第２回コロキューム

Web の進化で考える
IT の今までとこれから

Topic

第 1 部　Web の歴史

- 1969 年　インターネット誕生
- 1982 年頃　OSI・TCP/IP の登場
- 1992 年　IPv4 の枯渇
- 1993 年　World Wide Web の誕生
- 1995 年以降

第 2 部　Web のこれからの進化

- 2014 年を振り返る
- 技術的側面から考える Web
- Web の違う進化
- Extensible Web

第 3 部　質疑応答

講演者：及川卓也

グーグル株式会社

講演日 2015/4/22

招聘者　金井太郎　白井裕子

アーカイブ担当　乙黒雄斗　倉石　孝　白井裕子

早稲田大学実体情報学博士プログラムのコロキューム第 2 回は、グーグルのシニア エンジニアリング マネージャとして、活躍されている及川卓也さんをお招きし、「Web の進化で考える IT の今までとこれから」と題してお話を伺った。講演後の質疑応答では、学生から多数の質問が寄せられた。なお及川氏のリクエストにより学食で昼食会を開催し、L2 生と懇談された。

講演者紹介

及川卓也 （おいかわ たくや）

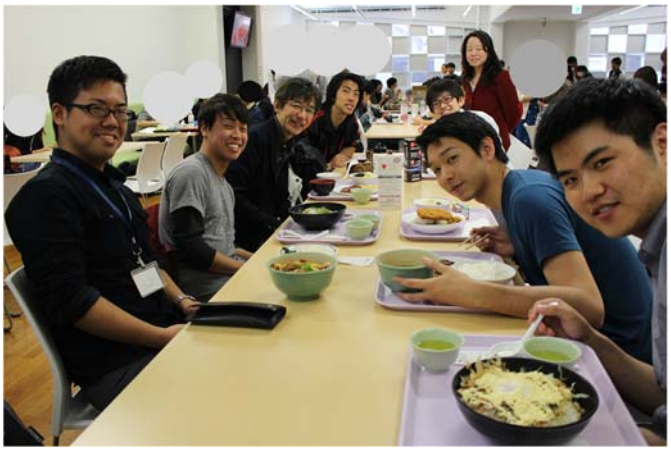
グーグル株式会社 シニア エンジニアリング マネージャ

早稲田大学理工学部卒業。卒業後は日本ディジタルイクイップメント株式会社(日本 DEC)に勤務。営業サポートやソフトウェア開発を担当し、1997 年よりマイクロソフト勤務。2006 年から現在のグーグル株式会社へ。

現在は Web ブラウザの Google Chrome 開発において Web 開発者のための API を提供している Blink を担当。またコミュニティ活動にも従事しており、IT で震災復興・防災を支援するコミュニティや HTML5 を推進するためのコミュニティを立ち上げている。

第1部 Web の歴史

Web の普及は、インターネットの一般への普及と重なるところが多い。2012 年にグーグルがインターネットの進化をインタラクティブに示した Web アプリケーション「Web の進化（Evolution of the Web（http://www.evolutionoftheweb.com/）」を公開したが、それを見ると、Web とインターネットの進化の過程が良く分かる。



学食での昼食会

■1969 年 インターネット誕生

1969 年、インターネットが誕生する。このインターネットは、正確に言うと、”the Internet”である。”Internet”単体では Internetworking、すなわちネットワークを相互接続することを意味する。この相互接続こそが当時も今もインターネットのコアな部分である。”the”が付加されることで、インターネットは、Network of Networks つまりネットワークをつなげるネットワークを意味する言葉となる。

インターネットの最初は ARPANET(Advanced Research Projects Agency Network)と呼ばれるネットワークシステムであった。この ARPANET は米国の先進的リサーチプロジェクトであり、最初のリンクは、カルフォルニア大学とスタンフォード研究所(SRI)の間で行われた。それぞれが持っているノード、すなわちネットワーク網における「節」の部分をつなげることがインターネットの起源である。余談になるが、この ARPANET のプロジェクトで、ユタ大学が使用したコンピュータは、私が勤務していた DEC の PDP10 である。

■1982 年頃 OSI・TCP/IP の登場

1982 年頃、OSI(Open Systems Interconnection)と呼ばれる規格が登場する。OSI はオープンな形で、ネットワーク間をつなぐコンピュータネットワーク標準である。当時、いろいろなベンダーが自社規格のネットワークを構築していたが、ネットワークは各社で閉じたものであった。このため、異なるベンダーのネットワーク同士をつなぐことができない状態であった。国際標準化機構(ISO)がこの現状を変えるために動き出し、国や企業の代表が規格をまとめた。最終的には、電話帳のような分厚い仕様書がたくさん並ぶまでになった。しかし、その複雑な仕様と標準化プロセスへの批判などもあり、できあがった規格である OSI はほとんど使われていない。この時、OSI と比較されていたのが TCP/IP である。ご存じの通り、TCP/IP は、現在のインターネットを支えるネットワークプロトコルだ。

■1992 年　IPv4 の枯渇

1992 年、インターネットの発展を考えるインターネット委員会、ISOC(Internet Society)の最初の国際会議となった INET'92 Kobe で IP アドレスの枯渇問題に対応するため IPv4(Internet Protocol Version 4)の後継プロトコルについて議論がされた。アドレス枯渇に備えるには、アドレス拡張が必要になる。マイクロソフトのクリスチャン・ウイテマは、IPv6 仕様を考えた一人であるが、彼は自著の中で、IP を諦め、下位プロトコルとして OSI のネットワーク層のプロトコルである CLNP(Connection Less Network Protocol)を使おうと、一度判断しかけたと書いている。しかし、その後、結論は覆され、今の IPv6 の原型が決まった。

先程、OSI を電話帳という言い方をした。印刷すると、何冊もの電話帳になるほどの複雑な規格を作り、それに基づき実装し、相互運用実験をする。プラグフェスタと言って、ベンダーが集まって実際に接続してみる場を用意することもあった。そこで繋がらなかったなら、実装を見直し、必要ならば仕様も再検討する。しかし OSI はこのやり方ではうまくいかなかった。技術者が見ると OSI のプロトコルの方が TCP/IP に比べて、明らかに優れた所もたくさんあった。TCP/IP は単純すぎるほど単純で、しかも冗長なところもあり、必ずしも効率が良いわけではなかった。しかし TCP/IP の単純さが決め手となった。

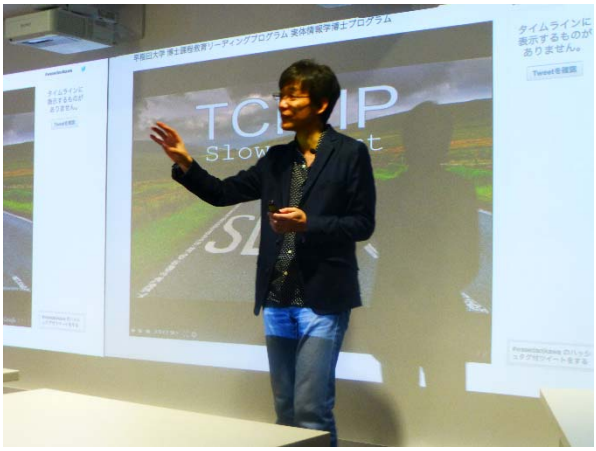
単純な方法から始め、必要に応じて機能を追加していく。実際の使用状況に応じて修正を加えていく。これこそが、本当に使われるものになっていくためのアプローチとして、インターネットが採用したやり方であり、TCP/IP が支持された理由である。

TCP/IP などのインターネットのコアプロトコルの標準化団体が IETF(The Internet Engineering Task Force)である。すでに知られているように、標準化には、デファクトとデジュールという二つの手法がある。IETF はデファクト的なやり方で、非常にカジュアルな仕組みを用いる。IETF は標準化プロセスの基本原則を"Rough Consensus and Running Code"という言い方をしているが、つまり「大まかな合意と動くプログラムがあれば良い」としている。これが IETF におけるコンセンサスビルディングの原則である。仕様を議論する際に、実装があるのかが常に問われる。

最低限の機能が動作確認できる Proof of Concept がなければならない。複数の動作する実装で相互運用性が証明されて、はじめてそれが標準化に足るものであると認められていく。「あなたの論はそれだけではダメです、実際に動くものを持ってきて下さい」この考え方がインターネットを支えている。インターネットの単純さこそが、その後の拡張性を持つことにつながり、今に繋がっている。

■1993 年　World Wide Web の誕生

Web は、1993 年に CERN(欧州原子核研究機構)に勤務していたティム・バーナーズ・リーが、所内で構造化された文章を共有したことから始まった。当時、私は米国ワシントン州レドモンドのマイクロソフト社に勤務していた。当時からマイクロソフト社内ではインターネットを使うことはできたが、まだ Web は無く、メールが中心であった。ある日、同僚から Web が出てきたことを教えてもらい、モザイクというブラウザで画像を見た。この時、Web の可能性を確信した。



■1995 年以降

1995 年 Windows 95 が発売される。これがおそらく、インターネットおよび Web が一般コンシューマに使われるようになった最初だろう。それに前後する形で商用のプロバイダもでてきた。

その後、Web は爆発的な普及を遂げていくことになるが、2005 年を境にさらにその利用方法が大きく変わっていく。一般コンシューマが使用するアプリケーションは、2005 年より前は、マイクロソフト オフィスのようなパソコンで動く、スタンドアローンのアプリケーションが、ほとんどであった。それが 2005 年以降は、Twitter や YouTube、Facebook のような Web アプリケーションになってきた。

この Web アプリケーションが普及する際も、先ほどの OSI と TCP/IP と同じような論争があった。それは XML(Extensible Markup Language)と HTML の間においてである。XML は、構造化された、セマンティックをもった形で情報を記述できるものだ。

例えば、いま Web アプリケーション開発者が、Twitter と連携したり、Google マップを組み込むような Web アプリケーションを書きたいと思った時には、REST (Representational State Transfer)を利用することが多い。REST で URI (”http://xxxx/yyyy/zzzz”の形式で表現できるインターネット上で一意の ID)に、パラメータを含めてリクエストを出すと、JSON(JavaScript Object Notation)と言われるフォーマットで

結果が返ってくる。これをブラウザ側で処理するというのが 1 つの典型的な Web アプリケーションの動作である。
しかし 2000 年前半は、Web アプリケーションといえば、XML をベースとした Web サービスが主流であった。これはマイクロソフトや IBM などが提唱した、いくつかの標準により構成される技術基盤であった。現在ではホーム AV 機器を接続するために使用されている DLNA (Digital Living Network Alliance)以外では、一般には普及しているとは言いがたい。誰もが自由に利用できるオープンな進化を遂げる技術としては、最初から仕様が複雑過ぎた。REST は、例えばブラウザのアドレスバーに、パラメータを含めた REST の URI を全部打ち込めば、結果が返ってくるという単純なもので、デバックもしやすい。本当にこれでスケールするのか、という話もあったが、やはりこの単純性に軍配が上がった。
その HTML も途中 XML をベースにした XHTML と呼ばれるものに、標準化団体やベンダーが舵を切った時があった。しかし人々はそこまで厳密なセマンティックを求めておらず、それより YouTube や地図アプリケーションのような動的な、もしくは対話的なアプリケーション開発の基盤となる技術を求めていた。このような背景から、すべてが HTML を主体とするものになっていく。
この間の標準化団体の動きとしては、Web の標準化団体である W3C(World Wide Web Consortium)が Web における文章の記述や情報を構造化する方に軸足を移していたため XHTML を支持していた。しかし Web アプリケーションを推進する Opera や Apple が、その動きには賛同できず、別団体 WHATWG(Web Hypertext Application Technology Working Group)を立ち上げる。これが HTML5 の最初となった。この WHATWG では、ブラウザのオフライン機能や Worker と呼ばれる JavaScript コードをバックグランド処理させる機能などを追加していき、多くの支持を集めた。その後、W3C も方針を変更し、HTML5 を推進し、XHTML は凍結される流れとなった。

■まとめ —Web とは何か？—

IETF と同じように W3C にも標準化の議論における基本原則がある。そこには「標準化における、いくつかのステージを上がり、標準勧告という最終ステージに行くまでに、相互運用可能な実装が 2 つ以上ないと標準にはならない」と書かれている。IETF が”Rough Consensus and Running Code”と言っているのと同じように、ここも、スペックだけではダメです、ちゃんとコードがあって、それは一つの団体からではなく、複数の人が作り、相互運用が可能であることを証明して下さいと、謳われている。今までの標準化は、標準化が完了したから実装しましょうという流れが中心であった。しかし、インターネットのコアの技術においては、標準化実装は一方向ではなくて、何回もイテレーション（反復）を繰り返していくことで、本当に使えるものにしていく。
「走りながら考える」というようにわかりやすく言い換えても良い。従来の開発では、仕様を考える人、実装する人、テストする人が別であり、それぞれの段階も分けられていることが多いが、全部を同時に走らせていくプロセスこそが、インターネットのような動きの速い技術においては必要となる。
同じようなコンセプト的として、TCP/IP の基本フィロソフィーの一つである Slow Start という考え方がある。IP や TCP、UDP などの通信プロトコルで一番あってはならないことは「繋がらない」ということである。今のインターネットには、言い方は悪いが、どこの馬の骨とも分からない中継器が沢山ある。ファイヤーウォールが入っているかもしれないし、プロキシサーバがあるかもしれない。途中の WiFi ルーターもどこのものかも分からない。何が起こるか分からない。そのため、いきなり高スループットで繋ごうとするのではなく、ちょろちょろつなぐのを試していく。「これで大丈夫」となった時から、通信時の送信単位であるウィンドウサイズを広げるなどしてできるだけスケールさせていき、スループットを速いものに持って行く。
繋がらないよりは、遅くても繋がった方が良い。この考えを一般に適用されると ”Think Big, Start Small, Scale Fast” でっかいことをやる時でも、まずは小さく始めて、その後、うまくいくと分かったら、素早くスケールさせていく、という考え方にも繋がる。これがまさに TCP/IP やインターネットの考え方そのものである。インターネットや Web を支える哲学としての、単純性をベースにした「繋がる」と言う考え方と、繋がったならば、それをいかにスケールさせるか、というこの 2 つの考えを、デザインとかアーキテクチャの部分から考えていくことが大事である。
さらに、もう一つ大事なのが、イノベーションのためのオープンなプロセスだ。IETF で作られている TCP/IP の標準は、RFC(Request for

Comments)という名前で、一つ一つの標準にすべて RFC 番号がついている。RFC のプロセスをスタートされたのがスティーブ・クロッカーという人物である。数年前に、このスティーブ・クロッカーがインターネットの殿堂入りした時にしたスピーチの中で RFC や IETF の議論は持続可能なオープンでソーシャルプロセスが大事である、と語っている。どこの組織でも起きえてしまうが、ともすると議論がどうしてもクローズになってしまい、いつも会話をしている人とだけで物事が決まっていくようなことになりかねない。そうではなく、次の世代、さらにその次の世代の人達が、プロセスに入ってもらうような、そういったオープンな仕組みを考えていくことが、イノベーションのためには必要である。

第 2 部　Web のこれからの進化

■2014 年を振り返る

2014 年は IoT(Internet of Things)が話題となり、ウェラブルコンピュータやドローン（マルチコプター）、シングルボードコンピュータと言った新しいデバイスも話題になった。また一方で、認証や同期、またはストレージのような従来 Web アプリケーションでは実現できなかった機能も比較的簡単に作れるようになってきている。BaaS(Backend as a Service)というものも出てきて、今はノンプログラミングで簡単にアプリケーションを開発できるようにもなってきた。長いこと標準化作業中だった HTML5 もようやく正式勧告が出た。

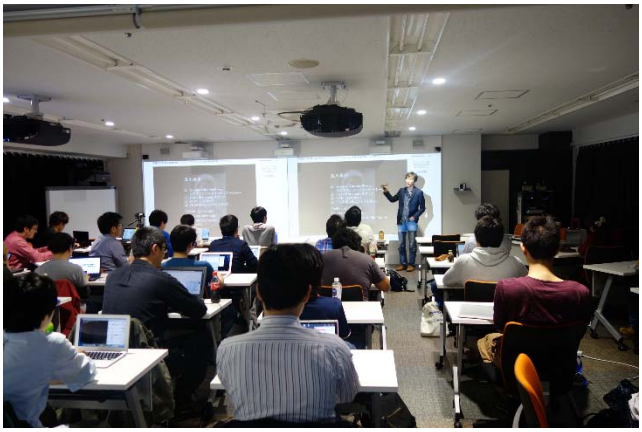
■技術的側面から考える Web

今一度、技術的側面から Web とは何かを考えてみよう。

技術的側面から見ると、Web は非常にシンプルで、関与するのは HTTP(Hypertext Transfer Protocol)クライアントと HTTP サーバの 2 つだけである。流れているプロトコルは HTTP で、URI と呼ばれるアドレスにリクエストを送り、そこに書かれているリソースがレスポンスとしてダウンロードされる。そして、ダウンロードされたリソースをクライアントが判断して処理する、これが Web である。

Web は、その歴史的な経緯でクライアント側に横長のディスプレイとマウス、キーボードがあることが前提となっていた。しかし、ウェアラブルデバイスやロボット、シングルボードコンピュータなども TCP/IP や HTTP をしゃべり、直接 Web にアクセスできるようになってきている。ここには横長ディスプレイもマウスもキーボードもない。また HTTP は基本的に 1 方向の通信であったが、WebSocket と呼ばれる双方向の通信が可能なプロトコルが登場し、その制御も JavaScript からも操作できるようになった。単純な Web の時代から、HTTP だけでなく、WebSocket などのほかのプロトコルも流れるような時代に移り変わってきている。WebSocket はバイナリーでやりとりができる。そうすると、クライアント側とサーバ側の両方が分かればいい、なんらからの独自のフォーマットを定義し、中はバイナリーで転送しあう。このような世界がすでに実現されている。WebSocket でやりとりされ、中身は HTML でもない、バイナリーデータが流れているだけのものを果たして Web と言えるのだろうか。このようなことを Web を進化させるにあたって、Web の標準化を進めている人々や Web 技術を実装している人々は考えている。

さらに、従来、マッシュアップとも言われる Web のアプリケーションの連携は、クラウド側の横連携が多かった。すなわち、自社サイトの中に他社が公開している API を利用して機能を組み込むなどの例だ。これも現在標準化と実装が進んでいる Service Worker と呼ばれる、特定の Web ページに張り付くのではなく、ブラウザ内で Web ページとは独立に動作する Worker により、クライアント側とサーバ側を連携させたハイブリッドなマッシュアップも可能となる。さらには、プッシュ通知やバックグラウンド同期という今までは Web アプリケーションでは不可能だったことも、この Service Worker を用いて実現できる。



このように、Web の進化は従来のシンプルさをベースに、確実に次のステージに移りつつある。

■Web とは、また違う技術の進化

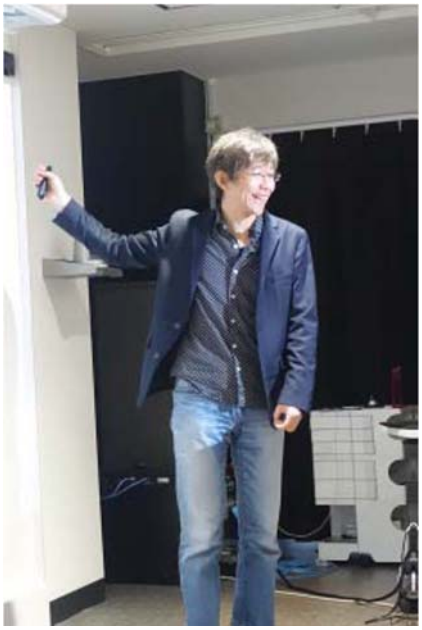
現在の Web には、従来の Web とは違う形の技術の進化として、利用シーンの広がりというのもある。HTML、CSS、JavaScript と言った Web 技術を、従来の Web とは直接関係ない、非常に高性能な汎用ソフトウェアの開発技術として使おうという流れだ。昔から JavaScript

を用いた汎用ソフトウェアの開発も行われていたが、高度なことをしようとすると、実際には多くの制約があった。HTML5 を始めとする技術の実用化が進み、様々なツールやフレームワークが登場した結果、通常のソフトウェア開発においても Web 技術を使うことが一般的になってきた。

組み込み系の製品を Web 技術で作るというようなことは、一昔前には考えられなかった。しかも、ここで言う組み込み製品は Web につなぐことなどまったく想定されていないこともある。そうなると、従来の Web の進化とは別の進化が必要となる。たとえば、デスクトップで使う普通のブラウザでならば、JavaScript のサイズが多少肥大化しても構わないが、組み込み製品で使うとなるとフットプリントを抑えなければならない。相矛盾するような要求が本来の目的とは違うところから出てきて、それにどう対峙していくかが新たな課題として出てきている。しかし、このように技術が様々な分野で使われるようになると、副次的な効果も生む。Web 技術を使える開発者人口が増加する。また汎用ソフトウェアとして追加された技術が、翻ってコアな Web 技術にも使われるようなことも起きうる。

また Web のプログラミングは DOM(Document Object Model)と言われるプログラミングモデルを基礎とする。HTML で記述された文章に効果を与えることから発展している。

Web は歴史的に横長のデスクトップのディスプレイを想定しているが、例えば、時計につけるとなったら、円形のほうが好ましい。これに対して、今の DOM のプログラミングパラダイムでいいのか、今の流れの延長でいいのだろうかという問題提起がある。Web が従来の Web から目的から離れて行った時に、多くの考えなければならないことに直面しているのが今日の Web 技術の状況である。



■Extensible Web

Web も登場から昨年で 25 周年を迎えた。そうなると、すでに使われなくなった機能や使えない機能がいくつか出てくる。例えば、Web 上での編集には多くの課題がある。Web 標準の課題としては、ブラウザ間で完全に互換性のある API が、ないことがあげられる。W3C の中でも編集、すなわち Editing に関してはドラフトの仕様はあるが、コンセンサスのとれた標準はない。つまり、中途半端な状態の実装が各ブラウザ内にあるが、満足して使える状況にはなっていない。ブラウザに期待できないので、Editing 機能を必要とする Web 開発者は、同じようなものを一から作らないといけないのだ。先ほど話したように、標準化と実装を両輪で回して行くのが Web の標準化のベストプラクティスであるのだが、それがうまくいっていない所が、この Editing を始めとしていくつもある。

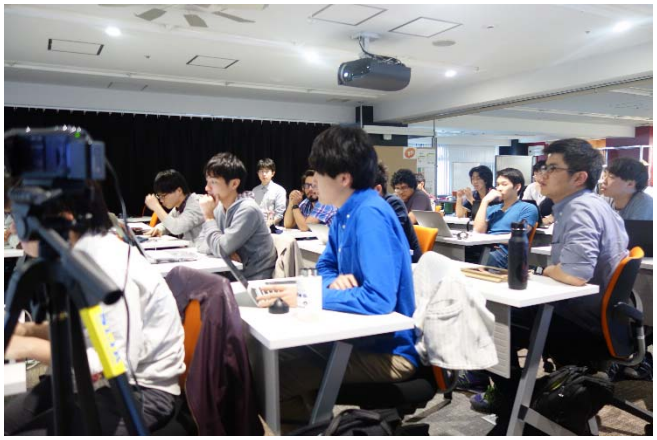
そこで、考えられているのが Extensible Web という考え方である。Web をもっと Extensible にしようという考えだ。例にあげた Editing の API は、ブラウザ内部のコードとして実装された API がどれもあまりうまく動いていないことが問題である。ある文字を Bold にすることさえ、ブラウザ間で完全に互換が取れないことがある。どういうことかと言うと、例えば Chrome 上で文字を Bold にして、次に同じ文章を Firefox を開いて Unbold、つまり Bold を取るというような単純なことさえできないことがある。複数の異なるブラウザ間で高度な Editing が実現出来ているのは、Web アプリケーションが自分で頑張っているためである。もし、もう一度 Web における Editing を見直すことができるならば、ある要素の 1 文字を選択できるだとかの低レベルの基本的な処理を規定するに留めて、それ以上は普通の DOM の API を利用することによって、より高度な機能が付加できるようにするのが良いだろう。必要なのは低レベル API であり、その上に必要なものは Web 開発者にどんどん自由に作ってもらう、そういう仕組みが Extensible Web である。

このような Extensible Web を実現することで、本当に必要な機能だけが組み込まれたミニマムなブラウザが完成する。ミニマムなブラウザに追加する形で、Web 開発者が作成した JavaScript ベースのコードが実際に使われ、そのコードは JavaScript でなく、Native に持った方が良いという議論になったに、標準化の土俵に乗せる。その結果として、ブラウザの機能に入れる形を取ったら良いのではないかという考え方が Extensible Web であり、これが Web で起きていることである。

実は、同じことは他の基本プラットフォームで起きたことでもある。例えば、私は Windows を開発していたが、Windows は奇跡的とも言えるくらいに信頼性が高い OS である。あれだけ世界中で、訳の分からないハードウェアが組み合わさり、様々な周辺機器が接続されている状況であっても、ほとんど落ちないで動いている OS など他にない。OS 自体の信頼性をあげる工夫は行われているが、Extensible Web と同じよ

うな拡張性の部分にも工夫が施されている。

例えば、周辺機器をサポートするためのデバイスドライバを開発するとき、開発者はーからコードを書く必要はない。キーボードやマウスは、昔ならすべて USB に繋がっていた。そうすると USB のドライバは開発者に書いてもらう必要はない。このようなキーボードやマウスデバイスは HID (Human Interface Device) と呼ばれるものだが、それらは挙動が類似している部分があるので、そのような部分もすべての開発者に書いてもらう必要はない。そうなると、一般的なキーボードやマウスに関しては、開発者が書く必要がある部分は極めて少なくなり、定義だけ書けば動くようになっている。こういった階層化と言われているものは、いろいろな所で行われている。クラスドライバやポートドライバと呼ばれるが、このように階層化されたドライバの最上位に本当にそのデバイスに特化した機能を実装するドライバがある。階層化をとる事により、どこまで OS 標準で持てば良いかも分かる。



Windows の一番最初も、最も下位のレベルしかなかった。その後、いろいろなデバイスドライバが出てきたときに、これは特徴が同じである、共通化できるところがある、それを OS 側で持つことが望まれている、というようなことが分かってきて、それらを上のクラスのドライバとして持って行く。こうすることで、信頼性や安定性を持たせた上で、同時に本当に必要な機能だけを開発者が書けば良くなる世界が実現された。これは Extensible Web でも同じである。標準と実装を並行して進めていく形を取る Web でも、Editing を始め、必ずしもうまくいかなかった所がある。もう一度、必要十分な機能を再定義して、本当に使われるコードを、ブラウザの中に取り込んでいこうとしている。これが Extensible Web という考え方である。

今日では、モバイルデバイスが非常によく使われている。スマートフォンが Web をなくすのではないかという記事があった。確かに、スマートフォンを使う人は、Web を、デスクトップほど使わなくなっている。代わりに何を使うか。モバイル Native アプリケーションだ。では、今開発者として Web アプリケーションではなく、モバイルアプリを書けば良いのだろうか。

ここに面白い調査結果がある。北米における調査で、1 ユーザがスマートフォンにインストールしているアプリは 33 個だが、そのうち実際に使われているのは 12 個と報告されている。端的に言うと、山ほどアプリがあっても、ほとんどものは使われていないのだ。ここで重要なのは、モバイルのアプリケーションの Web との連携だ。別にアプリの機能として Web につなげという訳ではなくて、ユーザのエンゲージメント、すなわちユーザの利用率を高めるところに Web との連携が必要になってくる。マニフェストファイルと言われる、簡単なマイクロデータを用意し、ユニークな URL に相当する所を埋め込むことによって、例えば、ある検索をした時に、インストールを促すような、そういった検索結果を実現できる。Web とモバイルアプリケーションとの連携を取ることで、ユーザのエンゲージメントを高めることがされている。

昨今、IoT が沢山でてきている、IoT が普及した近未来的な社会はどのようなものだろうか。ウェアラブルを持って町に出かけたとする。そこらじゅうのものが、IoT デバイスで、何かの形で、Web で接続され、自分に話しかけている。つまり何かの形でシグナルを出している。例えば、それはデジタルサイネージでも良いし、自動販売機でも良い。今のままだと、この目の前の自動販売機を操作するのに、アプリを立ち上げなければならなかったり、またはそもそもアプリをインストールしなければならないというようなことが起こりえてしまう。異なる IoT を探索し、操作を可能にするオープンな仕様は確立されていない。またサイネージなどですべての情報を出すことができず、続きは Web でというような利用形態もあるが、そのような場合、検索キーワードを示すことで利用者に検索を促したり、 QR コードがはってあることが結構ある。検索キーワードをタイプさせるのは利便性が悪く、QR コードはフィッシングの危険性を考えるとセキュリティ的に不安だ。QR コードが書き換えられていて、他のサイトに飛んでしまうことがある。

この IoT の課題を解決するために、グーグルが提案しているのが、Physical Web である。Physical Web が実現された世界では、スマートフォンの通知画面を開くと、自分の周囲にある IoT デバイスの一覧が表示される。例えば、そこに表示されたデジタルサイネージをタップすることで、追加情報として、詳しい情報を得ることもできれば、専用のアプリをインストールすることもできる。この裏にある技術は Apple の iBeacon と同じ、Bluetooth Low Energy を使った、UriBeacon というものだ。UriBeacon は URI をブロードキャストするものなので、通知画面には各 IoT デバイスのメタ情報が表示される。さらにタップして、追加の情報を得ることもアプリへの誘導も可能だ。こういう世界になると、一つ一つの IoT デバイス毎に事前にアプリをインストールする必要はない。最初のエントリーする部分は、Web が使っている技術をそのま

ま使っている。一種の IoT デバイス間の検索だ。これが実現できるのも、URI という Web のコアと連携されているからである。これが、Physical Web の考え方である。

■まとめ

インターネットはある意味、非常に緩い形で進化している。緩いとは、最初からリッチな標準をかっちりと作り、その後に実装するのではなく、仕様と実装を同時に作っていくサイクルによって、成長していることを意味する。そこにあるカルチャーは、大きく考え、小さくスタートし、できるだけ素早くスケールさせていくというものである。また、イノベーションを生むオープンなソーシャルプロセスも大切だ。IETF や W3C、WHATWG などでは、ある意味、いろいろな人が言いたい放題だが、こういったオープンなカルチャーを持てたがゆえに、今のインターネットの技術のイノベーションがある。

DOM 中心で本当にいいのか、HTTP 以外にいろいろなプロトコルが使われるようになったらどうなるか。Web とは何だろう。ぜひ考えて欲しい。最後のモバイルの話や Web で話したように、Web の本質としても、ユーザから見た文化的な側面からしても、一つのパーマネントな ID が重要だ。Physical Web における URI も非常に単純な記号でしかない。この URI をブロードキャストして、それを入手する。その後は、今まで使ってきた Web 技術の発展でいける、あとはモバイルのアプリのインストールを促したり、モバイルのアプリの中での表示につなげて行ったり、この URI の一つのパーマネント、グローバルな ID を活用していくことによって、モバイルとの連携を深めていくになる。Physical Web の真のパワーはそこである。

これからさらに Web 技術は進化していく。そこで、失ってはいけないのは、世界中の情報を集約し、誰からもアクセスできる基盤や様々な異なる機器制御というものを実現している、このパーマネントな ID であろう。今後の IT の進化も、この Web の進化の理由を探り、イノベーションの本質を理解するところから始まるだろう。

第 3 部　質疑応答

■Q　新しい Web システムを開発する際に、個々の環境を意識しているのでしょうか。(佐々木崇史)

■A　ユニバーサルにアクセス可能にするということは、常に考えている。

グーグルのミッションとして、世界中の情報をくまなく集めて世界中の人にユニバーサルにアクセシブルにするというのがある。ユニバーサルにアクセシブルにすることは非常に大事なことだと思っている。

たとえば、私のプロジェクトを例にお話しすると、Chrome も多くの人に使われるようになってきたので、グーグルが実現したいと思っていることを Chrome に実装してしまえば、多くの人に使ってもらえるだろう。しかし、世界中の人々に使ってもらうことは Chrome にだけ実装されているのでは現実的ではない。むしろ、逆にグーグルとしては極端なことを言えば、Chrome が使われなくても、最新の Web 技術を多くのベンダーが採用し、多くの人が使ってくれさえすればいいと思っている。

目的と手段という議論が、我々がものをつくるときによく出てくるが、多くの人に使って欲しいという目的の下、いろいろな手段を考えるのが大事だ。

グーグルの製品開発のやり方として、正解を最初から求めないというものがある。よくグーグルはどういう会社か聞かれたときに、Scientist の会社だと例えることがある。これは、会社の中に Doctor がたくさんいるという意味ではない。Science の考え方の一つとして、真理があり、その真理に対してのいろいろな仮説に対して実験や実証を繰り返すことによって真理に近づいていくというものがあるが、グーグルはまさにこの方法をとる。グーグルのエンジニアリングカルチャーとして Launch & Iterate と言われるものがある。Launch というのは、何かものを出すという意味で Iteration は反復して繰り返すという意味である。

グーグルは Web の会社なので Launch & Iterate が特に大事になる。Web アプリケーションでは、例えば UI 上の問題などは簡単なものであれば 1 日もあれば直すことができる。そのような部分に完璧を求めるよりも早く配布して、本当に自分たちが必要だと思った機能が、その通りに使われているかどうかを見ることが重要だ。Web ではログを見ればそれがすぐにわかる。

この Launch & Iterate の思想は Chrome にも取り入れている。なので、まず、自分たちが必要だと思うものを組み込んで、ユーザのフィードバックを受けて、完成度を高めていくという手法で開発をしている。

このような Launch & Iterate の思想で、異なる様々な環境に対応する機能の提供を進めている。

■Q　UI のレイアウトを変更する際に、気をつけていることは何か。(金井太郎)

■A　UI の変更で重要なことは、基本原則を守ることと、変更を信念を持って行うことである。

UI 変更には常に痛みが伴う。ユーザからの反発も強い。基本原則を守り、それなりの覚悟で行う必要がある。

Chrome の開発では 4 つの S という基本原則を持っている。Speed、Stability、Simplicity、Security の 4 つ。新機能を追加しようしてもこれらの基本原則に反していると行えない。自動化されたインフラが整備されているので、新機能を追加しようとしても、それがたとえば Speed を著しく阻害するものであったら、その追加は拒否される。

実際に UI の変更を行うと、基本的に文句しか来ない。しかし、乱暴な言い方をすれば、1 週間もすればみんな忘れてしまう。炎上してもひるんではいけない。UI 変更に満足している人はわざわざ声をあげない。声をあげない人たちがどう思っているかを知るのが大事だ。

常に使っている人たちは、毎日使っているので、少しずつ変えれば、みんなすぐ慣れる。しばらく使っていない人は、前の UI など忘れている。いずれにしろ、それなりの根拠がある UI 変更であれば、信念を持って行えば、多少の利用者からの不満の声に怯む必要は無い。しかし、自分たちの変えたいように変えるのではなく、Launch & Iterate を行い、結果がよくなければすぐに元に戻す。Web アプリケーション開発は配布してから反応を得るまでの周期がとても短いのでログを見て、変更が良いものかどうかすぐに分かる。声に出ない声を聴くことが大事である。

■Q　優秀な人材をマネジメントする上で気を付けている点を教えてください。(金井太郎)

■A　できるだけ意見を出せるような環境をつくることを心がけている。

まずリーダとマネージャは違う。マネージャは管理者なので、ある意味で、必要悪なつまらない管理仕事も多い。一方、リーダーシップに関しては、グーグルでは新卒の社員にもリーダーシップを求めている。プロジェクトの一部分を任せて責任を持たせている。マネージャは、そのマネジメントという役割と表裏一体のリーダという部分でも、製品の方向性であったり、開発の方向性を議論したり指示したりするのだが、できるだけ意見を吸い上げる、もしくは意見を出してもらえるようなカルチャーを作ることが大事だ。

自分の体験だと、プロジェクトで方針を決めるとき、過去の自分の経験で対処できるものがいくつか出てくるのだが、3 個の 1 個くらいは今までのやり方と違っていたり、気づかないようなやり方

で解決できるものがあった。業界経験やソフトウェア開発の経験が長かったりしても正解というのは常に変わっていく。正当法にこだわり過ぎないで、従来の考えを否定して考えることが大事だと思う。そのためには、物怖じしないで常に新しい意見を言い合える環境が必要だ。

■Q グーグルの開発プロセスで、ものづくりに応用できるようなものがあれば教えて欲しい。（安達眞聡）
■A 内部をソフトウェアにしておくことで、ハードウェアにおいてもスピードアップができると思う。ただ、開発に 10 年以上かかるような製造業ではグーグルの開発プロセスをそのまま適用するのは難しいとは思う。

■Q 新しいものを生み出す時に、どこに着目点を行けば良いか。（斉藤奨）
■A Creativity を保つことが大事。
いかに Creativity を保つかが大事だと思っている。グーグルでは Demo & Beer ということを行っている。Demo & Beer とは、1 か月に 1 回程度、プレゼンはしないで 5 分程度デモを見せ合うということをしている。他にもイノベーションワークショップという、半日から一日の間プロジェクト離れて、自分のオフィス以外でアイデアを出すということもしていた。
10%の向上ではなくて、10 倍にするためにはどうすれば良いか。普通のやりかたでは到底到達できないような高いゴールをあえて持つことで、全く違うアイデアが出ることもある。例えば Chrome OS という Web アプリに特化した OS を開発したが、その開発を表明した時に、ラップトップの蓋をあけ、電源を入れてから 10 秒でインターネットにアクセスできるようにするというのを掲げた。実際に世に出たものはそれを達成しているが、これはハードウェアからファームウェア、そして OS 本体にいたるまで、あらゆるコンポーネントがマイクロ秒ごとの最適化を積み重ねて実現したものである。
もう一つ、ラベリングを避けるということも行っている。従来からあるものに近いものであっても、あえて呼び方を変えてみることによって、既存のものとは違うものを作ることができる。たとえば Google 日本語入力という、いわゆる、かな漢字変換ソフトウェアがある。これは最終的にはわかりやすいようにと、日本語入力と呼んでいる。専門用語では IME と言う。だが、IME と呼ぶことによって、MS-IME や ATOK、ことえり等の既存の IME が頭に浮かんでしまう。なので、あえて IME と呼ばないようにしてみることで、自分たちの作っているものの本質が理解でき、従来の類似物から離れて、何を提供すべきなのかを考えることができる。
このように、目的を達成するための、従来とは違う別の方法を考えることが重要だ。

■Q これからたくさんの情報にアクセスしやすくなると思うが、セキュリティの点で気を付けていること、これからの課題を教えてください。
■A 正解はないと思う。
セキュリティと利便性はトレードオフの関係にある。1 つの完璧な正解を探すのは難しい。しつこくセキュリティだけを追及すると、逆にユーザはわからなくなってしまう。例えば、利用規約について、たくさんある場合に一個ずつ確認してもらうようにすると、よく読まずにすべてイエスにしてしまうということが起きてしまうだろう。セキュリティについてはより良い解決法を模索中だ。

■Q Academic と Industry の業界では目指している方向が違うと思うのですが、Industry 側から Academic 側への要請はあるか。（安達眞聡）
■A 人材の交流を増やすべき。
人材の交流を増やすといいと思う。また Academic の技術が Industry ではそのままでは使えないことも多いので、研究のほうも実際に社会でどう活かされるかを考えて、研究してみてもよいと思う。

※質問者のうち、本プログラム所属の学生のみ、氏名を記した。

実体情報学博士プログラム
http://www.leading-sn.waseda.ac.jp/