HTML5プロフェッショナル認定試験
レベル2ポイント解説無料セミナー

株式会社クリーク・アンド・リバー社 認定講師

高井 歩

本日の内容

- 試験概要
- XAMPP
- オフラインアプリケーションAPI
- 通信(XMLHttpRequest,WebSocket)
- Geolocation API
- Navigation Timing

HTML5プロフェッショナル認定資格とは

- 次世代のWebプロフェッショナルのスキルの向上に貢献するために、HTML5を活用したWebページやWebアプリケーションなどのデザイン、設計、構築に関する体系だった知識とスキルを備えたHTML5のプロフェッショナルを中立的な立場で公平かつ厳正に認定する資格制度です。

- Webデザイナー、Webプログラマー、スマートフォンアプリ開発者、サーバーサイドエンジニアなどの、Web開発プロジェクトやWebサービスに関わるあらゆるプロフェッショナルが対象です。

- 多くの企業が推進する次世代Web言語の認定資格として、HTML5のプロフェッショナルのスキルの向上に役立ちます。
また、企業内や研修機関での『技術力を担保する客観的基準』としても活用できます。

二つのレベル

HTML5 Level.1

マルチデバイスに対応した静的なWebコンテンツをHTML5を使ってデザイン・作成できる。

対象

- Webデザイナー/HTMLコーダー
- Webディレクター/グラフィックデザイナー
- フロントエンドプログラマー
- Webシステム開発者
- スマートフォンアプリ開発者
- サーバーサイドエンジニア

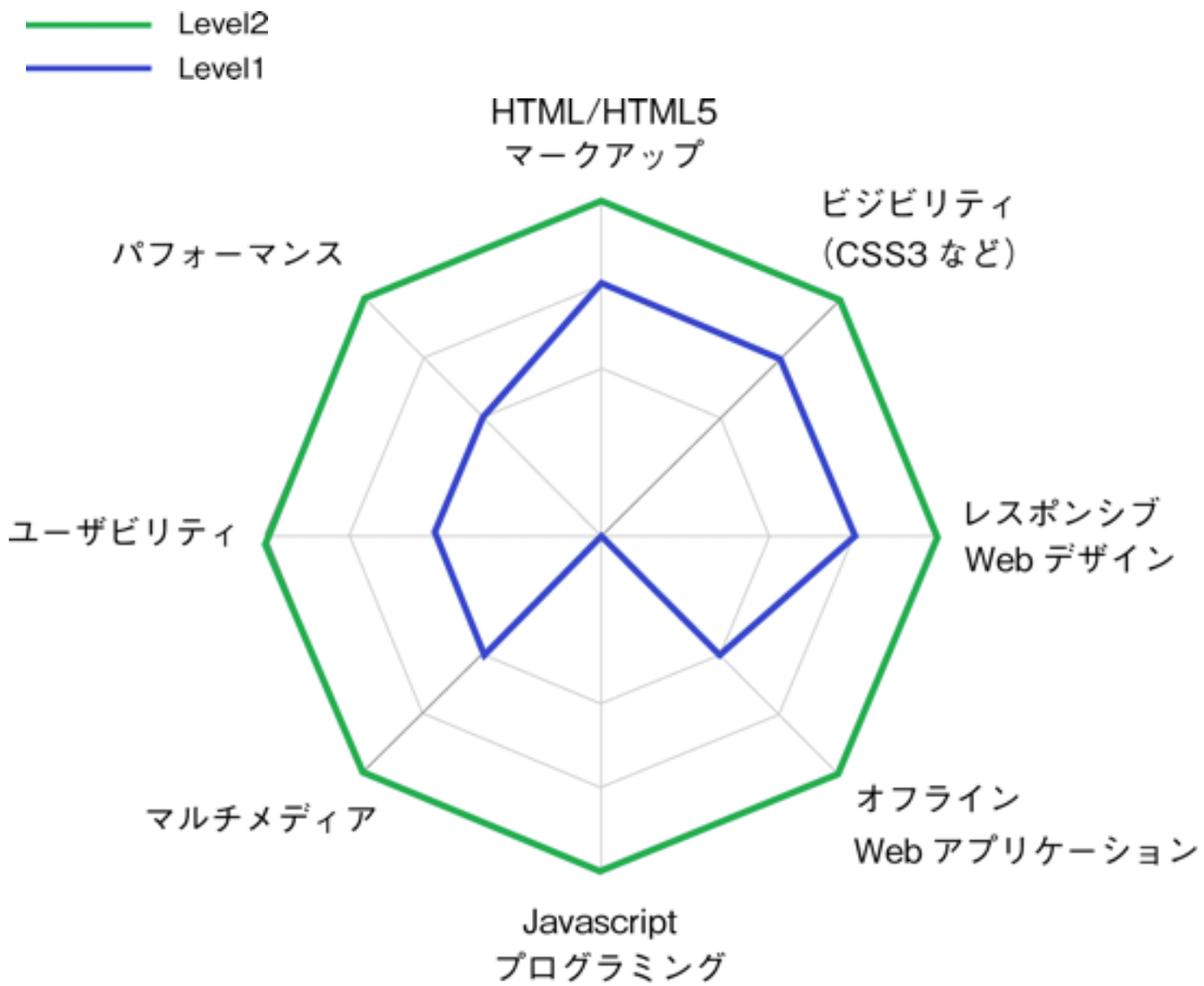
HTML5 Level.2

システム間連携や最新のマルチメディア技術に対応したWebアプリケーションや動的Webコンテンツの開発・設計ができる。

対象

- Webシステム開発者
- スマートフォンアプリ開発者
- フロントエンドプログラマー
- Webディレクター
- サーバーサイドエンジニア
- Webデザイナー/HTMLコーダー

レベル1とレベル2の資格体系



HTML/HTML5マークアップ
HTML5に関するタグの用途、構造の組み立て方に関する技術

ビジビリティ
JavascriptやCSS3などを用いて、デザイン仕様に沿った見やすい表示を行うための技術

レスポンシブWebデザイン
一つのソースで、スマートフォンなどの様々なデバイスの画面サイズに対応させるための技術

オフラインWebアプリケーション
通信が常時接続状態ではない環境でも、効率的にWebコンテンツを動作させるための技術

Javascriptプログラミング
Javascriptを使って、動的なWebコンテンツを作成する技術

マルチメディア
３D・動画・音声ファイルなどのマルチメディアコンテンツの表示・再生に関する技術

ユーザビリティ
ナビゲーション、地図表示など操作しやすいコンテンツを作成するための技術

パフォーマンス
データベースや、並列処理を使ってコンテンツを効率良く高速に動作させるための技術

レベル1とレベル2の資格体系

ベーシックレベル
HTML5プロフェッショナル向け

所要時間：90分
試験問題数：約60問
受験料：\15,000 （税抜）
認定条件： HTML5 レベル1試験に合格すること
認定の有意性の期限：5年間

認定名： HTML5 Level1 (Markup Professional)
試験名： HTML5 Level1 Exam

この資格の認定者は、下記のスキルと知識を持つWebプロフェッショナルであることを証明できます。

- **HTML5を使って静的なWebコンテンツを作成することができる。**
- **ユーザビリティ・ビジビリティの高いWEBコンテンツを設計・作成することができる。**
- **スマートフォンや車載システムなど、様々なデバイスに対応したコンテンツ作成ができる。**

アドバンストレベル
HTML5プロフェッショナル向け

所要時間：90分
試験問題数：未定
受験料：未定
認定条件：HTML5 レベル2試験に合格し、かつ有意なHTML5レベル1認定を保有していること。
認定の有意性の期限：5年間

認定名： HTML5 Level2 (Application Development Professional)
試験名： HTML5 Level2 Exam

この資格の認定者は、下記のスキルと知識を持つWebプロフェッショナルであることを証明できます。

- **動的に動作させて高いユーザビリティを実現するリッチユーザインターフェイスアプリケーションを作成することができる。**
- **マルチデバイスに対応し高パフォーマンスで動作する動的コンテンツを作成することができる。**
- **システム間連携を行いリアルタイムな情報を提供するアプリケーションを作成することができる。**

レベル2の出題構成(1)

主題	項目	重要度
JavaScript	JavaScript文法 Javascriptの概要 演算子 特殊数値 配列 制御文 関数 型・オブジェクト プロパティ スコープ	**10**
WebブラウザにおけるJavaScript API	イベント	**8**
	ドキュメントオブジェクト/DOM	**6**
	ウィンドウオブジェクト	**8**
	Selectors API	**4**
	テスト・デバッグ	**2**

レベル2の出題構成(2)

主題	項目	重要度
グラフィックス	Canvas(2D)	**6**
	SVG	**1**
マルチメディア	video要素,audio要素 (各要素の仕様や、マークアップの記述方法に関してはレベル1に含まれるため範囲外)	**2**
オフラインアプリケーションAPI	**アプリケーションキャッシュの制御**	**2**
Session History and Navigation	History API	**3**
表示制御	Page Visibility	**4**
	Timing control for script-based animations	**2**
ストレージ	Web Storage	**4**
	Indexed Database API	**2**
	File API	**2**

レベル2の出題構成(3)

主題	項目	重要度
通信	WebSocket	2
	XMLHttpRequest	4
Geolocation API	Geolocation APIの基本と位置情報の取得	2
Web Workers	並列処理の記述	4
パフォーマンス	Navigation Timing	4
	High Resolution Time	1

XAMPP

XAMPP

- Webサーバで良く使われるアプリケーションを簡単にインストールできるようにパッケージされたもの

- Apache(Webサーバ)、MySQL(データベース),PHP(プログラム言語),Perl(プログラム言語)

- Linux用、Windows用、Mac用がある

- https://www.apachefriends.org/jp/index.html

Webサーバが必要な項目

以下の項目については、動作を確認するために
Webブラウザだけではなく、
Webサーバとの連携が必要になる。

主題	項目	重要度
オフラインアプリケーションAPI	アプリケーションキャッシュの制御	**2**
通信	WebSocket	**2**
	XMLHttpRequest	**4**
パフォーマンス	Navigation Timing	**4**
Geolocation API	Geolocation APIの基本と位置情報の取得	**2**

必須では無いが、モバイル環境でのテストをする際にあると便利

オフラインアプリケーションAPI

アプリケーションキャッシュの制御 出題内容

知識問題 コードリーティング問題

オフラインでも動作可能なアプリケーションを設計するにあたって知っておくべき、状況の確認方法とキャッシュの 操作方法について理解をしている。

- アプリケーションキャッシュの概要
- アプリケーションキャッシュを利用する場合の注意点
- ApplicationCachオブジェクトの仕様
- ブラウザのネット接続状況に関する判別方法

アプリケーションキャッシュの概要

- HTML5 プロフェッショナル認定試験 Level1にも「オフラインWebアプリケーション」として出題

- Webサーバに接続できない状況でも、Webページを表示できるよう、**ローカルコンピュータに各種ファイルをキャッシュする**仕組み

- キャッシュするファイルのリストを**マニフェストファイル**と呼ばれるファイルに記述する

アプリケーションキャッシュを利用する場合の注意点

- マニフェストファイルを更新していないと、リソースを更新してもキャッシュは更新されない。

- マニフェストファイルを更新してから最初のリロードはキャッシュに保存されるだけで 表示に反映されない。2度目のリロードでキャッシュからリソースが読み込まれて表示が更新される。

- Webサーバ上で、マニフェストファイルに対して、MIME-Typeを設定する必要がある。

ApplicationCacheオブジェクトの仕様

ApplicationCache

種別	名称	概要
プロパティ	**status**	update()メソッド実行後、 **UPDATEREADY**の場合、キャッシュ更新可能
メソッド	**update()**	更新の試行(このメソッドで更新はされない)
	swapCache()	新しいリソースでキャッシュが更新される
イベント	checking	更新の試行が行なわれた
	error	マニフェストファイルが存在しないか、ダウンロード中にマニフェストファイルが更新された
	noupdate	更新を試行したがキャッシュは更新されない
	downloading	更新があり、リソースを取得する
	progress	リソースの取得中、1ファイル毎に発生
	updateready	新しいキャッシュが存在する
	cached	最初のキャッシュが行なわれた
	obsolete	マニフェストファイルが存在しない

ブラウザのネット接続状況に関する判別方法

window.navigatorオブジェクト

種別	名称	概要
プロパティ	**onLine**	ネットに接続していればtrue、 していなければfalse
イベント	online	未接続状態から接続状態になった
	offline	接続状態から未接続状態になった

試験範囲外

マニフェストファイルを使うための設定

- MANIFESTファイルを扱うためには、MIME-Typeを設定するためWebサーバの設定変更が必要な場合がある

  (ただしSafari,FirefoxはMIME-Typeを設定しなくてもキャッシュが動作する)

- XAMPPの場合は、xampp/etc/httpd.confに
  以下の太字部分を追加する。

```
<IfModule mime_module>
...中略...
    AddType text/cache-manifest .appcache
</IfModule>
```

Demo

問題1

オフラインアプリケーションAPIにおいて、ApplicationCacheオブジェクトでupdate()メソッドを呼び出した後、statusプロパティがどの値になっていれば、キャッシュを更新できるか選択しなさい。

A. UNCACHED

B. IDLE

C. CHECKING

D. DOWNLOADING

E. UPDATEREADY

通信

XMLHttpRequest

XMLHttpRequest 出題内容

知識問題 コードリーティング問題

XMLHttpRequestの特徴を理解し、通信を行い結果を適切に処理できるプログラムを読むことができる。

- XMLHttpRequestを利用した、WebサーバへHTTPリクエストを送信および結果の受信
- XMLHttpRequestオブジェクトと通信時に関連するイベントハンドラ
- XMLHttpRequestにおけるリクエストヘッダの設定
- XMLHttpRequestオブジェクトのステータス確認
- レスポンスデータに関する内容の確認、および用途にあった処理
- 取得データのブラウザによるキャッシュを防ぐ対策

XMLHttpRequestの概要

- HTML5 プロフェッショナル認定試験 Level1にも「**Ajax**」として出題

- **非同期通信**を行なうためのJavaScriptのオブジェクト。JavaScript内でWebサーバと通信を行なえる。

- 規格として、**Level1**と機能拡張された**Level2**(通称XHR2)がある。出題範囲には、Level1とLevel2の両方が含まれている。

- Level2では**クロスドメイン通信**、**バイナリデータ送信**が行なえることが主な特徴。また、イベント処理の方法が異なっている。

XMLHttpRequestでのHTTPリクエスト送信および結果の受信

XHR2での書き方

受信時の処理を記述

```
// XMLHttpRequestオブジェクトの生成
var xhr = new XMLHttpRequest();
// レスポンスデータのデータ形式を指定
xhr.responseType = 'text';
// 受信完了イベント
xhr.onload = function(ev){
    if(xhr.status == 200) { // レスポンスコード200
        alert(xhr.response);
    }else{
        alert('response code:'+xhr.status);
    }
};
// リクエスト開始
xhr.open('GET','http://example.com/sample.php');
// リクエスト送信 GETの場合は引数にnull,POSTの場合は送信するデータ
xhr.send(null);
```

Demo

XMLHttpRequestオブジェクト(1)

種別	名称	概要
プロパティ	**readyState**	リクエストの状態
	response	レスポンスデータ。データ形式はresponseTypeによる。
	responseText	レスポンスデータをテキスト形式にしたもの
	responseType	レスポンスのデータ形式。サーバに特定の形式でデータを要求する場合に使用する。
	responseXML	レスポンスデータをＸＭＬとして処理し、Documentオブジェクト形式にしたもの
	status	ステータス番号(成功すれば200)
	statusText	ステータス番号と補足メッセージ("200 OK")
	timeout	タイムアウト。ms単位で指定。0で無期限。
	withCredentials	クロスドメイン接続でクッキーやBASIC認証を必要とする場合はtrue
	upload	アップロード追跡のためのオブジェクト

XMLHttpRequestオブジェクト(2)

種別	名称	概要
メソッド	**abort()**	リクエストの中止
	open()	リクエストの開始
	send()	リクエストを送信
	setRequestHeader()	リクエストヘッダを設定する。 **open()の後、send()の前に呼び出す。**
	getResponseHeader()	指定したレスポンスヘッダを返す
	getAllResponseHeaders()	全てのレスポンスヘッダを返す
	overrideMimeType()	サーバから返ってくるMIMEタイプを上書きする。send()の前に呼び出す。
イベント	onreadystatechange	更新の試行が行なわれた

通信時に関連するイベントハンドラ

XMLHttpRequestEventTargetインタフェース

種別	名称	概要
プロパティ	**onabort**	**リクエストが中止された**
	onerror	エラーが発生した
	onload	**レスポンス受信完了**
	onloadstart	リクエストが開始された
	onprogress	データ受信中
	ontimeout	タイムアウトによりリクエストが失敗した
	onloadend	リクエストが終了した。onload/abort/errorと同時に発生する。

Level1では、readystatechangeイベントで全てのイベントを処理していた。Level2ではそれぞれ個別のイベントとして扱われるようになった。

Demo

XMLHttpRequestにおけるリクエストヘッダの設定

- **setRequestHeader**メソッドを使用する
- 全てのヘッダ項目を設定できるわけではない。
  (CookieやRefererなどは設定できない)
- open()の後、**send()の前**に呼び出す。
- POSTメソッドでリクエストを送信する場合の例

```
xhr.open('POST','http://example.com/sample.php');
xhr.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xhr.send('a=10&b=20');
```

Demo

XMLHttpRequestオブジェクトのステータス確認

readyStateプロパティ

値	状態	概要
0	UNSET	open()が呼ばれる前
1	OPENED	open()が呼ばれた
2	HEADER_RECEIVED	send()が呼ばれ、HTTPヘッダを受信、データ受信前。
3	LOADING	データ受信中
4	**DONE**	**受信完了**

Level1のみ対応のWebブラウザでは、readystatechangeイベントハンドラ内で、ステータスによる処理の分岐を行なう

XMLHttpRequestでのHTTPリクエスト送信および結果の受信

XHRでの書き方

受信時の処理を記述

```
// XMLHttpRequestオブジェクトの生成
var xhr = new XMLHttpRequest();
// 状態が変化する毎に発生するイベント
xhr.onreadystatechange = function(){
    if(xhr.readyState == 4){ // レスポンス受信完了
        if(xhr.status == 200) { // レスポンスコード200
            alert(xhr.responseText);
        }else{
            alert('response code:'+xhr.status);
        }
    }
};
// リクエスト開始
xhr.open('GET','http://example.com/sample.php');
// リクエスト送信 GETの場合は引数にnull,POSTの場合は送信するデータ
xhr.send(null);
```

レスポンスデータに関する内容の確認

- **status**による、HTTPレスポンスのステータス番号をチェックする。成功すれば200。

- **getResponseHeader**('Content-Length')による、コンテンツのサイズチェック。0の場合はデータを取得できていない。
(ただし、サーバ側のエラー表示がコンテンツとして送られてくることもあるため、0で無いなら正常とは言えない)

用途にあった処理(1)

Level1

データ形式	データ取得方法	概要
XML/HTML	**responseXML**プロパティ	HTMLDocumentオブジェクト、またはXMLDocumentオブジェクトとして取得される。
テキスト	**responseText**プロパティ	単純なテキストとして取得される。 Base64や、JSONと組み合わせることで任意のデータ形式に変換する。

用途にあった処理 Level2

Level2

データ形式	responseType	概要
テキスト	'text' 空文字列(未設定)	**responseText**に相当
バイナリデータ	'arraybuffer'	ArrayBufferオブジェクト
オブジェクト	'json'	データをJSON.parse()した結果
HTML/XML	'document'	**responseXML**に相当
Blob	'blob'	Blobオブジェクト

使用例

```
xhr.responseType = 'text';
xhr.onload = function(ev){
    if( xhr.status == 200 ) {
        alert( xhr.response );
    }
};
```

テキスト形式

Demo

取得データのブラウザによるキャッシュを防ぐ対策

GETリクエストの場合、Webブラウザにより、取得データがキャッシュされることがある。**同一ＵＲＬへのリクエスト**を行なうとキャッシュされたデータが返ってきてしまう。

対策1:リクエストヘッダでキャッシュを無効化する

```
xhr.setRequestHeader('Pragma', 'no-cache');
xhr.setRequestHeader('Cache-Control', 'no-cache');
xhr.setRequestHeader('If-Modified-Since', 'Thu, 01 Jun 1970 00:00:00 GMT');
```

対策2:クエリ文字列でURLを変化させる

```
var url = 'http://example.com/sample.php';
url += '?' + (new Date()).getTime();
```

Demo

問題2

```
var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function(){

    if(xhr.readyState == [A] ){

        if(xhr.status == 200) {

            alert(xhr.responseText);

        }

    }

};
```

左のソースコードにおいて、受信したデータを取得するために[A]に入れるべき値を選択しなさい

A. 1

B. 2

C. 3

D. 4

E. 5

WebSocket

WebSocket 出題内容

知識問題

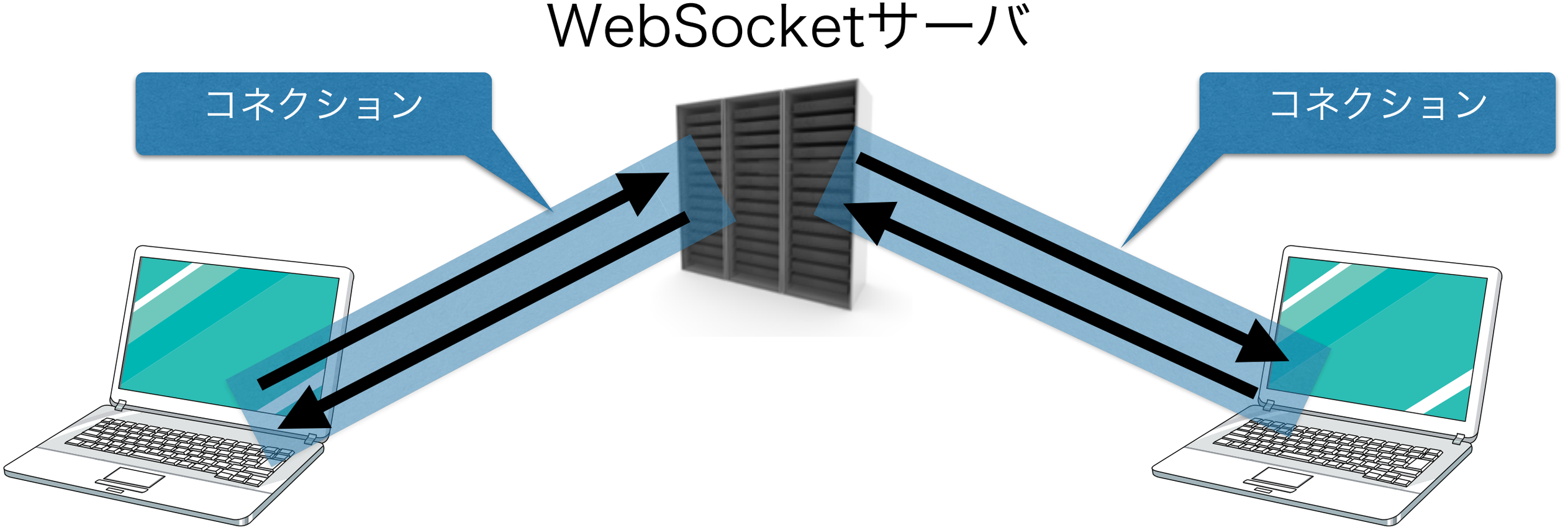
WebSocketの特徴を理解し、通信を行うにあたって必要な知識について理解している。

- WebSocketを使った通信の利点と欠点
- WebSocketにおけるイベント発生タイミング
- WebSocketを使ったサーバとの通信(クライアント側のコード)
  (WebSocket通信におけるサーバ側のコードについては、試験範囲外とする)

WebSocketの特徴

- HTTPはリクエスト/レスポンスで接続を毎回切断するが、**WebSocketは接続したまま、双方向に通信を行なえる。**

- **Webサーバではなく、WebSocket用のサーバが必要。**



WebSocketを使った通信の利点と欠点

- **利点**
  - 双方向通信が可能(HTTPやAjaxによる**ポーリング**に比べると通信量を減らせる上、リアルタイムに通信可能)
  - サーバから**クライアントへプッシュ**できる
- **欠点**
  - ブラウザの対応がまちまち
    (オブジェクト名や対応プロトコルのバージョン)
  - HTTPと異なる通信方式を使うため、ファイアウォール、プロキシ、セキュリティソフトで止められることがある。(HTTP/HTTPSと使うポートは基本的に同じ)
  - 長い間通信をしていないとセッションが切断されることがある。
  - 接続を維持するので、サーバの負荷が高くなる可能性がある。

WebSocketにおけるイベント発生タイミング

WebSocketオブジェクト

種別	名称	概要
プロパティ	readyState	コネクションの状態
メソッド	send()	データを送信する
	close()	コネクションを閉じる
イベント	**onopen**	コネクションを確立した
	onmessage	データを受信した
	onerror	エラーが発生した
	onclose	コネクションが閉じられた

WebSocketを使ったサーバとの通信(クライアント側のコード)

```
serverUrl = 'ws://127.0.0.1/';
```

接続先(http/httpsではなく、**ws/wss**)

```
var socket = new WebSocket(serverUrl);

socket.onopen = function(msg) { コネクション確立時の処理 };

socket.onmessage = function(msg) { データ受信時の処理 };

socket.onclose = function(msg) { コネクション切断時の処理 };

socket.onerror = function(msg) { エラー時の処理 };

socket.send(送信するデータ);
```

イベント設定

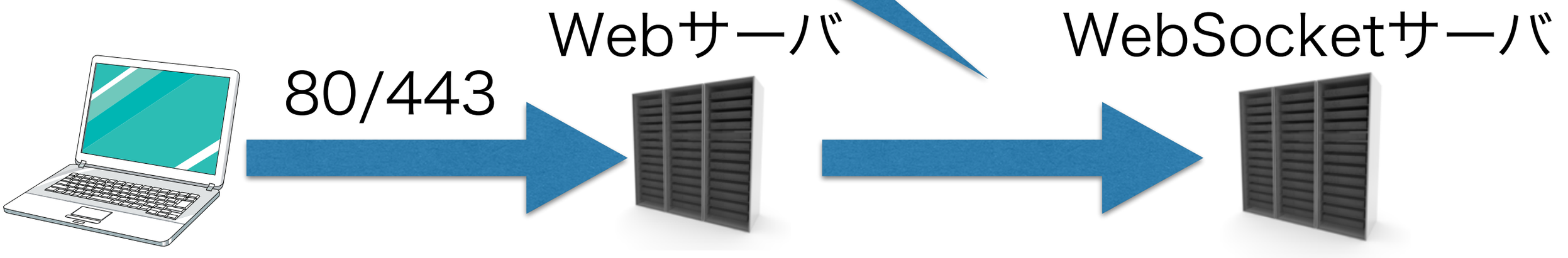
Demo

WebサーバとWebSocketサーバ

WebSocketの接続先の指定は、http/httpsではなく、
ws/wssを使用する。
ポート番号を特に指定しない場合は、wsは80番ポート、
wssは443番ポートを使用する。

http/httpsと同じ!

Webサーバと同居する場合
工夫が必要



ws/wssの通信だけ転送する

問題3

WebSocketの特徴として正しくないものを選択しなさい

A. サーバ側からプッシュできる

B. ポーリングを行なう

C. コネクションはクライアントから作成する

D. HTTP通信より軽量である

E. WebSocketサーバが必要である

Geolocation API

Geolocation API 出題内容

知識問題

Geolocation APIの概要と利用時の注意点について理解している。

- Geolocation APIの特徴と注意する点
- 端末における現在の位置情報を取得する方法
- 現在位置取得後のコールバック関数呼び出し

Geolocation APIの特徴と注意する点

- GPS(Global Positioning System)などを使用して現在位置を取得するためのAPI。

- 現在地の緯度、経度を取得できる。

- プライバシー保護の観点から、位置情報の取得にはユーザに確認が必要。

- ユーザの許可が得られない場合、もしくはデスクトップPC上での実行など、現在位置の取得が行なえないケースも考慮する。

WiFi
無線LAN AP
携帯基地局
IPアドレス

GeoLocation関連オブジェクト(1)

GeoLocationオブジェクト

種別	名称	概要
メソッド	**getCurrentPosition()**	現在位置の取得
	watchPosition()	現在位置を監視
	clearWatch()	監視を終了する

Positionオブジェクト

種別	名称	概要
プロパティ	**coords**	**Coordinaties**オブジェクト
	timestamp	現在位置を取得した時刻

GeoLocation関連オブジェクト(2)

Coordinatesオブジェクト

種別	名称	概要
プロパティ	**latitude**	**緯度**(単位:角度 北極+90.0 赤道0.0 南極 -90.0)
	longitude	**経度**(単位:角度 東経 + ロンドン 0.0 西経 -)
	accuracy	緯度、経度の精度(誤差 単位:メートル)
	altitude	高度(単位:メートル)
	altitudeAccuracy	高度の精度(誤差 単位:メートル)
	heading	方角(単位: 角度 北0 東90 南180 西270)
	speed	速度(単位:秒速メートル)

PositionErrorオブジェクト

種別	名称	概要
プロパティ	code	Coordinatiesオブジェクト
	message	現在位置を取得した時刻

端末における現在の位置情報を取得する方法

使用例：現在位置の取得

```
navigator.geolocation.getCurrentPosition( function(position) {
    var latitude = position.coords.latitude; // 緯度
    var longitude = position.coords.longitude; //経度
});
```

使用例：現在位置の監視

```
var geoloc=navigator.geolocation.watchPosition( function(position) {
    var latitude = position.coords.latitude; // 緯度
    var longitude = position.coords.longitude; //経度
});
navigator.geolocation.clearPosition(geoloc); // 監視の終了
```

現在位置取得後のコールバック関数呼び出し

```
getCurrentPosition(success,error)
watchPosition(success,error)
```

引数は共通

```
// PositionCallback
function (Positionオブジェクト){
    正常処理
}
```

```
// PositionErrorCallback 省略可能
function(PositionErrorオブジェクト){
    エラー処理
}
```

Demo

問題4

現在位置の継続的な取得を行なう、GeoLocationオブジェクトのメソッドを選択しなさい。

A. getCurrentPosition()

B. intervalGetPosition()

C. pollingLocation()

D. watchPosition()

E. watchLocation()

Navigation Timing

Navigation Timing 出題内容

知識問題

Navigation Timing APIを使って、発生している性能に関する問題を解決するための知識について理解できている。

- ユーザアクションに対する発生時刻の取得
- 画像の読み込み時間の計測
- ページの読み込み時間、DNSにおける名前解決などの各所要時間の計測

Navigation Timing APIの概要

- Webページの**読み込みに関する一連の動作**を**ナビゲーション(Navigation)**と呼ぶ

- ナビゲーションは、Webブラウザは現在表示しているページ情報の削除(アンロード)から、ロードの完了まで

- この各工程の**開始および終了時刻**を**ミリ秒**で取得する仕組みが**Navigation Timing API**

- window.**performance.timing** は **時刻情報**

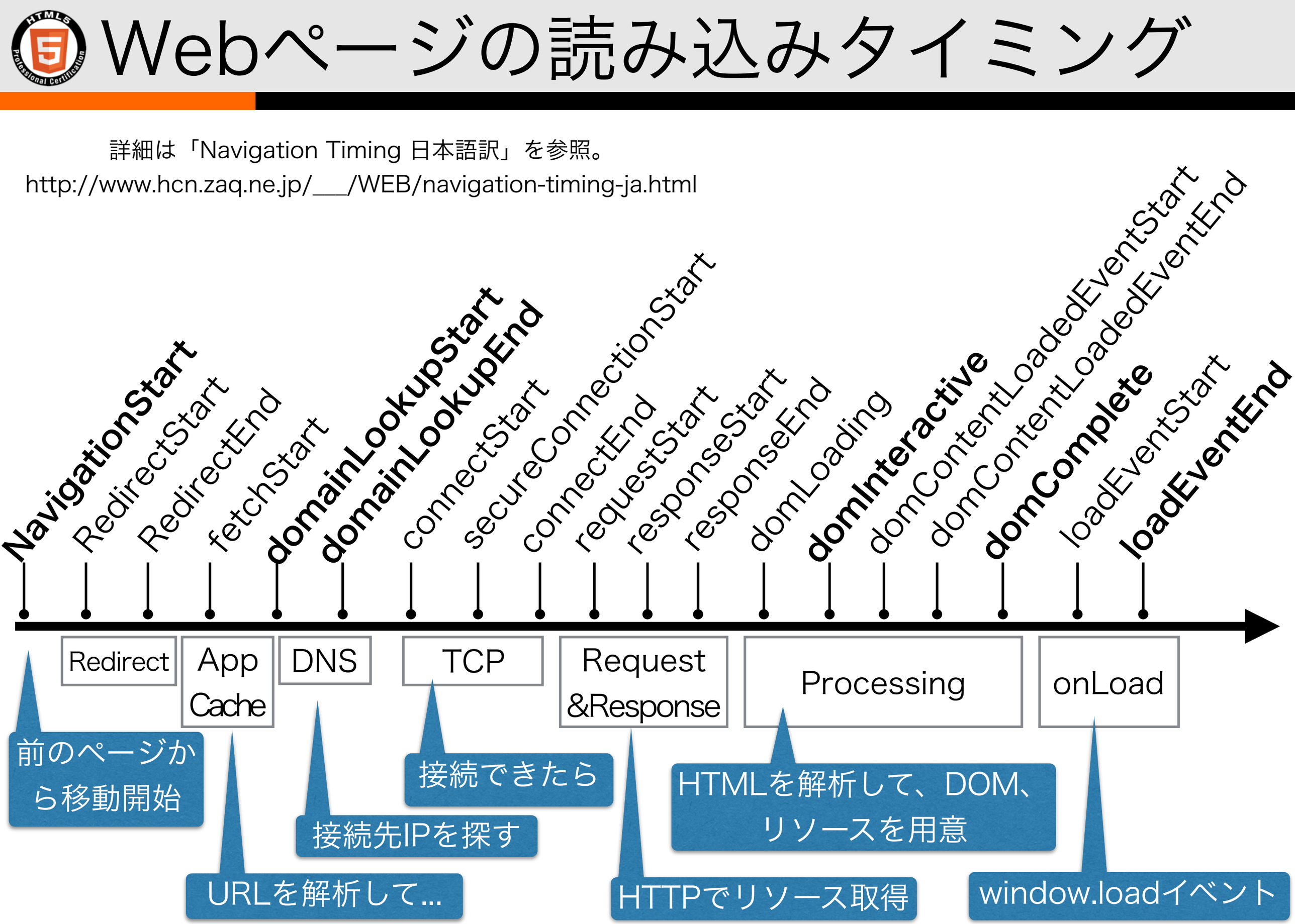
- window.**performance.navigation** は **ページ遷移情報**

PerformanceTiming

種別	名称	概要
プロパティ	**navigationStart**	**前のページから移動開始**
	unloadEventStart/End	unloadイベントの発生前後
	redirectStart/End	リダイレクトしていればリダイレクト前後
	fetchStart	読み込み処理開始(URL解析から)直前
	domainLookupStart/End	**名前解決の前後**
	connectStart/End	サーバへの接続確立前後
	secureConnectionStart	HTTPSによる接続開始直前
	requestStart	リクエスト開始直後
	responseStart/End	応答開始直後、応答終了または切断直後
	domLoading	HTMLの解析直前
	domInteractive	DOMの準備が整った(**リソース取得開始**)
	domContentLoadedEventStart/End	HTMLの解釈の前後(画像等と非同期)
	domComplete	**全てのリソースのダウンロード完了**
	loadEventStart/End	**HTMLの解釈、画像ロード完了後、window.onloadイベントの前後**

Webページの読み込みタイミング

詳細は「Navigation Timing 日本語訳」を参照。
http://www.hcn.zaq.ne.jp/___/WEB/navigation-timing-ja.html



ユーザアクションに対する発生時刻の取得

- 前のページでユーザがリンクをクリックするなどして、ページ遷移の切っ掛けになった時刻は、**navigationStart**プロパティにセットされている。

- performance.timing.**navigationStart**プロパティで、1970年1月1日0時0分0秒(UTC)を起点とした時刻を取得できる。

- ページ内のJavaScriptで取得できる、ページに関連する時刻としては最も早いと言える。

画像の読み込み時間の計測

- 元になるHTMLファイルの解析が完了し、画像等のリソースを取得できるようになる直前に**domInteractive**プロパティに時刻がセットされる。

- 画像の読み込みなどが完了すると、**domComplete**プロパティに時刻がセットされる。

- 以下の式で画像の読み込みに掛かった時間が取得できる
  performance.timing.**domComplete** -
  performance.timing.**domInteractive**

ページの読み込み時間

- ページの読み込み時間は**navigationStart**から**loadEventEnd**まで。(JavaScriptの処理やリソースの読み込みも含む)

- **完了時刻から開始時刻を引く**ことで所要時間を計算

- performance.timing.**loadEventEnd** - performance.timing.**navigationStart**

DNSにおける名前解決などの各所要時間の計測

- DNSにおける名前解決に関するプロパティは、performance.timeng.**domainLookupStart/End**

- **完了時刻から開始時刻を引く**ことで所要時間を計算
  performance.timing.**domainLookupEnd** - performance.timing.**domainLookupStart**

- ただし、Webブラウザに**名前解決のキャッシュがある場合**はdomainLookupStartとdomainLookupEndの**値は等しくなる**

Demo

問題5

performance.timing.navigationStartプロパティにセットされる時刻は次のどのタイミングのものか選択しなさい。

A. ページの表示が開始された

B. DOMの準備が完了した

C. 前のページでリンクをクリックした

D. HTMLのダウンロードが完了した

E. サーバへのコネクション作成が完了した

質疑応答

本日の内容

- 試験概要
- XAMPP
- オフラインアプリケーションAPI
- 通信通信(XMLHttpRequest,WebSocket)
- Geolocation API
- Navigation Timing

問題の解答

- 問題1: E / 更新データのダウンロードが完了して、キャッシュ更新可能になると、statusはUPDATEREADYになる。

- 問題2: D / readyStateプロパティが4になると受信完了

- 問題3: B / WebSocketはポーリングを行なわない

- 問題4: D / 現在位置の監視はwatchPosition()メソッドで行なう

- 問題5: C / navigationStartは、ページ遷移の切っ掛けになった行為のタイミングがセットされる。