Chromium を支える技術

開発インフラツアー

riesz@

Chromium の開発は多くのインフラに支えられています。Chromium Browser Advent Calendar 2017 21日目の今回はそれらを巡るツアーにご案内しようと思います。

Google Docs が重い場合は軽量版をご覧ください。

(参加を表明してから Jun Mukai 氏による 2012 年の記事と書きたいことが結構かぶっていることに気づきましたが、5年で変化したところも多いのでリスペクトしつつ投稿させていただきます)

目次

謝辞

バージョン管理

Chromium のソースコードは Git でバージョン管理されていますが、依存しているライブラリ群を取得するために depot_tools というツールチェインが必要です。依存ライブラリの場所とコミット ID は DEPS というファイルに deps という変数で記述されています。これをもとに depot_tools に含まれる gclient というツールがチェックアウトをします。git submodules のようなものです。

depot_tools にはコードレビューシステムへパッチをアップロードするツール git cl なども含まれています。

コードの閲覧・検索

ソースコードは Gitiles というツールによってブラウザで閲覧できるようになっています。

https://chromium.googlesource.com/chromium/src.git

左の branches にある master をクリックすると最新のコードが閲覧できます。

これに加え、1日目ですでに触れられているようにコード検索ツールがあり、開発時のコード閲覧には主にこちらが使われています。検索機能に加え、Linux でビルドするときの情報にもとづいたクロスリファレンスがはられており、シンボルをクリックすると宣言・定義や参照元の一覧が表示されます。閲覧 UI の右上には blame やコミットログのためのボタンがあり、Gitiles の該当ページにとべます。

Chromium 自体のソースコードが含まれる src/ ディレクトリには現在60万ファイルほどあるようです。

トップレベルにある src/ 以外のディレクトリにはコードのチェックアウトやビルドに用いるツール、継続インテグレーションを構成する build bot が使用するファイルなどが含まれています。

コーディングスタイル

Google のコーディングスタイルガイドなどベースとしているスタイルとプロジェクトでの例外的ルールが Chromium coding style にまとめています。

C++ の場合、Google C++ Style Guide をベースとし、C++11, C++14 で新しく導入された機能についてはその使用可否が https://chromium-cpp.appspot.com/ で理由とともにまとめられています。スタイルガイドに意見がある場合は cxx@ というグループで変更とその理由を説明して議論

し、合意に達したらガイドをアップデートするパッチをレビューに出すことで変更することができます。

WebKit からフォークしたという歴史的経緯から third_party/WebKit ディレクトリにある Blink レンダリングエンジンのソースコードは特別なコーディングスタイルに従ってきました。最近はスタイルの統一化が進んでいます。

コードレビュー

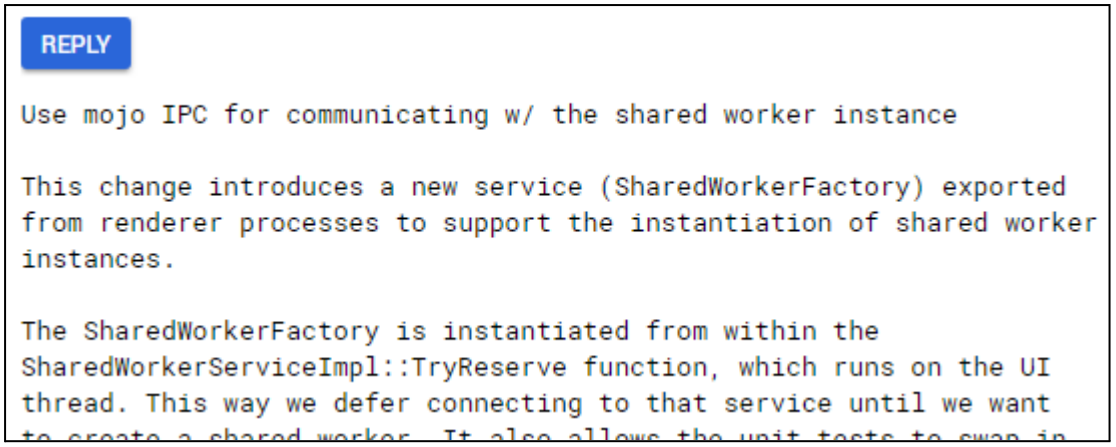
パッチの作者がそのコードのオーナーであるか否かに関わらず、パッチは原則レビューを終えてからコミットすることが求められています。Chromium では Gerrit を用いてコードレビューが行われています。Gerrit より以前は Rietveld を使用していました。

Chromium の Gerrit にはパッチ作成およびレビューを効率的に行うためにいくつかの機能が統合されています。

例として ID 680254 のレビューを見てみましょう。

左上にはパッチの作者 (Owner) やレビューを依頼されている人の一覧 (Reviewers)、その中で LGTM を出した人 (Code-Review) などが表示されています。

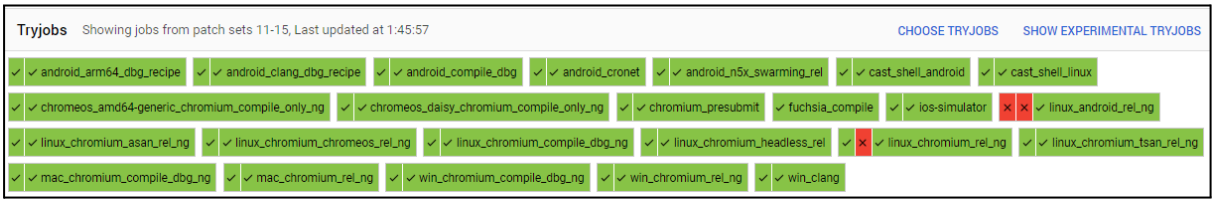
真ん中にはパッチの説明文があり、



その下にはパッチで変更されたファイルの一覧が表示されていて、ここで差分の確認やレビューコメントの入力ができます。

ログインしているとこれらに加え右上にいくつかボタンが表示されます。

コードを Gerrit アップロードして「CQ DRY RUN」というボタンを押すと trybot (tryserver) と呼ばれている build bot にパッチが送信され、パッチの適用結果がコンパイルされテストをパスするかが確認されます。Dry run の結果は Gerrit の Tryjobs という欄に表示されます。



十分な LGTM をもらってレビューが完了したら「SUBMIT TO CQ」というボタンを押して commit queue (CQ) にパッチを送ります。Commit bot が必要な dry run を実行し、それらをパスしたら自動的にリポジトリへパッチをコミットしてくれます。

コミットしたパッチが継続インテグレーションを壊してしまったときは「REVERT」というボタンを押すと自動的にリバートパッチを作成してコミットしてくれます。

PRESUBMIT スクリプト

パッチをコードレビューシステムへアップロードするとき、PRESUBMIT というスクリプトが自動的に実行されます。PRESUBMIT はパッチをスキャンし、変更後のファイルがコーディングスタイルに違反していないか、DEPS ファイル (1日目を参照) で指定された依存関係のルールに違反していないかなどを自動的にチェックし、開発者に知らせてくれます。

ウォッチリスト

ソースツリーのトップに WATCHLISTS というファイルが置かれています。このファイルには Python の dict リテラルの形式でファイルパスのパターンと通知先メールアドレスが記載されています。

誰かがコードレビューシステムへパッチをアップロードするとき、アップロードツールが自動的にこのファイルの内容にもとづいてパターンマッチを行い、変更されているファイルそれぞれに対応する通知先メールアドレスをコードレビューの CC 欄に加えます。

コードレビュー上のアクティビティは CC 欄で指定されているメールアドレスに通知されるので、WATCHLISTS を利用して特定のディレクトリやファイルの変更を監視できるわけです。

ドキュメンテーション

プロジェクト関連のドキュメントは長らく Google サイト で蓄積されていました。

http://dev.chromium.org/Home が Chromium プロジェクトのサイトです。どのようにすればコードを取得しビルドできるか、開発に参加するにはどうしたらいいか、といったガイドや、各種ツールへのリンク、設計書 (design docs) の一覧などがあります。

2014年ごろドキュメントをソースコードリポジトリに置こうという提案があり、現在では dev.chromium.org にあったドキュメントの多くが (Gitiles flavored) Markdown で書き直され、docs/ ディレクトリに配置されています。

今回の投稿内容の大半はこれらのドキュメントにも書かれています。

コミュニケーション

開発者間のコミュニケーションには Google グループ および IRC が使われています。私はあまり見ていないのですが Slack も実験的に使われているようです。

Technical Discussion Groups というページにグループの一覧があります。メインの chromium-dev@ に加え、分野ごとのサブリストが多数作られています。例えば net-dev@ では TCP, UDP, TLS, HTTP2, QUIC, DNS などを処理するネットワークスタックの実装に関するトピックが扱われます。

freenode IRC ネットワークに置かれた #chromium チャンネルでは継続インテグレーションの状況が bot によって通知されるほか、時折インタラクティブな議論が行われています。

課題管理システム

Chromium のために作られた Monorail というバグトラッカー (イッシュートラッカー) を使用しています。Monorail が導入される前は Google Code のバグトラッカーを利用していました。

https://bugs.chromium.org/p/chromium/issues/list

Chromium のバグトラッカーには crbug.com という URL shortener があり、バグ ID をパスとしてつけてアクセスすると対応するイシューのページにとべます。

crbug.com/123456

crbug の読み方が2日目の記事で詳しく紹介されていますのでそちらもご覧ください。

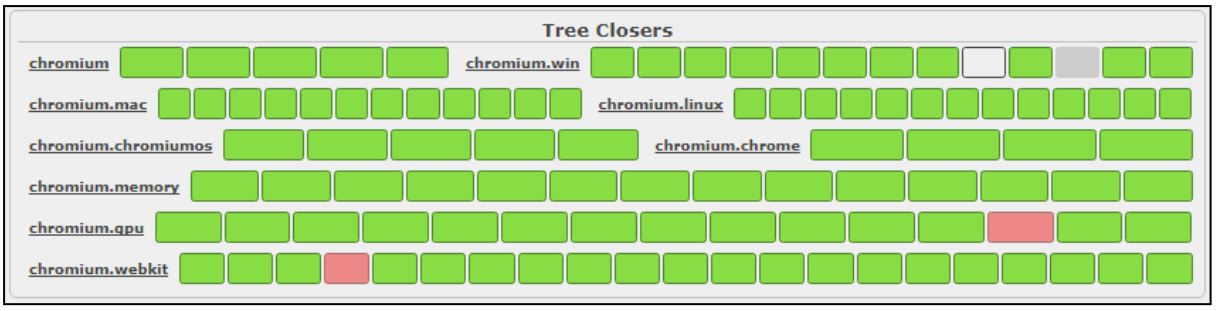
継続インテグレーション

継続インテグレーションの状況は https://ci.chromium.org/p/chromium/g/main/console で見ることができます。

「Chromium Main Console」というタイトルに続いてまずツリーの状態が表示されています。

継続インテグレーションで重大な異常が起きていない時はここに「Tree is open」と書かれて緑色になっており、commit queue によるコミットを受け付けています。コンパイルエラーなどの重大な異常が検出されたときはツリー状態が「Tree is closed」に設定され、コミットを原則禁止します。その時の監視担当者 (tree sheriff) が原因のコミットをリバートするなどして問題を修正した上でツリー状態を再度 open にするとコミット受け付けが再開されます。

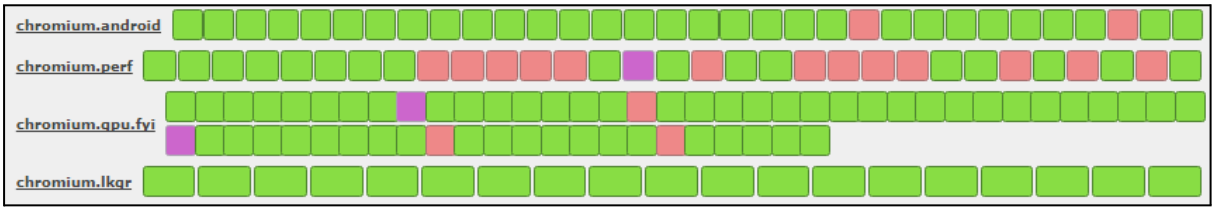
ツリー状態のすぐ下の「Tree Closers」というボックスには四角がたくさん並んでいます。この一つ一つが build bot です。



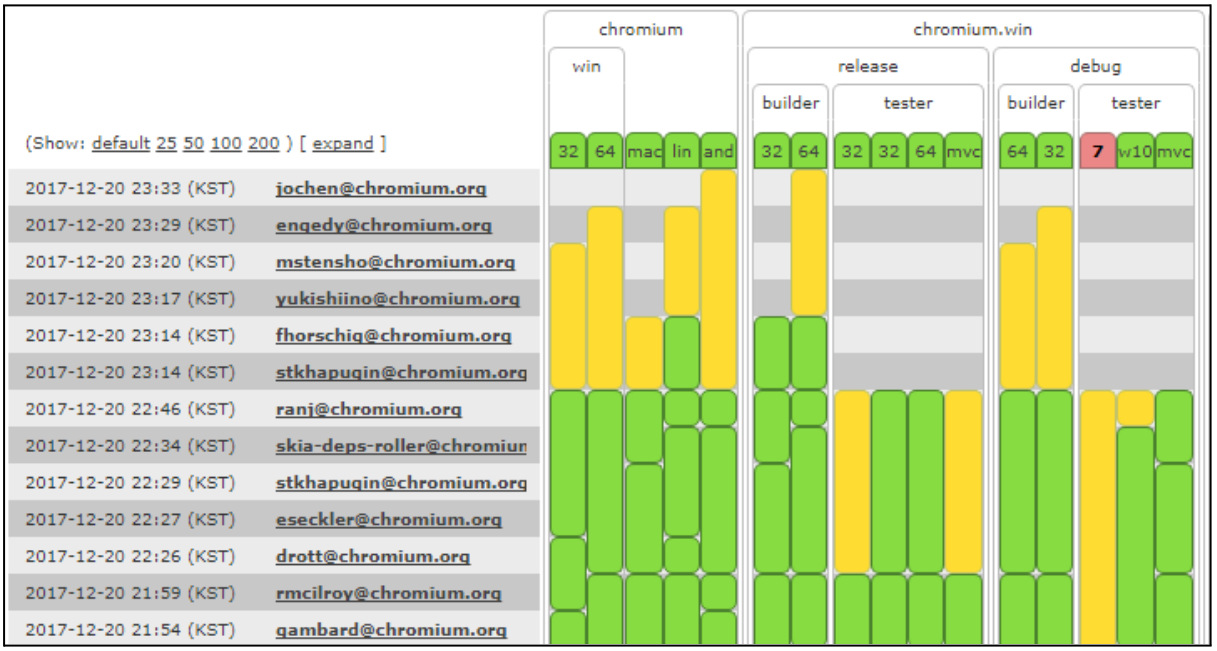
赤くなっている bot ではコンパイルやテストの失敗が発生しています。

Build bot をクリックすると最新のビルドの進行状況 (Steps and Logfiles) や、そのビルドから新たに含まれたコミットの一覧 (All Changes) が表示されます。ビルドはたくさんのステップからなり、それぞれのログや出力が保存されると同時に分析されています。たとえばテストを実行するステップで異常が起きたとき、失敗したアイテムの一覧と結果の確認ができるページへのリンクが表示されます。

「Tree Closers」ボックスのすぐ下には tree closer ではない build bot が並んでいます。



ページ下部のテーブルでは各行がコミットを、各列が build bot を、そしてテーブルにたくさん並んでいる縦長の棒の1つ1つが1回のビルドを表しています。あるビルドから新しく含まれたコミットがどれなのかを対応する棒がカバーすることで表しています。



Build bot には実行バイナリのビルドだけをするもの、テストを実行するもの、メモリリークを検出するもの、パフォーマンスの異常がないか確認するもの、などいろいろな役割のものがあります。例えば「chromium.win」グループに含まれる「Win 7 Tests x64 (1)」は Windows 7 の 64 ビット版上でテストを実行する build bot です。Android や iOS のための bot もあります。

ビルドの種類によっては実行にとても時間のかかる場合があり、そうした build bot では比較的多数のコミットをまとめてビルド (テスト) しています。そうしたビルドで異常が起きた場合は更なる bisect をしないと原因のコミットが特定できない場合もあります。

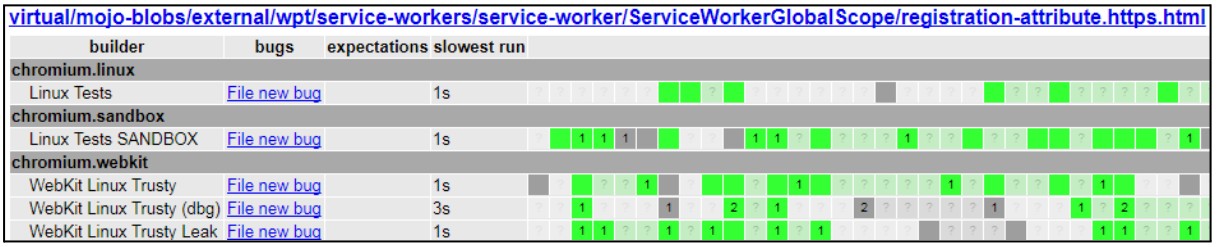
Build bot が測定したパフォーマンスのデータは https://chromeperf.appspot.com/ で閲覧可能です。異常を自動で検出し、バグトラッカーに報告したり bisect を行って原因と思わしきコミットを特定するシステムなどが稼働しています。

Flakiness dashboard

出来る限り成功・失敗が決定的なテストを書くようにしていますが、非同期処理のタイミングなどが原因で実行するたびに結果が変わってしまうテストを書いてしてしまうことがどうしてもあります。

こうしたテストは flaky test と呼ばれ、特に失敗の頻度が低い場合は継続インテグレーションで失敗が検出できたとしても一体どのコミットが原因かを特定するのが難しくなります。

Flakiness dashboard はテスト結果の履歴を見やすく表示してトレンドをわかりやすくし、テストが flaky かどうかの判断や原因コミットの絞り込みなどをしやすくしています。



例えばこのテストの場合、「WebKit Linux Trusty」bot において最新のビルドで失敗しつつ、その前のビルドでは成功しています。

Chromium が使用しているレンダリングエンジン Blink は WebKit レンダリングエンジンのフォークなので、歴史的経緯からソースコードやインフラに WebKit という文字が残ったままなものがあります。

ローテーション

プロジェクトのしごとの中にはローテーションを組んで持ち回りで担当しているものがあります。主なものが tree sheriff とバグトリアージです。

Tree sheriff

継続インテグレーションを監視し、コンパイルやテストが失敗したときにそれらを調査して対応するのが tree sheriff の仕事です。Build sheriff とも呼んでいます。

平日の各曜日に対し3人が異なるタイムゾーンから割り当てられており、各々が勤務時間内に sheriff を行うことでこの表にあるように24時間シフトを構成しています。

担当者には Google Calendar およびメールで自動的にリマインダーが送られるようになっています。

継続インテグレーションのステータス画面 (ツリー) を直接監視していた頃もありましたが、現在は Sheriff-o-Matic という補助ツールを経由して監視を行っています。Sheriff-o-Matic はコンパイルやテストが失敗している build bot の必要な情報のみをわかりやすく表示し、加えて sheriff が作業状況を書き残せるメモ機能や原因のコミットを自動的に推定して表示する機能 (FindIt) などを提供しています。残念ながら認証が必要なため UI を皆さんに見ていただくことはできません。

最近では FindIt による自動リバート機能も導入されました。

バグトリアージ

バグトラッカーに新しく報告された問題を確認し、優先度の確定や担当者の割り当てを行うのがバグトリアージ担当者の仕事です。Chromium 担当の Google 社員は原則 tree sheriff かバグトリアージのいずれかのローテーションに入っています。

Chromium はとても多くの要素から成り立っており、ひとりひとりが詳しい範囲は限られているため、段階的なトリアージが行われています。例えばあるバグがネットワークスタックに関連していそうだなと思った時は「Internals>Network」というラベルをそのバグに設定しておけば、ネットワークスタック専門のチームの誰かがあとで詳細な分析をしてくれます。

分散ビルド

Chromium は大量のソースコードから構成されており、パフォーマンスの高いワークステーションを用いてもビルドに時間がかかります。

残念ながら一般に広く公開はしていませんが、Google 社内では Goma という分散ビルドシステムを開発・運用しており、開発用ワークステーションに高い負荷をかけることなく短時間でビルドができるようになっています。

少し古いですが、2011年に鵜飼文敏氏が Goma の詳細を Google Developer Day で発表しております。

リリース

定期的に master からリリース用ブランチが作られ、これらのブランチには通し番号がついています。

Chrome のバージョン番号は Version Numbers で解説されているように MAJOR.MINOR.BUILD.PATCH という構造を持っていて、BUILD の部分がこのブランチ ID に対応します。

例えば、現在最新版の canary channel はバージョン 65.0.3299.0 ですが、このリリースが作られたブランチ 3299 の HEAD は以下です。
https://chromium.googlesource.com/chromium/src/+/refs/branch-heads/3299
このブランチは master のコミット ID bdab2c から作られています。master のコミット1つ1つには通し番号 (リビジョン番号) がついており、このブランチ元コミットのリビジョン番号 524906 です。これらコミット ID とリビジョン番号は Cr-Branched-From というタグに記載されています。

```
Cr-Branched-From:
   bdab2c10eaad8f821b6eac3c083da4f5fb4c20c8-refs/heads/master@{#524906}
```

リリースブランチには原則としてリリースにどうしても入れなければいけないバグ修正コミットのみがマージされます。現時点でブランチ 3299 には1つのパッチがマージされているようです。

こうしたブランチから canary channel や dev channel が定期的にリリースされ、Chrome Release Cycle にあるように約6週間に1回 stabilization ブランチが選ばれます。Stabilization ブランチからいくつか beta channel がリリースされていき、最終的に stable channel がリリースされます。

OmahaProxy

OmahaProxy というツールが Chrome のリリースシステムと連携してその状況を公開しています。

新しく追加した機能やバグ修正がいつリリースされたか、どのチャネルまでリリースされたかを確認するときに我々も使用しています。

各リリースチャネルの最新版がどこからブランチしたかは表の branch_base_commit および branch_base_position 欄で確認できます。

使用統計データ

Chrome は Google Chrome のプライバシー ホワイトペーパーでご説明しているように協力に同意していただいたユーザの皆様からいただいた使用統計データを性能向上のための方針決定に利用させていただいています。

例えば、数ある HTML や JavaScript API の機能の中でそれぞれがどれだけ多くの人に使われているかは UseCounter というクラスで集計され、HTML & JavaScript usage で閲覧可能にしています。最近計測精度を向上させる変更が加えられたため、2017/10/27 付近が不連続になっていますが、例えば非同期モードの XMLHttpRequest はユーザが訪問したサイト (述べ) のうち約 46%で使用されていることがわかります。

また、タブは平均していくつぐらい開かれているのか、WebSocket ではどのぐらいのサイズのメッセージが主に使われているか、といった統計は各機能の実装・改良方針の決定に役立ちます。

これらの統計項目は tools/metrics/histograms/ にある histograms.xml という巨大な XML ファイルで定義されています。

クラッシュが発生したときにいただいている障害レポートも自動的に分析し、バグ修正に役立てています。

Field trial

ある視点で性能向上が見込める変更を思いついても、その変更が思わぬところでパフォーマンス低下などの問題を引き起こしたりすることがあります。ある変更を含むリリースとそれ以前のリリースにおける統計データを比較するという方法もありますが、1つ1つのリリースにはその変更以外にも多くのコミットが含まれるため、確実にその変更が性能向上につながったかを常に判定できるわけではありません。Field trial はある変更を一部のユーザでオンにさせていただき、オフのユーザからのデータと比較することで統計的に有効性を確認する仕組みです。field_trial.h に説明が書かれています。

Origin trials

Field trial では一部のユーザの皆さんにある機能を有効にしていただくものでしたが、Origin trials はある新機能を実験参加申し込みをいただいたサイトへのアクセスのみで有効にする機能です。

新機能の導入を検討しているとき、その機能が本当にウェブ開発者・利用者の皆様の役に立つものなのか、その API は本当に使いやすく設計できているか、といったことを確認するには実際に利用していただくのが一番です。しかし、一旦機能を完全にリリースしてしまうと多くの人が利用し始め、廃止や変更が簡単にはできなくなります。Origin trials を利用することで新機能の導入

の是非が未確定なことや API の不安定性を理解していただいた上で実験に参加していただくことができるようになっています。

相互運用性

Chrome、Edge、Firefox、Safari といったウェブブラウザは互いにその性能を競い合い、その結果として皆さんによりよいソフトウェアを提供することができています。

しかし、HTML や JavaScript API の振る舞いがブラウザ間で異なることはプラットフォームとして望ましいことではありません。標準化団体や各ブラウザベンダーは長らくこの課題にいろいろな方法で取り組んできました。

最近こうした動きが加速し、現在では web platform tests (WPT) と呼ばれるテストスイートリポジトリでこれらのブラウザベンダーが共同でテストを開発しています。WPT に含まれるテストは HTML で書かれたテストケースをレンダリングして結果を確認するという形のもので、これは Chromium では layout tests と呼ばれていました。現在では新しくテストを書くときには可能ならば Chromium 専用のものではなく WPT へアップストリームできるものを書くことが推奨されています。

Interoperability チームの尽力により、WPT の開発は通常の Chromium 開発ワークフローと同じように行えるようになっています。WPT は Chromium 内では third_party/WebKit/LayoutTests/external/wpt/ というディレクトリに配置されています。ここに変更を加えるパッチをアップロードし、コミットするとこのように「Blink WPT Bot」という bot が W3C の WPT リポジトリへ自動的にマージしてくれます。

ちなみに、third_party/WebKit/LayoutTests/ ディレクトリにある external/ 以外のファイルが Chromium 専用の layout tests です。かなりの layout tests が WPT へアップストリームされましたが、依然として多数存在します。また、理由があって WPT へは移せないものもあります。

WPT を各ブラウザで実行した結果は wpt.fyi というサイトで閲覧することができます。

WPT を書いてみたい方は @horo 氏のWPTをChromeで実行してHTML5 APIを理解するや19日目をご覧になるのも良いでしょう。

謝辞

@nhiroki 氏、@horo 氏、@peria 氏にレビューをしていただきました。ありがとうございます。