# ブラウザ内仮想マシンを用いたWebページ操作の提案

Wu Dawei[1]　新城 靖[1]　清水 海大[1]

## 1. 序論

近年、Web ページからの情報抽出や、表示をユーザの好みに応じて変更する augmented browsing [1] の要求が高まっている。これらの要求に応えるため、JavaScript によるブラウザ拡張機能の作成、スクレイピングライブラリ、あるいはブラウザ自動化フレームワークの利用といった手法が一般的に用いられている。しかし、いずれの手法も独自の API を習得する必要があり、特にログインプロセスやセッション管理を伴う操作は複雑である。簡便な操作インタフェースが求められている。

我々は、Chrome DevTools Protocol (CDP) [*1] と FUSE (Filesystem in Userspace) [*2] を用い、Web ページの内容をファイルとディレクトリにマップするファイルシステム PageFS [2] を開発している。PageFS を利用することで、ユーザは使い慣れたシェルコマンドやスクリプト言語でファイルとして Web ページを操作できる。

しかし、PageFS は複数の課題を抱えている。まず、HTML 要素へのアクセスが XPath に限定されており、Web ページ操作に関する知識を必要とする。また、OS カーネル機能の FUSE に依存するため、動作環境が Linux や macOS に限定される OS 依存性の問題がある。さらに、ブラウザとは独立した外部プログラムとして動作するため、利用の手軽さを損なう。加えて、CDP で制御されるブラウザは、ボット対策システムによって自動操作が検知されやすい。

そこで本研究では、新しいシステム PageShell を提案する。PageShell は、PageFS のファイルシステムインターフェースという利点を継承しつつ、上記課題を解決する。XPath の知識が不要な、セマンティックファイルシステムの考え方に基づく柔軟なクエリを可能にする。

PageShell は、ブラウザ内仮想マシン、9P プロトコル [*3]、ブラウザ拡張機能の技術を組み合わせ、全ての処理をブラウザ内で完結させる。これにより、OS 非依存の高いポータビリティ、利便性、ボット検知耐性を備えた Web ページ操作環境を提供する。

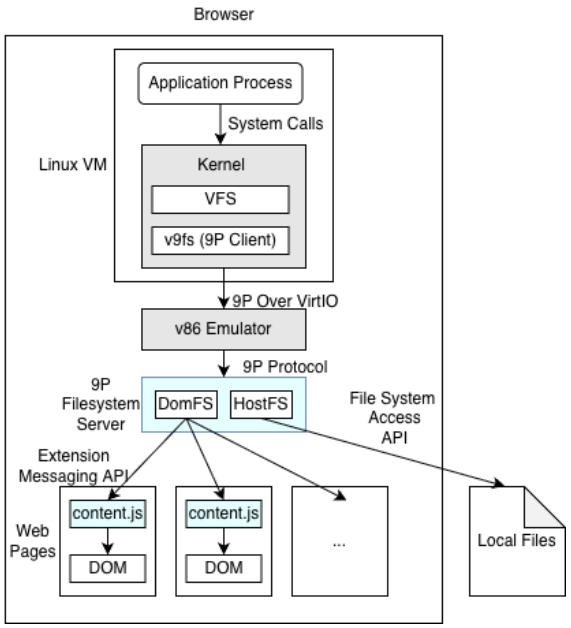


**図 1**　PageShell のアーキテクチャ

## 2. PageShell の構成と利用例

PageShell は、図 1 に示すように、すべてのコンポーネントがブラウザ内で動作する。主要なコンポーネントは以下の通りである。

- **v86 エミュレータ** [*4]**:** WebAssembly (WASM) [*5] 上で動作する x86 PC エミュレータ。OS 非依存性を保証する。
- **ゲスト OS:** v86 上で動作する軽量なカスタム Linux。Python や Ruby といったスクリプト言語の実行環境を備える。
- **9P ファイルシステムサーバ:** ゲスト OS からの要求を VirtIO と 9P プロトコルを通じて受け取り、処理する。
- **DomFS と HostFS:** 9P サーバのバックエンドとして動作する仮想ファイルシステム。DomFS はブラウザの DOM API を利用し、Web ページの内容をファイルシステムとしてゲスト OS に提供する。HostFS は File System Access API [*6] を利用し、ブラウザが動作している OS のファイルへのアクセスを提供する。抽

[1] 筑波大学
University of Tsukuba

[*1] https://chromedevtools.github.io/devtools-protocol/
[*2] https://github.com/libfuse/libfuse
[*3] https://github.com/ericvh/9p-rfc/
[*4] https://github.com/copy/v86
[*5] https://webassembly.org/
[*6] https://developer.chrome.com/docs/capabilities/web-apis/file-system-access

出した情報の保存やローカルデータとの連携を可能にする。

- **content.js:** 各 Web ページに注入され、DomFS の指示に基づき、ページ内の要素検索や書き換えを実行する。CDP ではなく標準の拡張機能 API を利用するため、ボット検知のリスクが低い。

図 2 に DomFS の利用例を示す。ユーザはタブ ID に対応するディレクトリ（例: `/mnt/dom/10`）に移動し、特殊なディレクトリ名を用いて Web ページ要素に対するクエリを行う。例えば、図 2 で、“`.class/price`” はクラス名が `price` である全要素、“`.find/yen`” はテキストに `yen` を含む全要素、“`.tag/table`” は `<table>` タグの全要素をそれぞれ意味し、これらの一致する要素を取得している。クエリ結果の各要素（例: “`.tag/table/1`”）はそれ自体がディレクトリであり、図 2 の `ls -a` コマンドの実行結果が示すように、内部に仮想ファイルやディレクトリを含んでいる。主な役割は以下の通りである。

- `.class/`, `.find/`, `.tag/`: セマンティックファイルシステムへのクエリを行うためのディレクトリ。
- `.txt`, `.html`: 要素のテキスト内容と HTML 構造を取得するためのファイル。
- `.csv`, `.tsv`: `<table>` 要素を CSV や TSV 形式で取得するためのファイル。
- `.attr/`: 要素の属性（例: `href`, `style`）をファイルとしてアクセス可能にするディレクトリ。
- `.p/`, `.c/`, `.a/`: 親、子、祖先要素へアクセスするディレクトリ。
- `.js`: 書き込まれた内容を JavaScript コードとして実行するためのファイル。要素を変数として参照可能。

ファイルシステムへの書き込みを通じて Web ページを操作することも可能である。例えば、図 2 で `echo` コマンドは、“`.tag/table/1/.attr/style`” ファイルに書き込むことで、対象 Web ページの要素のスタイルを変更している。

## 3. 実装

本研究では PageShell をブラウザ拡張機能として実装している。ゲスト OS とブラウザ拡張機能との間の通信プロトコルとして 9P を採用した。実装した主要な 9P メッセージは、セッション開始 (Tattach)、パス解決 (Twalk)、ファイル I/O (Topen, Tread, Twrite)、解放 (Tclunk) などである。

DomFS は、ユーザからのアクセス要求に応じて必要なファイルやディレクトリを動的に生成する遅延評価方式を採用している。この処理は主に Twalk と Tread によって起動される。

例えば、ユーザが `ls /mnt/dom/.class/price` を実行すると、ゲスト OS はまず “`.class/price`” のパスを解決するために Twalk 要求を送信する。9P サーバはこれを受け、DomFS が `price` をクエリパラメータとして保持する新しいディレクトリを動的に生成する。

```
$ cd /mnt/dom/10
$ ls .class/price
1    2
$ grep "" .find/yen/*/.txt
.find/yen/1/.txt:3,500 Yen
.find/yen/2/.txt:12,000 Yen
$ ls -a .tag/table/1
.a .attr .c .class .csv .find .html .js .p .tag .tsv .txt
$ cat .tag/table/1/.tsv
Product Name     Price
A    3,500 Yen
B    12,000 Yen
$ echo "color: red;" > .tag/table/1/.attr/style
$
```

**図 2** PageShell における Web ページの操作

次に、ゲスト OS がこの新ディレクトリに対し Tread 要求を発行すると、DomFS はこれをきっかけに content.js にクエリ（クラス名 `price`）を送信する。content.js は DOM API（`getElementsByClassName` など）を用いて要素を検索し、条件に一致した各要素から必要な情報（テキスト内容、HTML など）を抽出して返す。DomFS は、返された情報に基づき、“`.class/price`” ディレクトリ配下に個々の要素に対応する子ディレクトリを生成する。

## 4. 関連研究

Wildcard [3] は Web ページ上のデータをスプレッドシートのような表にマッピングする研究である。Web ページをより扱いやすい抽象モデルに変換するという思想は本研究と共通するが、本研究では操作の基本としてファイルシステムインターフェースを採用する点が異なる。

## 5. 終わりに

本研究では、先行研究の課題（XPath 限定、OS 依存性、利便性、ボット検知リスク）を解決し、ブラウザ内完結型アーキテクチャを採用した PageShell を提案した。ファイルシステムという簡便な UI で、OS 非依存かつボット検知に強い Web 操作環境を実現した。今後の課題は、セマンティックファイルシステムの機能拡充と性能評価である。

### 参考文献

[1] Iñigo Aldalur. Web Augmentation: A systematic mapping study. *Science of Computer Programming*, Vol. 232, p. 103045, 2024.
[2] 清水海大, 新城靖. Web ブラウザに表示されたページ内容を結合するためのファイルシステム. 情報処理学会第 36 回コンピュータシステム・シンポジウム, pp. 50–59, 2024.
[3] Kapaya Katongo, Geoffrey Litt, and Daniel Jackson. Towards end-user web scraping for customization. In *Companion proceedings of the 5th international conference on the art, science, and engineering of programming*, pp. 49–59, 2021.