        }

        public void setAddress1(String address1) {
                this.address1 = address1;
        }

        public String getAddress2() {
                return address2;
        }

        public void setAddress2(String address2) {
                this.address2 = address2;
        }

        public String getAddress3() {
                return address3;
        }

        public void setAddress3(String address3) {
                this.address3 = address3;
        }

        public String getKana1() {
                return kana1;
```

```
	}

	public void setKana1(String kana1) {
		this.kana1 = kana1;
	}

	public String getKana2() {
		return kana2;
	}

	public void setKana2(String kana2) {
		this.kana2 = kana2;
	}

	public String getKana3() {
		return kana3;
	}

	public void setKana3(String kana3) {
		this.kana3 = kana3;
	}

	public String getZipcode() {
		return zipcode;
	}

	public void setZipcode(String zipcode) {
		this.zipcode = zipcode;
	}
}
```

■DAO(『dao』パッケージに作成)
【ZipcodeDataDAO】：DBとの接続処理を担当するDAO

```
package dao;



import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import model.ZipcodeData;

public class ZipcodeDataDAO {
        // フィールド
        // 接続するデータベースのアドレス
        private final String DB_PATH = "jdbc:mysql://localhos
t:3306/zipcode20190218?useUnicode=true&characterEncoding=utf
8";
        // データベースのユーザー名
        private final String DB_USERNAME = "root";
        // データベースのパスワード
        private final String DB_PASSWORD = "root";
        // JDBCドライバ
        private final String JDBC_DRIVER = "com.mysql.jdbc.Dri
ver";

            //コンストラクタでJDBCドライバを読み込む
            public ZipcodeDataDAO(){
                    try{
```

```
                        Class.forName(JDBC_DRIVER);
                    }catch (ClassNotFoundException e) {
                        e.printStackTrace();
                    }
            }

        public ZipcodeData findZipcodeData(String zipcode) {

                ZipcodeData result = null;
                //DAOのお約束①：try-with-resources文の中でDBへ接続
                try(Connection conn = DriverManager.getConnect
ion(DB_PATH, DB_USERNAME, DB_PASSWORD)){

                    // SQLを用意
                    String sql = "";
                    sql += " SELECT";//取り出す項目を書く
                    sql += "          pref";
                    sql += "         ,city";
                    sql += "         ,street";
                    sql += "         ,pref_kana";
                    sql += "         ,city_kana";
                    sql += "         ,street_kana";
                    sql += "         ,zipcode";
                    sql += " FROM";//どこから取り出すか書く
                    sql += "         zipcode";
                    sql += " WHERE";//取り出す条件
                    sql += "         zipcode = ?";

                    // PrepareStatemant作成
```

```
                        PreparedStatement ps = conn.prepareSta
tement(sql);
                        // PrepareStatemant.setString()でSQLの?
に値を設定
                        ps.setString(1, zipcode);
                        // 実行
                        ResultSet rs = ps.executeQuery();

                        while(rs.next()){
                                String pref = rs.getString("pr
ef");
                                String city = rs.getString("ci
ty");
                                String street = rs.getString
("street");
                                String pref_kana = rs.getStrin
g("pref_kana");
                                String city_kana = rs.getStrin
g("city_kana");
                                String street_kana = rs.getStr
ing("street_kana");
                                String zipc = rs.getString("zi
pcode");
                                result = new ZipcodeData(pref,
city, street, pref_kana, city_kana, street_kana, zipc);
                        }

                //DAOのお約束②：SQLExceptionのcatch文を記述
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return result;
```

```
        }
}
```

**■テスト用クラス(『test』パッケージに作成)**
以上の２つを作成できたら、下記のクラスを用いて動作の確認を行う

**【ZipcodeDataDAOTest】：DAOとModelの動作確認用コード**

```
package test;

import dao.ZipcodeDataDAO;
import model.ZipcodeData;

public class ZipcodeDataDAOTest {

        public static void main(String[] args) {
                //クラスをインスタンス化する
                ZipcodeDataDAO dao = new ZipcodeDataDAO();
                //メソッドを呼び出す(引数に5300045の文字列を送る)
                ZipcodeData zd = dao.findZipcodeData("530004
5");
                //メソッドが返す値を受け取りsysoutで出力する
                if(zd != null){
                        System.out.println(zd.getAddress1());
                        System.out.println(zd.getAddress2());
                        System.out.println(zd.getAddress3());
                        System.out.println(zd.getKana1());
                        System.out.println(zd.getKana2());
                        System.out.println(zd.getKana3());
                        System.out.println(zd.getZipcode());
                }
        }
```

```
}
```

以下の内容が表示されれば成功

```
大阪府
大阪市北区
天神西町
ｵｵｻｶﾌ
ｵｵｻｶｼｷﾀｸ
ﾃﾝｼﾞﾝﾆｼﾏﾁ
5300045
```

**■BO(『model』パッケージに作成)**
**【ZipcodeDataLogic】：サーブレットクラスから呼び出され、DAOに指示を出すクラス**

```
package model;

import dao.ZipcodeDataDAO;

public class ZipcodeDataLogic {

	public ZipcodeData findZipcodeData(String zipcode){
		//クラスをインスタンス化する
		ZipcodeDataDAO dao = new ZipcodeDataDAO();
		//メソッドを呼び出す(引数にzipcodeを送る)
		ZipcodeData zd = dao.findZipcodeData(zipcode);
		return zd;
	}
}
```

**■テスト用クラス２(『test』パッケージに作成)**

**【ZipcodeDataLogicTest】：BOの動作確認用コード**

```
package test;

import model.ZipcodeData;
import model.ZipcodeDataLogic;

public class ZipcodeDataLogicTest {

	public static void main(String[] args) {
		ZipcodeDataLogic bo = new ZipcodeDataLogic();
		ZipcodeData data = bo.findZipcodeData("530004
5");

		if(data != null){
			System.out.println(data.getAddress1
());
			System.out.println(data.getAddress2
());
			System.out.println(data.getAddress3
());
			System.out.println(data.getKana1());
			System.out.println(data.getKana2());
			System.out.println(data.getKana3());
			System.out.println(data.getZipcode());
		}
	}
}
```

前回と同様、下記の内容がコンソールに表示されたら成功

```
大阪府
大阪市北区
天神西町
```

```
ｵｵｻｶﾌ
ｵｵｻｶｼｷﾀｸ
ﾃﾝｼﾞﾝﾆｼﾏﾁ
5300045
```

**■サーブレット(『servlet』パッケージに作成)**

**【ZipcodeServlet】**

```
package servlet;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.fasterxml.jackson.databind.ObjectMapper;

import model.ZipcodeData;
import model.ZipcodeDataLogic;

@WebServlet("/ZipcodeServlet")
public class ZipcodeServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;

        protected void doGet(HttpServletRequest request, HttpS
ervletResponse response)
                        throws ServletException, IOException {
                //リクエストパラメータとして送られてきた郵便番号を取り出
し
```

```
                String zipcode = request.getParameter("zipcod
e");

                //BOを用いてDAOをインスタンス化｀
                //データベースからその郵便番号と対応する住所の情報を取り
出し。
                //結果をZipcodeData型変数「result」に取得
                ZipcodeData result = new ZipcodeDataLogic().fi
ndZipcodeData(zipcode);

                //取得した住所の情報が入った変数resultを｀
                //ObjectMapperのメソッドを使ってJSONに変換
                String responseJSON = new ObjectMapper().write
ValueAsString(result);

                //  レスポンスのファイル形式｀文字コードを指定
                //(これを忘れるとJSONの中の日本語が文字化けしてしまう)
                response.setCharacterEncoding("utf-8");
                response.setContentType("application/json");

                //変換したJSON文字列をレスポンスとして返す
                response.getWriter().print(responseJSON);
        }
}
```

**■サーブレットの動作確認**
動的Webプロジェクトを右クリック→「サーバーで実行」後、
GoogleChromeなどのブラウザを起動して下記のURLにアクセスする。
http://localhost:8080/zipcode/ZipcodeServlet?zipcode=5300045

ブラウザにJSON文字列が表示されたら成功

## ■Unity側プログラムの制作

■下準備
●プロジェクトの作成
・プロジェクト名：『Zipcode』
・モード：『２D』

作成開始時のおまじない→シーンの保存(シーン名は任意)を済ませたら、以下のコードをスクリプトとして作っていく

【補足説明】：『ドキュメンテーションコメント』
文中の『///』で始まるコメントは、「ドキュメンテーションコメント」と呼ばれるもの。
その中に『<summary>』(メンバの説明)『<param>』(引数の説明)『<returns>』(戻り値の説明)などの指定のタグを書き含めることで、プログラムの説明を開発環境内に表示したり外部へ文書として書き出したりすることができる。
詳しい機能説明は割愛するので、興味があれば調べてみてください。

■Entity（通常のC#クラス）
【ZipcodeData】：WebAPIから受け取ったJSONを変換し、データとして使うための受け皿となるクラス
※『MonoBehaviour』クラスを継承させないことに注意。「JsonUtilityを使う条件」を満たした通常のC＃クラスであることに注目してほしい

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[System.Serializable]
public class ZipcodeData
{
    public string address1;
    public string address2;
    public string address3;
```

```
    public string kana1;
    public string kana2;
    public string kana3;
    public string zipcode;
}
```

**■Unityのゲームオブジェクトにアタッチするスクリプト達**

**【ConnectAPI】：Webに接続し、WebAPIを呼び出すクラス**

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class ConnectAPI: MonoBehaviour {
    /// <summary>
    /// URLのルートパス + ?zipcode=
    /// </summary>
    const string URL = "http://localhost:8080/zipcode/ZipcodeS
ervlet?zipcode=";
    /// <summary>
    /// DBから取得したzipcodeの情報
    /// </summary>
    ZipcodeData zipcodeData;

    /// <summary>
    /// 取得した郵便番号に関する情報
    /// 外部から参照できるようにする
    /// </summary>
    /// <returns></returns>
```

```
    public ZipcodeData Zipcode()
    {
        return zipcodeData;
    }

    /// <summary>
    /// ネット接続用の処理
    /// Getメソッド（JSONの取得）
    /// </summary>
    /// <param name="zipnum"></param>
    /// <returns></returns>
    public IEnumerator GetZipcode(string zipnum)
    {
        UnityWebRequest request = UnityWebRequest.Get(URL + zi
pnum);

        //レスポンスが返ってくるまで実行を待つ
        yield return request.SendWebRequest();

        //エラーが起きた場合はログを返す
        if (request.isNetworkError || request.isHttpError)
        {
            Debug.Log(request.error);
            Debug.Log(request.responseCode);
        }
        //正常に接続できた場合
        else
        {
            //デバッグ用表示
```

```
            Debug.Log("GetZipcode：" + request.downloadHandle
r.text);

            zipcodeData = JsonUtility.FromJson<ZipcodeData>(re
quest.downloadHandler.text);
        }
    }
}
```

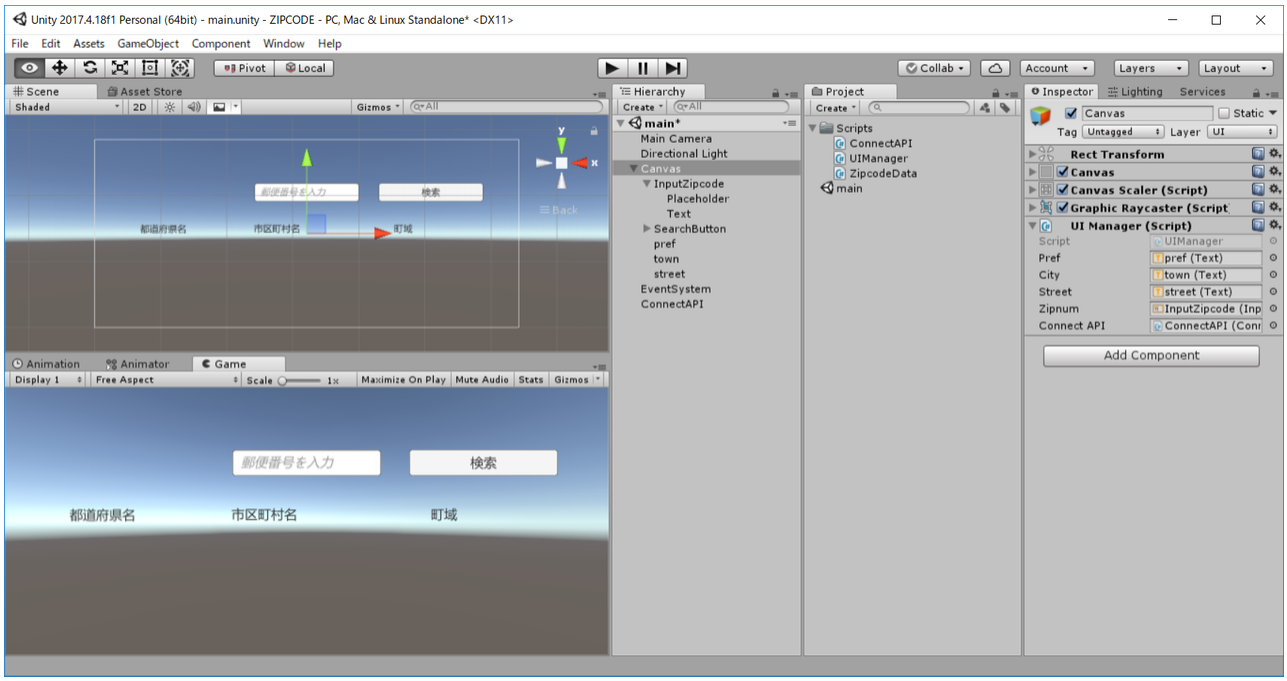
**【UIManager】：UIの表示を管理するクラス**

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class UIManager : MonoBehaviour {
    /// <summary>
    /// 都道府県名
    /// 表示用テキストボックス
    /// </summary>
    public Text pref;
    /// <summary>
    /// 市区町村名
    /// 表示用テキストボックス
    /// </summary>
    public Text city;
    /// <summary>
    /// 通り名
    /// 表示用テキストボックス
    /// </summary>
```

```
    public Text street;
    /// <summary>
    /// 郵便番号
    /// 入力用インプットボックス
    /// </summary>
    public InputField zipnum;
    /// <summary>
    /// API接続用管理クラス
    /// </summary>
    public ConnectAPI connectAPI;

    /// <summary>
    /// 郵便番号を入力欄から取得し文字列を返す
    /// </summary>
    /// <returns>文字列に変換した郵便番号</returns>
    public string Zipnum()
    {
        return zipnum.text;
    }

    /// <summary>
    /// 検索ボタン押下時
    /// </summary>
    public void OnSearchClick()
    {
        StartCoroutine(ConnectApiFindZipcodeData());
    }

    /// <summary>
    /// API接続呼出
```

```
    /// 郵便番号情報取得後画面表示
    /// </summary>
    /// <returns></returns>
    IEnumerator ConnectApiFindZipcodeData()
    {
        //郵便番号情報を取得
        //yield return でコルーチンを実行することでコルーチンの処理が終わるまで待機
        yield return StartCoroutine(connectAPI.GetZipcode(Zipnum()));
        //画面表示
        SetText(connectAPI.Zipcode());
    }

    /// <summary>
    /// 受け取ったデータを画面に表示
    /// </summary>
    /// <param name="data">郵便番号を元に取得した情報</param>
    public void SetText(ZipcodeData data)
    {
        pref.text = data.address1;
        city.text = data.address2;
        street.text = data.address3;
    }
}
```

**■Unityのエディタ上で行う作業**




## ●作成するGameObject

- 『**Canvas**』：各UIを表示する**Canvas**
  「Create」　>　「UI」　>　「Canvas」

- 『**InputZipcode**』：郵便番号の入力欄
  「Create」　>　「UI」　>　「InputField」　名前を変更　→
  『InputZipcode』

- 『**SearchButton**』：郵便番号の検索を開始するボタン
  「Create」　>　「UI」　>　「Button」　名前を変更　→　『SearchButton』

- 『**Pref**』：都道府県名の表示欄
  「Create」　>　「UI」　>　「Text」　名前を変更　→　『Pref』

- 『**City**』：市町村名の表示欄
  「Create」　>　「UI」　>　「Text」　名前を変更　→　『City』

- 『**Street**』：町域名の表示欄
  「Create」　>　「UI」　>　「Text」　名前を変更　→　『Street』

・**『ConnectAPI』：「ConnectAPI」スクリプトをアタッチするための空オブジェクト**

「CreateEmpty」　名前を変更　→　『ConnectAPI』

●スクリプトのアタッチ

・**『ConnectAPI』ゲームオブジェクトに「ConnectAPI」スクリプトをアタッチ**

・**『Canvas』に「UIManager」スクリプトをアタッチ**

→　**インスペクタービューから各public変数の内容を設定**

（変数ごとに同じ名前のゲームオブジェクトをドラッグ＆ドロップ）

・**『SearchButton』の「Button」コンポーネント内「OnClick」にイベントを追加**

→　**『Canvas』ゲームオブジェクトをドラッグ＆ドロップし、ボタンを押すと『UIManager.OnSearchClick()』メソッドが実行されるように設定**