



Microsoft<sup>®</sup>  
**ASP.net**<sup>™</sup>

# Web Making with ASP.NET

Hands on Tour  
in Hamamatsu

Microsoft Student Partners

# 自己紹介



# Yuki Ando

安藤 祐貴 (0x14歳)

Microsoft Student Partner



@m1zyuk1  
remy.slight

# 今日作るもの

- 雑談アプリ
  - 話しかけると戻ってくる
- 使うもの
  - docomo雑談対話API
  - Visual Studio
  - ASP.NET
  - C#
  - HTML, CSS, JavaScript



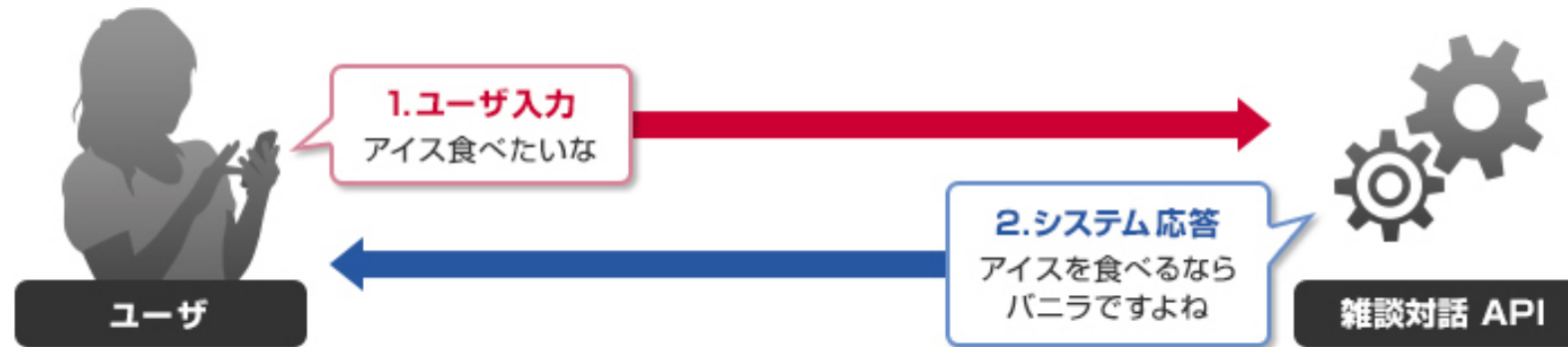
# docomo 雑談対話API

概要と使い方



- Application Program Interface
  - 命令や規約・関数等の集合
  - すでにある機能を用いる
  - Webを用いて提供される場合も(Web API)
- RESTfull API
  - Web APIの一種
  - RESTという原則に沿った設計

- ユーザの発話テキストを入力
- 入力に対して適切な応答を出力




- URLに対してデータを送信

- Type – JSON
- Required Data
  - API Key
  - utt
- Optional

context, nickname, nickname\_y, sex, bloodtype,  
birthdateY, birthdateM, birthdateD, age,  
constellations, place, mode, t

- URLに対してデータを送信
  - Type – JSON
  - Required Data
    - API Key
    - utt
  - Optional

context, nickname, nickname\_, sex, bloodtype,  
birthdateY, birthdateM, birthdateD, age,  
constellations, place, mode, t





- 必要な情報をサーバに送信
  - 認証情報(API Key)
  - 話しかけたい言葉(utt:こんにちは)
  - 他に投げたい情報(optional)
- 対話データが返ってくる
  - 決まった形(JSON)
  - システムの返答(utt:こんにちは！今日の調子はいかが？)

# ASP.NET

概要

# ASP.NETとは

- Microsoft製 Webアプリケーションフレームワーク
- .NET Framework上で動作している
- Webアプリ, Webサービスの開発・構築に
- Microsoft Azureとの連携も簡単
- 今回はMVCを使用



# ASP.NET MVCとは

- MVCパターンを使用
  - Model-View-Controllerを用いた構成
  - 詳細は手を動かしながら
- 使用可能言語
  - Visual Basic,C#,J#
  - おすすめはC#

# 雑談アプリ

Hands on

# おおまかな流れ

- プロジェクトの作成
- ASP.NET MVCの構造をしてみる
- Modelの作成
- Controllerの作成
- Viewの作成
- アプリの公開

# おおまかな流れ

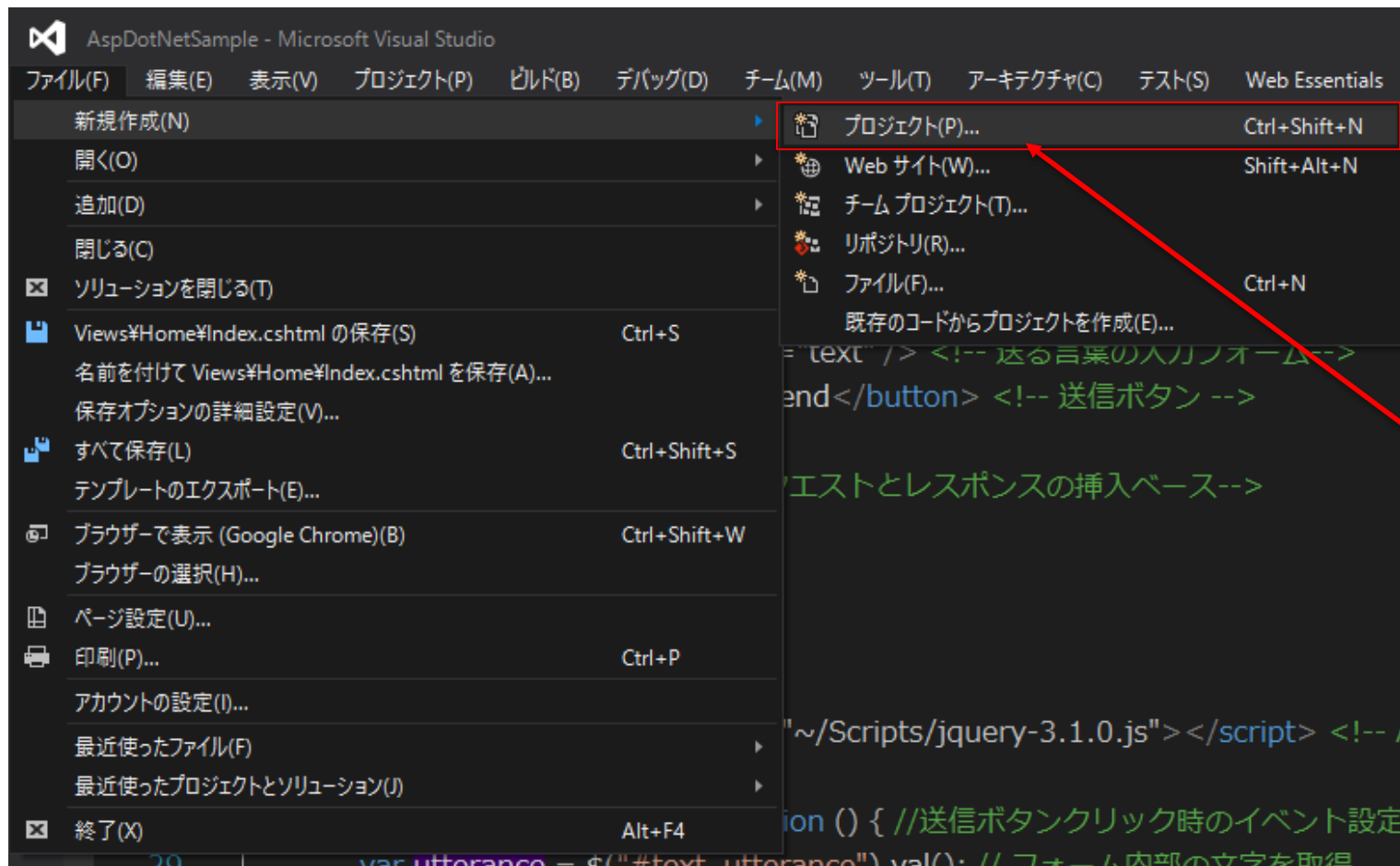
- 今日作るものの完成品
  - <https://github.com/mspjp/20160811HandsOnHamamatsu>
- ソースコードの読みにくい部分はこちらから

# おおまかな流れ

- プロジェクトの作成
- ASP.NET MVCの構造をしてみる
- Modelの作成
- Controllerの作成
- Viewの作成
- アプリの公開



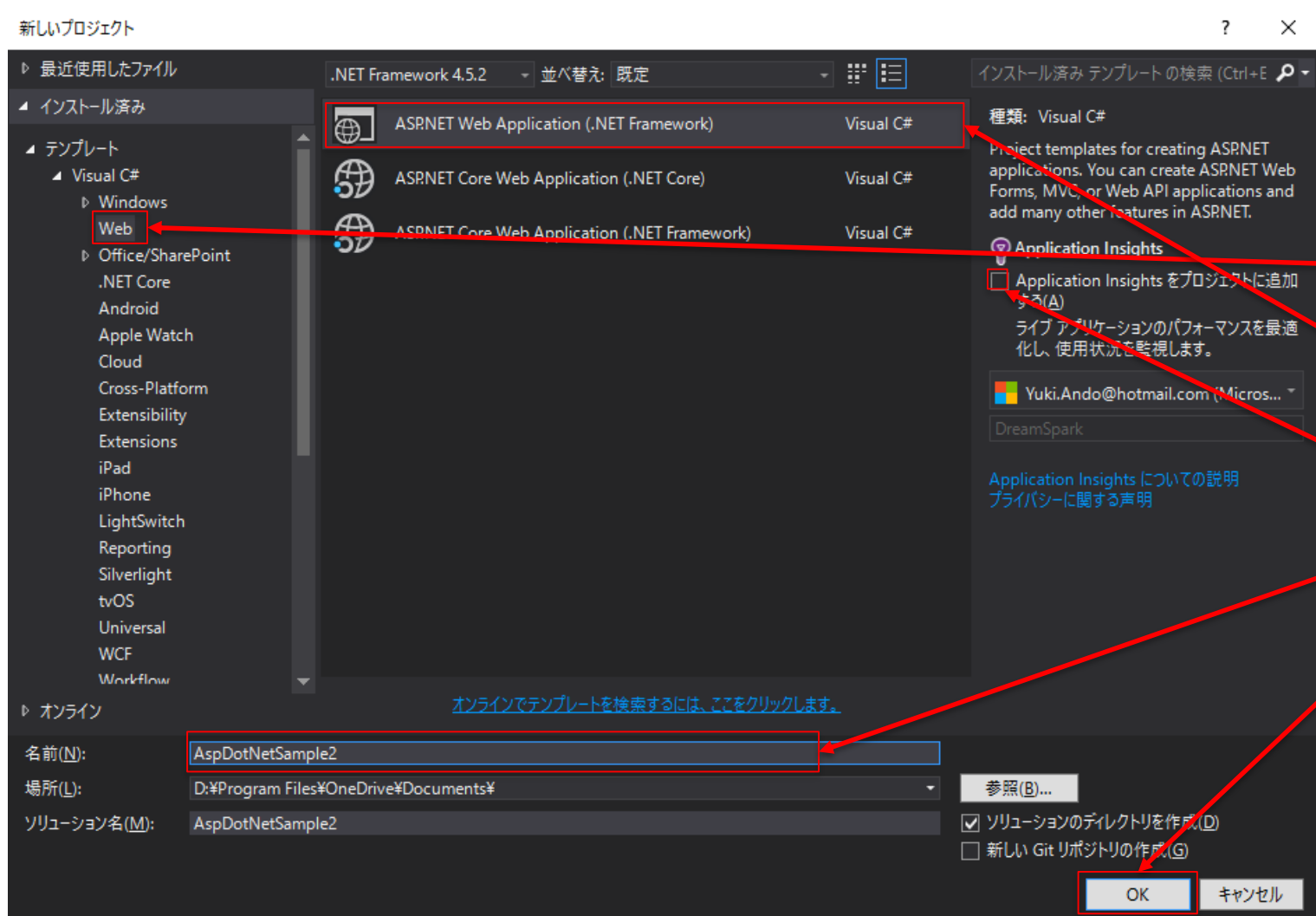
# 作業：プロジェクトの作成



1. Visual Studioを起動

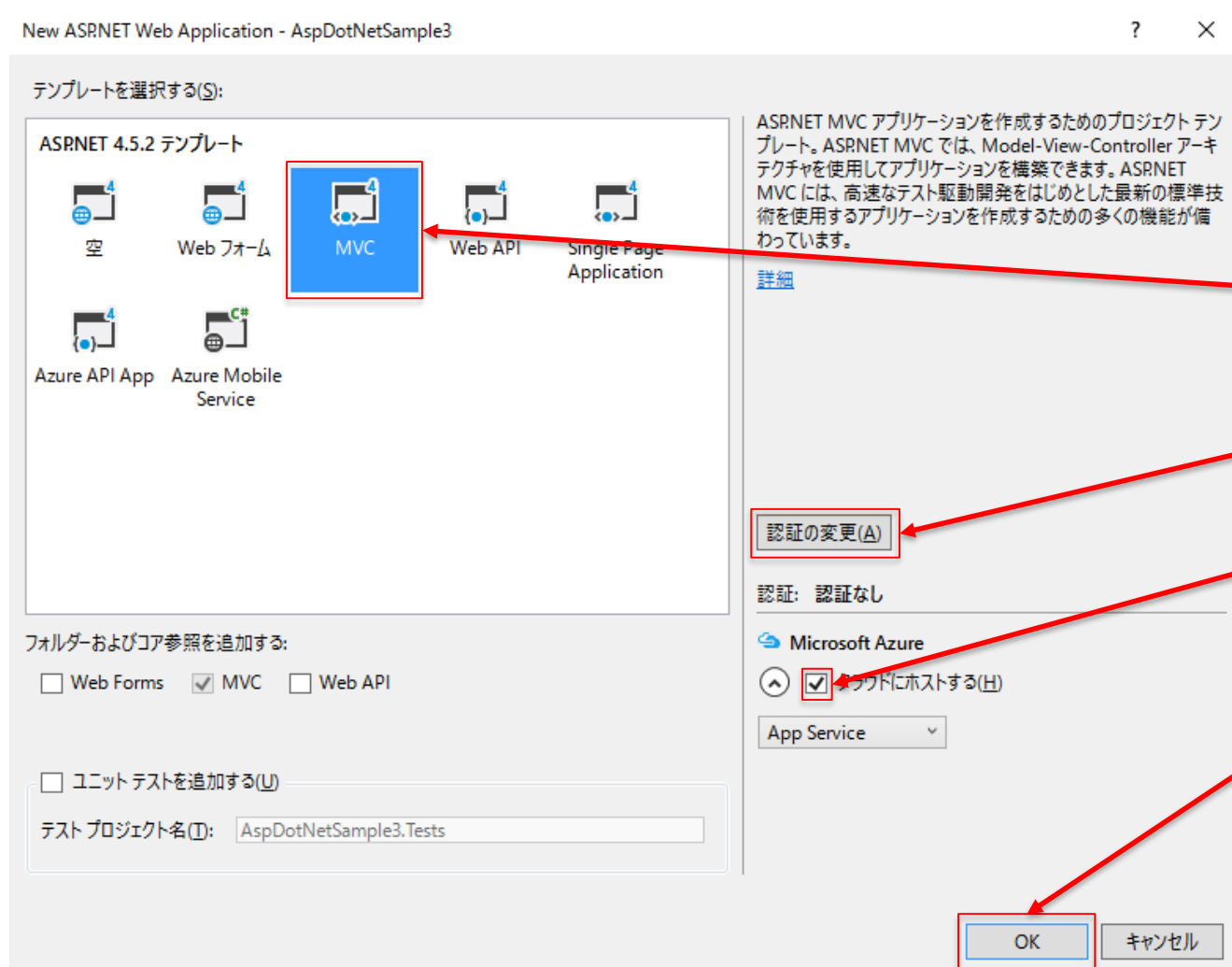
2. ファイルから新規作成  
->プロジェクトを選択

# 作業：プロジェクトの作成



1. テンプレートから Visual C# -> Webを選択
2. ASP.NET Web Application (.NET Framework)を選択
3. Application Insightsのチェックボックスを外す
4. 名前欄にソリューション名を入力
5. OKをクリック

# 作業：プロジェクトの作成



1. ASP.NET テンプレートから MVC を選択
2. 認証の変更をクリック (詳細は次スライドに)
3. クラウドにホストするのチェックボックスをチェック
4. OK をクリック

# 作業：プロジェクトの作成

認証の変更

ユーザー認証を必要としないアプリケーションの場合。  
[詳細](#)

☒ 認証なし(N)

☐ 個別のユーザー アカウント(I)

☐ 会社用および学校用のアカウント(S)

☐ Windows 認証(W)

OK キャンセル

1. 認証なしをチェック
2. OKをクリック

# 作業：プロジェクトの作成

**App Service の作成**  
Azure で Web アプリケーション、モバイル アプリケーション、REST APIなどをホストする

Microsoft アカウント  
Yuki.Ando@hotmail.com

ホスティング  
サービス

Web App の名前(M) 種類の変更 ▼  
WebApplication320160810023900

サブスクリプション(S)  
DreamSpark

リソース グループ(R)  
Default

App Service プラン(A)  
dreamspark (F1, Japan East)

【作成】 ボタンをクリックすると、次の Azure リソースが作成されます  
[他の Azure サービスの探索](#)

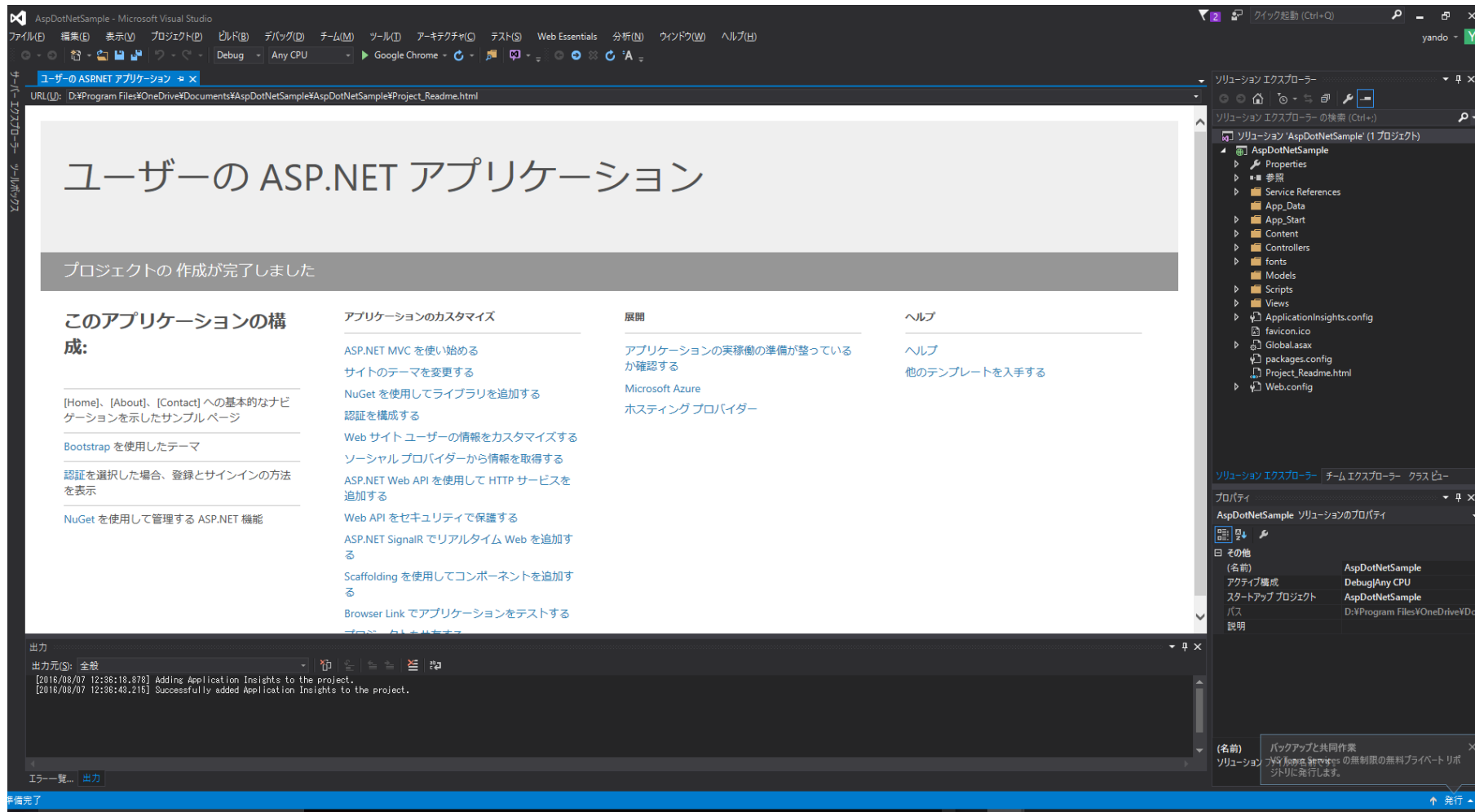
App Service - WebApplication320160810023900

支払制限を解除したが、従量制を使用している場合、追加のリソースをプロビジョニングすると料金に影響する可能性があります。 [詳細情報](#)

エクスポート(E)... 作成(C) キャンセル

1. Web Appの名前をお好きなように(重複不可) URLの先頭になります
2. サブスクリプションはDreamSparkに
3. リソースグループはDefault
4. App Serviceプランはdreamsparkに
5. 作成をクリック

# 作業：プロジェクトの作成



F5を押すと・・・

# 作業：プロジェクトの作成

[アプリケーション名](#) [ホーム](#) [詳細](#) [連絡先](#) [登録](#) [ログイン](#)

# ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

[Learn more »](#)

## Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

[Learn more »](#)

## Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

[Learn more »](#)

## Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

[Learn more »](#)

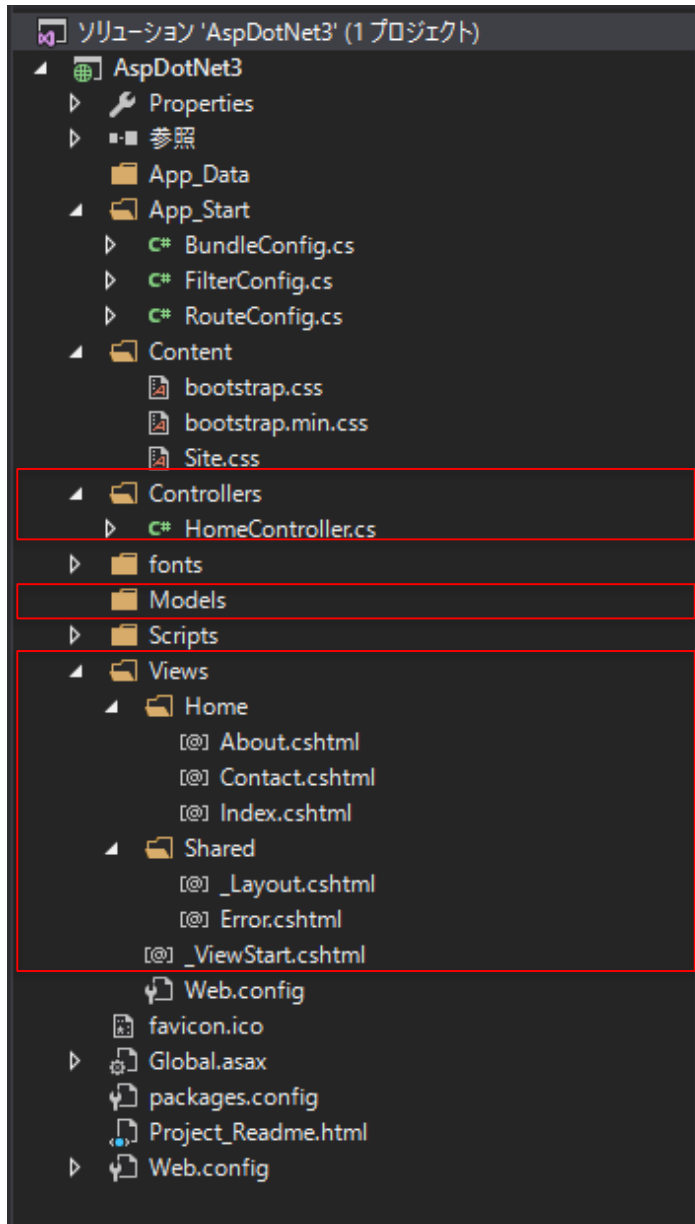
© 2016 - マイ ASP.NET アプリケーション

## 次のステップ

- プロジェクトの作成
- **ASP.NET MVCの構造をしてみる**
- Modelの作成
- Controllerの作成
- Viewの作成
- アプリの公開



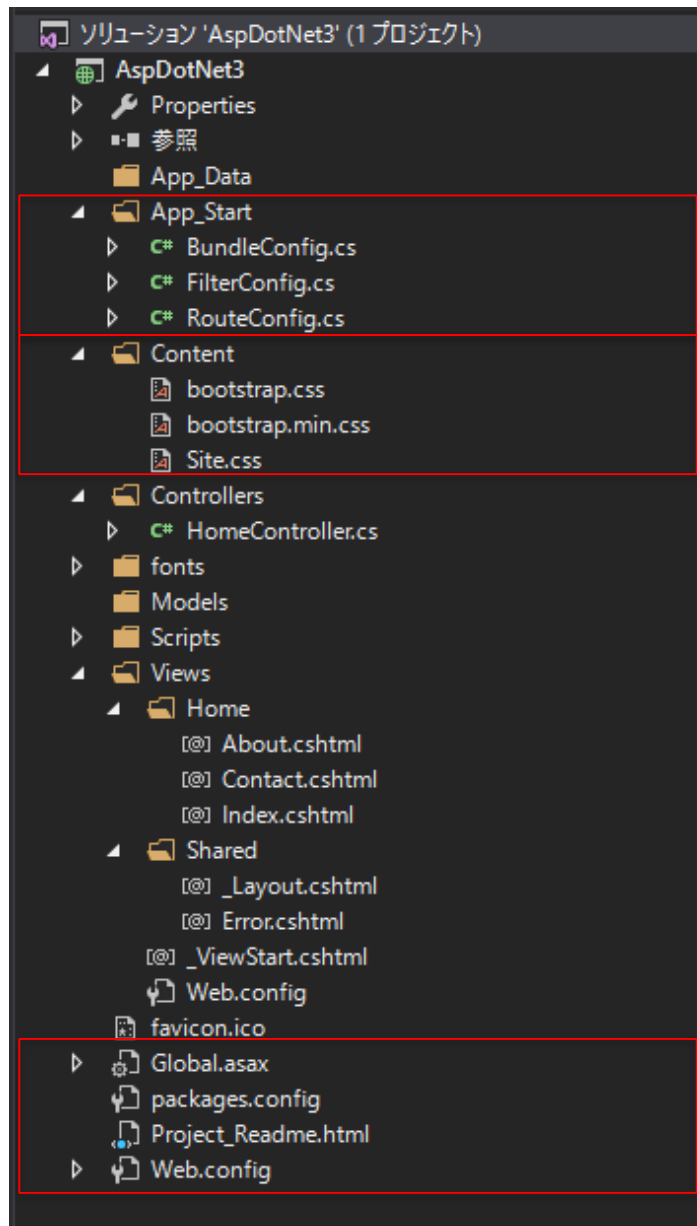
# ASP.NETのプロジェクト



最小構成にて作成した場合

- Controller
  - Homeが1つ
- Model
  - なし
- View
  - Home用のView
    - About, Contact, Index
  - Shared(共用のView)
  - \_ViewStart.cshtml(レイアウトの起点)

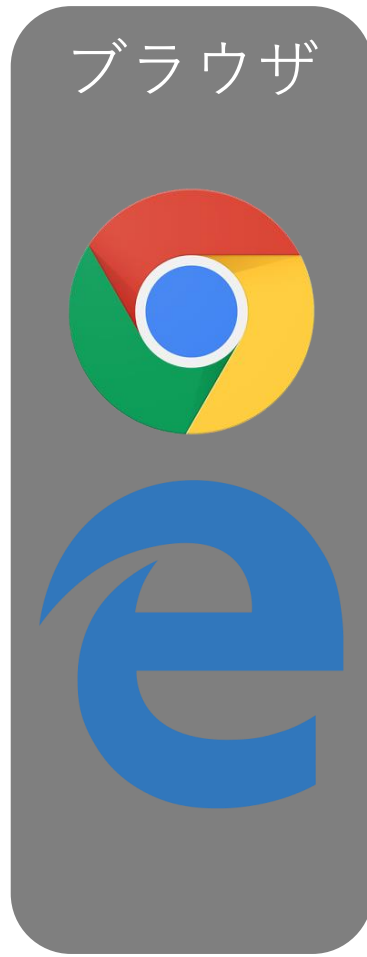
# ASP.NETのプロジェクト



最小構成にて作成した場合

- App\_Start
  - 初期設定用ファイル
- Content
  - Viewで用いる静的ファイル(CSS等)
- Webの設定
  - Global.asax(アプリ全体のイベント処理)
  - Packages.config(Nugetの管理)
  - Web.config(ASP.NETの構成設定)

# ASP.NET 動作の仕組み



URLアクセス  
Ex. /Home/Index

Viewの表示

Microsoft  
**ASP.net**

1. ルーティング解釈
  - Controller:Home
  - Method:Index
2. コントローラ実行
  - HomeControllerのIndexメソッド実行
3. Viewが返される
  - Homeフォルダ内のIndex.cshtml

# ASP.NET 動作の仕組み

```
public class HomeController : Controller
{
    0 個の参照
    public ActionResult Index()
    {
        return View();
    }

    0 個の参照
    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";

        return View();
    }

    0 個の参照
    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";

        return View();
    }
}
```

/Home/Indexに接続した場合

- Homeコントローラを確認
- Indexメソッドを見る
- 処理が行われる
  - return View()が行われる
- Views/Home/Index.cshtmlを返す
  - 命名規則より

# ASP.NET 動作の仕組み

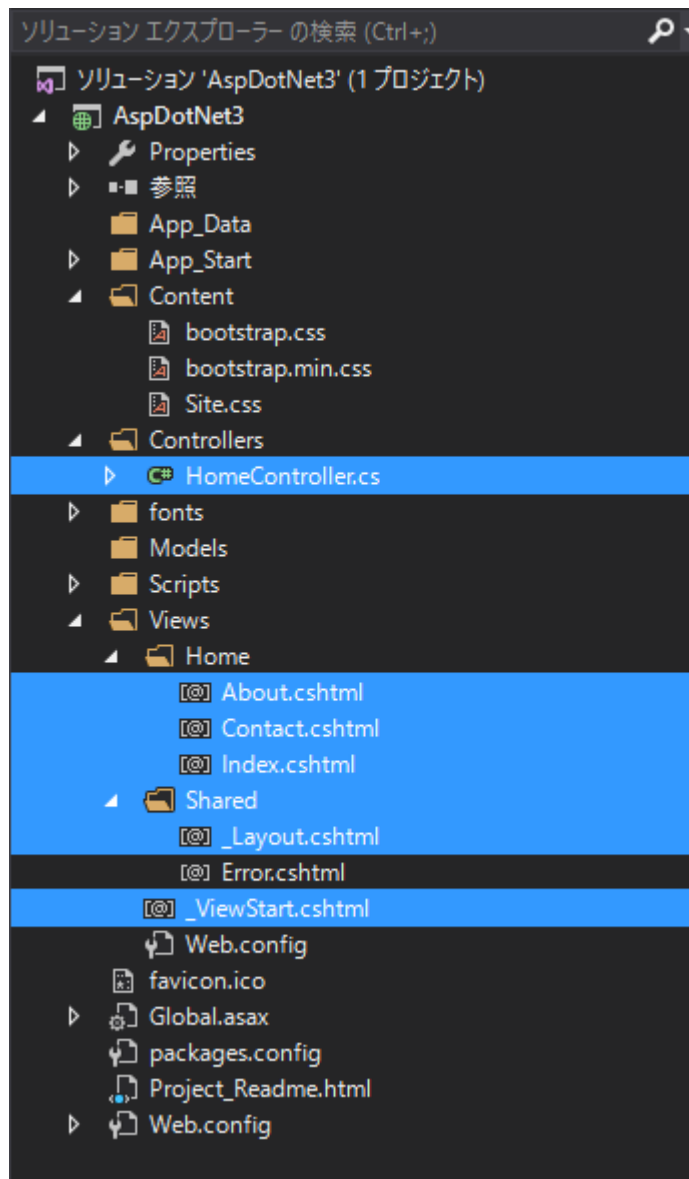
```
public class RouteConfig
{
    1 個の参照
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
        );
    }
}
```

## ルーティングの解釈

- Urlの解釈
  - /{controller}/{action}/{id}
- 未指定の場合
  - /Home/Index/

# 作業：今回使わないファイルの削除



## 削除するもの

- Controllers
  - HomeController.cs
- Views
  - Home
    - About.cshtml
    - Contact.cshtml
    - Index.cshtml
  - Shared
    - フォルダごと
  - \_ViewStart.cshtml

## 次のステップ

- プロジェクトの作成
- ASP.NET MVCの構造をしてみる
- Modelの作成
- Controllerの作成
- Viewの作成
- アプリの公開

- Model
  - そのアプリケーションが扱う領域のデータと手続きを表現する要素
  - データをとそのロジックをまとめたもの
- 今回使うもの
  - docomoAPIの型に合わせたもの2つ
    - DialogRequest.cs
    - DialogResponse.cs

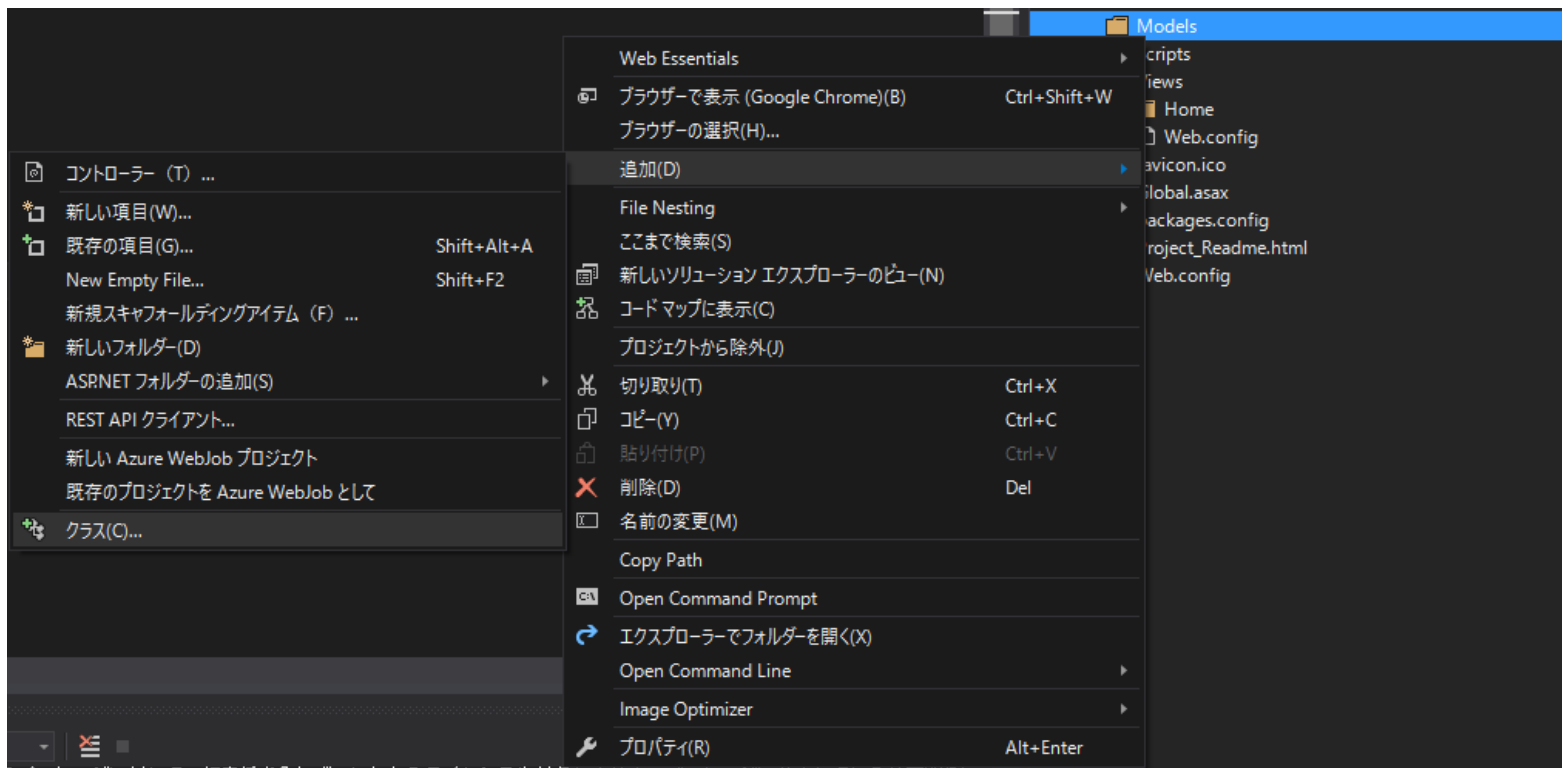


# 作業：クラスの実装

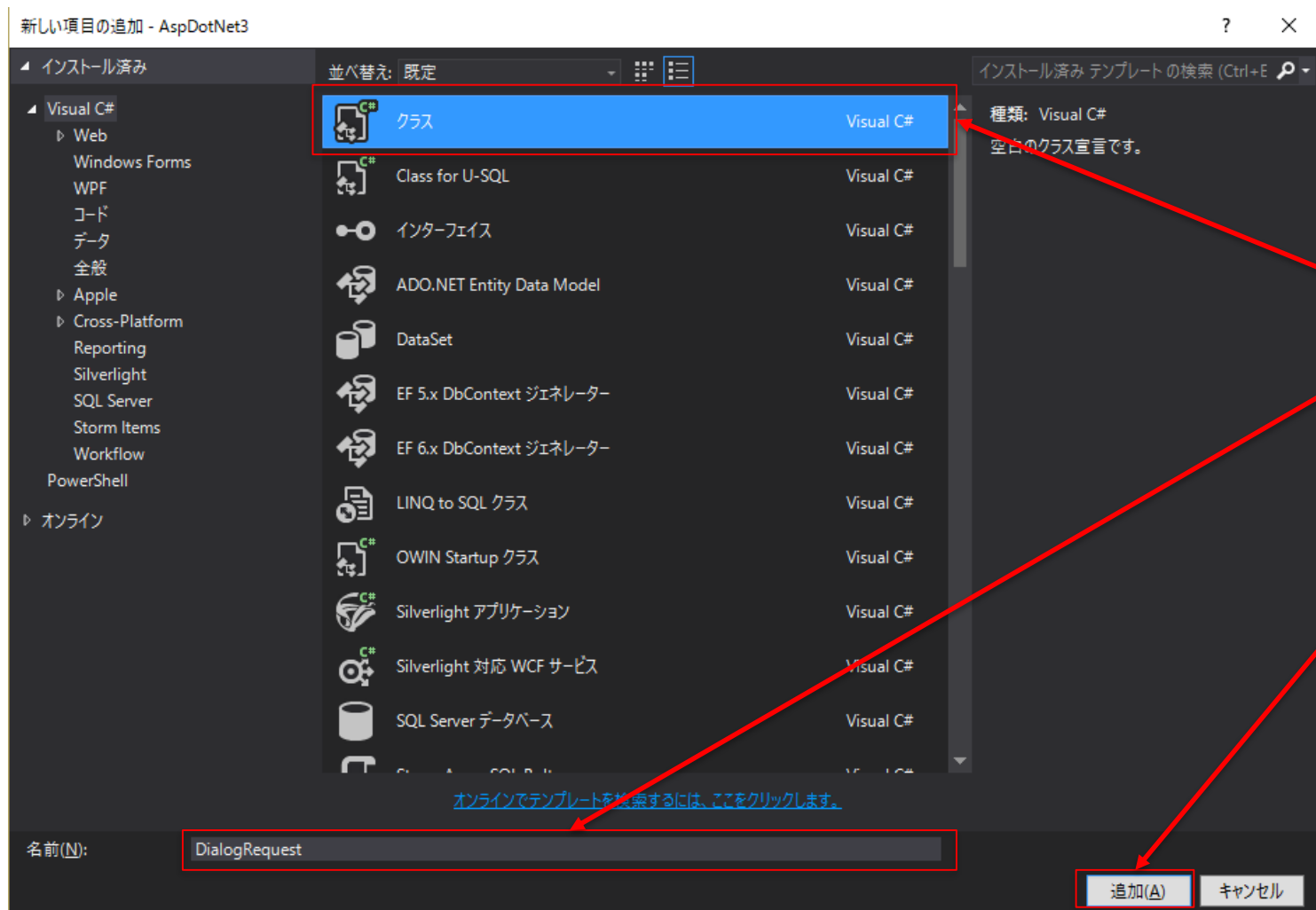
## クラスの実装

(2回やります)

- Modelsフォルダを  
右クリック
- 追加->クラス



# 作業：クラスの作成



## クラスの作成 (2回やります)

• クラスを選択

名前を入力

- DialogRequest
- DialogResponse

• 追加をクリック

# 作業：DialogRequest

```
public class DialogRequest
{
    2
    1 個の参照
    public string utt { get; set; }
    1 個の参照
    public string context { get; set; }
    0 個の参照
    public string nickname { get; set; }
    0 個の参照
    public string nickname_y { get; set; }
    0 個の参照
    public string sex { get; set; }
    0 個の参照
    public string bloodtype { get; set; }
    0 個の参照
    public string birthdateY { get; set; }
    0 個の参照
    public string birthdateM { get; set; }
    0 個の参照
    public string birthdateD { get; set; }
    0 個の参照
    public string age { get; set; }
    0 個の参照
    public string constellations { get; set; }
    0 個の参照
    public string place { get; set; }
    0 個の参照
    public string mode { get; set; }
}
```

- ソースコードの記入
  - docomo APIのフォーマット
  - 送信用データ

# 作業：DialogResponse

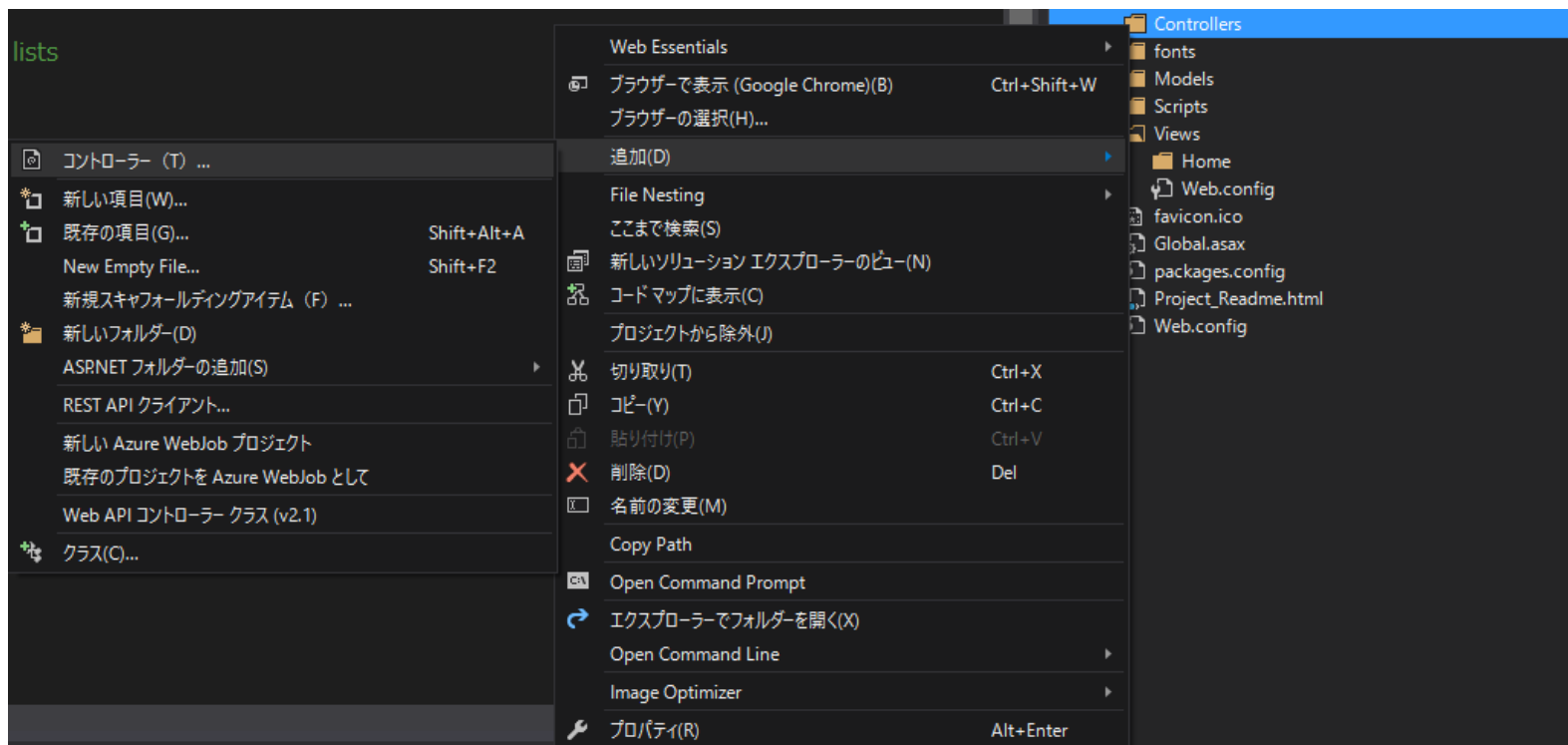
```
public class DialogResponse
{
    1 個の参照
    public string utt { get; set; }
    0 個の参照
    public string yomi { get; set; }
    0 個の参照
    public string mode { get; set; }
    0 個の参照
    public string da { get; set; }
    1 個の参照
    public string context { get; set; }
}
```

- ソースコードの記入
  - docomo APIのフォーマット
  - 受信用データ

## 次のステップ

- プロジェクトの作成
- ASP.NET MVCの構造をしてみる
- Modelの作成
- Controllerの作成
- Viewの作成
- アプリの公開

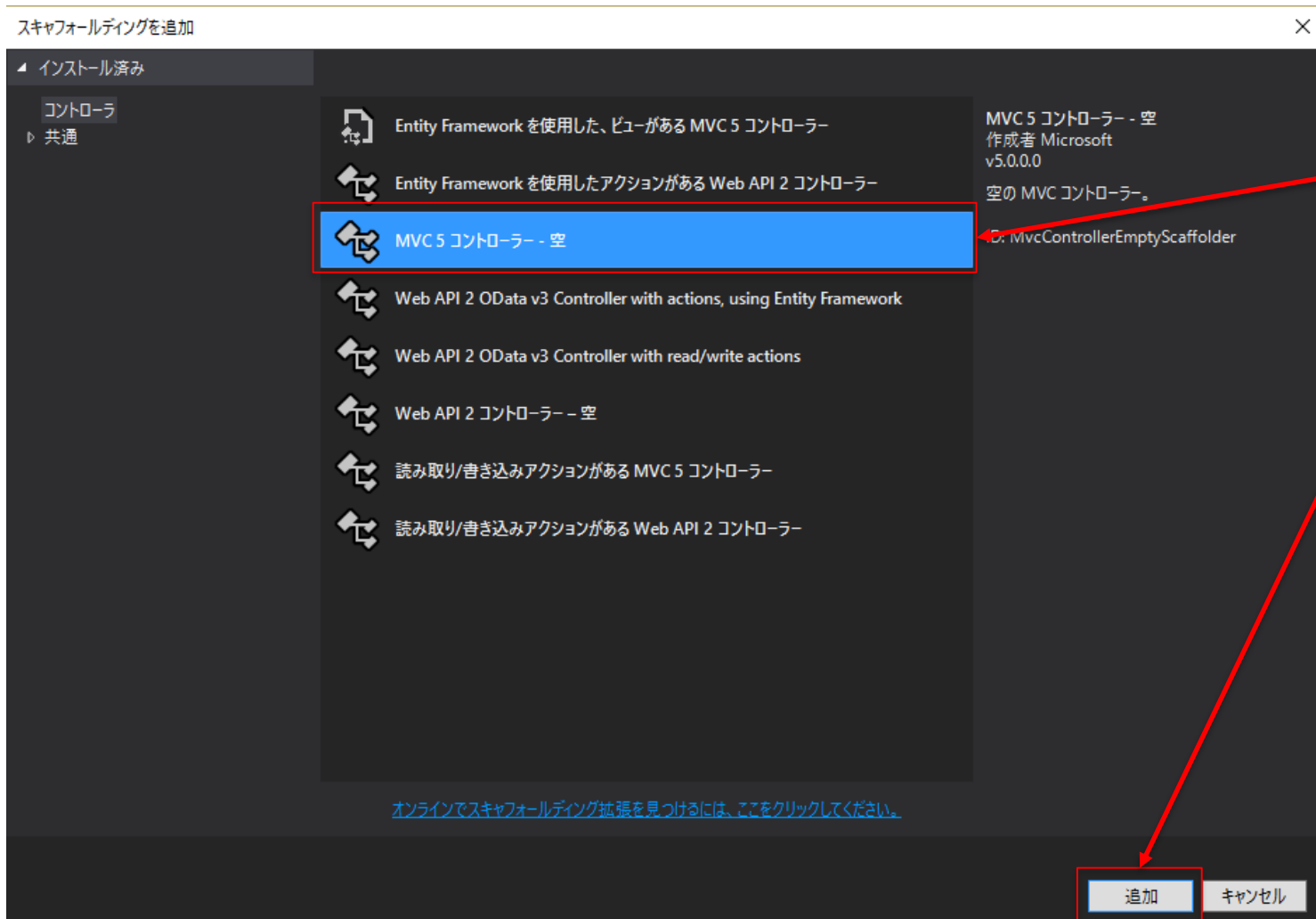
# 作業：コントローラの作成



コントローラの作成  
Controllers フォルダを  
右クリック

- 追加->コントローラ

# 作業：コントローラの作成



## コントローラの作成

MVCコントローラ-空を選択

追加をクリック

# 作業：コントローラの作成

## コントローラの作成

コントローラの追加

コントローラ名(C):

HomeControllerと入力

追加をクリック



# 作業：コントローラの書き換え

## コントローラの作成

- ソースコードの編集\_1
  - Global変数を1つ追加

```
0 個の参照 | Mizune、1 日前 | 2 人の作成者、3 件の変更  
public class HomeController : Controller  
{  
    static string previousContext = "";  
  
    // /Home/Index にアクセスが来たとき  
0 個の参照 | garicchi、4 日前 | 1 人の作成者、2 件の変更  
    public ActionResult Index()  
    {  
        // /Views/Home/Index.cshtmlを返す  
        return View();  
    }  
}
```

# 作業：コントローラの書き換え

[HttpGet]

0 個の参照 | Mizune, 1 日前 | 2 人の作成者, 3 件の変更

public ActionResult Api(string utterance)

{

```
// Docomo雑談API用の鍵
string apiKey = "6863673132504d57574b32583564796867516a4b624630386d61486e4163646c5442";
// apiKeyを含んだREST API用URL
string url = "https://api.apigw.smt.docomo.ne.jp/dialogue/v1/dialogue?APIKEY=" + apiKey;
// URLに送るPOSTパラメータクラスのインスタンス化
DialogRequest requestParam = new DialogRequest();
// 前回のデータをcontext変数に代入
requestParam.context = previousContext;
// 送る会話の内容をutt変数に代入
requestParam.utt = utterance;

// 作成したデータをstring型にシリアライズ(変換)
string requestJson = JsonConvert.SerializeObject(requestParam);
// 作成したデータをエンコード
StringContent requestContent = new StringContent(requestJson, Encoding.UTF8, "application/json");

// HttpClient(ネットコネクション)をインスタンス化
HttpClient client = new HttpClient();
// 非同期通信を行いデータを送信し,DocomoAPIの返回值を取得
var response = client.PostAsync(url, requestContent).Result;

// 返回值の内容を読み込む
var responseJson = response.Content.ReadAsStringAsync().Result;
// 読み込んだデータをクラス型に変換
DialogResponse responseParam = JsonConvert.DeserializeObject<DialogResponse>(responseJson);
// Viewで使えるようJsonを用意
JsonResult result = new JsonResult();
// JsonにDocomoから帰ってきたデータのうち uttを格納
result.Data = new { utterance = responseParam.utt };
// Viewでgetで取れるように(セキュリティ)変更
result.JsonRequestBehavior = JsonRequestBehavior.AllowGet;
// グローバル変数に今回の通信内容を保存
previousContext = responseParam.context;

// Jsonを返す
return result;
```

}

## コントローラの作成

- ソースコードの編集\_2
  - Api関数の追加
    - JavaScriptから呼び出す

# 作業：コントローラの書き換え

```
public ActionResult Api(string utterance)
{
    // Docomo雑談API用の鍵
    string apiKey = "6863673132504d57574b32583564796867516a4b624630386d61486e4163646c5442446438787731746c37";
    // apiKeyを含んだREST API用URL
    string url = "https://api.apigw.smt.docomo.ne.jp/dialogue/v1/dialogue?APIKEY=" + apiKey;
    // URLに送るPOSTパラメータクラスのインスタンス化
    DialogRequest requestParam = new DialogRequest();
    // 前回のデータをcontext変数に代入
    requestParam.context = previousContext;
    // 送る会話の内容をutt変数に代入
    requestParam.utt = utterance;
}
```

# 作業：コントローラの書き換え

```
// 作成したデータをstring型にシリアルイズ(変換)
string requestJson = JsonConvert.SerializeObject(requestParam);
// 作成したデータをエンコード
StringContent requestContent = new StringContent(requestJson, Encoding.UTF8, "application/json");

// HttpClient(ネットコネクション)をインスタンス化
HttpClient client = new HttpClient();
// 非同期通信を行いデータを送信し,DocomoAPIの返り値を取得
var response = client.PostAsync(url, requestContent).Result;

// 返り値の内容を読み込む
var responseJson = response.Content.ReadAsStringAsync().Result;
```

## 作業：コントローラの書き換え

```
// 読み込んだデータをクラス型に変換
DialogResponse responseParam = JsonConvert.DeserializeObject<DialogResponse>(responseJson);
// Viewで使えるようJsonを用意
JsonResult result = new JsonResult();
// JsonにDocomoから帰ってきたデータのうち uttを格納
result.Data = new { utterance = responseParam.utt };
// Viewでgetで取れるように(セキュリティ)変更
result.JsonRequestBehavior = JsonRequestBehavior.AllowGet;
// グローバル変数に今回の通信内容を保存
previousContext = responseParam.context;
// Jsonを返す
return result;
}
```

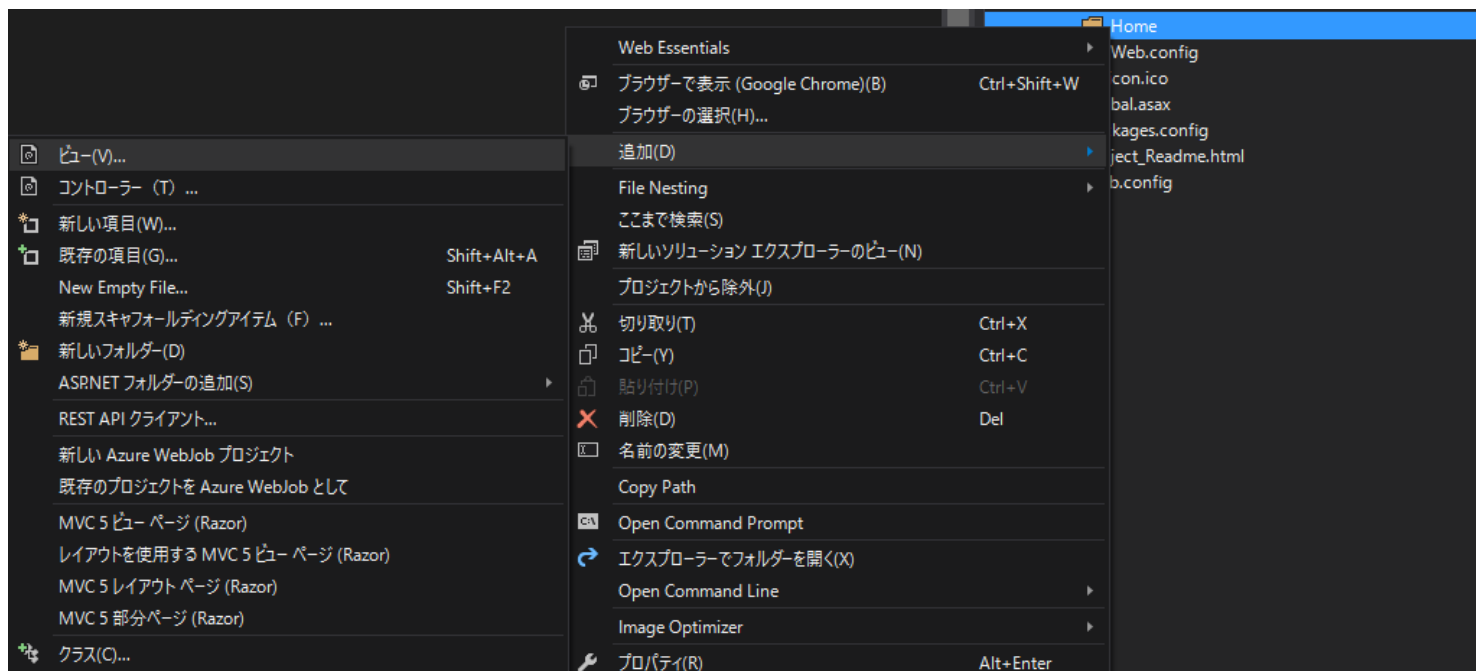
# 今回のコントローラの役割

- Index関数
  - Home/Indexに接続した際
  - Viewを返す
- Api関数
  - docomoApiと通信してデータを取得する
  - 取得したデータをJSONで返す

## 次のステップ

- プロジェクトの作成
- ASP.NET MVCの構造をしてみる
- Modelの作成
- Controllerの作成
- Viewの作成
- アプリの公開

# 作業：ビューの作成



## ビューの作成

- Home フォルダを  
右クリック
- 追加->ビュー



# 作業：ビューの作成

ビューの追加

ビュー名(N):

テンプレート(T):

モデルクラス(M):

オプション:

☐ 部分ビューとして作成(C)

☒ スクリプトライブラリの参照(R)

☐ レイアウトページの使用(U):

(Razor\_viewstart ファイルで設定する場合は空のままにしてください)

## ビューの作成

- ビュー名を"Index"に変更
- テンプレートはEmptyを選択
- レイアウトページの使用のチェックを外す
- 追加をクリック

# 作業：Viewの作成

```
@{  
    Layout = null;  
}  
  
<!DOCTYPE html>  
  
<html>  
  <head>  
    <meta name="viewport" content="width=device-width" />  
    <title>Index</title>  
  </head>  
  <body>  
    <div>  
    </div>  
  </body>  
</html>
```

生成されたView

# 今回Viewで使う機能

- Controllerのビュー呼び出し機能
  - Routingに沿ってビューを返す
  - Ex. `Home`コントローラの`Index`メソッド
    - `return View()`
    - `-> Home/Index.cshtml`が返ってくる
- JavaScript
  - Ajax

- Ajaxとは
  - 対話型Webアプリケーションの実装形態
  - JavaScriptのHTTP通信機能を使用
  - 非同期でサーバとデータのやり取りを行う
- Ex. Twitter
  - 画面をリロードせずにツイートが流れてくる

# 作業：ビューの書き換え

```
@{
    // レイアウト一括管理せず
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title></title>
    <link rel="stylesheet" href="~/Content/bootstrap.css" />
    <link rel="stylesheet" href="~/Content/Site.css" />
</head>
<body>
    <div><!-- 基本UI -->
        <input id="text_utterance" type="text" /> <!-- 送る言葉の入力フォーム-->
        <button id="button_request">Send</button> <!-- 送信ボタン -->
        <br />
        <ul id="ul_messages"> <!-- リクエストとレスポンスの挿入ベース-->
            <li id="list_messages"></li>
        </ul>
    </div>
```

ビューの詳細 1

# 作業：ビューの書き換え

```
<script type="text/javascript" src="~/Scripts/jquery-3.1.0.js"></script> <!-- API叩き用 jQuery -->
<script>
  $("#button_request").click(function () { //送信ボタンクリック時のイベント設定
    var utterance = $("#text_utterance").val(); // フォーム内部の文字を取得
    var parameter = {
      utterance:utterance
    }; //送信パラメータを取得
  });
```

# 作業：ビューの書き換え

```
$.ajax({ //ajax(非同期通信)開始
  url: "/Home/Api", // HomeコントローラのApiを
  type: 'GET', // HTTP通信をGETにて行う
  data: JSON.stringify(parameter), // JSONの文字列に変換
  dataType: 'json', // JSON型の使用宣言
  contentType: 'application/json', // content-typeをJSONに指定する
  error: function (data) { // 失敗時の動作
    alert(data); // アラートの出力
  },
  complete: function (data) { // 完了時の動作
    var json = data.responseText; //データを取得
    var obj = JSON.parse(json); // JSONをパース
    // UIにシステムから会話の内容を追加
    $('#list_messages').append(
      '<p class="system_box">'
      + '<span class="system_name">システム: </span>'
      + '<span class="system_utterance">' + obj.utterance + '</span>'
      + '</p>'
    );
  },
});
```

# 作業：ビューの書き換え

```
// UIに自分が送った会話の内容を追加
$('#list_messages').append(
  '<p class="user_box">'
  + '<span class="user_name">あなた: </span>'
  + '<span class="system_utterance">' + utterance + '</span>'
  + '</p>'
);

$('#text_utterance').val(""); //フォームを空に
});
</script>
</body>
</html>
```



# View上でやっていること

- UIの生成
  - フォームとボタン
- JavaScriptの実行
  - Sendボタン押下時
    - コントローラのメソッドを呼び出し(Api関数)
    - 成功すればデータを表示

# 動かしてみよう

- デバッグ
  - デバッグからデバッグなしで実行
    - ブラウザが開いたらOK



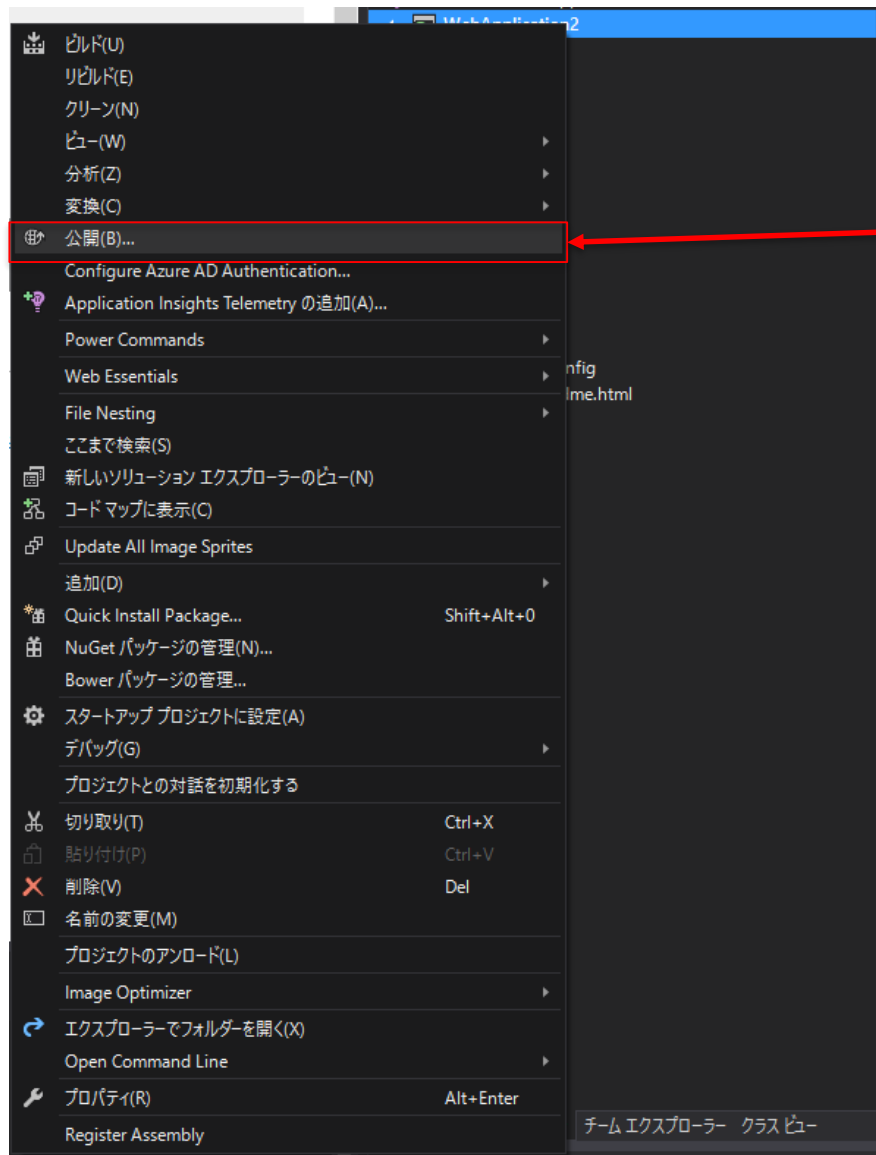
## 次のステップ

- プロジェクトの作成
- ASP.NET MVCの構造をしてみる
- Modelの作成
- Controllerの作成
- Viewの作成
- アプリの公開

# アプリの公開

- ここまで
  - ローカルホスト (PC内部で)
- ここから
  - パブリック (Azure上で)

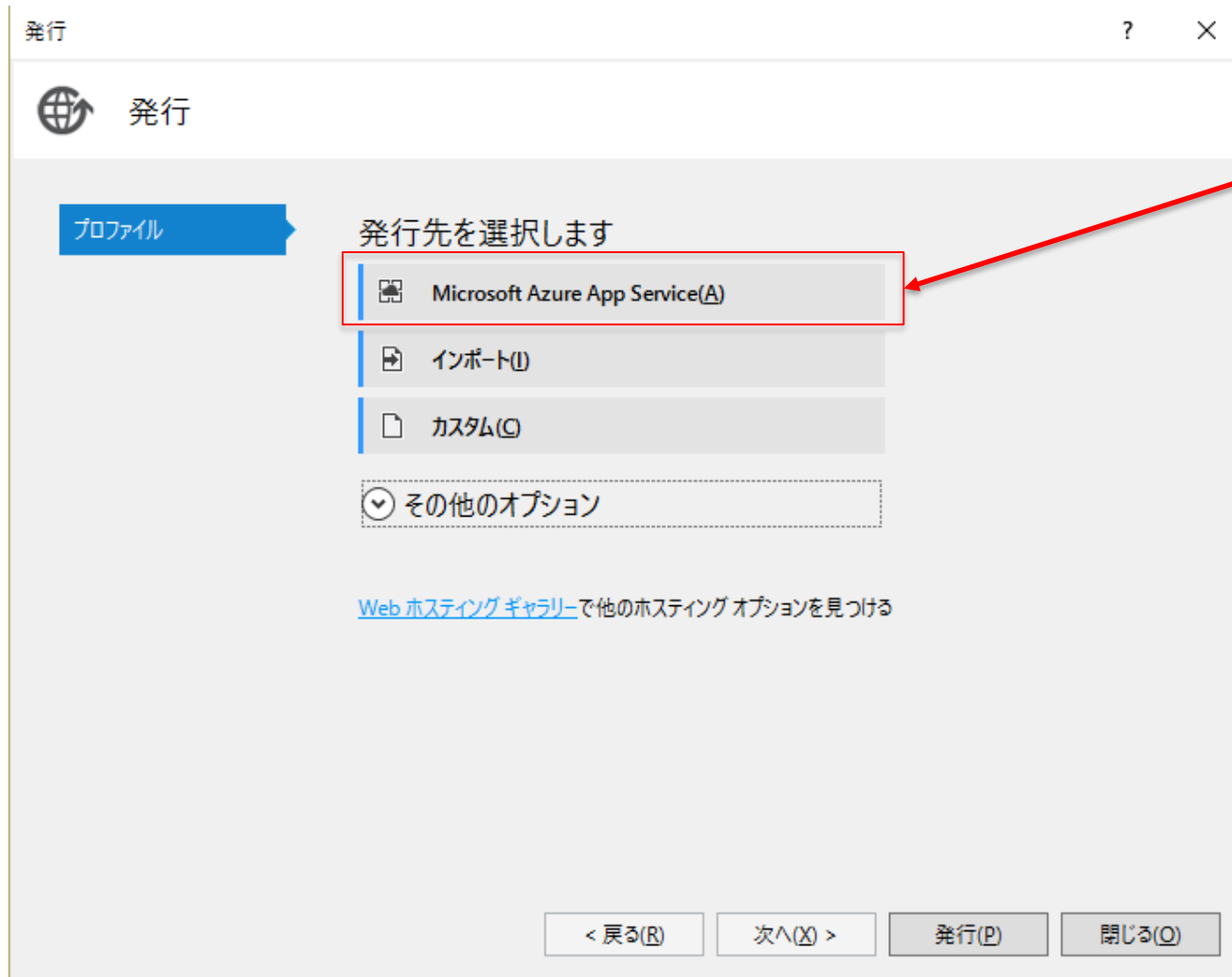
# 作業：アプリの公開



## アプリの公開

ソリューションを右クリック  
-> 公開を選択

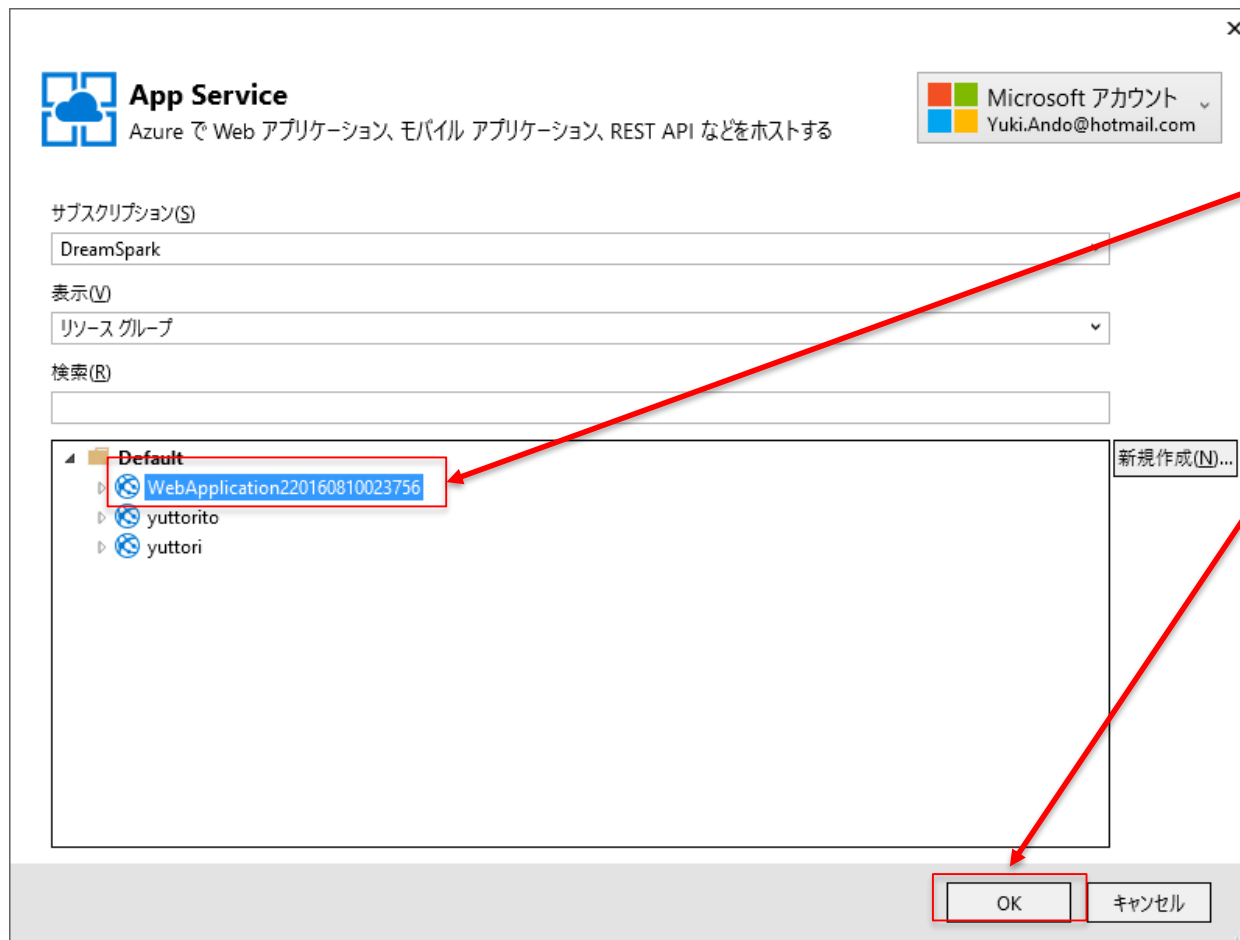
# 作業：アプリの公開



## アプリの公開

Microsoft Azure App Service  
をクリック

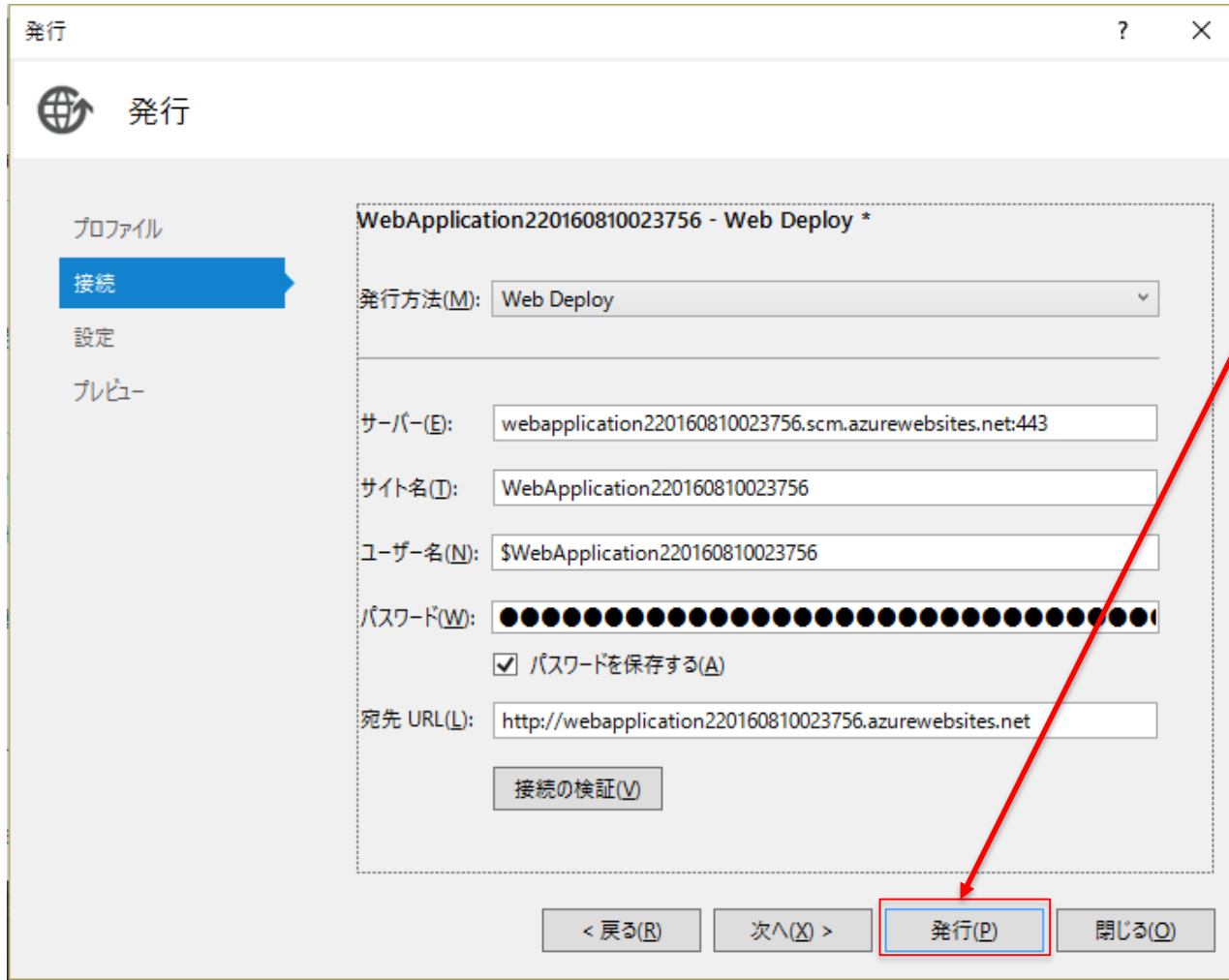
# 作業：アプリの公開



## アプリの公開

- プロジェクト作成時に生成したものを選択
- OKをクリック

## 作業：アプリの公開



# アプリの公開

発行をクリック



# Finish!

- Scaffoldingを最大限有効活用してみよう
- ユーザ管理入門

# まとめ

Summary

# まとめ

- ASP.NETは様々な機能を提供
- 全てを理解していなくても使える
  - ルーティングとデータ保持だけASP.NET
  - ビューはHTML,CSS頼りに
- ~~• お仕事にはよく使われるかもしれない~~

- ASP.NETは様々な機能を提供
- 全てを理解していなくても使える
  - ルーティングとデータ保持だけASP.NET
  - ビューはHTML,CSS頼りに
- ~~• お仕事にはよく使われるかもしれない~~

ASP.NETを楽しもう!

ご清聴ありがとうございました！

Thank you for listening!