Vol.2010-MPS-80 No.16
2010/9/28

Webアプリケーションのユーザ快適性向上を目指した一つのモデルの提案と解決

涌井智寛[†1]　沼崎隼一[†1]
三塚恵嗣[†1]　畠山正行[†2]

Web アプリケーションにおけるユーザの快適性を改善する技術は未だ途上にある．我々は，快適性の要因の 1 つである，Web アプリケーションにおけるユーザ操作への応答性に着目した．データ転送量単位とその頻度，そしてブラウザのレンダリング時間を考慮し，ファイルサイズで数 MB，HTML 文書で数万要素程度の，テキスト主体のデータを扱う Web アプリケーションの応答性を高めるための 5 つの工夫を考案した．これらの 5 つの工夫はテスト実装の結果から，Web アプリケーションでは古くから広く使われている簡易な技術のみで構成できた．そこで簡易な技術のみを前提とし，5 つの工夫を複合的に適用してテキスト主体の数 MB 程度のデータを扱う Web アプリケーション応答性良好にするための指針を，Web アプリケーションの 1 つの実装モデルとして提案する．この実装モデルに従い，実際に Web アプリケーションを開発してアンケートによるユーザ評価を行ったところ，良好な結果を得たので報告する．

A Model and it's Implementation aiming at user's Comfortable Operation Environment of Web Application on the Browser

Tomohiro Wakui,[†1] Toshikazu Numazaki,[†1]
Keishi Mitsuka[†1] and Masayuki Hatakeyama[†2]

The skill to improve user comfort of Web applications has been developed. We have focused our goal on the responsibility of the Web browser that is a factor of the usability for Web applications. Specially, we have focused on the amount of the transfer data, the frequency and the rendering time and, we have made up five ideas that improve the responsibility for the Web application. These five ideas consist of simple and traditional Web skills. In this paper, we propose an implementation model that are based on our five ideas and report on the result about our Web application that has been developed on our implementation model.

1. はじめに

Web アプリケーションの利用環境は広がり，インターネット接続端末も従来の PC だけでなく，スマートフォン[⋆1]や iPad[1)⋆2]なども普及を見せつつある．ブラウザ上でインタフェースとして動作する Web アプリケーションは，インタラクティブな GUI としての役割を果たす．高性能な PC だけでなく，スマートフォン，iPad などからも Web アプリケーションは利用されるため，ユーザ利用の快適性と実用性の観点から，Web アプリケーションの応答速度の改善は一つの課題である．Web アプリケーションのパフォーマンスチューニングは複雑であるため，具体的ガイドラインは有用と思われる．我々はユーザ快適性の 1 要因としてパフォーマンス向上を目的に，ブラウザでのユーザ操作への応答性を高める具体的な工夫を見出し，それらを複合して Web アプリケーションの 1 つの実装モデルとすることを狙う．

本論文では，2 章でブラウザ上の Web アプリケーションの現状技術，我々自身の問題経験と設計方針について，3 章で応答速度を改善するための具体的なアイデアと，それに基づく実装モデルの提案について，4 章で各工夫のテスト実装の評価について，5 章で実装モデルを適用した Web アプリケーションの結果の評価と考察について，6 章で結論と今後の発展的な問題への提案について述べる．

†1 茨城大学大学院理工学研究科
Graduate School of Science and Engineering, Ibaraki University
†2 茨城大学工学部情報工学科
Department of Computer and Information Sciences, Ibaraki University

⋆1 本格的なインターネット接続機能を備える携帯電話．性能や機能面では PC に劣るが，静的な Web ページだけでなく，インタラクティブな Web アプリケーションが利用できる．

⋆2 Apple が提供する無線通信機能とタッチパネルを備えるインターネット接続端末で，PC とスマートフォンの中間的な機能を持つ．

図 1 Web アプリケーションでの取り扱い対象データのイメージ
Fig. 1 Image of target data of Web application

2. ブラウザ上の Web アプリケーションの現状技術と本研究の設計方針

2.1 Web アプリケーションの設計・開発に関する現状技術

Web アプリケーションは，ブラウザで所定 URL へアクセスするだけで利用できるため，古くからあるスタンドアローンのデスクトップアプリケーションと比べて，ユーザの新規導入，ユーザ側のコードを最新の状態に維持することが容易等の特徴がある．一方で，通信や複数ユーザによるサーバリソースの共有など，複数の要素が複雑に絡むために，Web アプリケーションの設計・開発はより複雑となる面がある．

Web アプリケーションの開発の手間を軽減する技術は，prototype[2)⋆1]，jQuery[3)⋆2]，Ruby on Rails[4)⋆3]など多数ある．その一方で，パフォーマンスを引き出す (特に簡単にパフォーマンスを引き出す) 技術は不十分である．高いパフォーマンスを達成するためには，複雑な関係を考慮し，多様な技術を駆使してパフォーマンスチューニングを行わなければならない[5)]．もしパフォーマンスが悪ければ，ユーザは度々の待ち時間にストレスを感じ，Web アプリケーションの快適性は低いものとなる．

⋆1 Ajax フレームワーク，コードの記述を簡略化など，Web アプリケーション開発を支援するライブラリである．
⋆2 prototype と同様に Web アプリケーション開発を支援するライブラリ．
⋆3 Web アプリケーションを生成するフレームワーク．

2.2 我々の問題経験と開発の変遷

我々はこれまでに，分析，設計，実装を支援するアプリケーションを異なる設計で複数回開発した[6),7)]．その中の一つで Web アプリケーションをインタフェースとした設計では，以下の 2 点がユーザの不満の原因となった．

(1) サーバ側の処理が多く，サーバとクライアント間の通信で頻繁に待つこと．
(2) ブラウザ上の GUI の実装が素朴であり，取扱うデータ量の増加に伴ってブラウザの応答性の低下が体感できること．

サーバの負荷状況によって応答時間が伸び縮みし，ちょっとした選択や編集毎に数秒待つこともあった．ユーザからのヒヤリングでは，このストレスが改善できないならばデスクトップアプリケーションの方が良いという意見も強かった．

2.3 取り扱い対象データの特徴と規模

図 1 は我々が実際に設計・開発した Web アプリケーションのスクリーンショットである．図 1 に示したのはユーザの興味ある対象となる世界をオブジェクト指向パラダイムで離散化，および構造化して記述することを支援することが目的の Web アプリケーションであり，データを表組で表している．取扱うデータは，大きめの物ではファイルサイズで数 MB，HTML 文書で 1 万〜10 万要素程度となる．Excel や幾つかの Rails アプリ[⋆4]もそうであるように，複数の表と，それらの表を束ねるようなデータが使われる場面はいくつもある．Web アプリケーションで扱うデータとしても一つの定型と言える．

数 MB のデータは 100Mbps の帯域で数百ミリ秒で送受信できる．Web アプリケーションの利用開始時／終了時の待ち時間は数百ミリ秒で体感回数も少ない．そのため利用中に頻繁に利用される機能の応答性がユーザの快適性に対して支配的になると言える．

2.4 Web アプリケーション応答性改善の設計方針

我々が設計・開発した Web アプリケーションは，プログラム開発の分析，設計，実装を支援するものである．Eclipce などの開発環境の利用がそうであるように，我々の Web アプリケーションも短くて数十分，長ければ数時間単位で利用する．すなわち，Web アプリ

⋆4 Ruby on Rails で作られた Web アプリケーション

ケーションをしては長期滞在型に属する．そのため，データのロードやセーブなどアプリケーション利用開始／終了時以外には通信機能は殆ど使わずに済むと想定できる．数 MB 前後のデータであれば転送時間は数百ミリ秒程度であるから，編集時に頻繁に使われる機能の応答性向上を優先すべきと判断できる．

本研究ではユーザの快適性を改善するため，

(1) 一度の通信でやり取りするデータ量を増やして通信頻度を減らしつつ，

(2) 保持，操作するデータ量が増しても，ブラウザの応答性への影響を抑える工夫を施す事で問題解決をはかることとした．

2.5 ブラウザの応答性改良のアイデア

ブラウザでの処理時間は

(a) 要素のオブジェクト作成時間

(b) DOM(Document Object Model) ツリー操作[⋆1]時間

(c) 表示時間

の 3 つで決まる．これら 3 つは要素種類，要素数，要素の内容，属性値，要素のプロパティ，要素へのアクセス方法に左右される．

要素種類について例を挙げる．まず，HTML 文書は構造化された要素の集まりとして表現される．ブラウザで閲覧／編集可能な要素は input 要素[⋆2]，textarea 要素[⋆3]である．表組は table 要素[⋆4]，表の行は tr 要素[⋆5]，セルは td 要素[⋆6]で表現する．一般的な集約階層構造は div 要素[⋆7]によって表現される．

表 1 1 要素あたりのレンダリング時間 [ミリ秒]
Table 1 Rendering time per a element

要素種類	div	span	td	input	textarea
要素数 1000	0.037	0.042	0.055	0.37	10.29
要素数 10000	0.031	0.078	0.045	1.49	118.97

⋆1 HTML 文書内の要素にアクセスし，編集や削除などを行うこと．
⋆2 HTML 文書内で利用できる要素．ブラウザ上で GUI で内容の文字列を編集できる．
⋆3 HTML 文書内で利用できる要素．複数行の文字列が編集できる．小さなメモ帳の様なもの．
⋆4 HTML 文書の構成要素の 1 つ．内部にセルに相当する要素などを持つ，表組を表現する要素である．
⋆5 tr 要素は HTML 文書を構成する要素の一つ．table の内部で行を表現する要素である．
⋆6 td 要素は tr 要素の内部で個々のセルを表現する要素である．
⋆7 HTML 文書内で利用できる要素．自身が複数の要素をもったり，自身が他の要素にもたれる．

表 1 は div，span，td，input，textarea と要素種類ごとのレンダリング時間の差を示すものである．テスト環境は Windows XP，Core 2 Duo，2GB メモリ，Firefox 3.6.7 である (以降テスト環境は同一)．我々が取扱ったデータで言えば，単純に編集可能な要素を用いれば input 要素あるいは textarea 要素が 1 万個以上となる．10000 要素のレンダリングには，表 1 に示すように，input 要素 1 つあたりでは 1.49 ミリ秒，textarea 要素 1 つあたりでは 119 ミリ秒を要するのに対し，td，div 要素では 0.05 ミリ秒程度とその差は大きい．

先の 3 つの項目とそれを左右する 5 つの要因に対して，我々は以下の 5 つの工夫を考案する．

(1) (a) に対する工夫は，textarea 要素でなく span[⋆8]，div，td 要素で軽量化するなど，軽量な要素種類を使い，要素種類の変更に伴う差分を JavaScript[⋆9]で埋める．

(2) (a) に対する工夫のもう 1 つは，class 属性値[⋆10]にデータを置き，必要に応じて要素に変換することで要素数を減らす．変更に伴う差分は JavaScript で埋める．

(3) (b) に対する工夫は，特別な ID 命名規則によって ID アクセスする．

(4) (c) に対する工夫は，display プロパティによる選択表示 (基本は不可視)．要素のプロパティの観点からより軽量な状態を作る．

(5) (c) に対する工夫のもう 1 つは，固定幅 table レイアウトによるレイアウト最適化．

次章で具体的かつ詳細に説明するが，これらの工夫により，可能な限り多くのデータを一括してクライアントのブラウザへと送り，通信とサーバでの処理時間を減らすことで，ブラウザでの処理時間を快適性の支配的要因とする．

3. Web アプリケーションの応答性向上のアイデアの具体化

3.1 工夫 1：軽量な要素を用いておき編集時のみ内容を変更可能な要素にする

もっとも単純で編集可能な HTML 文書の構成は，編集対象の全てを textarea 要素とすることである．textarea，input 要素はデータの閲覧／編集する機能を完全に備える．div，td 要素は負荷が軽い一方で，閲覧はできてもユーザが動的に編集する機能は備えていない．

さて，アプリケーションでユーザが同時に GUI で内容が編集する要素は 1 つである．そ

⋆8 HTML 文書内で利用可能な要素の一つ．自身には内容と属性を持つが，要素は持たない．
⋆9 ブラウザで動くプログラム
⋆10 HTML 文書を構成する要素が持つパラメータの一つ．一般にはその要素の性質を示すキーワードを保持するために使う．キーワードは複数個もつ事ができる．

こで，通常は div，td 要素としておき，ユーザアクションによって編集対象として選ばれた要素のみを動的に input，textarea 要素に変更する．こうすることで表示 (かつ非編集状態の) 対象は軽量な要素で，編集対象の要素のみリッチな要素で構成できる．

ただし，ユーザ快適性実現のために td，div 要素で input，textarea 要素を代替するには「編集」機能という差分を加味し，十分短い時間で埋め合わせる必要がある．

3.2 工夫 2：データを class 属性値として持たせる

class 属性値は要素のパラメータの一つに過ぎず，画面に直接表示されるものでもない．そのため，要素数の増加はレンダリング時間に対する影響が大きいが，class 属性値の影響は小さい．そこで，同様のデータを持つにしても HTML の要素の class 属性値としてデータを保持すれば，データ量の増加に対するブラウザのレンダリング負荷上昇を抑える事ができる．通常は要素で表現するデータを class 属性値とするものであるから，高速化効果とは別に，それらを必要に応じてその形式を要素と相互変換することに要する時間が十分速い必要がある．

3.3 工夫 3：特別な ID 命名規則によって ID アクセスする

インタラクティブな動作では，アクセス (読み書き) する対象の要素をどのような方法で特定するかによって，ユーザの快適性は左右される．複数の要素にアクセスする場合，特段の工夫を施さないならば，class 属性値をチェックしつつ DOM ツリーを構成する要素を順に辿る．そのため，要素へのアクセスは虱潰しかそれに近いものとなり，取扱うデータ量が増加に対応できない．そこで，ID 名に工夫を施し，それに対応するアクセス方法をとる．

通常，ID アクセスは特定の 1 つの要素へアクセスするために使われる．ID アクセスを複数要素のアクセスに使うために，ID 名を，要素の (開発者の定義する) 種類と連番，および集約階層構造を表現するものとする．これにより frameElement1, frameElement2, frameElement3... という様に (null がかえってくるまで) 種類名と連番でアクセスして行く過程で，所定の要素に短時間でアクセスできる．

3.4 工夫 4：表示が必要になるまでは非表示状態を維持する

display プロパティ[⋆1](を none にすること) によって，要素を非表示状態にして表示対象から外す事でブラウザの負荷を下げる．ブラウザ上で取り扱うデータ量がある程度に達すると，その全体を 1 画面で表示しきる事はできず，スクロールなどの操作で所望の位置へ移動することも難しくなる．適当な部分的表示 (あるいは非表示) によってブラウザの負荷を下げる事は，ユーザビリティを下げずに，Web アプリケーションの応答性を向上させるための技術的対策となる．この方針では，表示／非表示の切換えそのものの時間が待ちを感じない程度に十分速い必要がある．

3.5 工夫 5：固定幅 table レイアウトを採用する

図 1 に示すように，我々が実際に開発した Web アプリケーションではオブジェクト指向で構造化されたのデータ[8] を表示し，それらを編集する．対象データは，各項目の画面上での幅はおよそ一定を期待してよいために，固定幅で table レイアウトする許容性が高い．table 要素は tr 要素が所定数の td 要素を保持する構造であるため，(tr 要素が td 要素分のデータを纏めて保持することで)class 属性値によるデータ保持が適用し易い．

3.6 実装モデルの提案

以上，データサイズで数 MB，HTML 文書で数万要素，テキストを主要する Web アプリケーションの開発に際し，ユーザ操作に対するブラウザ応答性を向上するために，

(1) 軽量な要素を用いておき編集時のみ内容を変更可能な要素に変換する
(2) データを class 属性値として持たせておき，必要な場合だけ要素へと変換する
(3) 特別な ID 命名規則によって ID アクセスする
(4) 非表示状態を Default にしておき，必要な場合だけ表示状態へと変換する
(5) 固定幅 table レイアウトを採用する

を複合的に用いる事を実装モデルとして提案する．

4. 5 つの工夫のテスト実装の評価

4.1 工夫を評価するためのテスト条件

評価のための実験の標準条件は以下のような HTML 文書に設定した．

(1) div 要素 10000 個とする．

⋆1 HTML の要素には，その要素の画面上の表現等を設定するためのパラメータを持ち，これをプロパティと言う．display プロパティは要素の表示の有無等を設定するパラメータである．

(2)　要素の内容は 5 文字とする．
(3)　class 属性と id の記述はソースに含まない．
(4)　display プロパティはデフォルト値とする．すなわち全要素は常に表示状態とする．
各工夫の効果のテストでは，工夫に対応する部分のみを基準から外し，標準条件の実装の処理時間と比較した．テスト環境は Windows XP，Core 2 Duo，2GB メモリ，Firefox 3.6.7 であり，JavaScript で時間を計測した．データ量は 1.8MB 程度となり，100Mbps の帯域ではこのデータを 150 ミリ秒程度で送受信できる．

4.2　工夫 1：編集時のみ内容を変更可能な要素にする効果

表 2　要素種類ごとのレンダリング時間 [ミリ秒]
Table 2　Rendering time of each Element kind

要素種類	div	input	textarea
レンダリング時間	207	14692	3041287
各要素に編集可能化機能を付与する	115	0	0
閲覧専用と編集可能の状態切換をする	4	0	0

テスト結果を**表 2** に示す．表 2 は div，input，textarea の各要素種類ごとのレンダリング時間の違いと，div 要素には備わっていない GUI での編集機能の差分を埋めるための時間を調べたものである．表 2 に示すように，同数の要素を単純にレンダリングする場合でも，要素の種類によってその時間が異なる．

しかし，編集する場面では，その差分となる (div 要素には備わっていない) 編集機能を埋める必要があり，その時間を加味して高速化効果を考える必要がある．表 2 に示す「各要素に編集可能化機能を付与」とは編集機能の差分を埋めるための機能を div 要素に追加するための時間であり，Web アプリケーションの利用時に一度だけこれを費やす必要がある．

また，「閲覧専用と編集可能の状態切換」とは，マウスクリックやマウスオーバーなどに応じて，インタラクティブに div 要素を編集状態に変更するために要する時間であり，ユーザが編集を行おうとする度に必要である．

したがって，ブラウザがそのデータを表示する場面において textarea 要素では 3041287 ミリ秒に対して，div 要素であれば 207 ミリ秒に差分を埋めるための 115 ミリ秒を足した 322 ミリ秒を要する．標準条件において，初期レンダリング時間に関しては，textarea との比では 9329 倍の高速化効果が，input との比では 45 倍の高速化効果がある．ただし，編集を行うにあたり textarea 要素は 0 ミリ秒で内容を編集を開始できるが，div 要素では毎回 4 ミリ秒を待つこととなる．

4.3　工夫 2：データを class 属性値として持たせる効果

HTML 文書では通常は**図 2** のようにデータを持つ．

```
<div>data0</div><div>data1</div><div>data2</div><div>data3</div>
<div>data4</div><div>data5</div><div>data6</div><div>data7</div>
<div>data8</div><div>data9</div>
```

図 2　データを要素で持つ
Fig. 2　Data in the element's content

データを class 属性値で持つ場合には**図 3** のようにデータを持たせる．

```
<div class="data0 data1 data2 data3 data4
            data5 data6 data7 data8 data9"></div>
```

図 3　データを class 属性で持つ
Fig. 3　Data in the class attribute

テスト結果を**表 3** に示す．表 3 は，標準条件における 10 個の要素の内容を，1 要素の class 属性値として持つ場合のレンダリング時間，及び図 2 と図 3 のような状態を相互変換するのに要する時間を調べたものである．

表 3　データを class 属性値に持たせるか否かによるレンダリング時間の違い [単位はミリ秒]
Table 3　Rendering time when data is given to class attribute value

データ格納先	要素の内容	class 属性値
レンダリング時間	207	17.2
各要素に編集可能化機能を付与する	115	(1 要素あたり)0.115
class 属性値と要素の相互変換機能を付与する	0	11.3
閲覧専用と編集可能の状態切換をする	4	4
class 属性値と要素の相互変換をする	0	(1class 属性値から 10 要素で)0.1

同等のデータであっても class 属性値でデータを持つ場合には要素数が 10%に削減されるため，表 3 に示すように，レンダリング時間も 207 ミリ秒から 17.2 ミリ秒へとおよそ 10%程度に短縮される．しかし，標準条件との差分を埋めるために，最初に一度だけ class 属性値と要素の相互変換機能を付与する必要があり，11.2 ミリ秒を要する．また，class 属性値として保持されるデータを表示／編集するためには，毎回 0.1 ミリ秒を要する．

なお，class 属性値と要素の相互変換にはさらに追加コストが掛かる．class 要素値を要素に変換して表示や編集を行うため，変換して「新たに」作成される要素に対しては，毎回，編集可能化機能の付与が必要だからである．そのため，変換毎に動的に編集可能化機能の付与する時間を要する．標準条件では最初にまとめて 115 ミリ秒を要するが，class 属性値でデータを持つ場合には Web アプリケーション利用開始時の 115 ミリ秒が掛からないかわりに，変換発生毎に 1 要素 (10 データ) あたり 0.115 ミリ秒の追加コストを要する．

つまり，1 つの class 属性値につき 10 の要素を押し込め，それら 100 個単位で展開して 1000 要素として表示するならば，展開機能の付与に 10 ミリ秒程度の追加，展開毎に 20〜30 ミリ秒を要するものの，要素数削減効果により初期レンダリング時間は 207 から 17.2 ミリ秒へ短縮される．class 属性から要素への展開時間をトレードオフに，初期レンダリングには 11 倍の高速化効果がある．

4.4 工夫 3：特別な ID 命名規則によって ID アクセスする効果

表 4 ID アクセスと class 属性によるアクセス時間の違い [ミリ秒]
Table 4 Access time by ID name or class attribute value

アクセスのセレクタ	class(素朴な実装)	class(jQuery)	特別な ID 名
アクセス時間	7.005	1.714	0.0006

表 4 にテスト結果を示す．表 4 は素朴な実装で class 属性値に基づくアクセス，jQuery で class 属性セレクタに基づくアクセス，特別な ID 名に基づくアクセスそれぞれが要する時間の違いを示すものである．表 4 では 10000 要素から所定の 1 要素へアクセスする時間を測定した．ただし，class 属性値によるセレクタの場合の処理時間は (そもそも虱潰し方式であるため) アクセス対象要素の数に依存しないが，id の場合には要素数倍の時間が掛かる．例えば，class 属性値によるセレクタでは 10000 個中 100 個でも 7.005 ミリ秒だが，ID では 0.06 ミリ秒となる．

ID アクセスでは終了を判定するためのアクセスが 1 度必要であるから，1 要素へのアクセスには 0.0012 ミリ秒，n 要素へのアクセスには 0.0006(n+1) ミリ秒を要する．表 4 に示すように，jQuery ライブラリとの比較では，10000 万要素のうちの 1 要素にアクセスするのであれば，1.714 ミリ秒に対して 0.0012 ミリ秒であるから 1428 倍，10 要素のアクセスであれば 1.714 ミリ秒に対して 0.0066 ミリ秒であるから 260 倍の高速化効果がある．

4.5 工夫 4：表示が必要になるまでは非表示のままにする効果

表 5 可視／不可視によるレンダリング時間 [ミリ秒]
Table 5 Rendering time of each display property

可視／不可視	可視	不可視
レンダリング時間	207	79
各要素に編集可能化機能を付与する	115	115
可視／不可視の状態切換え機能を付与する	0	1
閲覧専用と編集可能の状態切換をする	4	4
可視／不可視の状態切換をする	0	1

※ 1000 要素単位で可視／不可視の制御領域を分割．

表 5 にテスト結果を示す．表 5 は標準条件と可視／不可視の切換え機能を付加した場合のブラウザの処理時間を示している．表 5 に示すように，標準条件の常に可視の場合には，ブラウザが最初にその表示を終えるまでに 322 ミリ秒，動的な編集可能状態への切換え毎に 4 ミリ秒掛かる．一方，通常は不可視で必要に応じて，要素を部分的に可視状態へ切り替えるの場合には，レンダリング時間の部分は 207 ミリ秒から 79 ミリ秒へと短縮される．

しかし，標準条件との差分となる可視／不可視を切り替える機能の追加が必要となる．この機能の付与を Web アプリケーション利用時の最初に一度だけ行う必要があり，これに 1 ミリ秒を要する．また，利用中に動的に可視／不可視を切り替える処理を行わねばならず，これには毎回 1 ミリ秒を要する．ゆえに，Web アプリケーション利用開始時のイニシャルコストに 1 ミリ秒，動的な切換えコストに毎回 1 ミリ秒をトレードオフにして，可視／不可視の切換えには Web アプリケーション利用開始時のレンダリングについて 4 倍の高速化効果がある．

4.6 工夫 5：固定幅 table レイアウトにする効果

表 6 にテスト結果を示す．表 6 は table 幅を自動調整 (auto) と固定 (fixed) と変えて，そ

表 6 table 要素のレンダリング時間 [ミリ秒]
Table 6 Rendering time of table element

table レイアウト	auto*	fixed
平均文字数 10 のレンダリング時間	289	274
平均文字数 100 のレンダリング時間	454	414
平均文字数 1000 のレンダリング時間	2211	1820

※基準では div 要素が 10000 であるが，table では td が 10000 要素.
※ auto のレンダリング時間は最悪値.

れぞれ td 要素の内容を平均 10 文字，100 文字，1000 文字でのレンダリング時間を調べたものである．表 6 では，標準条件では div 要素が 10000 であるが，table では td が 10000 要素とし，表組表現におけるセル幅の固定とセル内容の文字数がレンダリング時間に及ぼす影響を表している．セルの内容の文字数は 1 文字に始まり，後段に行くに従って一定の増加率で単調に増加するものである．

```
<tr>
  <td>data0</td><td>data1</td><td>data2</td>
  <td>data3</td><td>data4</td><td>data5</td>
</tr>
```

図 4 td 要素自身がデータを持つ
Fig. 4 Data in each td element

```
<tr class="data0 data1 data2 data3 data4 data5"></tr>
```

図 5 tr 要素が纏めてデータを持つ
Fig. 5 Data set in tr element

table レイアウトでは，tr 要素の class 属性値に，その子要素である td 要素の内容を置くいう対応がとりやすい．tr 要素の下にくる td 要素の数と順序が一定だからである．具体的には**図 4** のような状態を，**図 5** のような形にすることで，データを単純に class 属性値として置くことができる．

table 要素におけるセル幅は原則として auto であるべきとされるが，印刷とのギャップを小さくする，あるいは画面サイズの固定された端末への対応を考慮するなどの場合，固定幅 table レイアウトには特段のパフォーマンス低下はない．むしろ柔軟幅 table レイアウトを基準に比較すると，最悪値との比では 1.06〜1.22 倍の高速化効果がある．

5. 結果の評価と考察

5.1 5 つの工夫の複合的な適用とその評価

5 つの工夫の応答性は要素数によって高速化効果が変わってくるので，あくまで目安として述べたい．本研究で述べたアイデアを全く使わない素朴単純な実装と比較すると，その具体的な高速化効果は以下の式で求められる．

[45〜9329] * 11.3 * 4 * [1.06〜1.22] = 2156〜514438

複合的に実装指針に従った場合には，全体として，50 万倍〜2000 倍の高速化 (あるいは同時に扱えるデータ量の増量) 効果がある．

インタラクティブな要素アクセスでは，jQuery ライブラリのセレクタとの比較で，数百倍から数千倍の高速化効果がある．一方で，編集操作や選択表示には毎回 4 ミリ秒や 20〜30 ミリ秒の待ち時間が発生するというトレードオフがある．

Web アプリケーション全体としては，先に述べたアイデアを適用する事で，素朴な実装に対して追加的に必要となる関数の割当に 200 ミリ秒，表示選択に 20〜30 ミリ秒，編集可能化に 4 ミリ秒を要するものとなり，初期レンダリングが 50 万〜2000 倍高速化される (数百ミリ秒のレンダリング時間で扱える要素数が 50 万〜2000 倍になると言い換えても良い).

5.2 我々の実装した Web アプリケーションに対するアンケートから見た応答性の評価

表 7 我々の Web プリケーションのユーザデータ [個]
Table 7 User data of our Web Application

対象世界	ファイルサイズ	div	table	tr	td
ATM の利用シミュレーション	590.08KB	653	107	599	2527
魚釣りのシミュレーション	495.98KB	537	99	510	2080
バッターとピッチャーにおける打撃シミュレーション	937.08KB	1005	123	906	3722
ファーストフード店の店員と客とのやり取りの流れ	1.15MB	1255	114	1140	4786

表 7 に我々のアイデアを実装した Web アプリケーションのユーザデータを示す．いずれ

も単純素朴な実装では数万要素となるところが数千要素，1MB 程度のサイズとなり，大きなものでも 2 万要素，3.2MB 程度であった．

およそ 50 名の大学 3 年生に利用してもらい，Web アプリケーションの応答性の善し悪しについてアンケートを取ったところ，500KB などデータ量が少ないユーザは「快適」「Web アプリケーションとしては快適」と，1MB 前後までのデータ量が達したユーザは「実用にならないほどではない」「鈍重」と回答した．なお,「実用にならないほどではない」「鈍重」と回答したものの 6 割がセーブの遅さを自由記述で指摘していたが．これはプログラムの一部の不備によるものであった．そのため，トレードオフの待ち時間は殆ど気にならず，高速化効果が体感できたと言える．

5.3 提案した実装モデルの有効利用

table レイアウトは，Excel や Rails アプリのような，表組でそのデータを閲覧・編集する様な Web アプリケーションと親和性が高い．table では tr 要素への class 属性値圧縮の利用が容易で，我々の提案する Web アプリケーション実装指針が適用し易い．

この実装モデルは HTML，CSS，JavaScript というブラウザが素の状態で扱える基本的な技術のみで実装できる．このシンプルさ故に，基本的なブラウザを備えるデバイスに広く適用可能性がある．携帯端末やスマートフォン，iPad などは処理能力の面で PC に劣り，レンダリング領域はデバイス毎に固定である．ゆえに．性能面で劣るデバイスでは十分に実行できなかった Web アプリケーションに適用する事で，その快適性向上の結果として実用性を獲得できる可能性がある．

5.4 本実装モデルの利用に必要技術

- HTML 文書が扱えること
- DOM 操作と動的スタイル適用が出来る程度に JavaScript が使えること
- デザイン性に応じた CSS を使えること
- HTML 文書をサーバ／クライアント間でやり取りを実装できること

図 6 本実装モデルの利用に必要な技術
Fig. 6 Skills needed to use our implementation model

本実装モデルに必要な最低限度の知識は**図 6** に示す 4 つである．これらは一度でも Web アプリケーションを開発を試みた事があれば，その殆どすべてを習得しているであろう技術である．すなわち，多少でも Web アプリケーションを開発した人であれば，我々の実装モデルを組み入れるだけで，これまでの習慣を大きくかえることなく，実現可能な Web アプリケーションの領域が広がる．これまで継続してきたやり方に 数個のノウハウを加えるという印象で導入できるところは，実際に有用性を発揮するための重要な一要因である．したがって，第 3 章で提案した実装モデル実体，上記の必要技術を含めて実装モデルとしたい．

6. 結論と今後の発展的な問題への提案

ファイルサイズで数 MB，HTML 文書で 1〜10 万要素，テキストを主要する Web アプリケーションの実装モデルを提案した．実際に，当該の実装モデルを適用してグラフィカルな Web アプリケーションを実装してユーザの評価を得た．簡易な技術のみを前提とし，一部にトレードオフを伴うパフォーマンス向上であるが，ユーザ評価にとくに悪いところは無く，十分な実用性があると実証できた．

今後は，スマートフォンや iPad など，画面サイズが一定で，より非力なハードウェアに対する適用性を検討したい．

参 考 文 献

1) http://www.apple.com/jp/ipad/
2) http://www.prototypejs.org/
3) http://jquery.com/
4) http://rubyonrails.org/
5) http://download.microsoft.com/download/e/c/d/ecd2e194-8560-4057-b56d-fd84680dad0c/PerfTestGuideJPN.pdf
6) 片野克紀，池田陽祐，畠山正行，オブジェクト指向設計記述言語 ODDJ の汎化階層構成への拡張とその実装，第 159 回 SE 研究会報告，2008-SE-159, pp.57-64, (2008).
7) 大木幹夫，片野克紀，三塚恵嗣，沼崎隼一，涌井智寛，加藤木和夫，池田陽祐，畠山正行，三言語独立のオブジェクト指向記述言語 OOJ の実装と検証，第 163 回 SE 研究会報告，2009-SE-163, pp.49-56, (2009).
8) 池田陽祐，畠山正行，三塚恵嗣，加藤木和夫，大木幹生，オブジェクト指向分析記述言語 OONJ の記述例を用いた簡潔な表現技法の開発，第 78 回 MPS 研究会報告，MPS78-17, (2010).