フロントエンド界隈 Overview

2020年7月

ネットワーククラウド本部 - 高井 実

この講義の目的

Vue / React / Angular のハンズオンに向けて

Vue / React / Angular は JavaScript の SPA (Single Page Application) 向けフレームワークです。
SPAとは何なのか、これらのフレームワークが登場した歴史を簡単に紹介します。
ただし、この講義では近年の情報は扱いません。
それよりも、土台となる1990年代から2000年代にかけての HTML と JavaScript の歴史を扱います。
その理由は次のページを参照。

この講義で話すこと – 第一部：Web歴史学

1章

ハイパーテキストの歴史

1945年に提唱されたMemexという思想、そして1960年以降のハイパーテキストの発明の歴史

2章

WWWの登場

1980年以降、ティム・バーナーズ＝リーによって発明されたWorld Wide Webにまつわる経緯

3章

ブラウザ戦争とHTML5

Internet Explorerの登場から現代の各種Webブラウザのシェア争い、Web技術の標準化の変遷

4章

SPAとフレームワークの登場

WebページのJavaScript主体で行うSingle Page Applicationという手法の登場について

「ファインマンの教え（その壱）」（20世紀の物理学者リチャード・P・ファインマンと父メルヴィル・P・ファインマンの逸話）

リチャードが子供の頃、森で鳥を見つけると父は、「リチャード、あの鳥の名前を知っているか？あの鳥はxxxと呼ばれているんだ」と話した後、「君はあの鳥の名前を今知ったが、あの鳥自体に関しては何も知ったことにならない。さあ、一緒に鳥を観察してみよう」と話したとされるエピソードがある。

名前を知っているだけでは、その物のことを本当に知っているとはいえない。

好奇心を持ってその物を特徴を観察したり、その特徴を持っている理由を歴史から解き明かしていくと面白いかもしれません。

この講義で話すこと – 第二部：Web技術概論

5章

HTTPリクエストとレスポンス

Webページにアクセスしてからページが表示されるまでの処理の流れについて

6章

ASTとDOM（ドム）

JavaScriptでWebページを操作するための基本概念である Document Object Model について

7章

JavaScript実践

実際にJavaScriptを書いて動かしてみます。そしてSPAでは何をやろうとしているのか説明します

8章

まとめ

このスライドの内容を振り返りつつ、質疑応答やフリートークの時間にします

「ファインマンの教え（その弐）」（20世紀の物理学者リチャード・P・ファインマンと父メルヴィル・P・ファインマンの逸話）

学生になったリチャードは、ラジオを考えるだけで直すことができた。むろん手をくださなかったのではなく、ラジオの裏をあけ、いじるたびに手を休めて考えた。また、三角法を解きながら、その数学記号が気にくわなくなって、自分で記号をつくった。

何であれモノやコトは動かしたらその意味を考えることである。また、それに対しては自分なりの表現が可能である。

学んだことは鵜呑みにするのではなく自分で考えることが重要で、自分の言葉で説明できることこそが真の理解といえるのかもしれません。

Cited from 松岡正剛の千夜千冊 https://1000ya.isis.ne.jp/0284.html -『ご冗談でしょう、ファインマンさん』

第一部：
Web歴史学

テッド・ネルソンと Project Xanadu

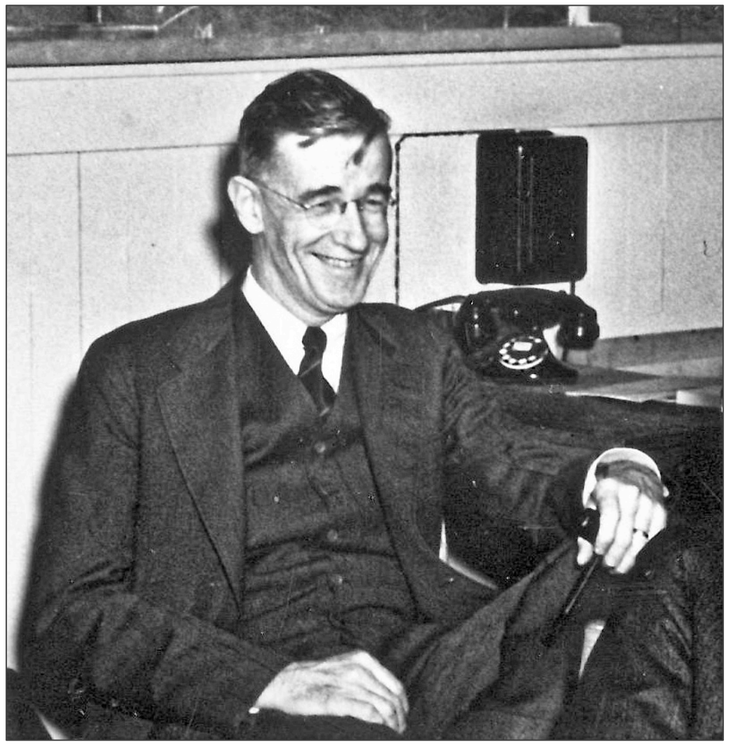
1章：ハイパーテキストの歴史

概要

1945年、ヴァネバー・ブッシュは情報検索機構 Memex の思想を "As We May Think" という論文で発表した。

1960年、テッド・ネルソンは Memex の構想に影響を受けて、これを発展させるザナドゥ計画を開始する。ザナドゥ計画は、世界初のハイパーテキストを開発するプロジェクトとなった。

(1) 記憶拡張機 Memory Extender - Memex



Vannevar Bush in 1940.
Cited from Wikimedia Commons.
https://commons.wikimedia.org/wiki/File:Vannevar_Bush_1940.jpg

1945年に公開されたヴァネバー・ブッシュの論文 "As We May Think"

以下は、山形浩生（やまがたひろお）氏による翻訳文書より引用
Cited from https://cruel.org/other/aswemaythink/aswemaythink.pdf

概要

第二次大戦が終わり、科学者は次の大きなプロジェクトを探す必要がある。計算機は進歩したが、計算だけではダメだ。科学はさまざまな成果を生み出しているが、いまや情報洪水でそれを十分に活用できていない。

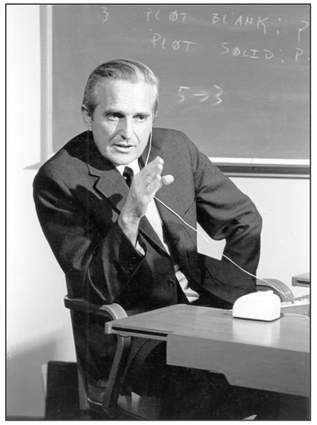
機械による情報記録、検索、関連づけを行うことで人類の知的可能性は飛躍的に向上する。記録のためのカメラ技術は発達してきて、現像不要で画像が作れる。それを保存するためのマイクロフィルム技術はある。そのフィルムの中身を拡大投影するブラウン管もある。そのマイクロフィルム内の情報を結ぶ経路を保存するためのリレー回路もある。

これを使って、情報検索と情報の結びつきや関連性まで記録保存できる新しい装置「メメックス」が実現できるのだ。それが発達すれば、いずれは各種信号を脳に直結することも可能になるかもしれない。

(2) テッド・ネルソンのザナドゥ計画

1960年、ハーバード大学の大学院生であったネルソンは、ブッシュの "As We May Think" から着想を得て、Memex に類するシステムの開発を開始した。Project Xanadu（ザナドゥ計画）である。

1965年、ネルソンは「ハイパーテキスト」と「ハイパーメディア」という用語を生み出す。逆説的ではあるがネルソンのザナドゥ計画とは、世界初のハイパーテキストシステムを開発するプロジェクトだった。



Douglas Engelbart in 1968.
Cited from Wikimedia Commons.
License CC BY-SA 3.0.
https://commons.wikimedia.org/wiki/File:SRI_Douglas_Engelbart_1968.jpg

ネルソンとは別に、1945年にブッシュの論文から影響を受けたダグラス・エンゲルバートは、1962年からスタンフォード研究所で NLS (oN-Line System) の開発に着手する。1968年に完成し、そのデモを行なった。

NLSは、世界初の実用的なGUIを伴うハイパーテキストシステムであった。

エンゲルバートはマウスの発明者としても知られる

このデモは革新的であり "The Mother of All Demos"（全てのデモの母）と呼ばれることとなった。

Ted Nelson
Cited from Wikimedia Commons.
License CC BY-SA 3.0.
https://commons.wikimedia.org/wiki/File:Ted_Nelson_cropped.jpg

(3) ハイパーテキストのその後

NLSを筆頭に、1970年代からはいくつかのハイパーテキストシステム、ハイパーメディアアプリケーションが登場した。

1987年には、アップルコンピュータがMacintosh向けに開発した商用のハイパーテキストシステムである HyperCard が披露された。これによりハイパーテキストの概念が広まることとなった。

ハイパーテキストシステムは、今でいう Wiki やブログ、バグトラッキングシステムのようなものに近い。
ページの編集はサーバ上からできる。ページ間の関連付けはRedmine等の関連チケット機能に似ている。

クローズドシステムであり全てのデータはデータベースに集約されている。

テキストとテキストの関連付けは、双方向にリンクするという概念を持っている。

(4) 余談：双方向リンク？

ネルソンが Project Xanadu で考えていたハイパーテキストは、今の WWW におけるハイパーテキストとは性質の異なるものだった。

ネルソンは全ての文献をひとつの巨大な電子図書館に収めて、文献間の引用・被引用関係、文献の著作権管理を動的に管理する構想があった。
それはトランスクルージョン、トランスコピーライトという用語でザナドゥ計画では扱われている。

Project Xanadu のハイパーテキスト構想については次の記事が参考になるかもしれない。

http://xanadu.com.au/ted/TN/WRITINGS/TCOMPARADIGM/tedCompOneLiners.html

https://gigazine.net/news/20140610-xanadu-54-development/

偉大なる Tim Berners-Lee

2章：WWWの登場

概要

1980年、バーナーズ＝リーはCERN内部向けのハイパーテキストシステムENQUIREの開発を行う。

バーナーズ＝リーはその後、ENQUIREの問題点の解決に取り組み、HTMLを提唱して1989年にWorldWideWebブラウザを公開した。

Webは産業の対象となり、1995年、第一次ブラウザ戦争が始まる。

文字ばかりのスライドなので

ここからは「いらすとや」の提供でお送りします。

https://www.irasutoya.com/

(1) ティム・バーナーズ=リーの発明

Timothy "Tim" John Berners-Lee
ティモシー・"ティム"・ジョン・バーナーズ=リー

ティム

字数の都合で ティム と表記

1980年、欧州原子核研究機構（CERN）に在籍。

「CERNの内部情報を、様々な環境・システム間でも共有できるシステムをつくって！」

と言われ、ENQUIRE（エンクワイア）を開発する。

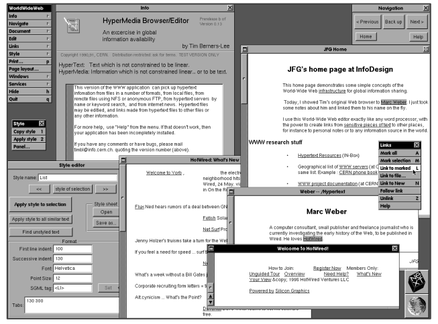
ティムは既存のハイパーテキストシステムの問題点と新たな可能性に気付く。

Tim Berners-Lee
Copyrighted by W3C.
https://www.w3.org/Press/Stock/Berners-Lee/

1989年、この仕組みを世界中で使えるようにするため Global HyperText Project を立ち上げる。

HTML というものを生み出し、World Wide Web という概念を提唱する。今の WWW (Web) である。

翌年12月、ティムは同名の世界初のWebブラウザ WorldWideWeb と CERN httpd を完成させる！



世界初のWebブラウザ
WorldWideWeb

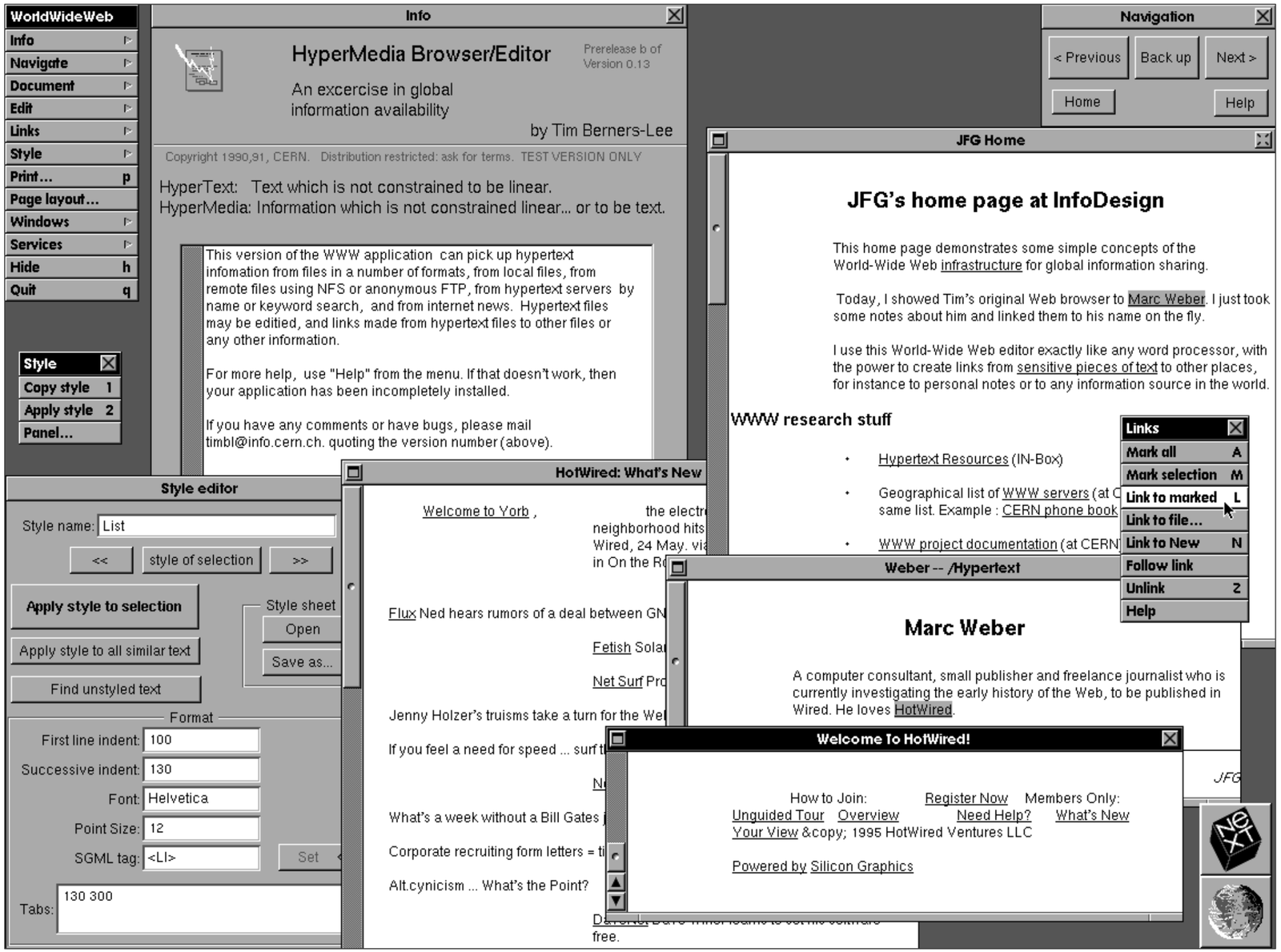
1989年リリースの OS NeXTSTEP 上で動作するGUIベースのWebブラウザ。

WorldWideWeb は1991年に公開された。

概念である World Wide Web と、ブラウザ WorldWideWeb がある。

混乱を避けるため、ブラウザは後に Nexus と改名された。

(2) 世界初のWebブラウザ - WorldWideWeb



https://www.w3.org/MarkUp/historical

Copyrighted by W3C.

Screen shot of Tim Berners-Lee's browser editor

1991年公開のブラウザ。

でも、画像はリンクを辿るとダウンロードできるだけでインライン表示はできない。

このブラウザはNeXTSTEP上でしか動かない。他のOSでは使えない。

これでは世界中の人にWWWを広められない!!

(3) libwwwの開発とNCSA Mosaicの登場

字数の都合で マークと表記 ↓

ティムはWorld Wide Web(WWW)の可能性を示すため、NeXT向けだったWWWのコンポーネントを移植性の高いC言語で書き直した。

1992年、ティムはWWW普及のため、OSによらないクロスプラットフォームなWWWライブラリとなる libwww をパブリックドメインで公開した。

そこに目をつけたのが、米国立スーパーコンピュータ応用研究所（NCSA）のマーク・アンドリーセン。

マークは libwww を基にして、画像のインライン表示や音楽再生をサポートしたWebブラウザ NCSA Mosaic を1993年に公開した。

画像が扱えるブラウザの登場でWebの利用者は急増した。これを機に世界にWebが知れ渡った。

libwww は Library WWW の意味。

Mosaic は日本では「モザイク」と表記されるが、ネイティブは「モ↑ゼイク↓」と言う。

「モ↓ゼエ↑ク↓」と言うと悟空になってしまう。
モザイクでいい。

NCSA Mosaic 3.0
Copyrighted by NCSA.
http://www.ncsa.illinois.edu/news/press

(4) ティムによるW3Cの創設

1993年にWebブラウザ Mosaic が登場し、また同時期にWebサーバ実装である NCSA HTTPd も開発されたことをきっかけに、そのブラウザの利用者がWebサイトを制作できるようになった。

しかし Mosaic が独自に「HTML要素タグ」を実装しまうと、Webサイト制作者は何のHTML タグが使えるのかわからず困ってしまう。

この状況を改善するため、ティムはHTMLの標準化のために World Wide Web Consortium (W3C) を1994年10月に創設する。

小話：Webサーバ実装 NCSA HTTPd に対する改善や機能追加がなかなか進まなかったことがきっかけで、それを行う目的の Apache Group が1995年に創られた。

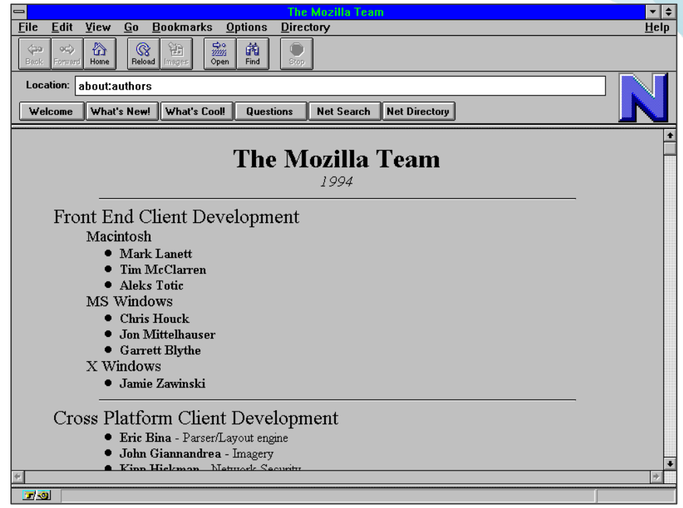
(5) マーク・アンドリーセンのその後

Marc Andreessen
Cited from Wikimedia Commons.
License CC BY 2.0.
https://commons.wikimedia.org/wiki/File:Marc_Andreessen.jpg

マークはNCSA在籍中にMosaicを開発し1993年に公開。その翌年に Mosaic Communications を創設した。

しかし数ヶ月後にNCSAがMosaicの権利を主張してきたためマークはMosaicの権利を破棄してNCSAと決別する。

1994年11月に Netscape Communications に企業名を変え、Mosaicに変わる新たなブラウザ Netscape Navigator を開発した。



Netscape Navigator 1.0
Cited from The Andrew Turnbull Network.
http://www.andrewturnbull.net/nscape1.html

Mozilla (the mascot of the Netscape)
Cited from The Mozilla Museum.
http://home.snafu.de/tilman/mozilla/

Mozilla の由来 – Netscape Communications のマスコット

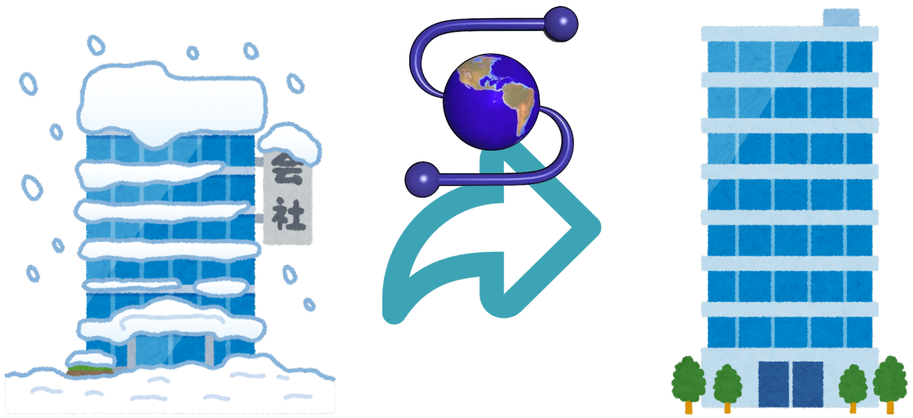
Netscape は The Mozilla Team によって開発されていた。ゴジラ (Godzilla) のもじりでもあるが、すでにブラウザシェアのトップだった Mosaic を追い抜くことを目指し Mosaic-Killer の意味を込めて名付けられたと言われている。(出典: Net Loss: Internet Prophets, Private Profits, and the Costs to Community – 2002. Nathan Newman)

(6) いっぽう そのころ・・・・

Spyglass, Inc. という1990年に創設された企業があった。この企業はNCSAの技術を商業化することを目的に設立された。

1994年にマークから Mosaic の権利を奪ったNCSAは、同年5月に Spyglass に Mosaic のライセンスを供与し独自のブラウザ開発に挑む。

そして Spyglass Mosaic が完成する。



Microsoft Internet Explorer

IE 2 のロゴ

Microsoft Internet Explorer

IE 3 のロゴ

Internet Explorer logos
Credit: Wikimedia Commons.
https://commons.wikimedia.org/wiki/Category:Internet_Explorer_logos

(7) いっぽう そのころ・・・

1994年、Web産業への参入を目論んでいた Microsoft は Spyglass からライセンスを取得して Spyglass Mosaic のコアをそのまま流用する形で Internet Explorer を開発する。

1995年、Internet Explorer が使える Microsoft Windows 95 が発売される。

インターネットが必需品となる今の生活がここから始まり、Netscape Navigator vs. Internet Explorer の第一次ブラウザ戦争が勃発する。

Mosaic は IE のご先祖様！

2章までのまとめ

ハイパーテキストの発案、World Wide Web の発明から
Netscape Navigator / Internet Explorer の登場まで。

1945

Memexとハイパーテキスト

Memex構想とテッド・ネルソンのザナドゥ計画。
そしてダグラス・エンゲルバートのNLS開発。
ハイパーテキストはここから始まった。
AppleもHyperCardを開発。スティーブ・ジョブズはNeXT社を創設しNeXTSTEPを開発する。

WWWの発明

ティム・バーナーズ＝リーによるENQUIREの開発、HyperText Project の発足、World Wide Web の概念、そして1989年、NeXTSTEP 上で動作する世界初のブラウザ World Wide Web を開発する。後に W3C を創設する。

MosaicとNetscapeの登場

マーク・アンドリーセンによる、画像表示をサポートした NCSA Mosaic のリリース。
そして NCSA に見切りをつけたマークは Netscape Comnunication を創設し1994年に Netscape Navigator を公開する。

Internet Explorerの登場

Spyglass Mosaic をベースに、Microsoft は Webブラウザ産業に参入する。
そして1995年に待望の Windows 95 を発表し、1996年末から Internet Explorer 入りの Windows 95 が爆発的な人気となる。

1995

余談：WWWの登場からNN/IEの登場まで

Netscape Navigator / Internet Explorer

1 NeXTSTEP と Machintosh の関係

Machintosh（マッキントッシュ）は Apple が開発する今の macOS の最初の姿です。初代 Machintosh は1984年に発売されました。

Macintosh は画期的ではあったものの当時はインターネットもなくコンピュータ需要がなかったため販売数が伸びず、スティーブ・ジョブズは責任を取る形で Apple 社を追放されます。追放されたジョブズは Apple とは別にコンピュータの開発を行うため1985年に NeXT Software, Inc. を創業します。そして1989年に NeXTSTEP を公開したのです。その後、ジョブズは Apple に戻り NeXTSTEP で設計した UI を Machintosh へ取り入れました。

2 Microsoft Windows 95 の戦略変更

ビルゲイツはインターネットの普及はまだ先であると考えていたため、初期版の Windows 95 では Internet Explorer は別売りオプションでした。

しかしその判断は誤りだったと気づき、Windows 自体の機能拡張と合わせて1996年末に Internet Explorer をバンドルした Windows 95 OSR2 というバージョンを販売します。これは実質的に Windows 96 でしたが、営業戦略上 Windows 95 のアップデートという形で "Windows 95" の名称は変えずにリリースしました。この戦略は成功し「Windows 95 があればインターネットが使える」と認知され今日の Windows 人気に繋がっています。

3 このスライドについて

このスライドの最後のページにも示していますが、「Sirius PowerPoint Template v1.0」というデザインテンプレートを使っています。

しかし、いらすとや画像を使い始めたあたりからスライドデザインの方向性を見失いつつあります。

そんな心配もしながら、ここまでの時系列を振り返った後に、
時代別のブラウザ戦争「第一次：1995年～2000年」、「第二次：2000年～2014年」、「第三次：2014年～現在」のお話をしていきます。

1995年から現在までのブラウザ史

3章：ブラウザ戦争とHTML5

概要

第一次ブラウザ戦争：1995年〜2000年（NN/IEの2強時代）
第二次ブラウザ戦争（前期）：2000年〜2008年（IE6全盛期）
第二次ブラウザ戦争（後期）：2008年〜2014年（Chromeの登場）
第三次ブラウザ戦争：2014年〜現在（HTML5時代）

Webブラウザ商戦が始まり、各ブラウザは独自の機能追加を行う。
それはWebの標準化とは反する道だった。WWWの運命はいかに。

(1) 第一次ブラウザ戦争：1995年～2000年

IE: https://commons.wikimedia.org/wiki/File:Internet_Explorer_4_and_5_logo.svg
NN: https://commons.wikimedia.org/wiki/File:Netscape_icon.svg

Microsoft
Internet Explorer

VS.

Netscape Navigator

NN と IE の2強時代

NN 4.x / IE 5 / IE 5.5 の登場
- Windows / Mac OS / UNIX の主要ブラウザ
- IE の Mac OS / UNIX サポートは IE 5 が最後となった

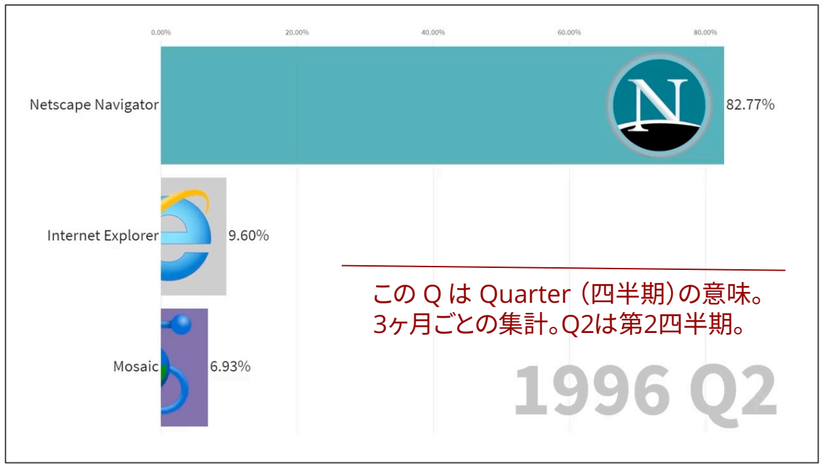
Opera 1.0 は1995年に公開されているが…
- この時代は NN / IE が99%以上のシェアを占めていた

1995年にリリースされた Netscape Navigator 2.0 では、ブラウザ上でスクリプトを動作させて動的にコンテンツを操作する仕組みが実装された。これは LiveScript という名称で開発されていたが、Netscape Communications Corporation と提携関係にあった Sun Microsystems（現Oracle）が開発し、当時人気を博していた Java にあやかって JavaScript という名称に改名される。

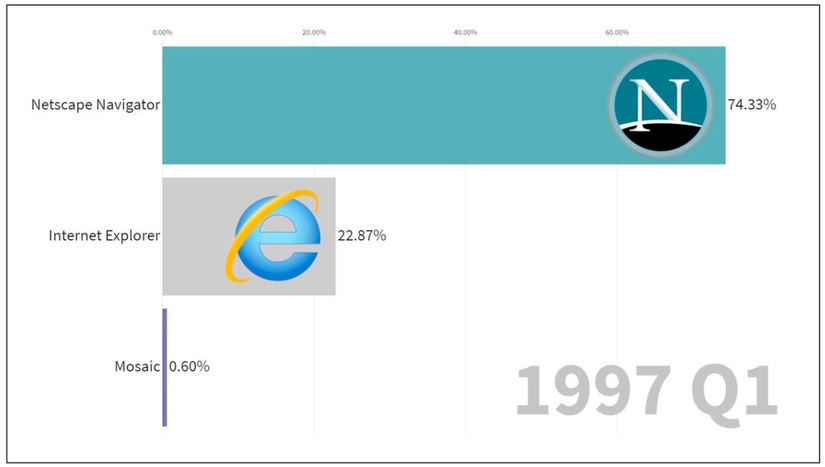
DHTML! since 1997.

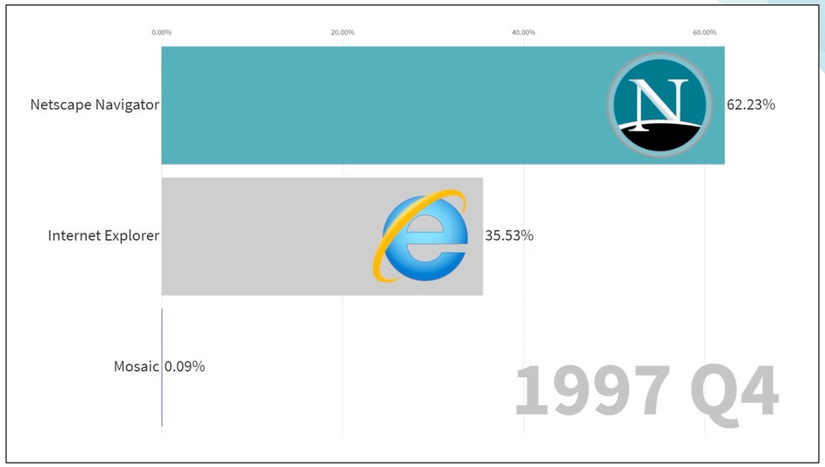
対抗馬の Microsoft も Internet Explorer 3.0 で JScript というスクリプト言語を実装していた。
ライバル社同士が競って独自の機能開発を行ったため、Netscape Navigator 向けのスクリプト実装は Internet Explorer で動作しないといった状況が当たり前であり、Webサイトを開発する側は2種類のコードを書かざるを得ない状況だった。処理用のAPIがNNはLayer、IEはDOMと異なる概念だった。

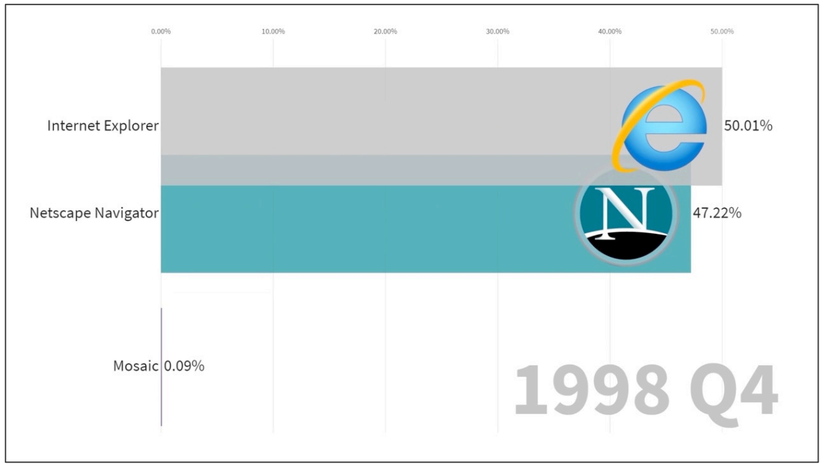
(2) 第一次：1995年～2000年のブラウザシェア

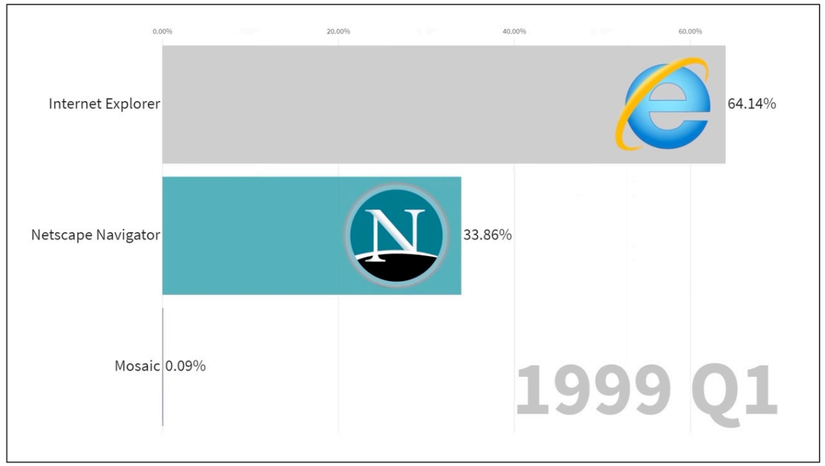


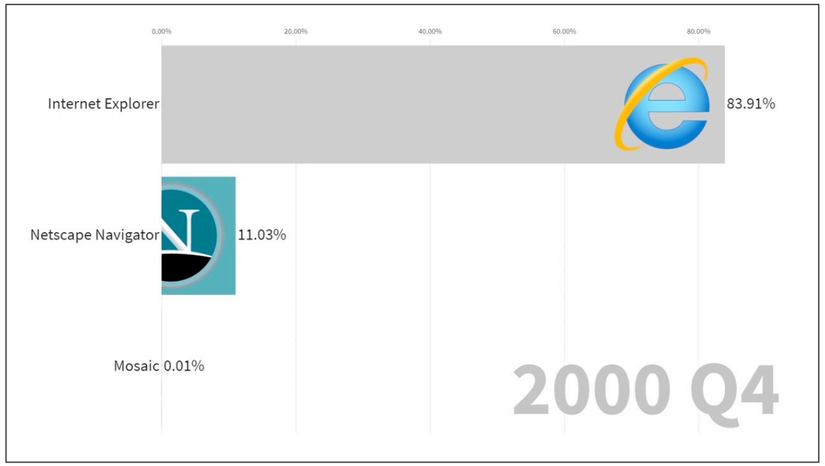
この Q は Quarter（四半期）の意味。
3ヶ月ごとの集計。Q2は第2四半期。











Cited from "Data Is Beautiful" - reddit
https://www.reddit.com/r/dataisbeautiful/comments/cxuah9/usage_share_of_internet_browsers_1996_2019_oc/
YouTube: https://www.youtube.com/watch?v=es9DNe0l0Qo

独擅場（どくせんじょう）という言葉よりも独壇場（どくだんじょう）が
市民権を得てしまい、NHKや教育出版でも後者を正としている。

Netscape Navigator / Internet Explorer / NCSA Mosaic およびその他のブラウザシェア。

IEはWindows 95にバンドルされ徐々にシェアを伸ばす。1998年7月、IE4が標準搭載のWindows 98がリリースされ、圧倒的なWindows時代にブラウザのシェアはIEの独壇場となった。

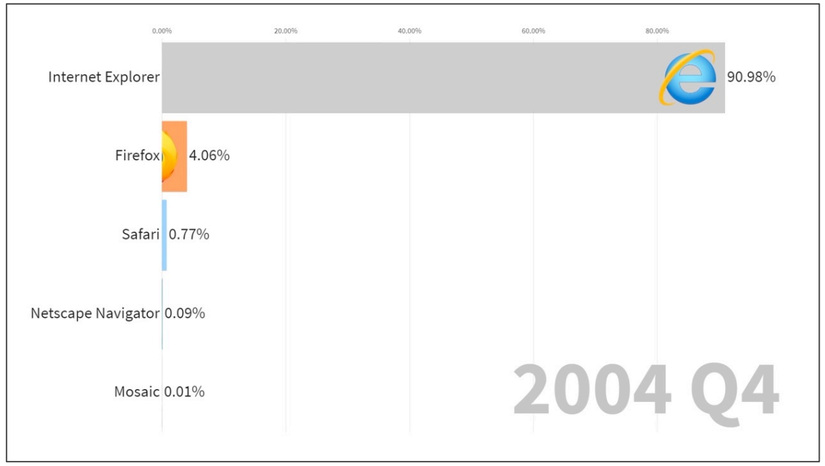
2000年時点でも、Operaを含むその他のブラウザは1%にも満たない NN / IE 2強時代だった。

(3) 第二次ブラウザ戦争前期:2000年～2008年

Mozilla Mascot in 1998. - Red design
Credit: en.wikipedia.org
https://en.wikipedia.org/wiki/Mozilla_(mascot)

～ Mozilla の逆襲 ～ Web標準の幕開け

生まれ変わった Netscape 7.0
Mozilla.org の真価 Firefox のリリース
Mac OS X 10.3 標準ブラウザ Safari の登場



W3CによるDOMなどの標準化、IE4がサポートしたHTML 4.01やCSS2、DOMなどと離れた実装をしていても今後の展望が見込めないNetscape CommunicationはNetscape 5.0の開発を断念しソースコードを公開する。Netscapeの開発者とコミュニティはそのソースコードを参考にしつつ全く新しいブラウザ開発のため Mozilla.org を創設する。

Mozilla.org はブラウザ Mozilla を開発した。Mozilla の成果は Netscape 6.0 / 7.0 へも利用された。後に、Netscape とは機能を独立した Mozilla 独自のブラウザを開発した。それは2004年に Firefox としてリリースされる。

他の企業もこの時代に大きく進展した。勢いに乗る Microsoft は2001年に IE6 を搭載した Windows XP を発売する。OS の対抗馬である Apple は2003年に Mac OS X (10.3) の標準ブラウザ Safari をリリースする。

Mozilla や Apple はWeb標準に準拠することを目指した。2006年には IE7 がリリースされるが、当時はブラウザのインストールが手間だったこともあり Windows XP にバンドルされた IE6 はその後も長年使われ続けることになる。

文書型宣言? DOCTYPEスイッチ? IE5.5 再現モード?

Webサイト制作の地獄時代。悪魔のキーワード「IE6対応」。飛躍的に発展したブラウザの機能が使えず、2001年のブラウザを考慮し続けることに。

(4) 第二次前期：2008年までの五大ブラウザとIE6

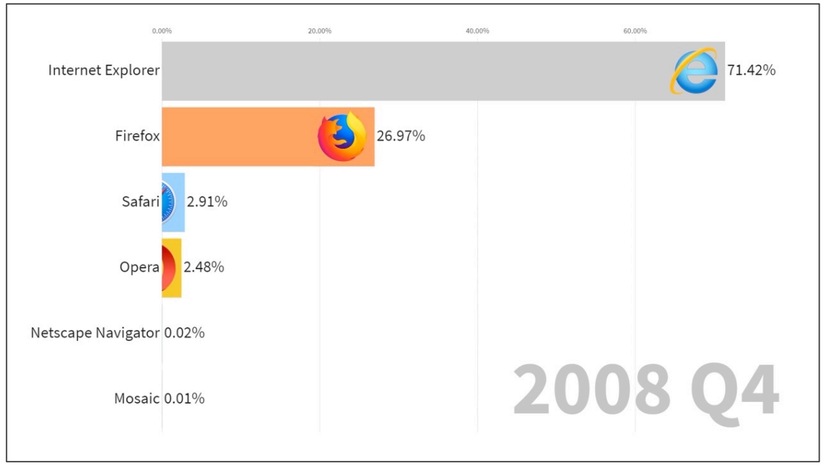
インターネットが普及

深夜定額のテレホーダイから
フレッツISDN/ADSL/CATV/FTTH
による常時定額通信の普及

Flash黄金時代　2ちゃんねる

個人サイト/ブログ　mixi（ミクシィ）

誰もがインターネットを使う時代に



Windows XPの時代、IE6は多くの人に使われ続けた。Windows Vistaは2006年に発売されたが、Windows XPは完成度が高く非常に優れたOSであったこともあって2001年から2010年近くまで現役OSとして使われた。2009年にはIE8がバンドルされたWindows 7が発売されたこともあり、IE6の利用者は次第に減るものの、IE7が登場してから3年経っているにもかかわらずIE6のブラウザシェアは2009年時点で20%以上もあった。

2005年、GoogleがJavaScriptの非同期通信（Ajax）を活用したコンテンツ Google Maps を公開した。Firefox や Safari、Opera などの「モダン」なブラウザは JavaScript を利用してリッチなコンテンツを開発することができた。

そのような新しいブラウザの機能、表現が登場する時代でもIE6の利用者がいた。「IE6の利用者がいる以上は、IE6でもきちんと動作するコンテンツを作らなくてはいけない」と、Webサイトの開発を発注するクライアント（顧客）は言う。IE6が完全に消えるまで、Webサイト制作者は「レガシー」なブラウザをサポートし続けなければならなかった。

出典：IE6の2009年のブラウザシェア https://gs.statcounter.com/browser-version-market-share/all/worldwide/2009 およびIE6対応をし続けてきたこの資料の著者の知識

JavaScript ライブラリ Prototype.js と jQuery の登場

2000年代のブラウザは、W3Cによる標準化の思想はあったものの、IE6を始め JavaScript API の標準化はそれほど実現できていなかった。

例えば「ボタンをクリックしたらコンテンツの表示/非表示を切り替える」という実装を JavaScript で記述する場合、IE と Firefox で呼ぶべき API が全く別のものだった。また、Google Maps では Ajax を使っているが、これと似た処理を実装するのはブラウザの実装差異のせいで非常に大変だった。

そうした状況を改善するため、実装の違いを吸収するライブラリが登場した。

2005年には Prototype.js が、それを追うようにして2006年には jQuery が登場した。どちらも、ブラウザの種類を気にせず簡単に DOM を操作するための API を提供する。

これは DOM Manipulation Library と呼ばれる。

IE: https://logos.fandom.com/wiki/Internet_Explorer – CC BY-SA
NN: https://commons.wikimedia.org/wiki/File:Netscape_7.2Logo.png – Public Domain
Safari: https://logos.fandom.com/wiki/Safari – CC BY-SA
Firefox: https://commons.wikimedia.org/wiki/File:Mozilla_Firefox_2004_Logo.png – MPL 2.0
Opera: https://logos.fandom.com/wiki/Opera – CC BY-SA


Safari - Apple


Firefox - Mozilla


Opera - Opera

(5) Web標準
W3C と WHATWG

1994年にW3Cを創設し、HTML仕様などの標準化を進めたバーナーズ＝リー。このおかげで1990年代後半にはHTML 4.01を始めとするWWW関連技術の土台が作られ、異なるブラウザでもWebページを利用することができた。

しかし第二次ブラウザ戦争に入り、Webブラウザ開発者が実現したいWebの方向性、Web開発者のニーズがW3Cの標準化には反映されていない状況に不満をもった Apple, Mozilla, Opera が結集し Web Hypertext Application Technology Working Group (WHATWG) を2004年に創設する。（出典：WHATWG https://whatwg.org/faq - What is the WHATWG?）

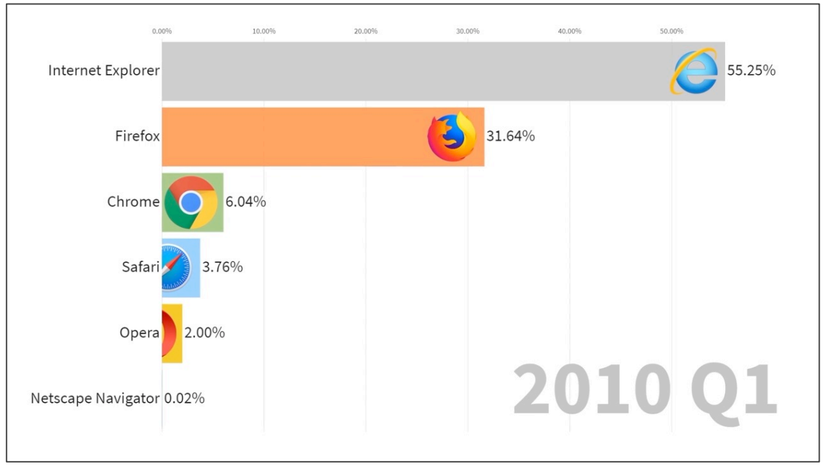
WHATWG は W3C の標準化をベースとしつつ、Webコンテンツに求められる表現方法やクリエイティブで発展的な機能仕様を、安全に利用できるようなセキュリティ仕様とともに積極的に策定し、WWWの最前線で新しい価値の創造を画策した。

この活動は2014年、WHATWGの創設から10年後に大きな成果を上げることとなる。

(6) 第二次ブラウザ戦争後期：2008年～2014年

Google Chrome の登場

各種モダンブラウザのWeb標準サポートの強化
IE8 の登場による IE6 / IE7 の衰退



レガシーブラウザが消失し、モダンブラウザが進化し続ける時代

2008年に Google Chrome が登場し、同年に進化した Firefox 3.0 がリリースされた。
2009年には IE8 と Windows 7 が登場し、IE6/7 と Windows XP の姿も消えていった。

それから数年、各ブラウザはさらに進化・Web標準化が進む。

2011年から2013年には1年ごとに IE9, IE10, IE11 がリリースされ、悪夢の9年間だった IE6 全盛期の状況は大幅に改善された。

この頃になると jQuery のようなライブラリを使わずともWeb開発がそれほど困らなくなり、より高度なJavaScript開発を行うための「フレームワーク」が登場し始めた。

JavaScript でアプリ開発!?
MVCフレームワーク
や
SPAフレームワーク

2005年以前の JavaScript は、Webページの文字や画像を「クリック」したら表示したり、文字サイズを動的に変更したりと、「ちょっとした振る舞い」を実現する用途で用いることが大半だった。

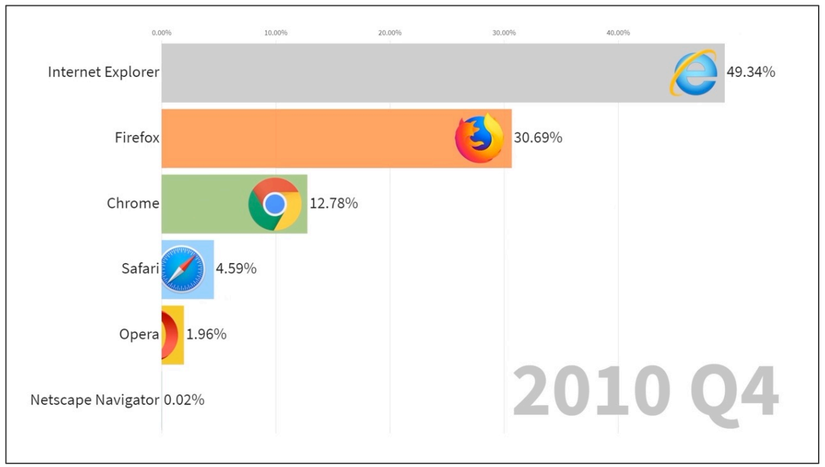
2005年、Google Maps によって Ajax の可能性が理解され始めると JavaScript 経由で別のWebページの情報を取得して表示するといった利用法が増え始めた。

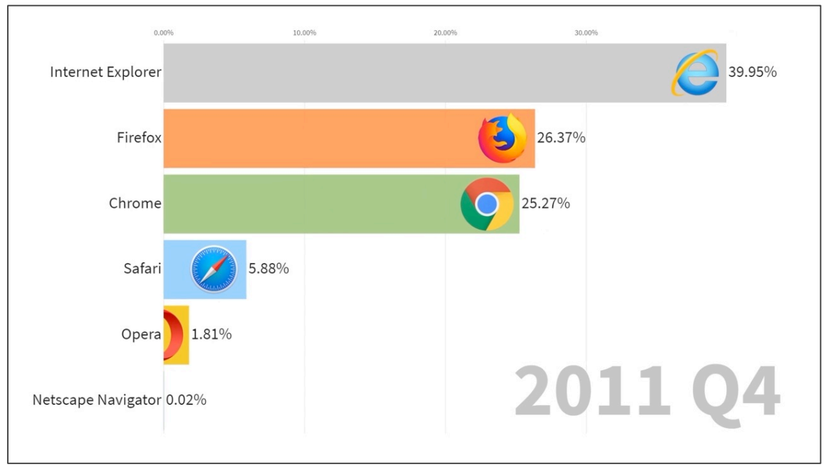
これらを応用すると、例えばECサイトの商品一覧ページで、画面遷移を伴わずに商品一覧を切り替えて表示するといったことが可能になる。

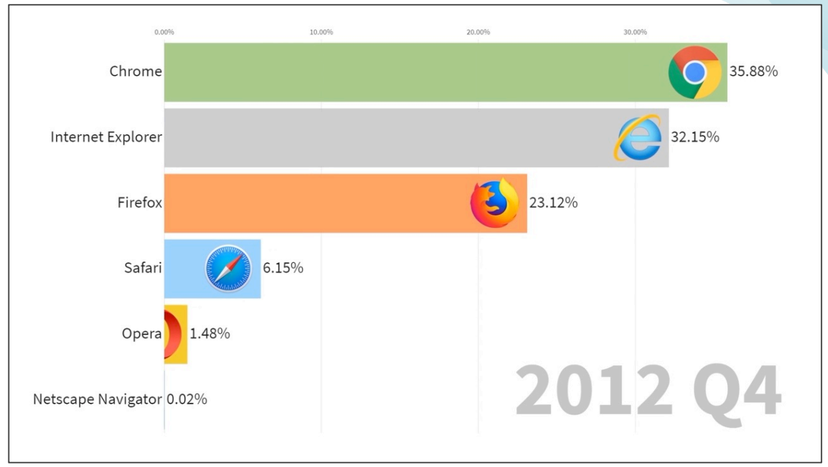
それを実現するために JavaScript でMVCパターンの設計ができる Backbone.js といったフレームワークが登場した。

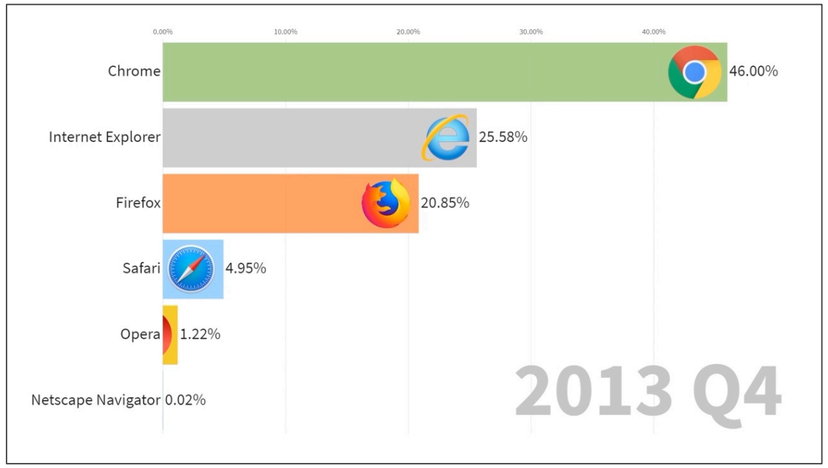
その後、Webサイト全体を同様に管理する手法が考案される。Single Page Application の登場だ。

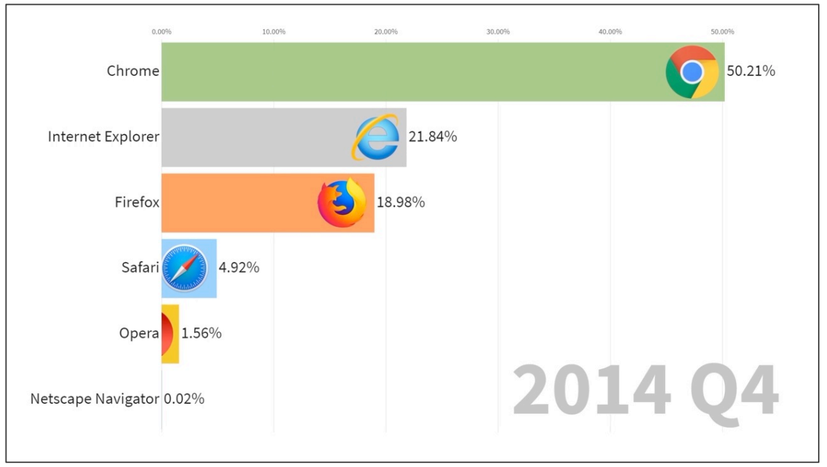
(7) 第二次：2008年～2014年のブラウザシェア











Cited from "Data Is Beautiful" - reddit
https://www.reddit.com/r/dataisbeautiful/comments/cxuah9/usage_share_of_internet_browsers_1996_2019_oc/
YouTube: https://www.youtube.com/watch?v=es9DNe0l0Qo

2008年に登場した Google Chrome は徐々にシェアを伸ばし、Windows XP 時代から Windows 7 時代に移り変わったことも影響して Windows ユーザからも多くの利用者を獲得する。そして2012年の3Q/4Qには、ついにIEからトップシェアの座を奪い、Google Chrome の時代を築いていく。

HTML5: https://commons.wikimedia.org/wiki/File:HTML5_logo_and_wordmark.svg – CC BY.

HTML5 の意味は狭義と広義があり
HTML に限らず CSS や DOM,
その他の関連技術を含めた概念は
広義の HTML5 を呼ばれる

(8) 第二次後期：HTML5の登場

「インタラクティブな機能だって？
それなら1998年頃からすでに Flash があったじゃないか！」

という意見がFlash黄金時代を知っている人から出てきそうだ。

2008年、日本でスマートフォン iPhone 3G が発売されたのを機にスマートフォン時代がやってきた。しかし iPhone は Flash に非対応だった。Flash が非対応であったことは一部の人々を落胆させた。

これは Apple の意図的な戦略だった。Flash はライセンスの問題で利権が絡み誰もが自由に使える技術ではなく、また、セキュリティ上の致命的な問題を含んでいたため、むしろ HTML5 に期待していたのだ。

Flash 終了の顛末は以下の記事「Flashにとどめを刺したのは」が詳しい。
https://www.itmedia.co.jp/enterprise/articles/1708/24/news018.html

HTML仕様である HTML 4.01 / XHTML 1.x は2000年頃に勧告され、第二次ブラウザ戦争中にずっと使われてきた。

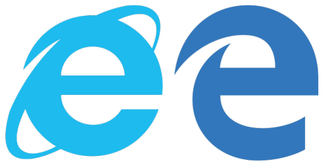
しかし JavaScript フレームワークを用いた開発が行われるようになった時代には、各種モダンブラウザが canvas 要素や video 要素といったインタラクティブな機能を実装してきた。SVG サポートや、従来のHTML要素の使い方も見直され、より使いやすいHTML仕様へと変わっていった。これらはWHATWGが何年も前から進めてきた成果である。

その集大成が HTML5 という名称で、2014年についに勧告された。HTML5 時代の幕開けである。

(9) 第三次ブラウザ戦争：2014年～現在

Browser Icons: https://logos.fandom.com/wiki/Logopedia - License CC BY-SA.

HTML5 時代の五大（七大）ブラウザ



IE11 & Edge

Safari

Google Chrome

Firefox

Opera & Vivaldi

IE11 は2011年リリースだが勧告される前の HTML5 を一部サポートしている。

Vivaldi は Opera 創設者が独立して開発し、2016年にリリースされたブラウザ。

2020年時点のブラウザシェア（全てデスクトップ版）

Google Chrome	70%
Mozilla Firefox	10%
Microsoft IE11 & Edge	10%
Apple Safari	6%
その他	4%

出典：statcounter - Desktop Browser Market Share Worldwide | Jan - Dec 2019
https://gs.statcounter.com/browser-market-share/desktop/worldwide/#monthly-201901-201912-bar

HTML5 時代に突入し、Webブラウザでの表現や可能性はそれ以前にも増して大幅な進展を遂げた。

canvas 要素や WebGL、CSS3 の transition によるインタラクティブなグラフィックス表現。
WebSocket や LocalStorage、また pushState 等の History API による高度な JavaScript 処理。
Same Origin Policy や Cross Origin Resource Sharing による安全なスクリプトの実行など。

また、アジャイルソフトウェア開発が採用されることが増え、一部のブラウザは数年に一度のリリースであったのが、6週間に一度というハイペースに変わってきた（ラピッドリリースと呼ばれる）。
Firefox や Google Chrome はブラウザのバージョンが2020年時点で "80" 近くにもなっている。

3章のまとめ

ブラウザ戦争とWeb標準化の歴史を20行でまとめます。

「IE6は9年前の腐った牛乳」- 2010. Microsoft Australia
Microsoft はセキュリティ上の理由で IE6 の利用停止を呼びかけた。

1995

天下のWindowsとIE

Netscape と IE の第一次ブラウザ戦争により JavaScript の独自機能拡張が進められた。Windows 98 & IE4 そして Windows XP & IE6 の時代。Internet Explorer がブラウザシェアのトップを維持していた。

五大ブラウザとIE6

Netscape は Mozilla を創り Firefox を生み出した。Apple も Safari を開発した。IE6 が生き続け、DOCTYPEスイッチという概念もあった。ブログやFlashなど消費者がコンテンツを生み出す中、WHATWG はWeb標準化に動き出す。

Chromeの登場とHTML5

Windows 7 の登場によりレガシーブラウザの利用者が激減し、モダンブラウザがサポートする最先端の機能を活用したWebアプリ開発が可能に。そして2014年、それらの集大成となる HTML5 の仕様が勧告された。

HTML5時代のWebサイト

jQuery のようなブラウザ差異を扱うライブラリの需要が減り、ページ遷移なしでリッチなUIを提供するため JavaScript ベースでWebサイトを開発する手法が流行する。その最新系が Single Page Application (SPA) である。

2020

Single Page Application の登場

4章：SPAとフレームワークの登場

概要

Web標準の JavaScript API と jQuery ライブラリの関係、
MVCフレームワークの登場、それを発展させたSPAフレームワークについて紹介していく。

Vue.js / React / Angular とは一体なんなのか。

(1) JavaScriptでできること（一例）

ブラウザにより実装が異なったり
サポートしていないことがある

イベントの登録(listen)と発火(fire)

「ブラウザがHTMLソースコードを解釈してDOMの構築が完了したら—」、「ボタンをクリックしたら—」と何らかのタイミングをきっかけに処理を行うための仕組みが「イベント」である。

対象に対して「イベント登録」をしておくと「発火」時に実行される。

どんな風に書くの？

標準の JavaScript API では、Document や HTMLElement に対して addEventLisnter(type, listener, useCapture) を呼び出す。type が "click" で listener にコールバック関数を渡せば、その対象をクリックした時に listener の処理が実行される。

DOMの操作

HTMLで書かれた文字列データをブラウザが解釈すると、内部で抽象構文木 (AST) を作成する。その一つとして DOM ツリーが作られる。JavaScript から DOM ツリーを操作することで動的に内容を変更したり、要素を追加したりすることができる。

どんな風に書くの？

let button = document.getElementById('button-1') で id="button-1" と設定された要素を取得できる。これに対して button.addEventListener(...) を呼べばイベントを登録できる。DOM を取得する API は他にも querySelector() などがある。

locationやhistoryの変更

Location API で現在のページ URL を書き換えることができる。URL のパスやクエリを書き換えるとページ遷移が発生する。

History API を用いるとページ遷移を発生させずに URL を書き換えることができる。SPA の真価を発揮するには必須機能だ。

どんな風に書くの？

location.href で現在の URL が取得できる。

location.href = 'https://example.com/' のように代入すればページ遷移する。history.pushState(state, title, url) とすれば、ページ遷移することなく URL のみを書き換えられる。

(2) jQueryでできること（一例）

ブラウザのサポート状況や
実装差異を気にせず実装ができる

イベントの登録(listen)と発火(fire)

「ブラウザがHTMLソースコードを解釈してDOMの構築が完了したら—」、「ボタンをクリックしたら—」と何らかのタイミングをきっかけに処理を行うための仕組みが「イベント」である。

対象に対して「イベント登録」をしておくと「発火」時に実行される。

どんな風に書くの？

取得した jQuery オブジェクトで on(type, listener) を呼ぶとイベント登録ができる。カスタムイベントも登録でき、jQueryオブジェクトで trigger(type) を呼ぶと発火できる。「クリックしたら」と登録しておいて trigger() を使えばクリックしたことにできる。

DOMの操作

HTMLで書かれた文字列データをブラウザが解釈すると、内部で抽象構文木 (AST) を作成する。その一つとして DOM ツリーが作られる。JavaScript から DOM ツリーを操作することで動的に内容を変更したり、要素を追加したりすることができる。

どんな風に書くの？

$() という jQuery メソッドを使う。このメソッドは「セレクタ文字列」を引数に取ると CSS 同様にマッチした要素を jQuery オブジェクトとして取得できる。let button = $('#button-1') として、button.on('click', function() {...}) でイベント登録ができる。

locationやhistoryの変更

Location API で現在のページ URL を書き換えることができる。URL のパスやクエリを書き換えるとページ遷移が発生する。

History API を用いるとページ遷移を発生させずに URL を書き換えることができる。SPA の真価を発揮するには必須機能だ。

どんな風に書くの？

jQuery は DOM Manipulator Library なので、DOM操作以外の機能は基本的に備えていない。

つまりこれは jQuery を使って簡単に書くことはできない。標準の JavaScript API で記述する。

(3) SPAでできることとは？

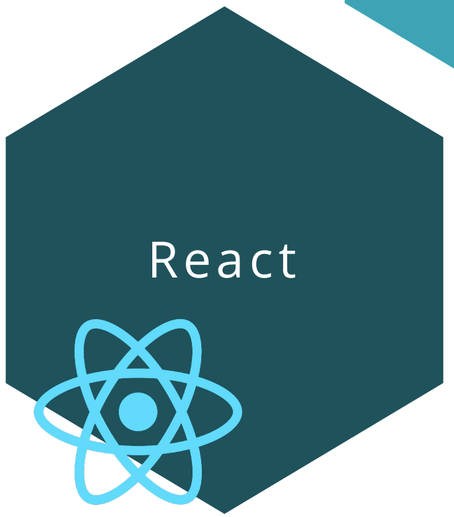
Single Page Application は
ページ遷移なしで機能を提供する手法

従来のWebアプリケーションは、何か操作をするごとにページ遷移（ページの読み込み）をするのが当たり前だった。

ページ遷移なしにシームレスかつリッチなUIを提供しながら機能（サービス）を提供するために、ページ遷移なしで従来のページ遷移後に必要な処理を全て JavaScript で擬似的に処理してしまうという方針で設計されたWebサイト（Webアプリケーション）を Single Page Application (SPA) という。

フルスタックフレームワーク

Angular はこれだけで全ての機能が完結するフルスタックなSPAフレームワーク。2009年に公開されたAngularJS (1.x)系と、アーキテクチャを一新し2016年に公開された Angular がある。Google によって開発されている。



Facebook開発の副産物

2011年、Facebookの広告管理機能のUI開発において、実装の煩雑化を避けるために考案されたコンポーネント。後にオープンソース化され2013年に公開された。Vue と同様にプログレッシブフレームワークである。

AngularJSの生まれ変わり

AngularJS の開発者の一人がAngularの主要な機能をシンプルに実装したいと考えて開発されたもの。部分的なアーキテクチャのみを提供するプログレッシブフレームワークである。2014年にリリースされた。

三大SPAフレームワーク

宗教上の理由により私から説明できることは以上です。

第二部：
Web技術概論

HyperText Transfer Protocol

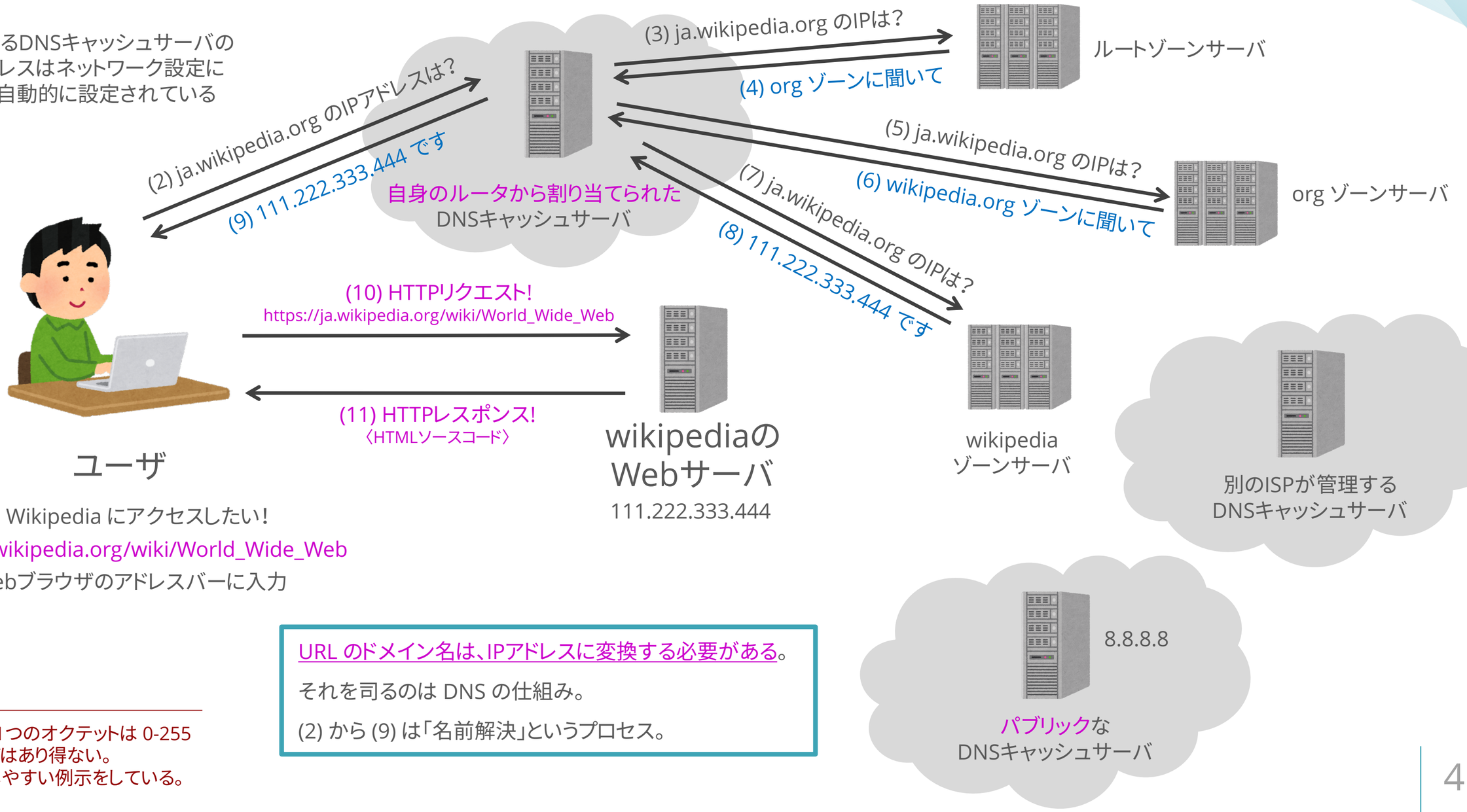
5章：HTTPリクエストとレスポンス

概要

Webブラウザのアドレスバーに URLを入力してWebページにアクセスし、Webページが表示されるまでの処理の流れを確認する。

HTTPリクエストとHTTPレスポンスの概念、HTMLリソースと、その他のCSSやJavaScript、画像などのリソースが読み込まれる段取りを確認する。

(1) DNSの名前解決とWebサーバへのリクエスト



URL のドメイン名は、IPアドレスに変換する必要がある。

それを司るのは DNS の仕組み。

(2) から (9) は「名前解決」というプロセス。

IPv4アドレスの1つのオクテットは 0-255 であり 333 などはあり得ない。
直感的に理解しやすい例示をしている。

(2) Webサーバへのリクエストは1回で終わらない

ユーザ

(1) HTTPリクエスト!
https://ja.wikipedia.org/wiki/World_Wide_Web

(2) HTTPレスポンス!
〈HTMLソースコード〉

```
<html>
...
<link href="/style.css" />
<script src="/script.js"></script>
...
<img src="/img/image.png" />
</html>
```

(3) ブラウザは受け取ったHTMLをパース（解釈）する
HTMLソース中にある外部リソースのURLを控える。

(4) HTTPリクエスト!
https://ja.wikipedia.org/style.css

(5) HTTPレスポンス!
〈CSSソースコード〉

wikipediaの
Webサーバ

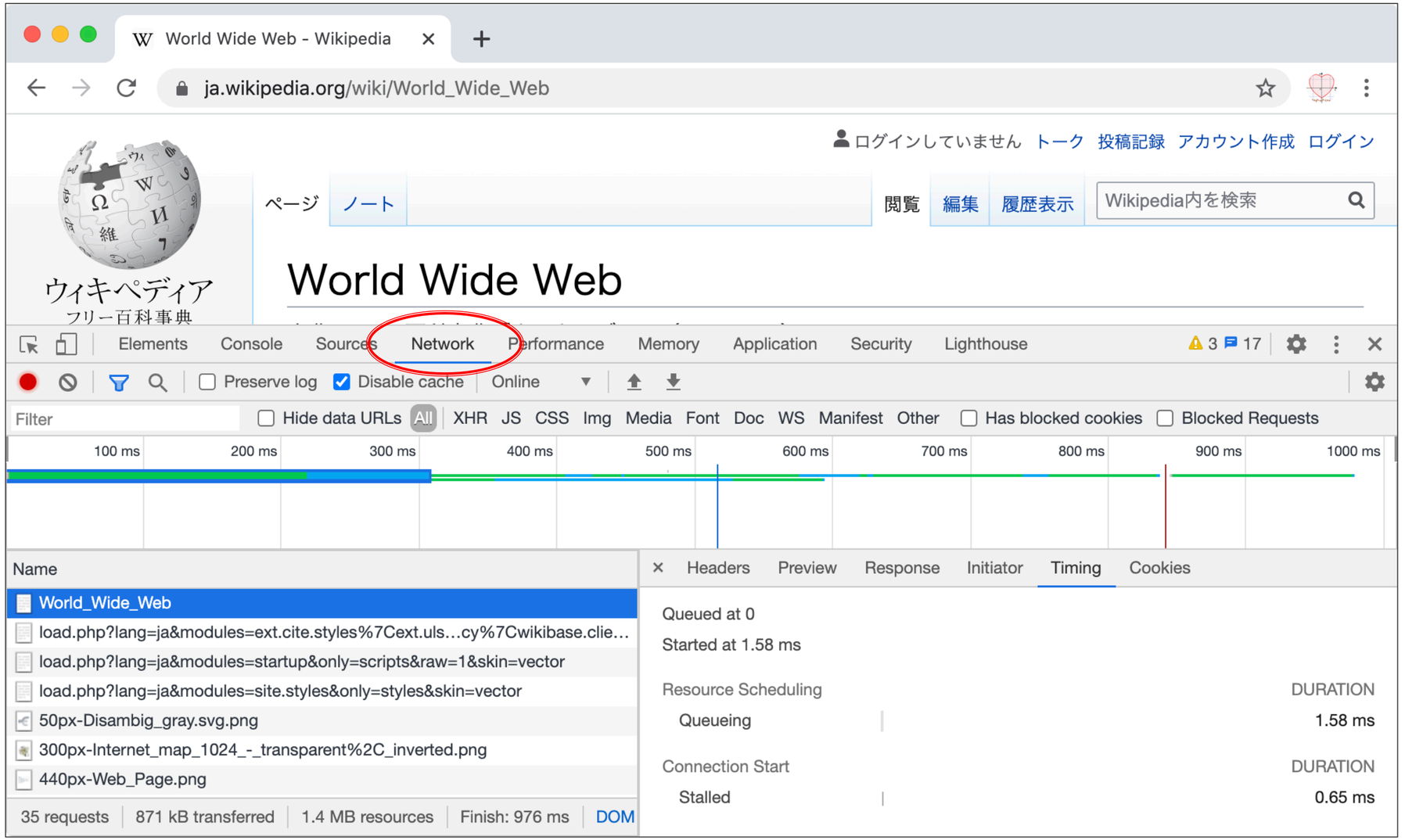
レンダリングをブロックしない非同期読み込み、およびDOMツリーやCSSOMツリー、レンダリングツリーはここでは説明しない。

JavaScript や画像も同様にHTTPリクエストを行いレスポンスを待つ。

全てのレスポンスを受け取ったら、ブラウザは全てのリソースを組み合わせてパースし、スクリプトの実行やスタイルシートの反映を行なった上でWebページを表示する。

ここでは簡単のため外部リソースをwikipedia サーバから取得する例を書いたが、別のWebサーバから取得する場合ももちろんある。

(3) 実際のHTTPリクエストの確認方法



使っているブラウザの開発者ツールから、Network パネルを開く。

左の図は Google Chrome の例である。

ブラウザによっては、そのリソースの読み込みに何秒かかったか、各リソース間で読み込み順序の優先度をどのように決めてリソース読み込みを行なったかといった情報を知ることができる。

Disable Cache でブラウザキャッシュを無効にすると挙動がわかりやすい。

読み込み速度を制御する Throttling 機能もある。（赤丸の右下の Online とあるところを変更する）

抽象構文木 Abstract Syntax Tree

6章：ASTとDOM（ドム）

概要

Webページを JavaScript から操作する際の重要な概念である Document Object Model (DOM) について解説する。

HTML ソースを受け取ったブラウザが内部でどのような処理をするのか、抽象構文木、DOMツリー、CSSOMツリー、レンダリングツリーについて説明する。

(1) 構文解析と抽象構文木(AST)の生成

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Document Object Model を理解する</title>
    <style>
     body { font-size: 16px; }
     p { font-weight: bold; }
     span { color: red; }
     p span { display: none; }
     li { color: blue; }
    </style>
  </head>
  <body>
    <p>Hello, <span>World.</span> Students.</p>
    <ul>
      <li>Good</li>
    </ul>
  </body>
</html>
```

ここでは body 要素タグHTMLに注目しよう。

```
body
├ p
│  ├ TEXT
│  ├ span
│  │  └ TEXT
│  └ TEXT
└ ul
    └ li
      └ TEXT
```

このような木構造で HTML が書かれている。

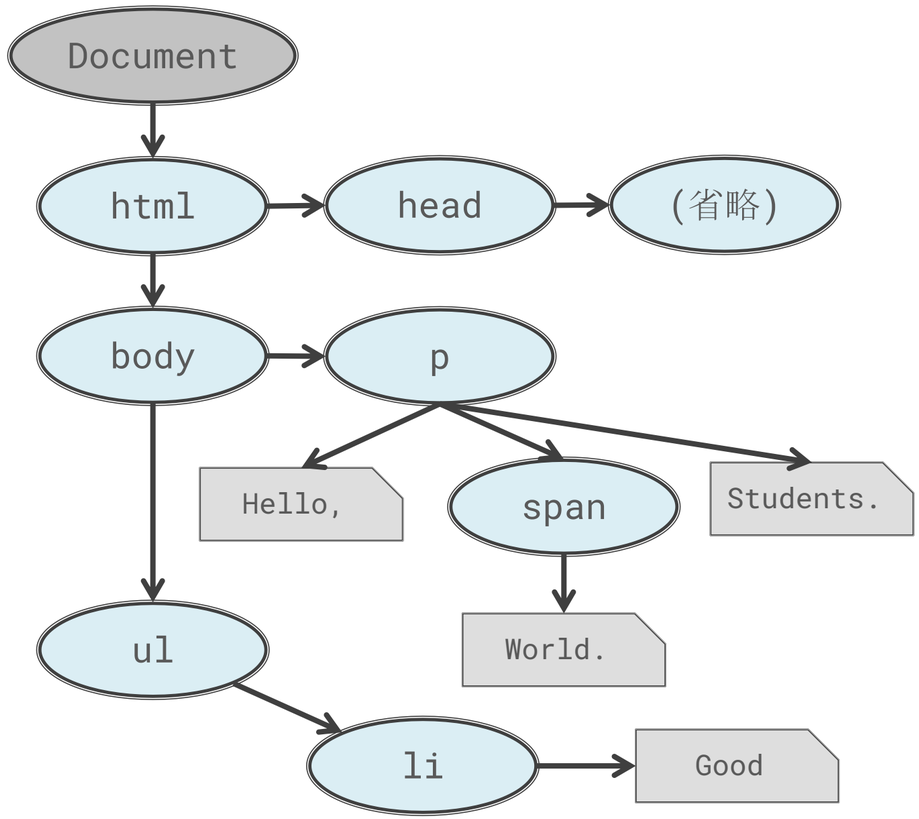
ブラウザは HTML ソースコードを受け取ると、その文字列をパース（構文解析）して、上記右側に示したような木構造（ここではbody以外を省略している）を内部的に生成する。このようなプロセスで生成した木構造のことを「抽象構文木」と呼ぶ。

HTML要素の構造に注目して生成した抽象構文木を Document Object Model (DOM) という。

(2) HTMLとDOMツリー

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Document Object Model を理解する</title>
    <style>
     body { font-size: 16px; }
     p { font-weight: bold; }
     span { color: red; }
     p span { display: none; }
     li { color: blue; }
    </style>
  </head>
  <body>
    <p>Hello, <span>World.</span> Students.</p>
    <ul>
      <li>Good</li>
    </ul>
  </body>
</html>
```

DOMツリー



ブラウザはHTMLを構文解析し、ドキュメント上のオブジェクトに関する抽象構文木を生成する。

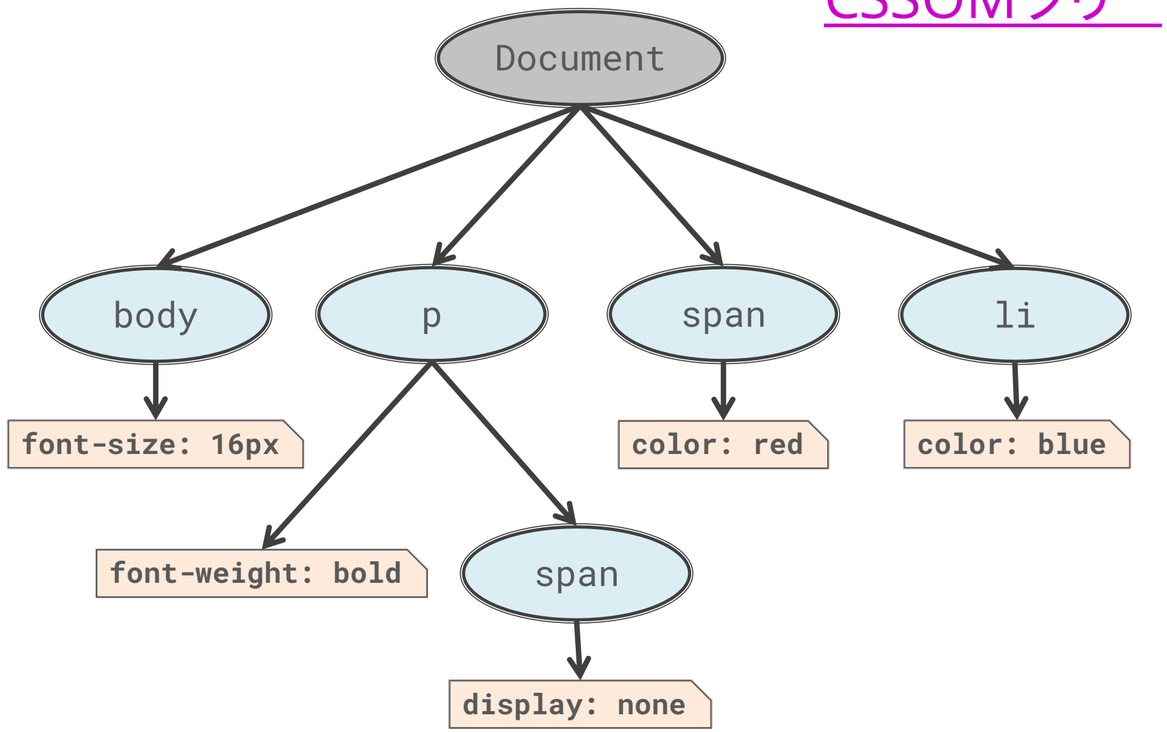
この抽象構文木をDOMツリーという。ツリーの要素は「ノード」と呼ばれる。

ノードにはDOMの頂点である Document Node、そして HTML を表現する HTMLElement Node と Text Node がある。

(3) CSSとCSSOMツリー

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Document Object Model を理解する</title>
    <style>
     body { font-size: 16px; }
     p { font-weight: bold; }
     span { color: red; }
     p span { display: none; }
     li { color: blue; }
    </style>
  </head>
  <body>
    <p>Hello, <span>World.</span> Students.</p>
    <ul>
      <li>Good</li>
    </ul>
  </body>
</html>
```

CSSOMツリー



ブラウザはスタイルシートを構文解析し、CSSに関する抽象構文木を生成する。

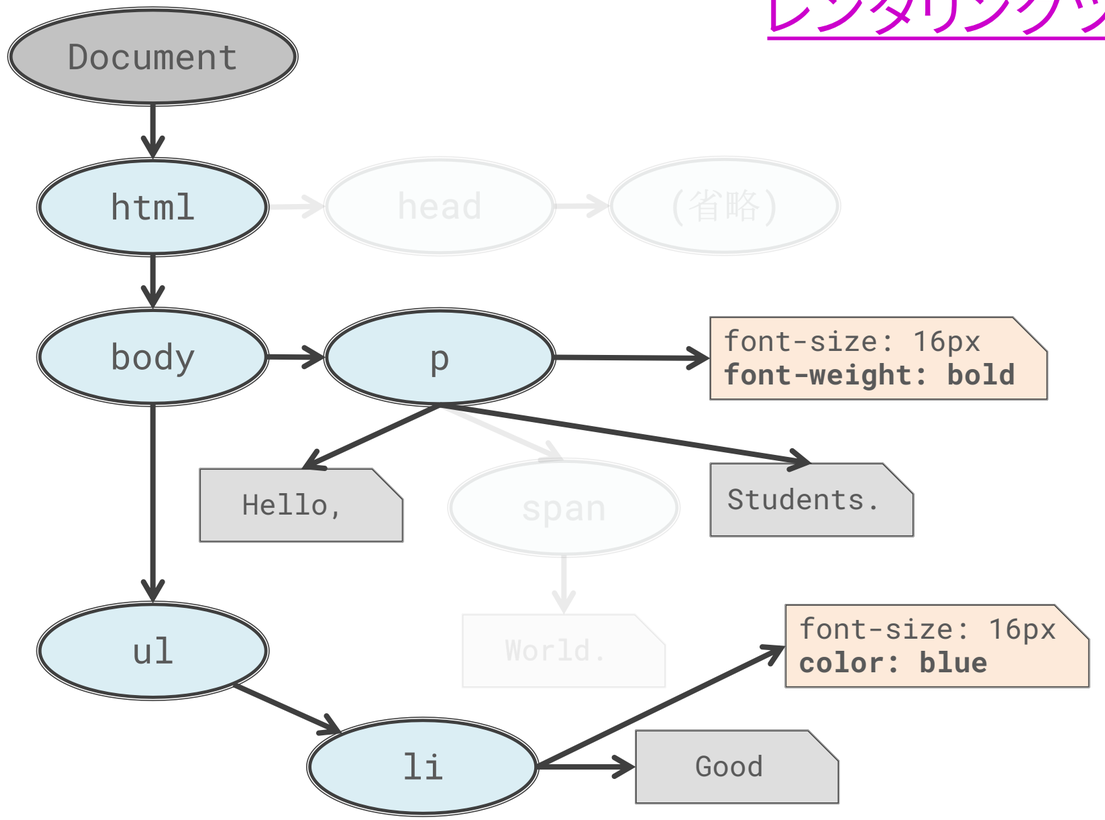
この抽象構文木をCSSOMツリーという。ツリーの要素は「ノード」と呼ばれる。

DOMツリーとは別にCSSOMツリーが作られているということを覚えておけば良い。

(4) HTML/CSSとレンダリングツリー

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Document Object Model を理解する</title>
    <style>
     body { font-size: 16px; }
     p { font-weight: bold; }
     span { color: red; }
     p span { display: none; }
     li { color: blue; }
    </style>
  </head>
  <body>
    <p>Hello, <span>World.</span> Students.</p>
    <ul>
      <li>Good</li>
    </ul>
  </body>
</html>
```

レンダリングツリー



ブラウザはDOMツリーとCSSOMツリーを生成すると、レンダリングツリーを構築する。CSSは継承されるプロパティもある。

レンダリングツリーにはDOMツリーのノードのうち、CSSOMツリーで描画が必要なノードのみが含まれる。

head要素や、p要素が先祖のspan要素は display: none によって描画されないためレンダリングツリーには含まれない。

(5) DOMを操作するということ

JavaScript ではDOMツリーのノードを取得することができる。

取得したノードに子ノードを追加したり、指定したノードを移動したり削除することができる。

HTMLにおける要素内容を JavaScript から動的に書き換えるという処理は、DOMツリーのノードを変更しているということになる。

JavaScript からノードに対するスタイル情報を書き換えることもできる。その場合はCSSOMツリーが更新され、レンダリング結果にも影響する。

DOMツリー、CSSOMツリー、レンダリングツリーのより詳しい説明は Google Web Fundamentals が詳しい。

https://developers.google.com/web/fundamentals/performance/critical-rendering-path/constructing-the-object-model?hl=ja

レンダリングツリーからブラウザに実際に描画する際のプロセスには、「ペイント」と「フロー」という概念がある。CSS を動的に変更するような処理を実装する場合、リペイントやリフローに関する知識がなければ処理コストの設計においてパフォーマンスを低下させてしまうかもしれない。

No Experience, No Knowledge

7章：JavaScript実践

概要

実際に JavaScript を動かしてみよう。

知識がないからプログラムが書けないのではない。

プログラムを書かないからプログラムが書けないのである。

知識は学びによって得られるが、本を読むだけが学びではない。
経験をすることは本を読むよりも大きな学びが得られるのだ。

(1) JavaScript を動かそう

1_index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>JavaScript演習 - 1</title>
    <script src="1_script.js"></script>
  </head>
  <body>
    <h1>1章：ハイパーテキストの歴史</h1>
    <ul class="key-persons">
      <li id="bush">ヴァネバー・ブッシュ</li>
      <li id="nelson">テッド・ネルソン</li>
      <li id="engelbart">ダグラス・エンゲルバート</li>
    </ul>
    <div class="description">
      <p>WWWの土台となるハイパーテキストの歴史。</p>
    </div>
  </body>
</html>
```

1_script.js

```
alert('JavaScriptを呼び出した！');
```

JavaScript を実行する例です。

alert メソッドはブラウザのグローバルメソッドとして定義されています。親オブジェクトは window です。

つまり、より安全に書くなら window.alert() と書けます。

JavaScript のグローバルスコープにコードを書いてしまうと、関数や変数を定義した際にグローバルの名前空間を汚染してしまいます。それを防ぐために無名関数でラップして即時実行させるパターンがあります（後述のIE6対応コードのスライドを参照）。即時関数とも呼ばれます。
JavaScript には Strict モードという概念があり、スコープの先頭で 'use strict'; と記述することでそれを有効にできますがここでは省略します。

(2) HTMLの内容をアラート表示してみよう

2_index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>JavaScript演習 - 2</title>
    <script src="2_script.js"></script>
  </head>
  <body>
    <h1>1章：ハイパーテキストの歴史</h1>
    <ul class="key-persons">
      <li id="bush">ヴァネバー・ブッシュ</li>
      <li id="nelson">テッド・ネルソン</li>
      <li id="engelbart">ダグラス・エンゲルバート</li>
    </ul>
    <div class="description">
      <p>WWWの土台となるハイパーテキストの歴史。</p>
    </div>
  </body>
</html>
```

2_script.js

```
alert('JavaScriptを呼び出した！');

const description_paragraph = document.querySelector('.description > p');
alert(description_paragraph.textContent);
```

description ブロックの p 要素のテキストを alert 表示する例です。document はグローバル変数です。window.document です。

しかしこのコードはうまく動きません。JavaScript 1行目の alert は動きますが、その次の alert を実行しようとして description_paragraph.textContent が参照できずにエラーになります。HTMLElement に対しては textContent を参照できます。

document.querySelector() を呼び出した時点では description の要素が期待通り取得できていないことが原因です。

このスクリプトが実行されるのは HTML の script 要素タグが解釈されるタイミングです。この時点では body 要素の DOM の構築が完了していません。この時点では、DOMツリーに HTML の 6 行目までの要素しか含まれていないのです。

(3) アラート表示を成功させるには

3_index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>JavaScript演習 - 3</title>
    <script src="3_script.js"></script>
  </head>
  <body>
    <h1>1章：ハイパーテキストの歴史</h1>
    <ul class="key-persons">
      <li id="bush">ヴァネバー・ブッシュ</li>
      <li id="nelson">テッド・ネルソン</li>
      <li id="engelbart">ダグラス・エンゲルバート</li>
    </ul>
    <div class="description">
      <p>WWWの土台となるハイパーテキストの歴史。</p>
    </div>
  </body>
</html>
```

3_script.js

```
alert('JavaScriptを呼び出した！');

document.addEventListener('DOMContentLoaded', function() {
  const description_paragraph = document.querySelector('.description > p');
  alert(description_paragraph.textContent);
});
```

DOMツリーの構築が完了してから JavaScript を実行すれば、先ほどの問題を回避できます。

DOM API では、それを実現するための仕組みが用意されています。「イベント」の仕組みにおける DOMContentLoaded イベントです。

このイベントが発火したタイミングで処理が実行されるようにイベント登録を行うことで、HTML の全要素がDOMツリーに含まれてから JavaScript を実行することができます。

似たイベントに load イベント（onload イベント）がありますが、load イベントはDOMツリーの構築が終わり、さらに外部リソースを全て読み込み終わった後に発火する点が異なります。画像の読み込みが完了した後で画像に関する処理を行いたい場合などは load イベントを使います。

(4) 機能実装してみよう – クリックしたら文章表示

4_index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>JavaScript演習 - 4</title>
    <style>.key-persons > li { cursor: pointer; }</style>
    <script src="4_script.js"></script>
  </head>
  <body>
    <h1>1章：ハイパーテキストの歴史</h1>
    <ul class="key-persons">
      <li id="bush">ヴァネバー・ブッシュ</li>
      <li id="nelson">テッド・ネルソン</li>
      <li id="engelbart">ダグラス・エンゲルバート</li>
    </ul>
    <div class="description">
      <p>WWWの土台となるハイパーテキストの歴史。人物名をクリックして詳細表示。</p>
    </div>
  </body>
</html>
```

4_script.js

ハイライトはこのコードの解説ポイント

```js
document.addEventListener('DOMContentLoaded', function() {
  // クリック対象の要素を取得する
  const bush = document.getElementById('bush');
  const nelson = document.getElementById('nelson');
  const engelbart = document.getElementById('engelbart');

  // 操作したい要素を取得しておき変数にキャッシュする
  const persons = document.querySelectorAll('.key-persons > li');
  const description_paragraph = document.querySelector('.description > p');

  // description の中のテキストを変更し、クリックされた要素をハイライトする関数
  function clickAction(text, clickedElem) {
    // 全ての人物のスタイルをリセット
    persons.forEach((elem) => elem.style.cssText = '');
    description_paragraph.textContent = text;
    clickedElem.style.cssText = 'background: yellow';
  }

  // クリック時のイベントを登録する（"this" を使えば bush を二度書かずに済む）
  bush.addEventListener('click', function() {
    clickAction('1945年にMemex構想を発表。"As We May Think" の著者。', bush);
  });
  nelson.addEventListener('click', function() {
    clickAction('1960年にザナドゥ計画を創始。ハイパーテキストの考案者。', nelson);
  });
  engelbart.addEventListener('click', function() {
    clickAction('1968年にマウスを発明し、革新的なハイパーテキストのデモを発表。', engelbart);
  });
});
```

クリックされる度に persons や description_paragraph を探すのは無駄なので、プログラム全体で一度だけ要素取得（DOM走査）を行うよう変数キャッシュを活用している。

ここまでの知識を活かした応用篇です。人物名をクリックしたら、その人物に関する説明文を表示する機能の実装です。

要素を取得するには document オブジェクトが持つ DOM API を用います。getElement(s)By... や querySelector(All) メソッドがあります。

HTML5 の仕様が公開された時代であれば割と自然に実装することができますが、第二次ブラウザ戦争時代は querySelector メソッドは存在していない状況でした。DOM API の実装が貧弱で、ブラウザの種類によっても呼び出すべきメソッドが異なっているという状況でこのような実装をすることはとても大変です。そこで登場した救世主が prototype.js や jQuery という DOM Manipulation Library でした。

(5) jQueryによる実装 – クリックしたら文章表示

5_index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>JavaScript演習 - 5</title>
    <style>.key-persons > li { cursor: pointer; }</style>
    <script src="jquery.js"></script>
    <script src="5_script.js"></script>
  </head>
  <body>
    <h1>1章：ハイパーテキストの歴史</h1>
    <ul class="key-persons">
      <li id="bush">ヴァネバー・ブッシュ</li>
      <li id="nelson">テッド・ネルソン</li>
      <li id="engelbart">ダグラス・エンゲルバート</li>
    </ul>
    <div class="description">
      <p>WWWの土台となるハイパーテキストの歴史。人物名をクリックして詳細表示。</p>
    </div>
  </body>
</html>
```

5_script.js

ハイライトはこのコードの解説ポイント

```
$(function() {
  // クリック対象の要素を取得する（jQueryオブジェクトの変数名に $ を付ける流派がある）
  const $bush = $('#bush');
  const $nelson = $('#nelson');
  const $engelbart = $('#engelbart');

  // 操作したい要素を取得しておき変数にキャッシュする
  const $persons = $('.key-persons > li');
  const $description_paragraph = $('.description > p:first');

  // description の中のテキストを変更し、クリックされた要素をハイライトする関数
  function clickAction(text, clickedElem) {
    // 全ての人物のスタイルをリセット
    $persons.removeAttr('style');
    $description_paragraph.text(text);
    $(clickedElem).css({'background': 'yellow'});
  }

  // クリック時のイベントを登録する
  $bush.on('click', function() {
    clickAction('1945年にMemex構想を発表。"As We May Think" の著者。', this);
  });
  $nelson.on('click', function() {
    clickAction('1960年にザナドゥ計画を創始。ハイパーテキストの考案者。', this);
  });
  $engelbart.on('click', function() {
    clickAction('1968年にマウスを発明し、革新的なハイパーテキストのデモを発表。', this);
  });
});
```

jQuery を使って書き直しました。

4_script.js のコードとほとんど変わらないことが分かるでしょうか。これが HTML5 時代に jQuery が不要と言われる理由です。ただ一方で jQuery を使う理由もあります。ネイティブの JavaScript（生JS、Pure JS、Vanilla JS とも。）よりも短く書きやすいメソッドが用意されています。また、jQuery ではマッチする要素がなく取得できなかった場合でも例外処理が不要な Null オブジェクトパターンで実装されています。

参考までに、次のページで jQuery が大流行した IE6〜IE8 全盛期の時代に jQuery を用いずに Vanilla JS で実装する例を紹介しましょう。

特別なフレーバーのないベーシックな味。バニラ。だから Vanilla JS と呼ばれる。水をDHMOと呼称するようなジョークの一種だった。http://vanilla-js.com/

(6) jQueryを使わずにIE6対応してみた

6_index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>JavaScript演習 - 6</title>
    <style>.key-persons > li { cursor: pointer; }</style>
  </head>
  <body>
    <h1>1章：ハイパーテキストの歴史</h1>
    <ul class="key-persons">
      <li id="bush">ヴァネバー・ブッシュ</li>
      <li id="nelson">テッド・ネルソン</li>
      <li id="engelbart">ダグラス・エンゲルバート</li>
    </ul>
    <div class="description">
      <p>WWWの土台となるハイパーテキストの歴史。人物名をクリックして詳細表示。</p>
    </div>
    <script src="6_script.js"></script>
  </body>
</html>
```

IE6 向けに DOMContentLoaded をフォールバックするのは面倒なので script 要素タグを body 要素の末尾に記述することで script 以前の要素のDOMツリーの構築を完了させることを利用しています。

JavaScript の処理では、IE6-8 向けのフォールバック関数を自作しています。今回の機能実装程度であれば右記のコード量で済みますが、もう少し実装量の多い機能の場合はフォールバックの実装が膨れ上がります。DOM API の再発明になりかねません。それをやったのが jQuery です。

コードの解説は次のページで行います。

6_script.js

```
(function() {
  // クリック対象の要素を取得する
  var bush = document.getElementById('bush');
  var nelson = document.getElementById('nelson');
  var engelbart = document.getElementById('engelbart');

  // 配列の forEach を自作する
  function forEach(array, callback) {
    for (var i = 0, len = array.length; i < len; ++i)
      if (callback.call(array[i], array[i], i) === false) break;
  }

  // class 属性値での要素の絞り込み関数を自作する（IEはバージョン9からサポート）
  function getElementsByClassName(argClsName, argElem) {
    if (argElem == null) argElem = document;
    var resultElems = [], allElems = argElem.getElementsByTagName('*');
    forEach(allElems, function(elem) {
      forEach(elem.className.split(/¥s+/), function(clsName) {
        if (clsName === argClsName) { resultElems.push(elem); return false; }
      });
    });
    return resultElems;
  }

  // 操作したい要素を取得しておき変数にキャッシュする
  var persons = getElementsByClassName('key-persons')[0].getElementsByTagName('li');
  var paragraph = getElementsByClassName('description')[0].getElementsByTagName('p')[0];

  // HTMLElement に text をセットする関数を自作する
  function setTextContent(elem, text) {
    elem.innerHTML = '';
    elem.appendChild(document.createTextNode(text));
  }

  // description の中のテキストを変更し、クリックされた要素をハイライトする関数
  function clickAction(text, clickedElem) {
    // 全ての人物のスタイルをリセット
    forEach(persons, function(elem) { elem.style.cssText = ''; });
    setTextContent(paragraph, text);
    clickedElem.style.cssText = 'background: yellow';
  }

  // イベントを登録するラッパー関数を自作する
  function addEvent(elem, type, listener, useCapture) {
    if (elem.addEventListener) {
      elem.addEventListener(type, listener, useCapture);
    } else {
      elem.attachEvent('on' + type, listener);
    }
  }

  // クリック時のイベントを登録する（"this" を使えば bush を二度書かずに済む）
  addEvent(bush, 'click', function() {
    clickAction('1945年にMemex構想を発表。"As We May Think" の著者。', bush);
  });
  addEvent(nelson, 'click', function() {
    clickAction('1960年にザナドゥ計画を創始。ハイパーテキストの考案者。', nelson);
  });
  addEvent(engelbart, 'click', function() {
    clickAction('1968年にマウスを発明し、革新的なハイパーテキストのデモを発表。', engelbart);
  });
}());
```

(6) IE6対応コードの解説

6_script.js

```
(function() {
  // クリック対象の要素を取得する
  var bush = document.getElementById('bush');
  var nelson = document.getElementById('nelson');
  var engelbart = document.getElementById('engelbart');

  // 配列の forEach を自作する
  function forEach(array, callback) {
    for (var i = 0, len = array.length; i < len; ++i)
      if (callback.call(array[i], array[i], i) === false) break;
  }

  // class 属性値での要素の絞り込み関数を自作する（IEはバージョン9からサポート）
  function getElementsByClassName(argClsName, argElem) {
    if (argElem == null) argElem = document;
    var resultElems = [], allElems = argElem.getElementsByTagName('*');
    forEach(allElems, function(elem) {
      forEach(elem.className.split(/¥s+/), function(clsName) {
        if (clsName === argClsName) { resultElems.push(elem); return false; }
      });
    });
    return resultElems;
  }

  // 操作したい要素を取得しておき変数にキャッシュする
  var persons = getElementsByClassName('key-persons')[0].getElementsByTagName('li');
  var paragraph = getElementsByClassName('description')[0].getElementsByTagName('p')[0];

  // HTMLElement に text をセットする関数を自作する
  function setTextContent(elem, text) {
    elem.innerHTML = '';
    elem.appendChild(document.createTextNode(text));
  }
```

6_script.js の続き

ハイライトはこのコードの解説ポイント

```
  // description の中のテキストを変更し、クリックされた要素をハイライトする関数
  function clickAction(text, clickedElem) {
    // 全ての人物のスタイルをリセット
    forEach(persons, function(elem) { elem.style.cssText = ''; });
    setTextContent(paragraph, text);
    clickedElem.style.cssText = 'background: yellow';
  }

  // イベントを登録するラッパー関数を自作する
  function addEvent(elem, type, listener, useCapture) {
    if (elem.addEventListener) {
      elem.addEventListener(type, listener, useCapture);
    } else {
      elem.attachEvent('on' + type, listener);
    }
  }

  // クリック時のイベントを登録する（"this" を使えば bush を二度書かずに済む）
  addEvent(bush, 'click', function() {
    clickAction('1945年にMemex構想を発表。"As We May Think" の著者。', bush);
  });
  addEvent(nelson, 'click', function() {
    clickAction('1960年にザナドゥ計画を創始。ハイパーテキストの考案者。', nelson);
  });
  addEvent(engelbart, 'click', function() {
    clickAction('1968年にマウスを発明し、革新的なハイパーテキストのデモを発表。', engelbart);
  });
}());
```

IE6 対応をしているにも関わらず、なんとなく雰囲気はもとのコードに似ていることが分かるでしょうか。

IE6 などのレガシーブラウザ対応では、モダンブラウザでの実装に近いコードを書いておき、そのままでは動かない部分をラッパー関数でフォールバックすることで見通しの良いコードになります。ラッパー関数の実装を隠してしまえば、そのコードはモダンブラウザ向けに書いたものとほとんど変わりません。

それを実現しているのが jQuery なのです。フォールバックだけでなく便利なユーティリティも提供しています。jQuery 使いたくなりましたか？

(6) SPAの登場

先ほどの JavaScript の例は、第一次ブラウザ戦争時代から可能だった JavaScript の使用例です。

2005年の Google Maps の登場、そして2014年の HTML5 の登場で JavaScript の可能性は大きく進展しました。

第三次ブラウザ戦争時代、ページ遷移なしで Google Maps のようにシームレスかつインタラクティブにWebコンテンツを利用する手法が模索されました。

そして登場した jQuery とは別の新たなフレームワークが、AngularJS, React, Vue.js そして生まれ変わった Angular です。

「あなたが作りたいのは
Webサイトですか？ それとも
JavaScriptアプリケーションですか？」

(7) SPAによって実現されること

SPAは「銀の弾丸」ではない

サーバサイドでHTMLを返す
従来の設計が適切なケースは多い。

SPA の用法・用量は適切に。

Single Page Application は何が便利なのでしょうか。SPA は本当に必要なのでしょうか。

その答えの一つは Google Maps にありそうです。

地図情報を提供するWebサービスが JavaScript を活用しなかったら表示領域を変更するごとに明示的なHTTPリクエストページが発生し、ブラウザが一瞬真っ白になりページが再表示されるという挙動になります。（Applet や Flash などを使わない限り）

これは地図を閲覧するユーザからしたら煩わしい動作です。では Canvas を活用すればよいのか？

このメリットをHTML形式のままWebサイト全体に適用したのが SPA というアプローチです。

Do you have any questions?

8章：まとめ

概要

このスライドの内容を振り返っておきます。

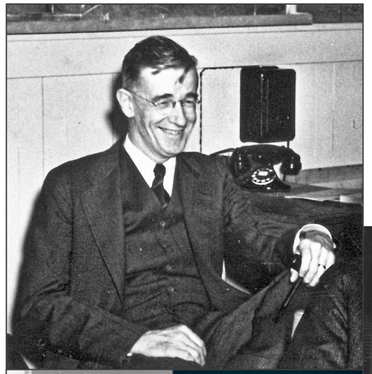
この講義を受けて興味や関心の対象が増えましたか？

疑問に思うことはありましたか？

このスライドでは深く触れられなかったトピックもあります。
興味を持ったことは意欲的に調べてみてください。

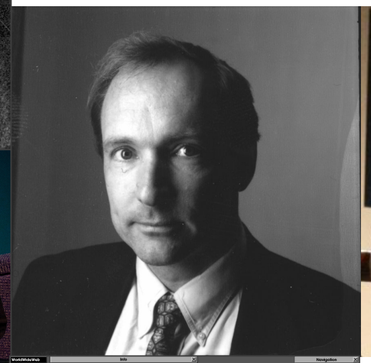
まとめ：フロントエンド界隈概論

DNSとHTTPリクエストとHTTPレスポンス。
DOMツリー、CSSOMツリー、レンダリングツリー。
DOM API とイベント、非同期通信 Ajax、Google Maps の登場。



1945年、ヴァネバー・ブッシュによる Memex 構想の公開。

ハイパーテキストの原点
"As We May Think"



1960年、テッド・ネルソンによる Project Xanadu.

ハイパーテキスト・ハイパーメディアの考案。

1968年、ダグラス・エンゲルバートによるマウスの発明、革新的なハイパーテキストシステムのデモンストレーション。"全てのデモの母"

1989年、ティム・バーナーズ＝リーによる World Wide Web の発明。

Global HyperText Project, W3Cの創設。

1994年、NCSA が Spyglass へライセンス付与。

1995年、Internet Explorer の誕生。

1993年、NCSA在籍中のマーク・アンドリーセンによる NCSA Mosaic の開発。

インライン画像のサポート。
WWW の普及に貢献。

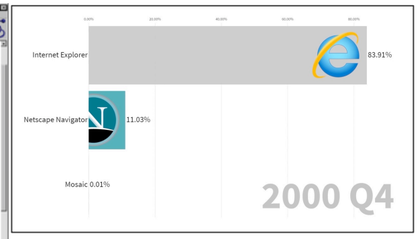
1994年、マーク・アンドリーセンが Mosaic Communication を創設、Mozilla Group を結成。

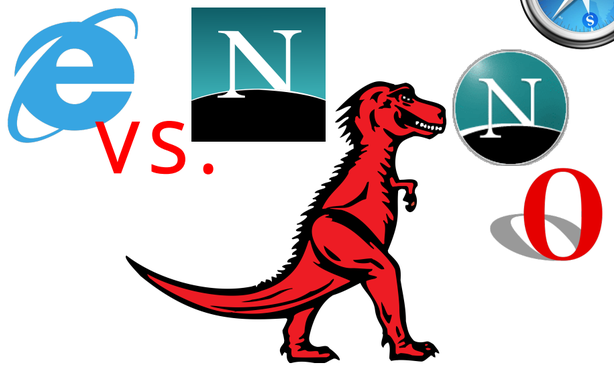
Netscape Communication への改名と Netscape Navigator の開発。

1995年～2000年。

第一次ブラウザ戦争。

Windows 98 と IE4 の圧倒的シェア。





Mozilla.org の創設。

Mozilla, Apple, Opera による WHATWG の結成。

第二次ブラウザ戦争。

2000年～2008年。

2008年～2014年。

「9年前の腐った牛乳」と jQuery の時代。

2014年、HTML5 の勧告。

第三次ブラウザ戦争。

Google Chrome 旋風。

Single Page Application の流行。
Vue.js / React / Angular の登場。

注：画像のライセンスは別スライドに記載

Thank you

部分的にでも興味を持った内容があれば自分で調べてみてください。

あなたが受け取った情報は、全てが正しいとは限りません。
内容を理解するには、自分の言葉で説明することが必要です。

このスライドのデザインテンプレートについて

Sirius PowerPoint Template v1.0

- このスライドは thepopp.com で提供されているデザインテンプレートを用いています。
  - https://thepopp.com/templates/sirius/
  - Icon generated by flaticon.com under CC BY. The authors are: Stephen Hutchings.
- このデザインテンプレートは作者である秋咲准氏が著作権を保有しています。
  - 再配布したり販売したりすることはできません。
  - 利用については、商用・非商用にかかわらず誰でも自由にダウンロードして使うことができます。
- このスライドでは Spica Neue Font Family および Roboto Mono のフォントを利用しています。
  - https://thepopp.com/fonts/
  - https://fonts.google.com/specimen/Roboto+Mono