# Ajaxとは

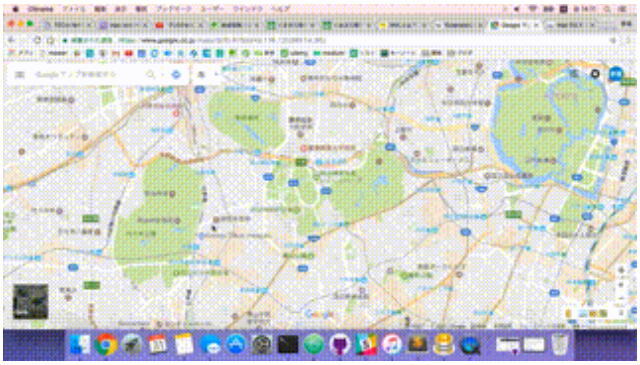
Ajaxとは「Asynchronous JavaScript + XML」の略

**Asynchronousとは、非同時性の、非同期の**

つまり、「JavaScriptとXMLを使って非同期にサーバとの間の通信を行うこと。」

んん？
詳しく内容を追っていきましょう。

# そもそも非同期通信とはどんなものか？



このように画像の遷移のない通信を非同期通信と言います。
**同期処理は一瞬画面が白くなって、画面を切り替わることを言います。**

こういったことをするためにAjaxという仕組みが必要です。

# どうしてこんなことができるのか？

大枠をざっくり先に説明します。

## 同期通信の場合

webブラウザからサーバーにリクエストを通信し、レスポンスが戻ってくる。
この時に、すべての情報を通信しているので、一瞬画面が白くなる。
=>**サーバーからレスポンスが返ってくるまでは他の作業はできない。**

## 非同期通信の場合

webブラウザから一部の情報をリクエストするので、
それ以外の部分は変わらない。なので画面が白くなることがない。
=>**サーバーからレスポンスが返ってこなくても他の作業ができる。**

ほー
ってことは**一部の情報をサーバーに送信して、それを受け取り反映させる仕組み**をAjaxっていうんだね。

でもどうやってやっているの？

# Ajaxを支える仕組み

ここで登場する技術が

- XMLHttpRequest
- JavaScript
- DOM
- XML

Ajaxというのは一つの機能でできているのではなく、
複数の機能が組み合わさって実装しています。

それぞれ機能を見ていきましょう

## XMLHttpRequest

> クライアントとサーバーの間でデータを伝送するための機能をクライアント側で提供する API です。
> ページ全体を再読み込みすることなく、URLからデータを読み出す簡単な方法を提供します。
> このAPIによって、ユーザの作業を中断させることなくWebページの一部を更新することができます。
> (MDNより)

要するに、ブラウザ上でサーバーとHTTP通信を行うためのAPIです。

このAPIを使って実装をするのがJavaScriptです。
でもなんでJavaScriptなの？

## JavaScript

それは先程のXMLHttpRequestがjavascriptの組み込みオブジェクトだからです。
組み込みオブジェクトとは、あらかじめ定義されているオブジェクトのことですね。

Ajaxという名前にある通り、非同期通信はjavascriptを使わないと実装できないのです。

## DOM

> Document Object Model (DOM) は、HTML および XML ドキュメントのための API です。これはドキュメントの構造的な表現を提供し、内容や表示形態の変更を可能にします。端的に言えば、Web ページをスクリプトやプログラミング言語とつなぐような機構です。
> (MDNより）

つまり、Ajaxを使って動的なWebページを作成するときに、HTML・XML上のどの要素を変更するかを指定します。そこでDOMはHTMLやXMLを「ツリー構造」として展開し、アプリケーション側

に文章の情報を伝え、加工や変更をしやすくするものです。

**ツリー構造とは、データ構造の一つで、一つの要素(ノード)が複数の子要素を持ち、一つの子要素が複数の孫要素を持ち、という形で階層が深くなるほど枝分かれしていく構造のこと。**

ここで出てくるXMLとはなんなのでしょうか？

## XML

Extensible Markup Languageの略。
文書やデータの意味や構造を記述するためのマークアップ言語の一つ（HTMLと似たようなもの)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<?xml-stylesheet type="text/xsl" href="testxsl.xsl"?>
<おこづかい帳>
  <支出>
    <内容>
      <日付>1月20日</日付>
      <交通費>780</交通費>
      <食費>980</食費>
      <嗜好品>250</嗜好品>
    </内容>
    <内容>
      <日付>1月21日</日付>
      <交通費>950</交通費>
      <食費>1200</食費>
      <嗜好品>300</嗜好品>
    </内容>
    <内容>
      <日付>1月22日</日付>
      <交通費>500</交通費>
      <食費>1500</食費>
      <嗜好品>250</嗜好品>
    </内容>
  </支出>
</おこづかい帳>
```

XMLはタグを自由に設定でき、そのタグに意味づけをすることができます。

データのやりとりで「XML」を使えば、複数のデータを同時にやりとりしても、どのデータがどの要素なのか一発で判明できます。

ただ現在では、AjaxにはXMLの代わりにJSONという型がよく使われています。
Ajaxという名前にXMLが入っているので、他の型は使えないように思いますが、
そうではないんですね。

## Json

JavaScript Object Notationの略。
軽量のデータ交換フォーマットで、人間にとって読み書きが容易で、マシンにとっても簡単にパースや生成を行なえる形式です。

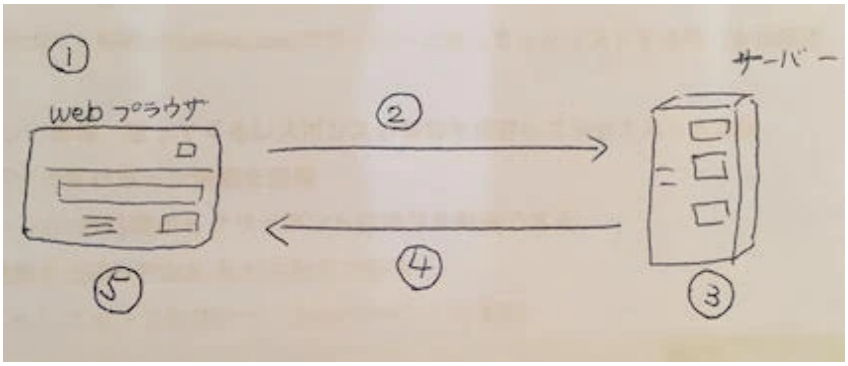
```
{ "name"   : "John Smith",
  "sku"    : "20223",
  "price"  : 23.95,
  "shipTo" : { "name" : "Jane Smith",
               "address" : "123 Maple Street",
               "city" : "Pretendville",
               "state" : "NY",
               "zip"   : "12345" },
  "billTo" : { "name" : "John Smith",
               "address" : "123 Maple Street",
               "city" : "Pretendville",
               "state" : "NY",
               "zip"   : "12345" }
}
```

現在ではJSONを使用して、非同期通信を行うのが主流となっています。

ん～
なんだか色々な言葉が出てきたので、まとめて欲しいな～

# まとめ（流れの整理）

ここまで説明してきたものが、どのように関係して、
Ajaxの機能を実装しているかをまとめます。




①ページ上で任意のイベントが発生（ボタンクリックなど）

②JavaScript + XMLHttpRequestでサーバーに対してリクエストを送信（非同期通信）

**ほしい情報、返ってくるレスポンスの情報を指定してリクエスト**

③サーバーで受け取った情報を処理

**サーバーの処理中もクライアントは操作を継続できる**

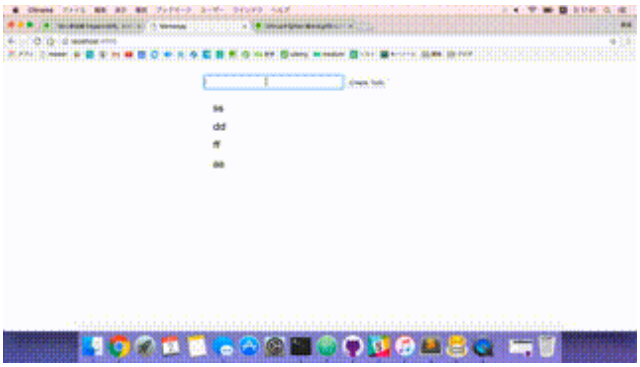
④処理結果をJSONやXMLなどの形式で応答

⑤受診したレスポンスを受けて、DOMでページを更新

**更新のあった部分だけを書き換えるため、画像が一瞬白くなることはない。**

このように機能が結びついてAjax機能を実装しているんですね。

# サンプルアプリで動きの確認



文字を入力したらその文字が表示されるアプリを通してAjaxを確認してみます

index.html.erb

```
<div class="contents">
  <%= form_for @todo, html: { class: 'form js-form' } do |f|%>
    <%= f.text_field :content, class: 'form__text-field js-form__text-field' %>
    <%= f.submit class: 'form__submit js-submit' %>
  <% end %>
  <ul class="todos">
    <%= render @todos %>
  </ul>
</div>
```

_todo.html.erb

```
<li class="todo"><%= todo.content %></li>
```

todos.controller.rb

```
class TodosController < ApplicationController

  #indexアクションはTodoの一覧ページとTodoの入力ページを兼ねているので、新しいTodoを作成するためのイ
  def index
    @todo = Todo.new
    @todos = Todo.order('created_at ASC')
  end

#createアクションが呼ばれると何かしらのTodoが入力されていた場合（contentが空でない場合）は、Todoを保
  def create
    @todo = Todo.new(todo_params)
    if @todo.save
      respond_to do |format|
```

```
        format.html { redierct_to :root }
        format.json { render json: @todo}
      end
    else
      render :index, alert: 'Todoを入力してください'
    end
  end
#respond_to doを使用し、リクエストされたformatによって処理を分けるようにします。今回はhtmlと非同期通信
  private
  def todo_params
    params.require(:todo).permit(:content)
  end
end
```

todo.js

```
$(function() {
  function buildHTML(todo) {
    //以下はセレクタの中身を新規に作るという意味
    var html = $('<li class="todo">').append(todo.content);
    return html;
  }
//submitイベントを使い、フォーム（js-from）が送信された時に処理が実行されるようにイベントを設定。
  $('.js-form').on('submit', function(e) {
    e.preventDefault(); //フォームが送信された時に、デフォルトだとフォームを送信するための通信がされ
    var textField = $('.js-form__text-field'); //class js-form__text-fieldを代入
    var todo = textField.val(); //js-form__text-fieldのフォームに入力された値を取得し、todoに代入
    //$.ajax関数は、戻り値として XMLHttpRequestオブジェクトを返します。
    //ここでサーバーに対しての通信を行う。情報の指定（ここではdataに格納）、送信先、データの型（Json）
    $.ajax({
      type: 'POST',
      url: '/todos.json',
      data: {
        todo: {
          content: todo
        }
      },
      dataType: 'json' //データをjson形式で飛ばす
    })
    //↓フォームの送信に成功した場合の処理
    .done(function(data) {
      var html = buildHTML(data);
```

```
      $('.todos').append(html); //$.append関数は操作後はDOMに要素が追加された状態になる。
```

```
      textFieldval(''); //
    })
    //↓フォームの送信に失敗した場合の処理
    .fail(function() {
      alert('error');
    });
  });
});
```

以上の流れでAjax機能が動いています。

すごい仕組みだな～