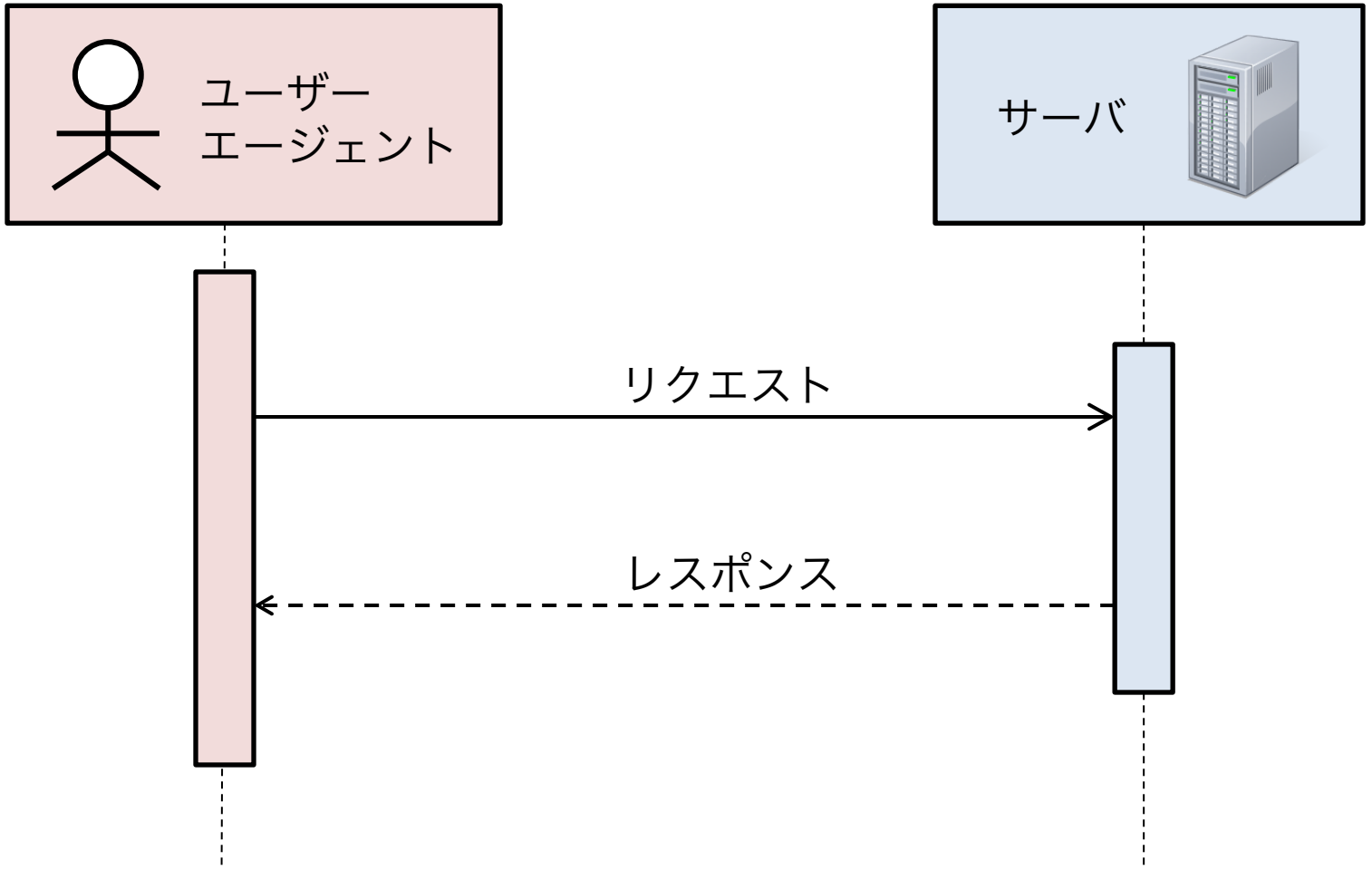
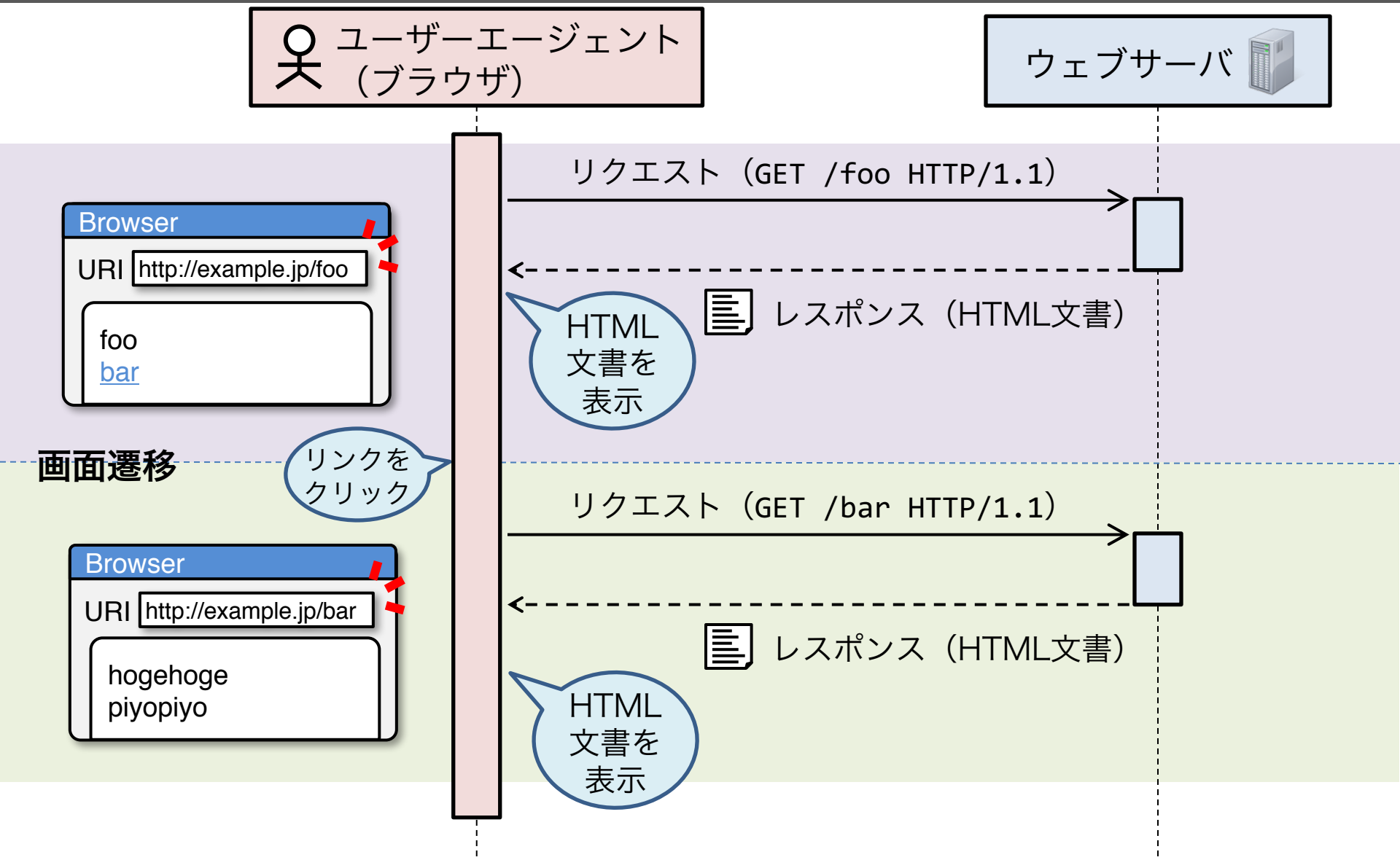
# HTTP（再掲）

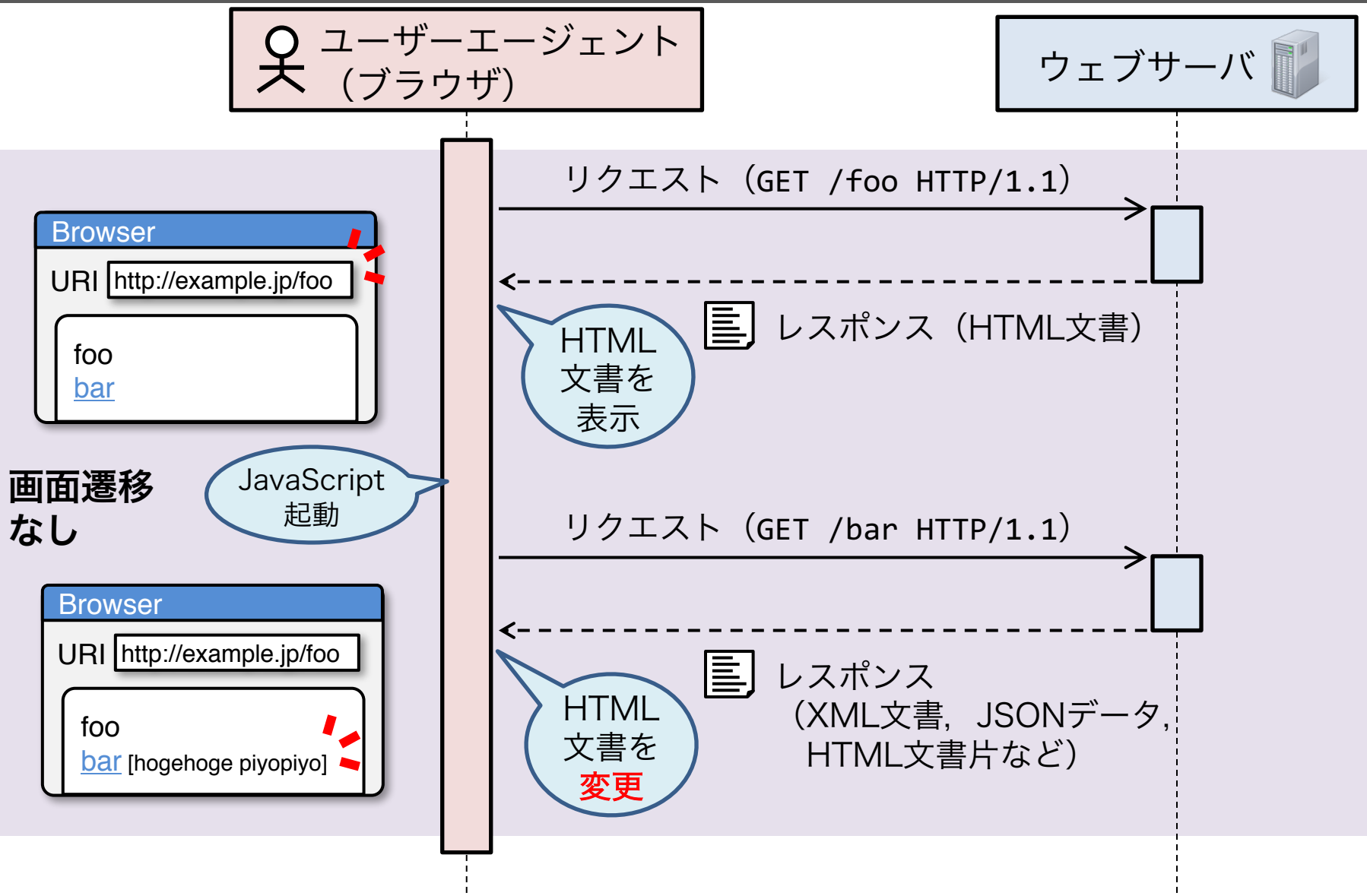



- **特徴: ステートレス（状態を持たない）**

# Webアプリにおける画面遷移



# Ajax を用いた非同期遷移



# Ajax (Asynchronous JavaScript + XML)

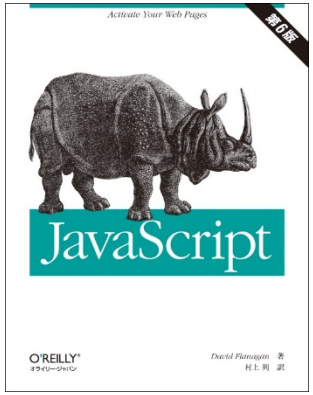
- **ブラウザ上で非同期通信を行う技術の総称.**
  - 定義は揺れている.
    - 本来は，XMLHttpRequest 等を経由して，JavaScript から XML を非同期にやりとりすることを指すが，現在では XML を利用しなくても Ajax と呼ばれる場合が多い.
    - たまに Asynchronous ですらないものが Ajax と呼ばれることもある……
- **典型的なアプリケーションとして以下がある.**
  - Google マップ
  - Gmail

# サンプル

- https://localhost:9001/sample.ajax/

# JavaScript

- **JavaScript とは?**
  - ウェブブラウザ上で動作する，（現在のところ）もっとも一般的なスクリプト言語である.
  - プロトタイプベースのオブジェクト指向言語である.
    - Java はクラスベースのオブジェクト指向言語である.
    - 本実験では JavaScript のオブジェクト指向については掘り下げない.
      興味があれば下記書籍等を参照すること.
- **仕様: ECMA-262**
  - http://www.ecma-international.org/publications/standards/Ecma-262.htm
  - 日本語版: http://www2u.biglobe.ne.jp/~oz-07ams/2002/ecma262r3/
  - おすすめ書籍
    - JavaScript第6版，オライリージャパン
      978-4-87311-573-3



# 対話的開発環境の整備

- **Chrome**
  - アドレスバー右部から「ツール」 > 「デベロッパー ツール」を選択し, ビューを表示.
  - Console を選択.
- **Firefox**
  - アドレスバー右部から「開発ツール」 > 「Webコンソール」を選択.
- **Safari**
  - Safari > 環境設定 > 詳細 > 「メニューバーに”開発”メニューを表示」にチェックを入れる.
  - 「開発」 > 「エラーコンソールを表示」を選択.

# JavaScript 文法

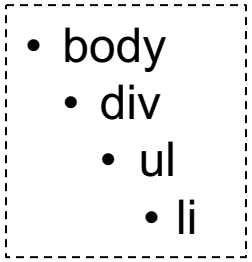
- **Java（C）に近い.**
  - `if(true) { ... } else { ... }`
  - `switch (value) { case 1: ... case 2: ... default: ... }`
  - `for(var i = 0; i < 10; i++) { ... }`
  - `foo.method();`
  - `foo.property = value;`
- **変数に型がない.**
  - `var a = 1;`
  - `var b = "string";`
  - `b = true;`
- **関数が第一級オブジェクトとなる.**
  - `function foo(a) { alert(a); }`
  - `foo(1);`
  - `var bar = function(a) { alert(a); };`
  - `bar(1);`

# JavaScript を動かすには

- **script 要素を利用する.**
  - 外部ファイル読み込み
    ```
    <script type="text/javascript" src="path/to/script.js"></script>
    ```
  - インライン書き込み
    ```
    <script type="text/javascript">alert(1);</script>
    ```
- **イベントハンドラとして直書きする.**
  - `<span onclick="alert(1)">hogehoge</span>`
- **一般的な方法**
  - 関数の定義等を特定の .js ファイルに記述し，<head> 内で外部ファイルとして読み込む.
    - 例えば，　/public/javascripts/exp4todo.js
  - イベントハンドラで，上記で定義した関数を起動するなど，短いスクリプトを記述する.

# DOM (Document Object Model)

- **HTML文書（やXML文書など）に対する API**
  - 例：既存の文書に新しい要素を追加する.
  - 例：既存の文書の要素を参照する，削除する，変更する.
- **Level 分けされている．下記はLevel 1 である.**
  - http://www.w3.org/TR/REC-DOM-Level-1/
  - 日本語訳: http://www.doraneko.org/misc/dom10/19981001/cover.html
- **HTMLは木構造を成す.**
  - DOM での根要素は document




```
<body>
  <div>
    <ul>
      <li> ... </li>
    </ul>
  </div>
</body>
```

- **参考**
  - https://developer.mozilla.org/ja/DOM
  - DOM document https://developer.mozilla.org/ja/DOM/document
  - DOM element https://developer.mozilla.org/ja/DOM/element

# DOM: いくつかの例

- 特定の ID を持つ要素のノードオブジェクトを取得：
  - `var node = document.getElementById("idname");`
- ノードの親ノードや子ノードを取得：
  - ```
    var parent = node.parentNode;
    var children = node.childNodes; // children[0], children[1], …
    ```
- 新しいノードを追加：
  - ```
    var newnode = document.createElement("hr");
    node.appendChild(newnode);
    ```
  - `var text = document.createTextNode("テキストノード");`
- 既存のノードを削除：
  - `parent.removeChild(node);`
- 既存のノードを新しいノードに交換：
  - `parent.replaceChild(newnode, node);`
- ノードを複製：
  - `var cloned = node.cloneNode(true);  // trueでdeep copy`

# XMLHttpRequest

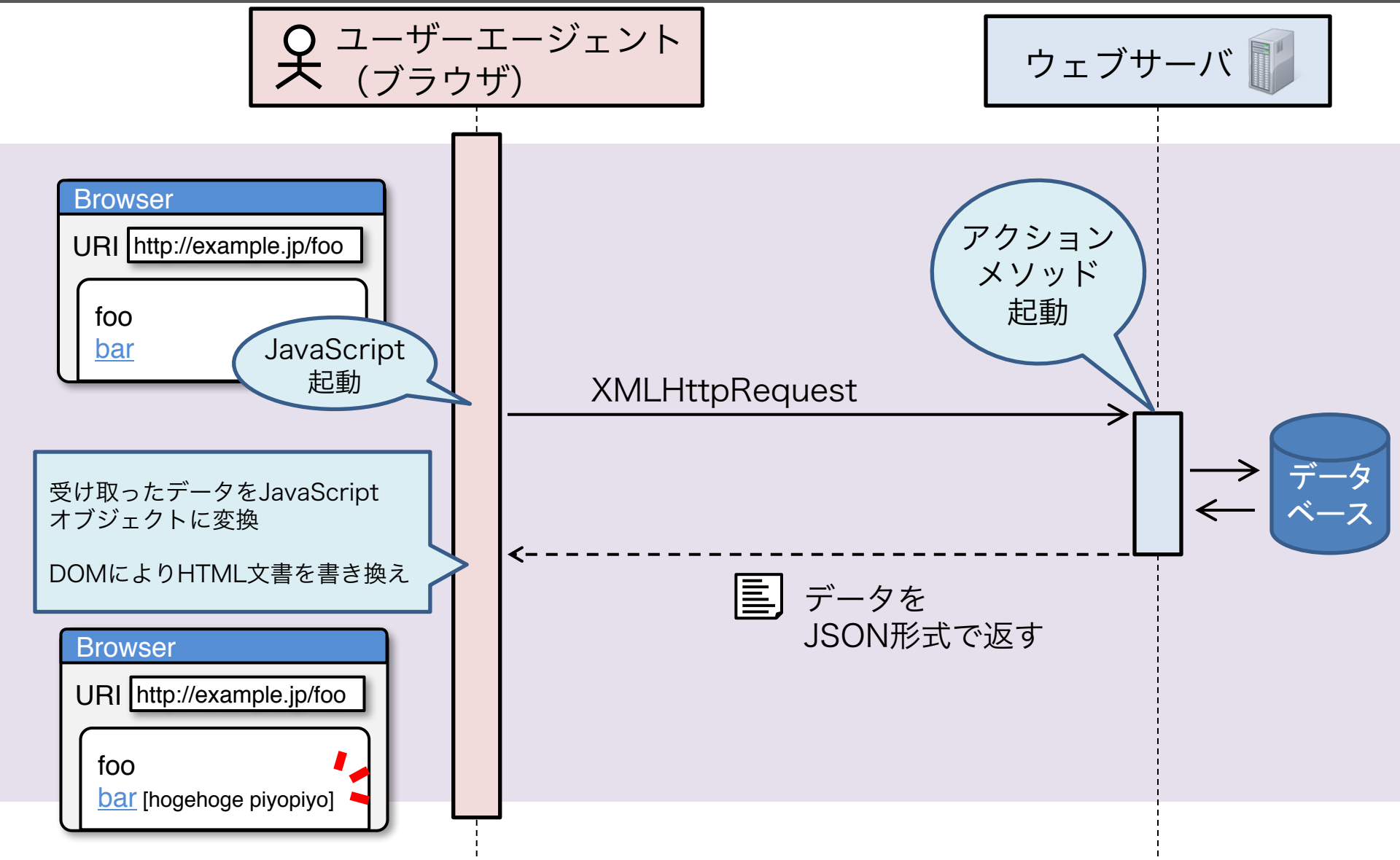
- **非同期通信のためのAPI**
  - http://www.w3.org/TR/XMLHttpRequest/

```
function enc(s) {
    return encodeURIComponent(s).replace(/%20/g, '+'));
}

var req = new XMLHttpRequest();          // オブジェクトの宣言
req.open("POST", "/path/to/uri");
req.onreadystatechange = function() {    // コールバック関数の宣言
    if (req.readyState != 4) return;     // readyState == 4: 終了
    if (req.status != 200) {             // 応答が200（OK）かを確認
        alert("failed."); return;
    }

    var body = req.responseText;         // body には応答の本文が入っている
    var data = eval("(" + body + ")");   // JSON が入っているなら解釈
    ...
};
req.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
                                         // Content-Type の指定
req.send(["param1=" + enc(value), "param2=" + enc(value)].join("&"));
                                         // パラメータ指定（不要なら無引数）
```

# Ajax を用いた非同期遷移



# JSON (JavaScript Object Notation)

- **JavaScript のオブジェクトリテラル風のデータシリアライズ方式**

```
Map<String, Object> result = new HashMap<>();
result.put("a", "b");
result.put("c", "d");
```



JSON

```
{"a": "b", "c": "d"}
```

- **Java世界での扱い**
  - Java オブジェクト を JSON に変換する必要がある.
    - renderJSON() メソッドを利用する.

```
Map<String, Object> result = new HashMap<>();
result.put("key", value);
// 普段は render() するところで
renderJSON(result);
```

- **JavaScript 世界での扱い**
  - JSON を JavaScript オブジェクトに変換する必要がある.
  - 例えば，eval を利用する.
    - `var obj = eval("(" + json + ")");`

# 本実験で(公式に)扱わないもの

- **ECMAScript 2015/6**
- **Fetch API (XMLHttpRequest 後掲)**
  - 各自が使う分にはOKです.

- **各種 JavaScript ライブラリ**
- **関連言語からのトランスパイラ**
  - 各自が使う分にはOKです，が,
    [B11] の実装には使わないでください．それ以外はOK.
    - 少なくとも1例は生の実装を含めることという意図です.