修士論文

Webページにおけるテキストレイアウトの国際化に関する研究と実装

Research and Implementation of Text Layout Internationalization on Web Pages

青山学院大学大学院 理工学研究科 理工学専攻 知能情報コース

Course of Information and Knowledge Engineering, Graduate School of Science and Engineering, Aoyama Gakuin University

塩澤 元

Shiozawa, Hajime

目次

第1章　序論

1.1　背景

近年，Web ページとして文書を公開することが広く一般化している。2008 年には Web ページの総数が 1 兆ページを超えたとされ [1]，2010 年現在ではさらに多くの Web ページが存在していると考えられる。また，以前と比べて様々な種類の Web ページが登場している。特に，従来紙の印刷物が担っていた役割を Web ページが担う場面が増えている。青空文庫[1] や Project Gutenberg[2] では，過去の出版物を Web ページで公開している。Wikipedia のような百科事典，Web 上にあるコンピュータのマニュアルや新聞のニュースページもその一種である。加えて，2010 年は電子書籍元年とされ，今後さらにこのような Web ページの数と種類は増えると考えられる。また，この流れを後押しするようにデバイスの進化も進んでおり，iPad やスマートフォンの登場によって Web ページを見る環境が飛躍的に向上している。

しかし，Web ページは紙と異なり，様々な表記体系のテキストの全てを表現することができない。また同時に，書籍や書類などのような高品位の組版を必要とする文書を作成することができない。そのため，ある表記体系特有の組版を持ったり，高品質な文書を作成する際には LaTeX，Microsoft Office Word，Adobe Indesign などの文書作成に特化したソフトウェアを利用することが多い。しかし，HTML で文書を作成することは，データの構造化，ハイパーリンクの利用，画像や動画の埋め込みの手軽さなどの多くの利点がある。加えて，Web にはすでにインターネットを通じ世界中に公開するための枠組みが提供されており，文書の流通という点で幅広く用いられている。さらに，Web ブラウザはソフトウェアとして普及しており，コンピュータをはじめ，スマートフォンなど多くのデバイスに搭載されている。このような Web の利点を生かすためには，Web 上で世界の様々な表記体系が持つすべてのテキストレイアウトを表現できるようにし，なおかつ紙媒体の出版物と同じように高品位な組版が可能なものを提供することが課題である。

[1] http://www.aozora.gr.jp/
[2] http://www.gutenberg.org/

1.2 目的

以上のような背景から，本研究はWebページのテキストレイアウトの国際化を目的とし，特に日本語とWebに親和性の高い，『ルビ』と『縦書きを可能にする書字方向選択』の二つに注目している。本研究において『テキストレイアウト』とは，『従来の伝統的な印刷文化における組版』と『Webページ特有のレイアウト』の両方を含んだ，テキストの配置のことを指す。この目的を達成するため，実際にWebブラウザ上にこれら二つのレイアウトを表示する機能を実装する。まず，現在のWebにおける具体的な問題点を挙げ，それらの問題点を解決するように実装を行う。特に，既存のブラウザに機能を付け加える形で実装することで，すでにあるWeb技術の資産を生かすようにする。また，実装を行う課程で得た知見について述べ，それらをWeb技術の仕様策定などに生かす。本研究では，実装対象ブラウザとしてオープンソースソフトウェアのMozilla Firefox[3]を採用した。

ルビ表示の実装ではまずルビのためのレイアウトモデルを提案し，それを基準として実装する。また，他のブラウザで実装が遅れているような詳細なスタイル指定についても対応することを目指す。加えて，多くの非整合なマークアップを考慮し，適切な表示ができるような対応を目標とする。書字方向選択の実装ではまず実装のレイヤを定義し，各部分で必要な実装を行う。特に日本語の書字方向である右か左の縦書きのみではなく，考えられる書字方向全てを実装することを目的とする。そして，単に紙媒体と同じような書字方向を可能にするレイアウトを実装するだけでなく，Webページに特有のレイアウトを考慮することも目標とする。

1.3 本論文の構成

本論文は，本章を含め全6章で構成される。第1章では研究の背景と目的を述べた。第2章では，文字と組版について一般的な解説とそれらに関するコンピュータの技術について述べる。第3章では，Webで第2章で述べた項目をWebでどのように指定するかを解説する。また，現時点でそれらのWebの技術がどのような課題を持っているかについて述べる。第4章ではそれらの課題を解決するための実装の詳細について述べる。第5章では本実装によって可能になったレイアウトを示し，その評価について述べる。最後に第6章では本研究による成果と，今後の課題について述べる。

[3] http://mozilla.jp/firefox/

第2章　文字と組版の国際化技術

本章ではテキストの構成要素となる文字と組版，またそれらをコンピュータで扱うための国際化技術について解説する。

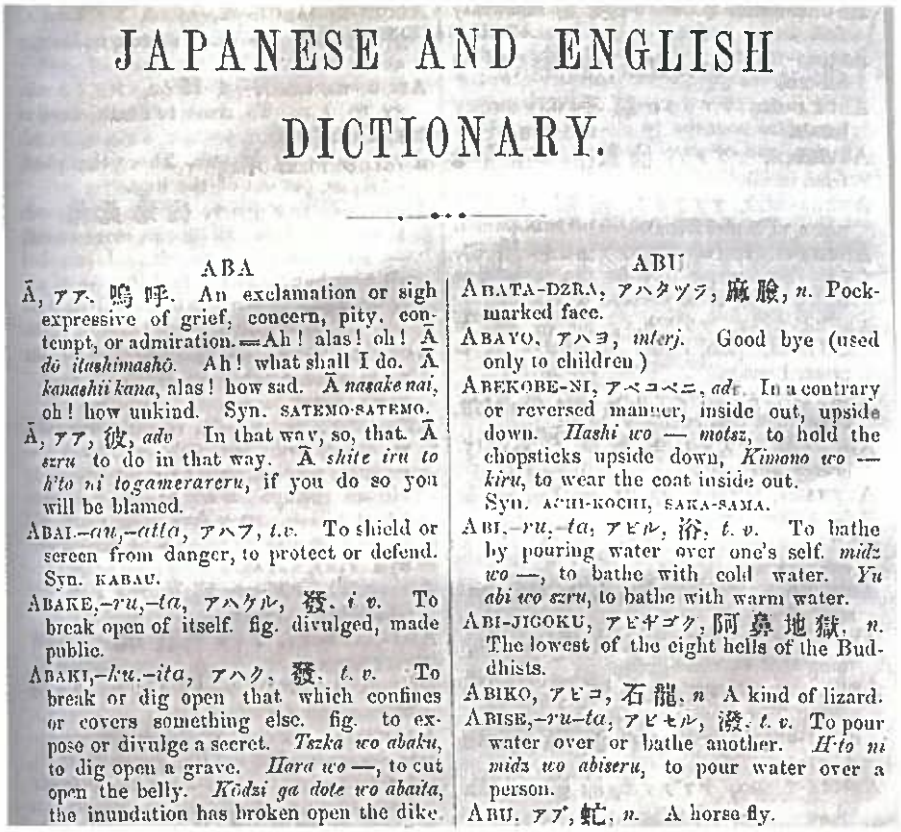
2.1　テキストの国際化とコンピュータ

2.1.1　コンピュータ上のテキスト

テキストは文字によって表現される情報であり，それらは何かの媒体に記録されている。昔はテキストを綴るための記録媒体としていろいろなものを用いていた。骨や亀の甲の上，石や粘土の上，貝殻の上，動物の皮の上，木の皮の上などである [2]。その後，紙が発明され，その製法が世界中に広まり，これらの記録媒体は全て過去のものとなった。今では紙がテキストの記録媒体としての地位を確立している。しかし，20 世紀末よりテキストを電子情報として記録し，コンピュータを使ってそれを表現することが可能になった。そのため，コンピュータでテキストを表現するために様々な技術が開発されている。例えば，紙の印刷出版が独壇場であった書籍でさえも，電子書籍として利用できるようになることが多くの人に期待されている [3]。

2.1.2　多言語テキスト

記録媒体が変化するにつれ，テキストを気軽に利用することが可能になっていった。その結果，テキスト内に異なる言語が含まれるような『多言語テキスト』が登場した。例えば，図 2.1 は明治時代の日英辞書である。このように近代では，テキストはすでに単一の言語のみによって書かれるものではなくなった。このような多言語の交流によって新たな文化が生み出されている。例えば日本語のテキストは他の言語の影響を顕著に見ることができる。日本語の書き言葉自体，完全にオリジナルのものではなく，中国語を源流として独自に発展させた（漢字，ひらがな，カタカナ）言語であることが知られている [4]。また，日本語においては明治時代まで，縦書きが一般的だったが，西欧文化を取り入れるなかで自然と横書き



JAPANESE AND ENGLISH

DICTIONARY.

ABA

Ā, アア. 嗚呼. An exclamation or sigh expressive of grief, concern, pity, contempt, or admiration.—Ah! alas! oh! *Ā dō itashimashō.* Ah! what shall I do. *Ā kanashii kana,* alas! how sad. *Ā nasake nai,* oh! how unkind. Syn. SATEMO-SATEMO.

Ā, アア, 彼, *adv* In that way, so, that. *Ā szru* to do in that way. *Ā shite iru to hito ni togamerareru,* if you do so you will be blamed.

ABAI,–*au*,–*atta*, アハフ, *t.v.* To shield or screen from danger, to protect or defend. Syn. KABAU.

ABAKE,–*ru*,–*ta*, アハケル, 發, *i.v.* To break open of itself. fig. divulged, made public.

ABAKI,–*ku*.–*ita*, アハク, 發, *t.v.* To break or dig open that which confines or covers something else. fig. to expose or divulge a secret. *Tszka wo abaku,* to dig open a grave. *Hara wo —,* to cut open the belly. *Kōdzi ga dote wo abaita,* the inundation has broken open the dike.

ABU

ABATA-DZRA, アハタツラ, 麻臉, *n.* Pock-marked face.

ABAYO, アハヨ, *interj.* Good bye (used only to children)

ABEKOBE-NI, アベコベニ, *adv.* In a contrary or reversed manner, inside out, upside down. *Hashi wo — motsz,* to hold the chopsticks upside down, *Kimono wo — kiru,* to wear the coat inside out. Syn. ACHI-KOCHI, SAKA-SAMA.

ABI,–*ru*,–*ta*, アビル, 浴, *t.v.* To bathe by pouring water over one's self. *midz wo —,* to bathe with cold water. *Yu abi wo szru,* to bathe with warm water.

ABI-JIGOKU, アビヂゴク, 阿鼻地獄, *n.* The lowest of the eight hells of the Buddhists.

ABIKO, アビコ, 石龍, *n* A kind of lizard.

ABISE,–*ru*–*ta*, アビセル, 潑, *t.v.* To pour water over or bathe another. *H:to ni midz wo abiseru,* to pour water over a person.

ABU, アブ, 虻, *n.* A horse fly.

図 2.1: James C. Hepburn による『和英語林集成』(出展：文献 [6], p.154.)

を認めるようになった。その後，日本政府公用文の作成に取り入れられるなどして [5]，日本語の横書きはすでに公式な表記体系のひとつとなっている。

また，インターネットと Web が発展した現代社会において，この多言語テキストの量と重要性はさらに増していくと考えられる。インターネットを用いれば，物理的な距離の制約をほとんど受けずにテキストを交換することができる。そして，Web では世界中で同一のシステムを用いるため自然と多言語テキストが誕生する。例えば図 2.2 のように，実際に Twitter[1] というサービスでは世界中の人々がそれぞれの言語を用いて言葉を書き，一つのページでそれらを表示することができる。また，デジタル化されたデータは半永久的に保存できるため，過去のテキストのデジタル化などが積極的に行われている。その際に，世界中の言語をテキストとしてデジタル化できることは文化の継承という点において非常に重要である。このような理由から，多言語テキストの量と必要性は増大していくと考えられる。

[1] http://twitter.com/

図 2.2: 様々な言語で発言される人々のつぶやき (Twitter)

2.2 文字とその技術

2.2.1 文字と文字コード

ここからはテキストを構成する文字と，コンピュータ上でそれを扱う技術について解説する。文字はテキストを表現する際に意味を持つ最小の構成要素とされ，概念的にその表現形式（見た目）とは別のものとして考えられている [7]。例えば，図 2.3 に示すように，“A” という表現は『ラテンアルファベットの大文字 A』という人間が持つ抽象的な概念を表現するひとつの形でしかない。

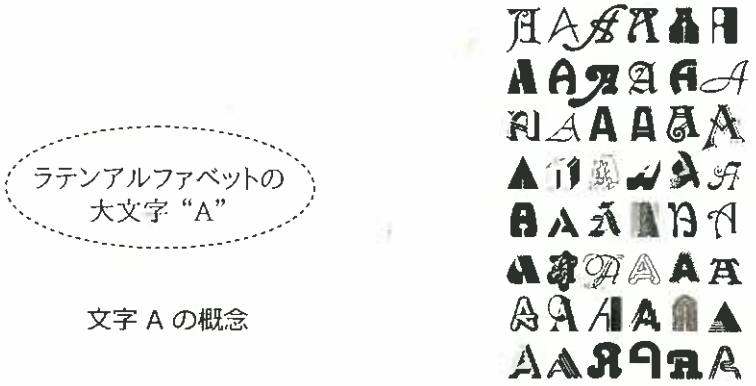


図 2.3: 文字の概念と文字の様々な表現 (右図出展：文献 [8], p.234.)

文字コードとはコンピュータ上で文字を扱うために利用される技術である。文字コードは，対象とする文字群を定義する『符号化文字集合』と，文字をコンピュータで扱うデータ表現へと変換するための『文字符号化変換方式』で構成される。

符号化文字集合では，抽象的な文字ひとつひとつに順序を与え，非負整数値の集合への対応付けを行う。これは，文字を抽象的な存在からより具体的な存在に還元することを目的としている。この中で文字に対応づけられた整数値はコードポイントと呼ばれ，コードポイントを持つ文字を符号化文字と呼ぶ。

文字符号化変換方式では，符号化文字集合で文字に与えられたコードポイントを，コンピュータで表現できるバイトデータなどに対応させるための変換方法を規定する。ここで変換先のデータの長さは一定とは限らない。同じ長さのデータへ変換する場合には，『固定長』符号化方式異なる長さのデータへ変換する場合には，『可変長』符号化方式と呼ぶ。また，同一の規格でこの二つ両方を定義している場合もある。

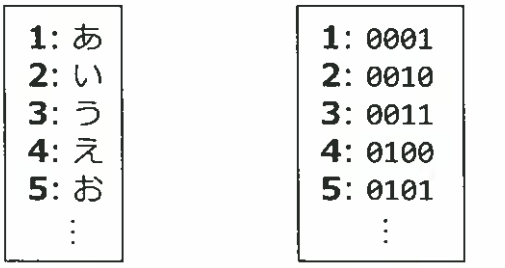


図 2.4: 符号化文字集合と文字符号化変換方式

文字コードの歴史

ここからは，文字コードの歴史と国際化について述べる。文字の符号化という概念は古くから存在していた。人類史で初めて，符号化通信が行われたのは紀元前 350 年頃のギリシャであると言われている [9]。その後，0 と 1 のバイナリデータへの符号化が主になったのは，電気の発見のためである。Alfred Vail はモールス電信機のための文字符号を考案し，日本では小田又蔵と勝海舟がオランダ人技術者らとともに和文モールス符号を作成した [10]。これらの技術が現代の文字コードの基礎となった。

その後，世界では様々な文字コードが考え出されている。現在の日本で文字コードとして幅広く知られているのは ASCII [11]，Shift_JIS [12]，日本語 EUC [13] などであるが，以前は企業独自の文字コードが乱立していた [10]。また，日本以外では中国の GB2312 [14]，インドの IS13194 [15]，アラビア諸国の ASMO449+ (ISO/IEC 8859-6) [16]，キリル文字のための KOI8 [17] や ISO10754 [18] など，言語ごとにそれぞれ文字コードが存在し，同じ言語内でも複数の文字コードが存在するような状態であった。

このように，全て言語に対してそれぞれの文字コードを考案することで，世界の様々なテキストをコンピュータで扱うことができる。しかし，これでは同一テキスト内で異なる言語の文字を扱うことができないうえ，対応していない文字コードのテキストは処理することができない。このような問題を解決するために，世界中の文字を一つの文字コードで扱う国際文字符号化集合が必要となった。

国際文字符号化集合と Unicode

上述したような問題を解決し文字コードの乱立を防ぐために，国際標準化機構 (ISO) は漢字を含む世界の文字を統一的に扱える国際文字符号化集合の作成を決定した。1990 年に既存

の様々な文字コード規格との整合性と拡張性を考慮した4オクテット(32ビット)符号方式・DIS10646第一版を策定した[19]。しかしほぼ同時期に，Xerox社のJoseph D. Beckerを中心とするグループがUnicodeと呼ばれる多言語をサポートする2オクテットバイト(16ビット)符号の文字符号を開発していた[20]。これによってISOのDIS10646第一版は標準化一歩手前の段階で，同じように多言語をサポートするUnicodeとの統合を要求された[21]。その結果，ほぼUnicodeのデザインの採用する形でISO/IEC 10646が採択され[10, 22]，Unicodeが事実上の国際文字符号化集合という形になった。

2010年現在，最新のUnicodeのバージョンは6.0あり，109,449文字を収録している[7]。前述のように初期のUnicodeは16ビット文字集合であったがバージョン2.0以降より，21ビットの文字集合として規定されている。そこで，16ビットで表現しきれなかった文字は未定義領域を利用したサロゲートペアという方式で表現している。Unicodeで定義される文字集合の代表的な符号化方式は，UTF-8 [23]とUTF-16 [24]である。UTF-8は文字を8ビット，1〜6バイトの符号へ変換し，UTF-16は16ビットまたは32ビットの符号へ変換する。UTF-8において，1バイトで表現される文字はASCIIと完全に互換であるので，一般的なテキストの符号化方式としてUTF-8が多く使われている。一方，UTF-16はOSやプログラミング言語の内部エンコードとして使われることが多い。

2.2.2 グリフとフォント

すでに図2.3で示したように，文字は抽象的な概念であり，実際の表現形はグリフ（字形）と呼ばれる。フォントとはコンピュータ上でグリフを表現するための書体データの集合である。もともとは同じ書体デザインの活字の一揃(そろ)いを指す言葉であり，これと区別するためにデジタルフォントと呼ばれることもあるが，本論文ではフォントと表記する。文字コードはコンピュータで文字を扱うために用いられ，フォントはコンピュータでグリフを表現するために利用される。コンピュータ上で人間が視覚的に文字を認識するためには，フォントの技術が必要不可欠である。

フォントフォーマット

フォントは書体データの集合であるが，そのデータ記憶の形式として『ビットマップフォント』と『アウトラインフォント』の二種類がある。ビットマップフォントでは図2.5のように，グリフはドットのマトリックスで構成され，個々のドットはオンまたはオフに設定し，グリフを表現する。デメリットとしてポイントサイズが限定されること，文字数やマトリッ

クス数の増加によってメモリ使用量が極端に多くなることが挙げられる。しかし，小さなポイントサイズや低解像度の出力装置での相性がよいといわれている。代表的なビットマップフォントとして，Adobe Systems による BDF フォント [25] などがある。

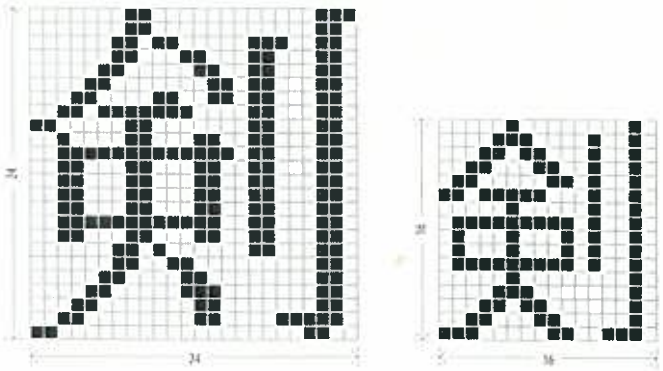


図 2.5: ビットマップフォントのイメージ (出典：文献 [26], p. 374.)

一方，アウトラインフォントではグリフの情報をビットデータではなく，その外形のみを保持する。図 2.6 に示すように，アウトラインフォントではグリフの形に関する情報を線分，円弧および曲線の連続として数学的に記述し，それを記録している。これによってビットマップフォントとは異なりスケーラブル（拡大・縮小可能）なフォントとなる。

図 2.6: アウトラインフォントのイメージ (出典：文献 [26], p. 376.)

この方式を用いた代表的なフォントは Adobe Systems による PostScript フォント [27], Microsoft と Apple Computer が共同で開発した TrueType フォント [28], また, これら二つを統合した OpenType フォント [29] などが挙げられる。

多言語テキストのためのフォント

コンピュータ上で多言語テキストを実現するためには, 符号化された全ての文字のグリフが, フォントによって表現できるようになることが必要である。なぜなら, たとえ符号化によってコンピュータの内部でその文字を表現できたとしても, フォントがなければその文字を実際に表示することができないからである。

対応フォントなし　　対応フォントあり

図 2.7: チェロキー語版 Wikipedia の表示画面

例えば, 図 2.7 はチェロキー語版の Wikipedia[2] をブラウザで表示したものである。この場合, 図の左側の状態では適切なフォントがないため, 該当するグリフを表示できずにフォントによる文字化けが発生している。しかし, このページ自体は UTF-8 で適切に符号化されているので, チェロキー語を含むフォントである FullAboriginal Unicode Fonts [30] をインストールすることで, 同図の右側のようにテキストを表示することができる。このように, 多言語テキストを扱うためには全ての文字のグリフに対するフォントを用意し, それらを利用できるようにすることが必要である。

そこで, 1995 年に Microsoft は世界の文字のグリフ全てに対応する Arial Unicode MS [31] というフォントセットを開発した。Arial Unicode MS は, Unicode バージョン 2.0 に収録さ

[2] http://chr.wikipedia.org/

れている全ての文字に対するフォントを持っていたが，これ以降のこのフォントセットは開発は継続されていない。Microsoft と Apple Computer のフォント開発者は，統一された単一フォントセットというアイディアは，『メモリ使用量が膨大になること』，『単一のデザインで世界中の文字を扱うことには無理があること』などを理由に，今後このようなフォントの開発はないと述べている [32]。

これに対する解決策として，個々の文字体系別にフォントを分割して用意しておき，適切なフォントを選択するという方式が考えられる。現在は各コンピュータにフォントデータをおいておくのが普通だが，Web にフォントデータおき，それを使う WOFF [33] という技術も開発されている。これを使えば，Web 上からフォントデータを自動的にダウンロードできるので先ほどのチェロキー語の Wikipedia のような問題は起きない。

この『適切なフォントの選択』は特に漢字において非常に重要である。Unicode では中国，韓国，日本で利用される漢字を統合している。そのため，U+76F4 U+9AA8 U+4EBA という単一の文字列を表示する場合でも，図 2.8 のようにフォントごとに表示されるグリフが異なるからである。そのため，中国語のテキストであれば中国語，日本語のテキストならば日本語というような選択が大事になってくる。

図 2.8: フォントの違いによる CJK 統合漢字のグリフの違い

これに対する解決策として Dürst らは適切なフォント選択と圧縮を行う方式を提案し，実装している [34]。また，Trager は Linux におけるフォントコンフィグデータを XML で構造化することで，これらの問題を解決しようとしている [35]。また，東南アジアで用いられる言語やアラビア語では一対一に文字とグリフが対応しない言語もある。そのため，合成文字や筆記体の文字コードとグリフの対応を記憶しておくフォントコントロールのソフトウェアの開発も進んでおり，今後の発展が期待されている [36]。

2.3 組版

2.3.1 組版の概要

ここでは組版の概要について解説する。組版とは文字や図などのテキストを構成する要素を適切な位置に配置する行為のことを指す。組版によって文字を一定の形式に固定させることができ，それによって初めて人はテキストを読むことができる。組版の目的は単に文字を配置するだけでなく，テキストの誤読を防ぎ，読みやすいように配置することである。例えば一つ一つの文字の間隔を適切にすることは，鮮明さや明快さを向上することになる。読みやすいグリフを選択したり，ひとつの行に対して適切な文字数を設定することは可読性を向上することになる。加えて，組版にはテキストの美しさを向上させる芸術的な要素もある。人はテキスト自身を読む前に，その組版によってテキストの印象が決定している。このような点において組版は，人がテキストを読む際の必要不可欠な要素である。

組版は Gutenberg による鉛合金の鋳造活字を用いた活版印刷の発明によって始まった。日本の組版は，明治時代に本木昌造（しょうぞう）が蝋型電胎法（ろうがた）という母型製作法を習得し，オリジナル金属活字を鋳造することによって始まった。その後，写真植字や電算写植を得て，現在ではコンピュータ上で組版をする DTP (Desktop Publishing) が一般になっている [37]。組版という言葉も活版印刷時代の用語であり，もともとは『版面を組む』ことを意味している。このように，組版を実現するデバイスは時代と共に大きく変化しているが，組版規則自体に大きな変化はない [38]。組版はデバイスによらないものであり，伝統的な組版規則は長い歴史を通して培われた非常に理にかなったものになっている。

組版は表記体系ごとに異なるルールを持っているのが普通である。欧文組版の場合には Oxford Rule [39] や Chicago Rule [40] のような組版規則が有名である。日本語の組版に関しては，日本規格協会が『JIS X 4051: 日本語文書の組版方法』というタイトルで，組版規則の規格化を行っている [41]。しかし，実際には出版社ごとに異なる明文化されれていない規則（ハウスルール）を持っているのが一般的である [42]。そこで，JIS X 4051 では一般的な規則を述べ，細部の項目では処理系定義などとしている。

以下の項では，本研究において取り扱う『ルビ』と『書字方向』という二つの詳細について解説する。

2.3.2 ルビ

ルビとは文章中の任意の文字に対して振り仮名，説明，異なる読み方などを，文章中の文字より小さな文字で付随させるものである。一般的にルビは漢字圏の言語において用いられる。ここでは各言語におけるルビの用いられ方と，それに対する組版規則について説明する。

ルビの役割

日本語のルビの役割　日本語におけるルビの源流は漢字の採用時までさかのぼる。ルビは当初，日本語の文書中の漢字を音読するべきか，訓読するべきかを指定するために用いられた[43]。このような背景から，ルビは漢字に読みを与える手法として発達した。昭和初期まで新聞や書籍などでは，全ての漢字に対してルビを振る『総ルビ』が主流であった。しかし，コストの問題や，ルビを振ることへの否定的な意見[44]があり，漢字の一部のみにルビを振る『パラルビ』が主流になった。また，当用漢字制定の際には『振り仮名は原則として使わない』と告示され[45]，現在は一部の漢字のみにルビをつける『パラルビ』が主流である。

しかし，日本におけるルビは発音記号という役割以上の美術的，意味的性質を保持している[46]。例えば中世の経文では本文の脇にルビを振り，本文を墨書，ルビを朱書し，視覚的な美を表現する役割を担っている。ルビは同様に文学上の表現や美しさを高める役目も果たしている。図2.9はJames Joyceによって書かれた*Finnegans Wake*の柳瀬尚紀による日本語訳『フィネガンズ・ウェイク』の一部である。この作品は多言語の表現方法を用いた英語の文学作品であり，翻訳本である『フィネガンズ・ウェイク』ではルビを効果的に用いて，元の文の表現を生かすように日本語訳を行っている。同図を見ると，単なる読み方の記述のほかにも『ドーユー案出ス嘆奴(アンダ　タンド)あり』のような特殊なルビの記述がある。この部分の原文は“*Thigging thugs were*”であり，thiggingはデンマーク語で『請う・物乞いをする』という意味であり，thungは『強盗・泥棒』の意味である。そのため，原文の直訳は『物乞いをする強盗がいた』である。しかし，原文は“*Thigging thugs were*”の発音と，アイルランド語で“You understand”を意味する“*tuigeann tú*”をかけている。その結果，『案（を）出（す）』と『嘆（く）奴』の二つの単語で『物乞いをする身分の低い者』を表現し，それにルビを用いて強制的に音読みさせることで原文を表現する形をとっている。他にも『ドグラ・マグラ』[3]や，英詩を漢文訳＋ルビという独自の表現方法で翻訳した『鉄道株投機屋連』[4]などが挙げられる。

[3]夢野久作，『ドグラ・マグラ』，角川文庫，1976.
[4]William M. Thackeray，平井呈一訳，『床屋コックスの日記 馬丁粋語録』，1951. に収録

図 2.9: ルビを文学的表現に用いた例 (出典：文献 [49], p. 17.)

また，明治時代には文明開化の流れと共にルビには啓蒙的役割があった。漢語や英語も庶民にとっては新しい言葉だったので，それらには緻密なルビがつけられている [47]。たとえば，当て字である『瑞西』などの国名や『TELEGRAPH』などの外来語に対して用いられた。日本語におけるルビはこのように単なる発音記号としての機能的な側面だけでなく，意味の多重化や異化の作用を持つことで，日本語の多様化という点において重要な役割を果たしている。また，現代のコンピュータ時代においても漢字語圏のみに用いる発音記号としての役割のみでなく，インラインの注釈 (inline annotation) として再評価する動きもある [48]。

中国語のルビの役割　中国語において発音記号を表しはじめたのは 1930 年代である。発音をあらわすための文字体系として，注音符号 (Bopomofo) と呼ばれる文字を制定した。しかし，時代が進むにつれ中国やシンガポールでは注音符号は廃止され，現代では台湾のみで用いられている。現在，中国やシンガポールでは拼音（ピンイン）と呼ばれるラテンアルファベットと声調記号を組み合わせた発音記号が使われている。ここではまずこの注音符号と拼音という二つの文字体系について説明する。

注音符号は，ある音にあう漢字の一部を取り出して利用され，Unicode において U+3100-U+312F の位置に割り当てられている。注音符号の最初の四文字がㄅ，ㄆ，ㄇ，ㄈとなっており，これが b，p，m，f の音に対応するので Bopomofo とも呼ばれる。

拼音は，1958 年に中国が制定したものであり，中国語の発音を示す国際的な標準となっている [50]。ラテンアルファベットとウムラウトなどの声調記号を用いて表されており，日本語のローマ字のような存在である。表 2.1 は注音符号と拼音との対応表である。表 2.2 は，「美」の読み方を注音符号と拼音で表したものである。

表 2.1: 注音符号と拼音の対応表

注音符号	ㄅ	ㄆ	ㄇ	ㄈ	ㄉ	ㄚ	ㄜ	ㄧ	ㄨ	ㄛ	ㄓ	ㄔ	ㄕ	ㄩ	ㄝ
拼音	b	p	m	f	d	a	e	i	u	o	zh	ch	sh	ü	ê

表 2.2: 拼音と注音符号の例

美	ㄇㄟˇ	měi
漢字	注音符号	拼音

これらは，日本語のひらがなと同じように発音を表すために用いられる。しかし，日本語のひらがなのように単体で文書に現れることはほとんどない。よって，日本語の場合と違い一般的な文書ではほとんど用いられない。例えば，日本では大人が読むような小説であっても，前述した文学上の表現などのためにルビを用いることが多い。しかし，中国語ではそのような場合はほとんどない。主に小学生向け，または中国語を学ぶ人のための教科書で用いられることがほとんどである。

ルビの組版

日本語のルビ組版　日本語のルビの組版は日本工業規格である JIS X 4051 によって一定の基準が定められている [41]。以下では，JIS X 4051 で規定されるルビの組版について説明する。ルビは図 2.10 のように，二つの要素によって構成されてる。文書中の文字である『親文字』と，ルビを表す文字の『ルビ文字』である。基本的に，ルビ文字のサイズは親文字のサイズの半分にすることが推奨されている。

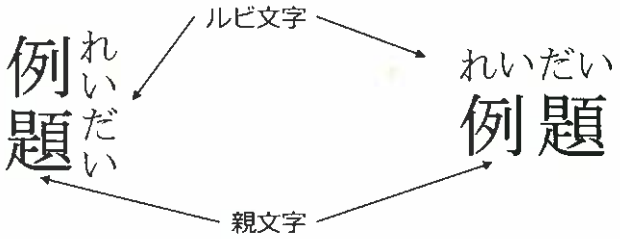


図 2.10: 親文字とルビ文字

ルビはその組み方によって，以下のように 3 種類の形に分けることができる（図 2.11）。

- モノルビ
- グループルビ
- 熟語ルビ

モノルビは親文字列 1 文字ごとに対応させたルビである。モノルビでは基本的にルビ文字を親文字の中心に配置する。親文字列の長さがルビ文字列未満の場合には，図 2.12 のようにルビ文字を，親文字群の前後の文字にかけて配置することができる。しかし，縦書きの場合に限って親文字の先頭とルビ文字の先頭を揃（そろ）える肩つきルビというものがある。この

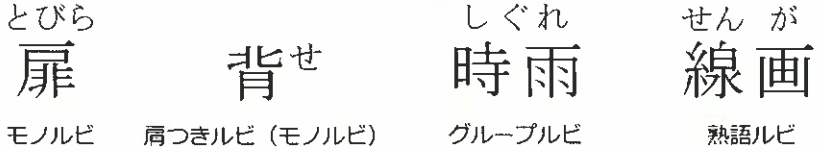


図 2.11: モノルビ，肩つきルビ（モノルビ），グループルビ，熟語ルビ，

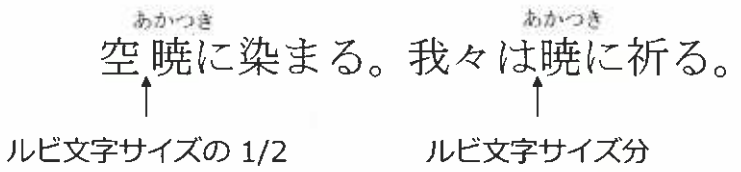


図 2.12: 異なる量のルビかけ

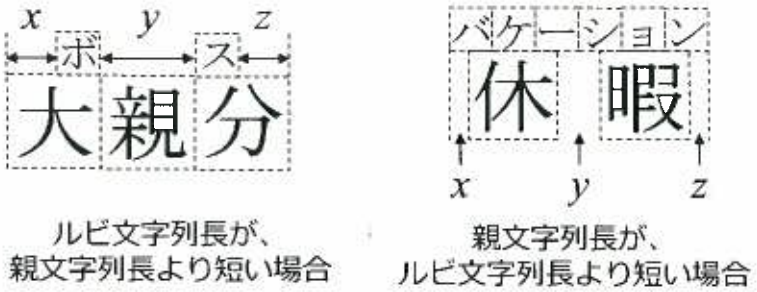


図 2.13: グループルビの処理例

際，ルビ文字の種類，そのサイズによって親文字群の前後の文字にかけてよい量が異なる。図 2.12 の例では，それぞれルビかけの量が異なっていることが確認できる。また，この量は行頭や行末にルビが来た場合でも異なるので注意が必要である。

グループルビは 2 文字以上の親文字列全体にまとめてつけたルビである。熟語で個々の漢字ごとに読みがわけられない場合に，グループルビを用いることが多い。グループルビでは図 2.13 のようにルビ文字列と親文字列の長さの関係によって配置のパターンが異なる。図中の x は『親文字列の先頭からルビ文字列の先頭までの空き量』，y は『ルビ文字間の空き量』，z は『ルビ文字列の最後尾から親文字列の最後尾までの空き量』を示している。このとき，y は文字の数によって複数ある場合もある。もし，親文字列長がルビ文字列長より短い場合には，これら三つの説明文中のルビ文字列と親文字列の言葉を入れ替える。グループルビを組む際には，これらが $x : y : z = 1 : 2 : 1$ という比率になるように空き量を調整しなければならない。また，これは行頭や行末の場合で配置が異なることを注意しなければならない。

熟語ルビとは，モノルビがつく親文字群が熟語を構成するルビのことである。図 2.14 のように熟語ルビにおいても，親文字列長とルビ文字列長の関係によって組み方が異なる。個々

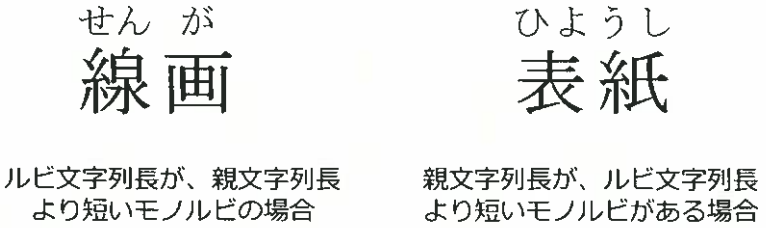


図 2.14: 熟語ルビの処理

のモノルビ全てが，ルビ文字列長が親文字列長より短い場合にはベタ組みを行う。一つでも，親文字列長がルビ文字列長より短いモノルビがある場合には，全ての親文字を組み合わせて新たな親文字列とし，全てのルビ文字を組み合わせて新たなるルビ文字列とする。この場合はグループルビと同様な処理である。もし，熟語ルビを複数行に分割する場合はモノルビとモノルビ間で改行を行うことができる。また，熟語ルビに関しては W3C によって公開されている日本語組版の要件の付録に，より詳細な説明が記載されている [51]。これらのほかにも JIS X 4051 では，ルビを両側に配置する場合の配置方法も規定している。

中国語ルビの組版　中国語では拼音の振り方に一定の基準が存在しているが，その組版に明文化された規格はない。そこで，中国語や台湾語でルビが利用されている例を紹介し，それらを解説する。中国語のルビの場合，注音符号と拼音で組版が異なる。拼音の場合，日本語ルビの場合のアルファベットの規定と同じような組み方を行う。もし，ルビとしてではなく一般的な発音のみを書く場合には原則的に単語を単位として分かち書きをする規則になっている。また，単語が固有名詞か一般的なものであるかどうかの判断のために一定の基準が標準として決められている [52]。図 2.15 に拼音のルビの振り方の一例を示す。

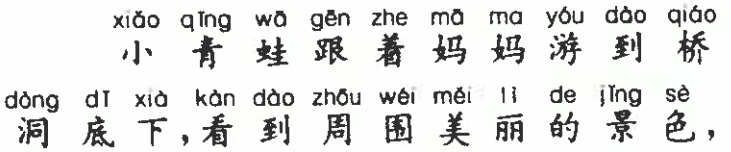


図 2.15: 拼音ルビのある文章 (出典：文献 [53])

注音符号の場合は組版が特殊である。まず注音符号の構成について説明する。注音符号は子音字と声調記号と母音字とで構成され，これらを図 2.16 のように縦書きに並べることで

一つの発音を示す。そのため，横書きであっても図 2.17 のように個々の親文字に対して縦書きされた注音記号を振ることが基本である。しかしながら，現在コンピュータでこれを表現することができない場合が多く，図 2.18 のように注音符号を表記する場合もある。

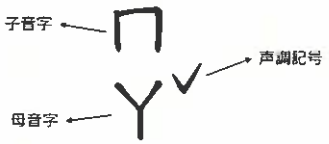


図 2.16: 注音符号の構成と組み方

我ㄨㄛˇ在ㄗㄞˋ日ㄖˋ本ㄅㄣˇ玩ㄨㄢˊ兒ㄦˊ了ㄌㄜ˙三ㄙㄢ個ㄍㄜˋ禮ㄌㄧˇ拜ㄅㄞˋ。

図 2.17: 注音符号の振られた文章 (出典：文献 [54])

克己復禮 ㄎㄜˋ ㄐㄧˇ ㄈㄨˋ ㄌㄧˇ kè jǐ fù lǐ
儒家的修養方法之一。指約束克制自身合於禮。
克己愛人 ㄎㄜˋ ㄐㄧˇ ㄞˋ ㄖㄣˊ kè jǐ ài rén
控制自己的私欲和有害於別人的言行，
克拉瑪依 ㄎㄜˋ ㄌㄚ ㄇㄚˇ ㄧ kè lā mǎ yī
城市名。在新疆西北部，以盛產石油而一個以工業為主的工礦城市。

図 2.18: 注音符号を横書きした例 (出典：文献 [55])

2.3.3 書字方向

書字方向の概要

縦書きや横書きなどのようにグリフを配置する方向のことを『書字方向』という。グリフを上から下へ配置するレイアウトを『縦書き』，左から右へ配置するレイアウトを『横書き』と呼ぶ。しかし，このように単純な『横書き』や『縦書き』という分類では，様々な種類の書字方向を適切に分類することはできない。

例えば，図 2.19 の左図は英語のテキスト，右図はアラビア語のテキストである。この図において，線矢印はグリフの進む方向，太矢印は行の進む方向を示している。この二つを比較すると，グリフは同じ横方向に進んでいるので，横書きである。しかし，グリフの進む方向がお互いに逆である。英語のテキストでは左から右，アラビア語のテキストでは右から左である。このように，同じ横書きであっても英語などの『左横書き（行：上から下)』，アラビア語などの『右横書き（行：上から下)』が存在する。

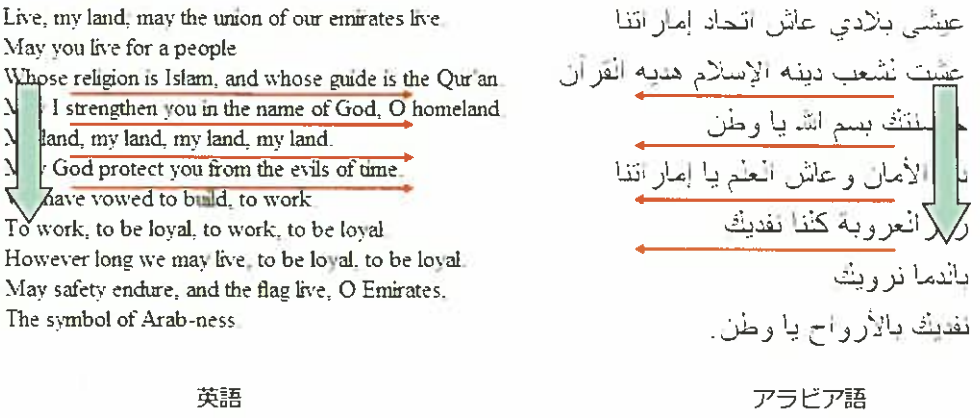
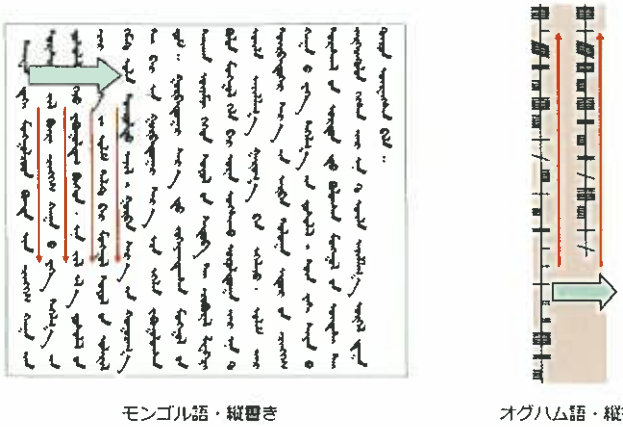


図 2.19: 英語のテキストとアラビア語のテキストの書字方向

(出典：http://www.nationalanthems.info/ae'.htm)

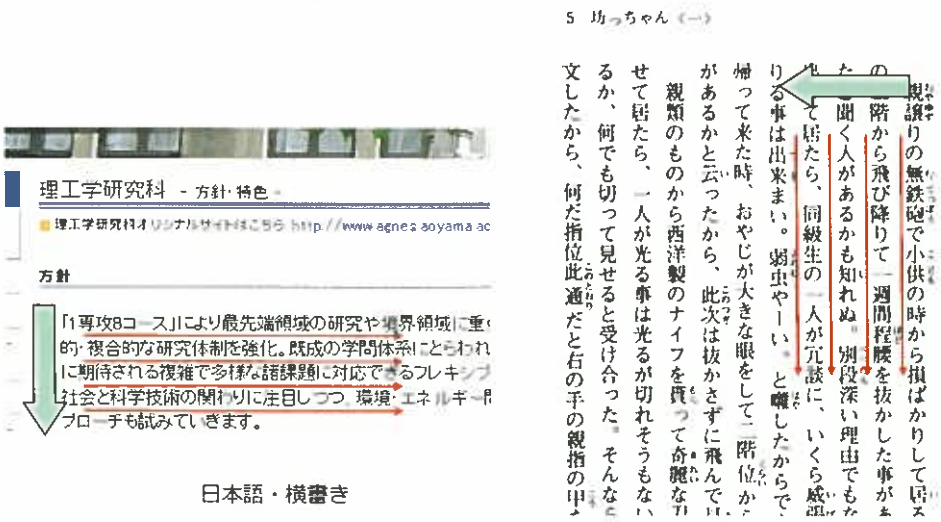
また，横書きと同様に縦書きであっても表記体系によって，いくつかの種類に分類することができる。図 2.20 の左図はモンゴル語のテキスト，右図はオグハム語のテキストである。モンゴル語のテキストは，日本語と同じようにグリフの進む方向は上から下である。しかし，行の進む方向（太矢印）は日本語と異なっており，左から右である。これを『上縦書き（行：左から右)』とする。また，オグハム語という中世初期のアイルランドで用いられたテキストは，グリフを下から上に書き進める。これを『下縦書き（行：左から右)』とする。



モンゴル語・縦書き　　オグハム語・縦書き

図 2.20: モンゴル語のテキストとオグハム語のテキストの書字方向

(左図出典：文献 [56], 右図出典：文献 [57])



日本語・横書き　　日本語・縦書き

図 2.21: 横書きと縦書きの日本語テキスト

(左図出典:http://www.aoyama.ac.jp/graduate/science/index.html,

右図出典：文献 [58])

これらを踏まえると世界には以下のような書字方向がある。

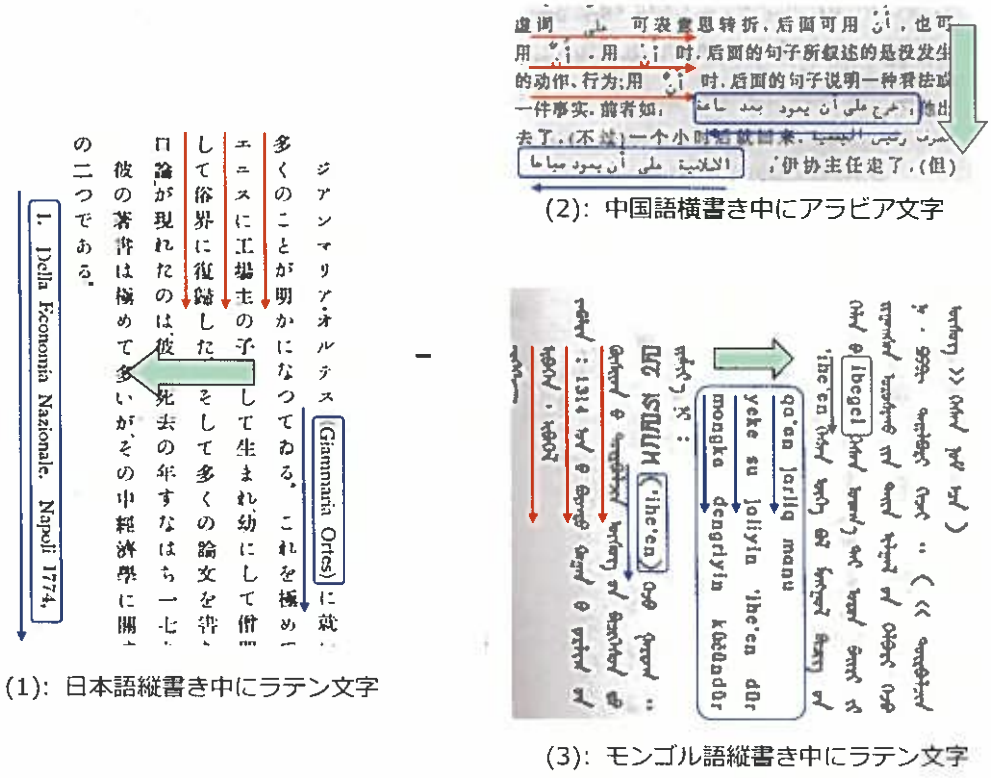
1. 右横書き（行：上から下）：アラビア語など
2. 左横書き（行：上から下）：英語など
3. 上縦書き（行：右から左）：日本語など
4. 上縦書き（行：左から右）：モンゴル語
5. 下縦書き（行：右から左）：オグハム語

また，同じ言語でも一つの書字方向だけを持つとは限らない。この特徴が顕著なのは日本語である。もともと日本語では縦書きのみであった。しかし，明治時期より横書きを取り入れ，図 2.21 のように，2 と 3 の書字方向が可能である [59]。また，昭和初期までは右横書きをする場合もあった。このように日本語は二つの書字方向が一般的である点で非常に特徴的である。

多言語テキストの書字方向

次に多言語テキストにおける書字方向について解説する。図 2.22 には，ある言語のテキスト内に外国の言語が表記されている場合のテキストの例を示している。同図において，実線矢印は主言語のテキストのグリフの進む方向であり，点線矢印は他言語のグリフが進む方向を示している。同図を見ると，それぞれの場合で主言語の配置とは少し異なる方法で配置されていることが分かる。

例えば，(1) の場合は文字の進む方向を示す実線矢印と点線矢印は同じ方向である。しかし，ラテンアルファベットの一つ一つのグリフは回転していて，日本語の文字とは配置が異なっていることが分かる。(2) の場合は，実線矢印と点線矢印が異なる方向に進んでいるが，それぞれのグリフは同じ方向を向いている。この場合，どの部分のグリフの進む向きを反転させるのかということが問題となる。この問題は BIDI 問題として知られており，Unicode によってアルゴリズムが提案され解決方法が示されている [60]。(3) の場合は (1) の場合と同じように，矢印の方向は同じである。しかし，外国の言語のグリフが回転されている。この三つの違いを見れば分かるように，多言語テキストの配置の方法は，言語ごとに異なっている。



(1): 日本語縦書き中にラテン文字

(2): 中国語横書き中にアラビア文字

(3): モンゴル語縦書き中にラテン文字

図 2.22: 様々な多言語テキストの例 ((1) の出典：文献 [61], (2) と (3) の出典：文献 [62].)

書字方向の分析と指定

このように複雑な書字方向を指定し，これらをコンピュータ上で扱うため，Etemad は書字方向の分析とその指定方法について述べている [62]。それによれば，書字方向は以下の三つのプロパティによって構成される。

1. ブロック方向 (block progression)
2. 行内方向 (inline progression)
3. グリフ角度 (glyph orientation)

それぞれのプロパティの概要を図 2.23 に示す。ブロック方向は，ひとつひとつの行をどちらに進めていくかを決定するものである。行内方向は，ひとつひとつのグリフをどの方向に進めて配置していくかを決定するものである。グリフ角度は，グリフの回転の角度を決定するものである。一般的にグリフは直立状態であり，値は 0 度となる。この三つのプロパティを定めることによって，書字方向を分析することができる。例えば，これらのプロパティを

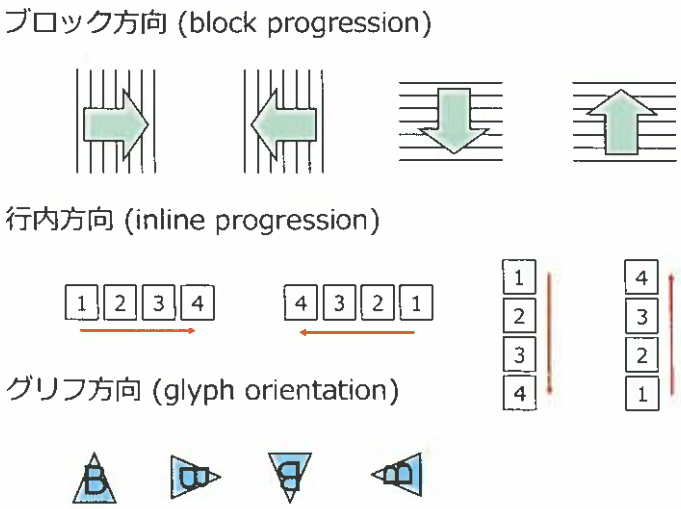


図 2.23: テキストレイアウトの三つのプロパティ

を定めることによって，書字方向を分析することができる。例えば，これらのプロパティを用いると，本論文の書字方向は以下のようになる。

- ブロック方向：上から下
- 行内方向：右から左
- グリフ角度：直立 (0 度)

これらのプロパティを用いることで，書字方向を指定することができるようになった。そこでこの三つのプロパティを使って，ある表記体系がとりうるべき適切な書字方向を規定規定することができる。その規定は以下の二つである。

1. ブロック方向 (block progression) と方向性 (directionality) の組み合わせ
2. テキスト回転規則 (text rotation scheme)

1 ではその表記体系がとるべき，ブロック方向と行内方向の組み合わせを規定する。例えば現代の日本語であれば，『ブロック方向：上から下，方向性：左から右』と『ブロック方向：右から左，方向性：上から下』という二種類の組み合わせが可能である。しかし，一方アラビア語などの右横書き（行：上から下）言語の場合は『ブロック方向：上から下，方向性：左から右』の組み合わせしか許されていない。表 2.3 では代表的なブロック方向と方向性の組み合わせと，それ対応する言語を示す。

表 2.3: ブロック方向と方向性の組み合わせと対応する言語

ブロック方向	方向性	言語
上から下	左から右	英語，日本語，中国語
上から下	右から左	アラビア語，ヘブライ語
右から左	上から下	日本語，台湾語
左から右	上から下	モンゴル語
左から右	下から上	オグハム語

最後に，2 のグリフ回転規則について説明する。これは主言語のテキスト内に，他言語があった場合のグリフの回転を決定するための項目である。図 2.24 は他言語が混入した場合に用いる一般的なグリフの回転規則の概要である。他言語の文字列は，主言語のブロック方向の始まりの方向に向けて，グリフを回転させるのが一般的である。例えば，同図ではラテンアルファベットの文字列とアラビア文字の文字列が日本語のテキストに混入している。これらは全て，グリフをブロック方向の始まりの方向に回転する。このようにすれば，同図のようにラテン文字の部分は上から下へ，アラビア文字の部分は下から上に進むようになる。

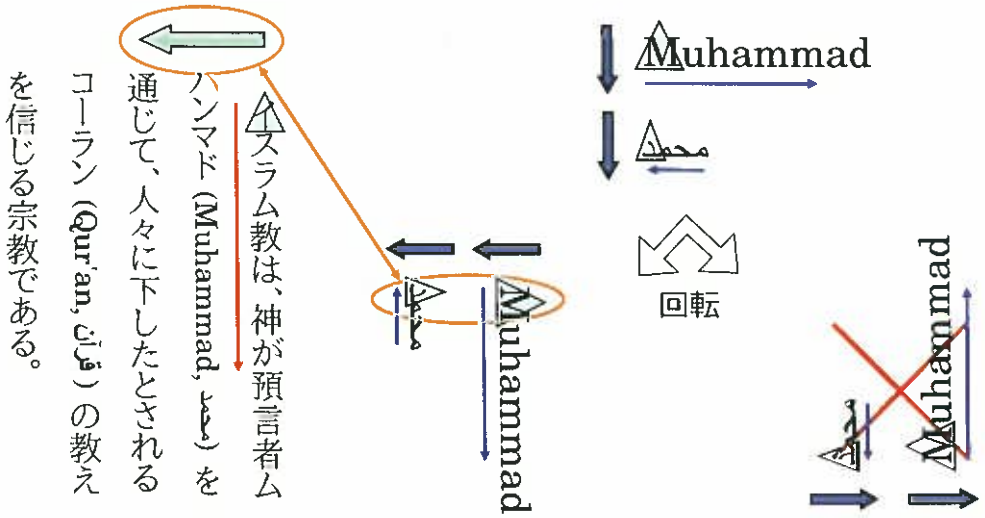


図 2.24: 一般的な他言語のグリフ回転規則

しかし，この汎用的な規則が適用できない場面もある。例えば，図 2.25 で示すように，日本語組版で用いる『縦中横』や略字を書く場合の特別な配置などの場合である。『縦中横』は，和暦や何かの略字などを一文字分の高さで左横書きするものである。また，同図の右の例ように，略字を書く場合に一般的にはその部分のグリフを回転させずに用いることもある。これらのような場合には，上述した一般的なグリフの回転規則とは異なるため，何らかの形で明示的な指定が必要である。

図 2.25: 縦中横と略字表示の際のグリフ回転

第3章　Webの技術

本章では，Webの技術の概要とルビと書字方向選択をどのように指定するのかを解説する。また，ここではそれらの視点から現在のWebが抱える問題点を指摘する。

3.1　WebとHTML

3.1.1　概要

World Wide Web (Web) は，1994年にTim Berners-Leeによって開発された，インターネット上で文書を共有することを目的に作られたシステムである[63]。Webはインターネットで最も普及しているシステムの一つであり，HTMLと呼ばれる言語でマークアップされたファイルを扱う。HTMLはHyper Text Markup Languageの略であり，マークアップ言語の一種である。HTMLのマークアップのアイディアは，文書の電子化のために開発されたSGML[64]というマークアップ言語の影響を強く受けている。図3.1はHTML文書の一例である。HTMLは<>で囲まれたタグと呼ばれるものを利用して，テキストをマークアップする。例えば同図の『HTML文書』という文字列は<h1>というタグでマークアップされていて，『見出し1』という意味を持っている。また，というタグで囲まれた『HTML』という文字列は『強調』という意味を持っている。

```
<html>
  <head>
    <title>HTML 文書の一例</title>
  </head>
  <body>
    <h1>HTML 文書</h1>
    <p>これは Web で用いられる <em>HTML</em> 文書の一例です。</p>
  </body>
</html>
```

図3.1: HTMLのマークアップ例

HTMLには様々なバリエーションがある。現在はHTML4.1 [65] とXHTML1.0 [66] という二つのバージョンが主に用いられている。しかし最近では，Webアプリケーションのプラットフォームとなることを目的とし，従来のHTMLよりもマルチメディアに関連する機能を強化したHTML5 [67] というバージョンが策定中である。

3.1.2 特徴と利点

構造化された文書

HTMLは構造をもった文書である。タグの入れ子構造になっており，図3.1のマークアップによる構造は図3.2の木構造と同等である。そして，この木を操作するためDOM (Document Object Model) というAPIが仕様として定義されている [68]。DOMは機能のレベルによってLeve1からLevel3まで定義されている。DOMの操作によって，ページを動的に変更することができる。この機能は現在のWebページでは欠かせないものになっている。その他の利点として，構造化されていることによって他のファイルへの変換が比較的容易であることがあげられる。実際に，出版データをXMLファイルでデータを保持してき，最終的に他のファイルへ変換し，印刷するという形をとる企業もある [69]。

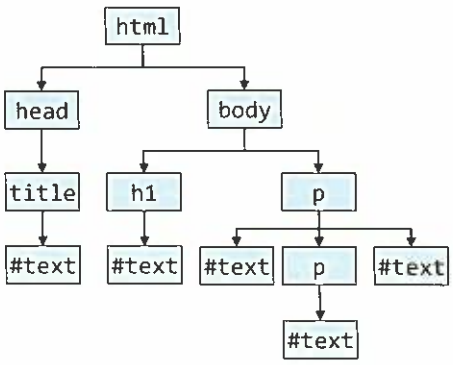


図3.2: HTMLの木構造（DOMツリー）

スタイルの分離と動的レイアウト

HTML文書は構造のみを保持するように設計されており，そのスタイル（見た目）とは分離されている。HTMLのスタイルを決定するため，スタイルシートという技術があり，異なるスタイルシートを同一のHTMLに適用することができる。そのため，同一のHTMLファイルを異なるスタイルで見ることが可能である。また，Wordのdocやpdf形式とは違い，

ページ固定されたレイアウトではない。ウインドウ幅やDOMの操作によって様々なレイアウトに動的に変更することができる。最近は一般的なPCディスプレイだけでなく，スマートフォンや電子書籍リーダなどの異なる表示環境があることを考えると，これらの利点は非常に重要である。

高機能なブラウザ

HTMLを閲覧するソフトウェアとして一般的なのはWebブラウザである。Webブラウザは現在，非常に高機能になっており，例えば音楽や動画などを簡単に再生することができる。また，Google Docs [1] ではWebブラウザ上で文書ファイルや表計算ファイルを閲覧し，編集できるようになっている。他にもFlashやJAVAプラグインなどのような，他言語の実行環境との連携も非常に簡単にできる。このようにWebブラウザは非常に高機能になっており，様々な機能を一般的なソフトウェアよりも簡単に用いることができる。それによってHTMLは従来の，主に文字のみで構成される文書以上の環境を提供することができる。

ハイパーリンク

HTMLはハイパーリンクという参照構造をもっている。一般的にハイパーリンクを持つ文書をハイパーテキストと呼び，その概念は1945年にBushが発表したMemexというシステム[70]のアイディアが基になっている。Bushが言及した文書間のリンク(path)，検索のための索引(index)などは現在のWebの特徴とよくあてはまる。Webでは単方向であるが文書への参照を実現することができる。これを用いて，クリックで参照先の文書へネットワークを介してすぐに移動することができる。この機能は人々の情報や文書の扱いに関して劇的な変化を与えている。docやPDFなどのファイル形式であってもハイパーリンクを表現することができるが，実際に用いられている機会は少ない。Webのハイパーリンクの方がより強力で，汎用的であるといえる。

国際化への対応

Webは世界中で同一のシステムを使うことを目的としているため，国際化に重きがおかれたシステムである。もともと，Webは西ヨーロッパのラテン文字のみを扱う文字コードのISO/IEC 8859-1 [71]のみを利用することが想定されていた。そのため，各国の文字コードに

[1] http://docs.google.com/

対応させた独自のページやブラウザが乱立した。そこで，1997 年に HTML で統一的に多言語を扱うための枠組みが IETF で仕様化された [72]。現在では多くの Web ページが Unicode を用いるようになっており，W3C によって HTML や XML などのマークアップ言語で Unicode が使えるような仕組みが提供されている [73, 74]。

実際，現在では Web は世界の言語を最も幅広く利用しているコンピュータシステムである。例えば Wikipedia ではすでに 276 言語版のページの作成されている[2]。このように，Web ではすでに国際化を実現するための技術が多く存在している。また本研究において更なる国際化を目指す点でも，HTML の国際化技術の豊富さは非常に有用である。

3.2 CSS

3.2.1 概要

CSS (Cascading Style Sheets) は HTML や XML のスタイルを指定するための仕様である。CSS はセレクタ，プロパティ，値という三つの項目を指定することでスタイルを指定することができる。図 3.3 は CSS でのスタイル指定の一例である。CSS では，`div#id` をセレクタ，`color` をプロパティ，`red` を値と呼ぶ。セレクタは HTML 文書中のどの範囲に対してスタイルを適用させるかを指定する項目である。この図の場合では，『`div` 要素で，かつ `id` 属性として `target` を持つ要素』という指定になる。プロパティはスタイルの特性を表すもので，この場合では『要素中の文字列の色』を示している。値はプロパティに対する具体的なスタイルの指定であり，この場合は『赤色』を示している。すなわち，この CSS の指定では『`div` 要素で，`id` 属性として `target` を持つ要素内の文字列の色を赤色にする』というスタイル指定をしたことになる。

```
div#target {
  color: red;
}
```

図 3.3: CSS の指定

CSS も HTML と同様にいくつかのバージョンが存在している。現在，CSS 2.1 [75] が最もメジャーな CSS のバージョンである。各ブラウザベンダーもこのバージョンを CSS の基

[2] `http://ja.wikipedia.org/wiki/Wikipedia:全言語版の統計`

本仕様とみなして実装を行っている。また，最近では CSS 2.1 を各機能ごとに仕様を独立させ，分離させた CSS 3 というバージョンが策定中である。CSS 3 は膨大な CSS 2.1 の仕様を機能ごとに各モジュールに分離し，それらを独立化して仕様化している。特に CSS 3 では，非常多様なレイアウトを可能にしようとしており，多段組 (CSS 3 Multicolumn Module) [76]，フォントに関する詳細指定 (CSS 3 Fonts Module) [77]，など，Web ページに紙媒体の組版に匹敵するような機能を持たせるような仕様の策定が進められている。CSS 3 についての各モジュールの現状は CSS Current Work[3] で仕様の一覧と，策定の進捗具合を確認することができる。

3.2.2 CSS の特徴

ボックスモデル

CSS ではボックスモデル (Box Model) と呼ばれるレイアウトモデルに従ってスタイルを決定する。各要素は，ボックス (box) と呼ばれる矩形領域に対応している。図 3.4 はボックスの概要図である。ボックスは内容 (Content) とその周りに周辺領域 (Padding, Border, Margin) を持つ。また，これらのボックスは視覚整形モデル (Visual Formatting Model) というモデルに従って配置される。CSS に関連する全ての表示はこれらのモデルにしたがって行われ，例えば行分割や表 (Table) もこのボックスモデルにしたがってレイアウトが決定されている。

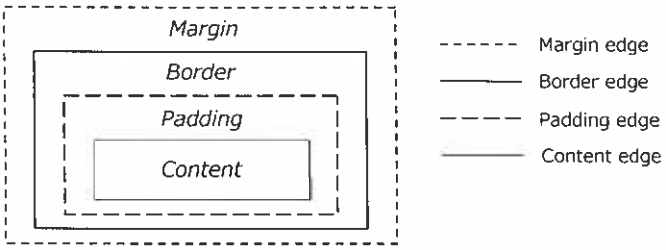


図 3.4: CSS ボックスモデル

他のスタイルシート

スタイルを指定するために用いるスタイルシート言語にはいくつか種類がある。XSL-FO [78] は，XML をベースとしたスタイルシート言語である。XSL-FO では，特に複数のページで構成される媒体を想定されて設計されている。そのため，印刷物並みの精細で高品位な組

[3] http://www.w3.org/Style/CSS/current-work

版の指定をすることが可能である。CSS が Web ページを表示する際のスタイル指定を前提とした技術であるのに対し，XSL-FO は紙媒体に印刷するために考え出された技術である。両者の違いは利用の仕方にも現れている。HTML は Web ブラウザで表示することが一般的であるが，XSL-FO を用いた場合は最終的に PDF 形式のファイルへ出力することが一般的である。すなわち，CSS は Web のため，XSL-FO は印刷のために利用されている。しかし，最近は Web と印刷の中間に位置する電子書籍という分野が脚光を浴びている。電子書籍形式の一つである EPUB はマークアップとその組版に XHTML と CSS を採用している [79]。そのため，CSS 3 では様々な組版技術をサポートするための数々の仕様が策定されようとしている。また，XSL-FO のほかにも DSSSL [80] というスタイルシート言語がある。これはプログラミング言語 Scheme を基に開発されたスタイル意味指定言語であり，SGML か XML で記述された文書のためのスタイルシート言語である。

3.3 ルビのマークアップとスタイル指定

本節では Web においてルビを表示するためのマークアップについて説明する。HTML でルビを扱うために最初の出された提案は Dürst による Internet Draft [81] である。現在では W3C によっていくつかの仕様が提案，または勧告されている。XHTML でルビを使うための XHTML Ruby Annotation [82]，HTML でルビを使うための HTML5 Ruby Markup [67]，CSS でルビ表示を指定するための CSS3 Ruby Module [83] である。HTML や XHTML は，前述のようにルビのマークアップのために用いる。CSS ではルビの表示と詳細なスタイルの指定を行うことができる。

3.3.1 二種類のマークアップ方法

XHTML によるマークアップ

XHTML では図 3.5 に示すように二つのマークアップ手法を用意している。簡単ルビ (Simple Ruby) と複雑ルビ (Complex Ruby) とよばれるものである。この分類はあくまでマークアップ手法の分類であり，組版上のモノルビやグループルビなどの分類とは異なる。簡単ルビは，<ruby> 内に一つの <rb> と一つの <rt> を持つ構造である。これによってモノルビやグループルビを表現することができる。複雑ルビは複数の <rb> と <rt> がある場合や，上下にルビを付けたい場合（縦書き時は右と左）に用いるものである。複数の <rb> のまとまりを <rbc> で囲い，同様に複数の <rt> の集まりを <rtc> で囲う。このように，

<rtc> や <rbc> は個々の <rb> や <rt> をまとめるために用いるタグである。<rtc> が二つ用いることもでき，その場合には暗黙的に二つ目の <rtc> のルビは下側に表示する。複雑ルビを用いることで対字ルビ，上下両側につくルビを表すことができる。また図 3.6 に示すように，rbspan を用いると上側は対字ルビ，下側はグループルビという表現が可能になる。

```
簡単ルビ：
    <ruby><rb>気質</rb><rt>かたぎ</rt></ruby>
複雑ルビ：
    <ruby>
      <rbc><rb>気</rb><rb>質</rb></rbc>
      <rtc><rt>き</rt><rt>しつ</rt></rtc>
    </ruby>
```

図 3.5: XHTML ルビのマークアップ

```
<ruby>
    <rbc><rb>気</rb><rb>質</rb></rbc>
    <rtc><rt>き</rt><rt>しつ</rt></rtc>
    <rtc><rt rbspan='2'>かたぎ</rt></rtc>
</ruby>
```

図 3.6: 両側にルビつける場合のマークアップ

HTML5 によるマークアップ

ルビは HTML5 で，図 3.7 のようにマークアップする。XHTML のルビマークアップとの最大の違いは，<rb> タグがないことである。また，XHTML のように複雑ルビが定義されていない。そのため，<rtc> や <rbc> のタグがなく，両側にルビを同時に配置することが出来ない。HTML5 では <rb> タグは定義されていないが，インラインとなる要素を <ruby> 直下に配置することができる。そのため，色を変えたい場合などは などでタグ付けすることが可能である。また，親文字列のテキストとルビ文字列を示す <rt> を複数個用いると，対語ルビを表現することができる。最後に，HTML5 は現在策定中の仕様であり，今後これらのマークアップ方法が変更される可能性があることを指摘しておく。

```
対語ルビ：<ruby>気質<rt>かたぎ</rt></ruby>
対字ルビ：<ruby>気<rt>き</rt>質<rt>しつ</rt></ruby>
```

図 3.7: HTML5 ルビのマークアップ

3.3.2 スタイルの指定

本項では CSS3 Ruby Module で定義されているプロパティについて説明する。CSS3 Ruby Module では，ルビの詳細なスタイルを指定することができる。なお，2010 年 1 月現在の CSS Ruby Module は Editor's Draft の段階であり，プロパティ，またはその値などが変化する可能性があることを指摘しておく。

ルビのボックスモデル

ここでは，ルビを表示する際のボックスモデルについて説明する。CSS で表示構造を指定するためには，ボックスの表示形態を指定する `display` プロパティを用いる。ルビのための `display` プロパティとして以下のプロパティ値が用意されている。

1. `ruby`
2. `ruby-base`
3. `ruby-text`
4. `ruby-base-container`
5. `ruby-text-container`

これらについて，図 3.8 に示すルビのボックスモデル図を用いて説明する。なおこのボックスモデルは XHTML のルビマークアップを参考にしており，簡単ルビ，複雑ルビ両方を扱うことができる。簡単ルビの場合，そのボックスモデルは `ruby`，`ruby-base`，`ruby-text` で構成される。複雑ルビの場合，上述した全てのプロパティで構成される。これらは XHTML のマークアップと対応して `<ruby>`，`<rb>`，`<rt>`，`<rbc>`，`<rtc>` と対応付けて考えることができる。以下ではルビの詳細なスタイルしてを行うための各プロパティについて説明する。

ruby-position プロパティ

`ruby-position` はルビ文字列をどの場所に配置するかを指定するためのプロパティである。プロパティの値としては，ルビを行頭側（横書きでは上，縦書きでは右）につけ

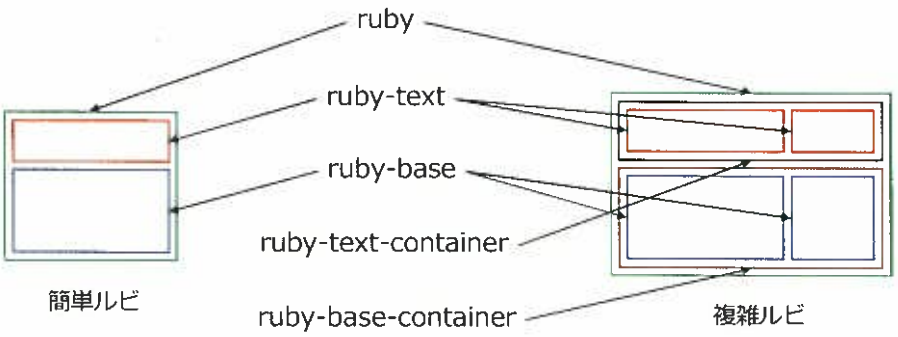


図 3.8: ルビのボックスモデルと display プロパティ

る before，行末側（横書きでは下，縦書きでは左）につけるための after，注音符号のため常に右側にルビを振る bopomofo，特別なスタイルを与えずにそのまま親文字列の後ろに表示する inline がある。

ruby-align プロパティ

ruby-align はルビ文字列や親文字列の寄せ方を指定するプロパティである。基本的な指定として左寄せ (left) や，中央寄せ (center)，右寄せ (right) がある。他にも，ルビを均等割りで配置する distribute-letter，ルビを JIS X 4051 で規定されている 1:2:1 の割合で配置する distribute-space，行端に寄せを合わせる line-edge，ルビ文字列や親文字列の種類に応じて自動的に規定された方法でルビを配置する auto などがある。

ruby-overhang プロパティ

ruby-overhang はルビかけ量を指定するためのプロパティである。プロパティ値として，ルビをかけない none，文頭側にのみルビをかける start，文末側にのみルビをかける end，両方にルビをかける auto がある。ルビをかける量は，日本語組版の要件 [51] で規定された方法で求める。

ruby-span プロパティ

ruby-span は，隣接するルビ文字列，または親文字列との結合数を指定するプロパティである。ruby-span を用いると，Table の colspan 属性のように働き，右に隣接するル

ビ文字列，または親文字列を結合する。それによって，図 3.9 の左の例に示すように対字ルビとして指定されたルビを，一部の部分だけグループルビとして表示することができる。また，ruby-span を用いると，同図の右の例ように片方が対字ルビ，片方がグループルビというような両側ルビを表示することも可能である。

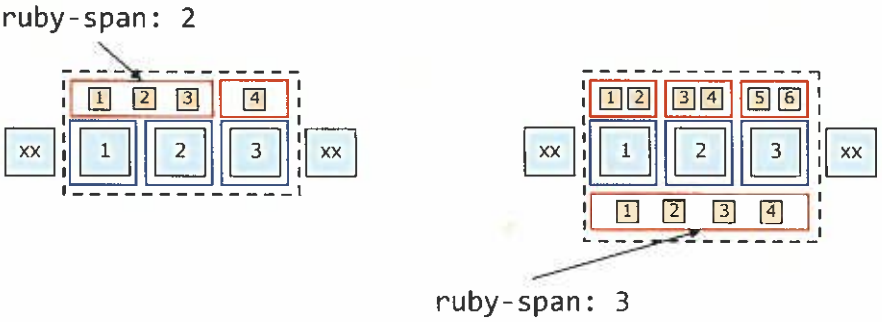


図 3.9: ruby-span 指定された場合の表示例

3.4 書字方向スタイルの指定

HTML に対して書字方向を指定するため，CSS3 Writing Mode Module という仕様の策定が進められている [84]。この CSS3 Writing Mode Module は 2.3.3 項で述べた Etemad による書字方向の指定を参考にしている。CSS3 Writing Mode Module において，書字方向は以下の三つのプロパティで指定する。

1. `writing-mode`
2. `direction`
3. `text-orientation`

図 3.10 はこれら三つのプロパティが示す項目の概要である。以下ではこれらのプロパティについて説明する。また，この仕様についても 2010 年 1 月の時点で Working Draft 段階であるため，変更される可能性があることを指摘しておく。

writing-mode プロパティ

`writing-mode` は 2.3.3 項で述べた書字方向指定のプロパティにおけるブロック方向 (Block Progression) に対応するプロパティである。これを指定することで，行の進む方向を

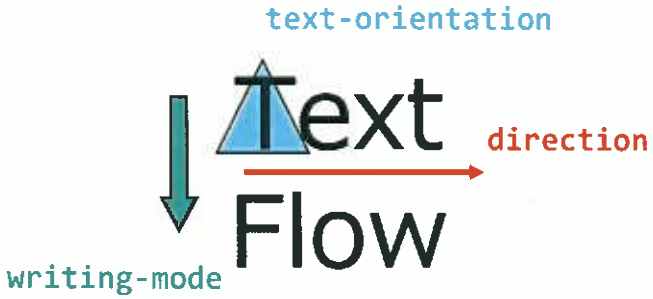


図 3.10: CSS で書字方向を指定するための三つのプロパティ

指定することができる。writing-mode では，以下のように三つのプロパティ値を指定することができる。

1. horizontal-tb
2. vertical-rl
3. vertical-lr

horizontal-tb は横書きで，上から下へのブロック方向を指定する。vertical-rl と vertical-lr は，両者とも縦書きを指定するものである。vertical-rl は右から左への縦書きのブロック方向，vertical-lr は左から右への縦書きのブロック方向を指定する。

direction プロパティ

direction は 2.3.3 項で述べた書字方向指定のプロパティのうち，行内方向を指定するためのプロパティである。direction は，以下のように二つのプロパティ値を指定できる。

1. lr
2. rl

lr は左から右へとグリフを配置する行内方向を指定するものであり，rl は右から左へとグリフを配置する行内方向を指定するものである。これらは，Unicode によって提供されている BIDI 問題を解決するアルゴリズム [60] を指定する unicode-bidi というプロパティと組み合わせて利用される。これらは縦書きの場合には，左を上，右をしたと読み変えて考える必要がある。すなわち，lr は上から下，rl は下から上を意味する。

text-orientation プロパティ

主言語中に他言語の文字列が来た場合に，どの方向に回転させるかを指定するためには text-orientation プロパティを用いる。text-orientation では，以下のようなプロパティを指定することができる。

1. vertical-right
2. upright
3. rotate-right
4. rotate-left
5. rotate-normal
6. auto

vertical-right は，縦書きにおいて本言語以外のものは時計方向に 90 度回転させる。upright は，縦書きにおいて一つの一つのグリフを回転させない指定をするプロパティ値である。rotate-right は，日本語など右から左の縦書きにおいて用いる，時計方向に 90 度回転させるプロパティ値である。rotate-left は，モンゴル語の左から右の縦書きにおいて用いる，反時計回りに 90 度回転させるプロパティ値である。rotate-normal は，書字方向が vertical-rl の場合には rotate-right プロパティを選び，vertical-lr の場合には rotate-left を用いるプロパティ値である。auto は，基本的には vertical-right と同一の処理を行うプロパティである。

論理プロパティ

論理プロパティは before や after などのプロパティ値を用いてブロック方向の指定に依存するように，位置指定を行うプロパティである。それに対して，物理プロパティは通常の left や right などを用いて，ブロック方向の指定に依存しないで位置指定を行うものである。論理プロパティには次の 6 つのプロパティ値が提案されている。

1. logical-width
2. logical-height
3. start
4. end
5. before
6. after

以下では，各プロパティの示す概念について解説する。logical-width は行の長さを表し，logical-height は全行を積んだ高さを表している。start は行頭方向を表し，end は行末方向を表している。before は先頭行の方向を表し，after は末尾行の方向を表している。図 3.11 はこれらの論理プロパティの概要図である。図において，"/" の左側は物理プロパティでの指定値，右側は論理プロパティでの指定値を示している。同図にある二つの場合を比較すると，物理プロパティは指定されている位置に変化はないが，論理プロパティは位置が変化していることが分かる。

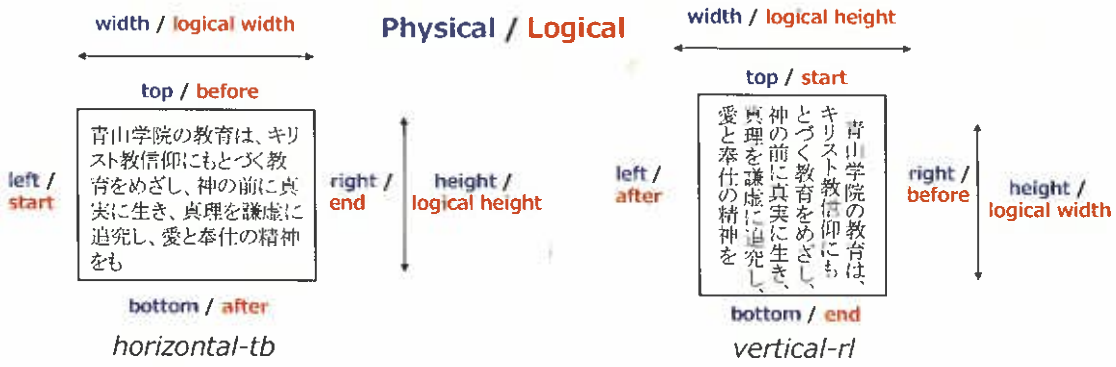


図 3.11: 物理プロパティと論理プロパティの関係

論理プロパティは，横書きと縦書きをでスタイルシートを変更する手間などをはぶくために用いる。例えば，行頭に余白を設定したい場合，物理プロパティのみでは横書きと縦書きで異なるプロパティを用いる必要がある。横書きの場合には margin-left を用いて，縦書きの場合には margin-top を用いなければならない。しかし，論理プロパティの margin-start を用いれば一つの指定で行頭に余白を設定することができる。特に，日本語は横書きも縦書きも可能であり，異なるプロパティを指定することは大きな手間になる。そこで，このような論理プロパティの提案が行われている。

3.5 Web のテキストレイアウトに関する課題

本節では本研究において解決するべき Web のテキストレイアウトに関する課題について述べる。まず，Web に関する一般的な課題について述べ，そして，特に本研究と関係の深いルビと縦書きを可能にする書字方向選択に関しての課題について述べる。

3.5.1 Webのテキストレイアウトの重要性と課題

近年，Webの発達によって，様々な種類のWebページが増えてきている。特に増えているのは，従来は紙媒体であった情報をWebページとして公開することである。たとえば，Google Books [4] では，EPUB形式で電子書籍を販売している。ACMは会誌である *Communications of ACM* をHTML形式で配布している[5]。日本経済新聞社はWebサイトでニュースを配信するサービス[6]を始めている。他にも，料理のレシピなどをWebで公開したり，ブログで自分の日記や意見をWebで公開し，多くの人がそれを読んでいる。このような背景から，Webの利用時間は年々増加しており [85]，紙媒体の印刷物と同程度の割合でWebページが読まれていると考えられる。

しかし，現状では紙の印刷物並みの組版品質を持った文書をWebで作成することはできない。Webにはすでに紙媒体では実現できない数々の素晴らしい特徴がある。この利点をそのままにして，さらに紙の印刷物と同程度の組版が可能になればより発展した文書形式となると考えられる。もちろん，各表記体系ごとの伝統的な組版の全てをWebで実現できるようにするのは不可能かもしれない。しかし，すでに多くの人々は，紙に印刷された文書の読みやすさに慣れている。また，伝統的な組版は各表記体系の読みやすさを考えて作れられており，Web上においてもその点は非常に重要だと指摘されている [86]。このような点で，従来の組版がWebで表現できることは多いに意味があることである。

加えて，文化の多様性を継承するというの意味で，組版自体の国際化も大いに意味のある仕事である。Webやコンピュータ技術は欧米で発達したため，一般的に欧米の文化にのみ対応しているという点が非常に多い。もちろん，文化はそのような制限とともに変化していくものである。しかし，様々な表記体系が持つ多様性がなくなるのは，大変残念なことである。実際このような背景から，W3Cでは日本語に関する組版の要件を未来に残すために文書化しており [51]，その目的は以下のようなものである。

> すべての文化集団は，独自の言語，文字，書記システムを持つ。それゆえ，個々の書記システムをサイバースペースに移転することは，文化的資産の継承という意味で，情報通信技術にとって非常に重要な責務といえよう。この責務を実現するための基礎的な作業として，このドキュメントでは，日本語という書記システムにおける組版上の問題点をまとめた。

以下では，Webのテキストレイアウトの対応が遅れている原因について解説する。

[4] http://books.google.com/
[5] http://cacm.acm.org/magazines
[6] http://www.nikkei.com/

浅い歴史

Webは非常に利用者の多いコンピュータシステムであるが，1994年に開発された比較的新しい技術である。そして，すでに述べたように，単なる文書の閲覧をするより画像や動画などのマルチメディアの媒体としての機能のほうが強い。そのため，紙の印刷物を対象とした伝統的な組版をWebに再現するということよりも，従来の紙の印刷物では実現できないWeb独自の技術の開発に重きが置かれていた。このような状況を背景として，Webの組版の品質というものは注目されづらく，まだまだ対応が遅れているのが現状である。

国際化への考慮

Webは様々な言語で利用されるひとつの巨大なシステムであるため，新たな技術を開発するためには，国際化の観点からの様々な議論が必要である。たとえばWebで組版に関しての技術を考える際は，その表記体系の事情だけでなく，世界中の表記体系と関係を考えなければならない。例えば，アラビア語では右から左に文字を進めなければならないが，日本語の組版をWeb技術に取り入れる場合に，それらの言語に対しても適切なレイアウトが保障されなければならない。この事実は技術の利用者からは好ましいものである。しかし，この点が新しいWeb技術の開発を難しいものとする原因のひとつでもある。

動的なレイアウト

Webページのレイアウトは，CSSのボックスの視覚整形モデルにしたがって決められる。このモデルでは，事前にレイアウトが固定化されているわけではなく，ユーザがページを表示する環境(User Agent)に依存して決定される。一方，紙の印刷物は固定化されたレイアウトであり，一度決定されたレイアウトは変更されないという性質を持っている。そのため，複雑な場合でも事前にレイアウトを決定してしまえば問題が起きない。しかし，Webではそのようなことができないため，より複雑なレイアウト規則が必要となる。

たとえばウインドウサイズが変更されれば，それにともなって改行位置も変更される。また，JavaScriptによってユーザの好みに合わせてスタイルが変化する可能性もある。このような場合の全てにおいて，どのようなレイアウトにするかをブラウザ側で計算しなければならない。この点において，静的なレイアウトを決める規則よりも，さらに複雑な規則を考える必要がある。

ユーザの意識

現在の Web ページを何らかの公式な書類を作るために用いる機会は多くない。なぜなら，公式な文書，書類などは，格式が重要になるため品質の高い組版が必要になるからである。例えば，書類，履歴書，論文などが例に挙げられる。多くの人は，必要に応じて専用の文書作成ソフトウェアを使っている。例えば，Word や LaTeX，Adobe InDesing などである。そのため，Web は品質の高い組版を必要とする文書を作成するための媒体ではないという意識が少なからず存在する。しかし，Google Docs [7] の登場によって，Web で文書を作成しデータを保存しておくという環境も整えられ始めている。このような流れが加速することで，Web でも品質の高い組版が必要であるという意識を高めることができる。

3.5.2 ルビの活用とその課題

Web におけるルビの必要性

Web 上でのルビの必要性は以前よりも増していると考えられる。なぜなら，以前と比べて様々な種類の Web ページが存在するからである。例えば，従来の新聞は一部の漢字にのみルビを振る『パラルビ』を用いている。その慣習にならい，新聞社や通信社が提供するニュース記事の Web ページでは，一部の漢字に読みがなを与えている。しかし，その場合には多くは『延坪（ヨンピョン）島』のように単に漢字に続けて () 内に読みを併記する形で表示していることが多い。このように簡単なルビを振る必要性に加えて，書籍や文学などに対してルビを振る場合のような，より詳細な指定が可能なルビも求められている。

また，利用者に関しても，幅広い年齢層が Web を利用するようになっている。例えば Web ページを教科書の教材として用いる場合に，漢字の読みが分からない子供のためにはルビが必要である。また，視覚障害者や一般の印刷物を読むことが困難な人々のため，DAISY というデジタル録音図書の規格 [87] がある。これは，XHTML をベースに文書を保存する形式になっており，録音図書という性質上，読みを与えるルビは必須である。

Web 独自の活用例

上述のような場合を考えると，紙媒体のルビを再現することが一つの目的となる。しかし，Web の性質を考えればさらに便利なルビを提供することができる。例えば，Web では

[7] https://docs.google.com/

動的にレイアウトを変更することが可能なので，ルビの表示と非表示を切り替えることができる。また，ルビのマークアップ自体は，コンピュータの力を借りることも可能である。

具体的なソフトウェア例として，振り仮名インジェクター[8] というソフトウェアは形態素解析器を用いて漢字の読みを解析し，DOM 操作によって自動的に Web ページにルビを振ることができる。また，Yahoo! JAPAN が提供するルビ振り API[9] では小学生の学年を指定することで，その学年のレベルにあった振り仮名を返してくれる。その結果を用いれば，同一の Web ページを小学生の学年に合わせたルビを振るということも可能である。

ブラウザの対応

ここでは，Web でルビを利用する際に最重要となる，Web ブラウザにおけるルビ表示への対応について述べる。現在のルビの対応状況を，マークアップに関しては表 3.1 に，スタイル指定に関しては表 3.2 に示す。

表 3.1: ルビのマークアップのブラウザの対応 (シェア出典：文献 [88])

ブラウザ	シェア	HTML5	XHTML (簡単)	XHTML (両側)	XHTML (複雑)
Internet Explorer 8.0	60%	○	○	-	-
Firefox 3.6	23%	-	-	-	-
Chrome 8.0, Safari 5.0	13%	○	○	-	-
Opera 11.0	2%	-	-	-	-
Amaya 11.3	-	-	○	○	-

Microsoft Internet Explorer は最初にルビの表示を実装したブラウザであり，1999 年に初めて登場した。その後 10 年ほど，他のブラウザはルビの表示に対応していなかったが，2009 年に Safari と Google Chrome の二つのブラウザでルビの対応が始まった。現在，簡単なルビのみならば約 7 割のブラウザで表示することができ，Web ページの製作者がルビを用いることに抵抗を感じなくなり始めている。実際，いくつかの新聞社の Web ページでは読みがなの記述にルビを用いている。しかし，二番目にシェアの大きい Mozilla Firefox ではルビの表示は対応していない。この Firefox がルビのマークアップに対応することで，単純なルビの表示ならばほぼ 100%のブラウザで表示できると言えるだろう。

[8] https://addons.mozilla.jp/firefox/details/6178
[9] http://developer.yahoo.co.jp/webapi/jlp/furigana/v1/furigana.html

表 3.2: CSS3 Ruby Module の対応

ブラウザ	ruby-position	ruby-align	ruby-overhang	ruby-span
Internet Explorer 8.0	-	○	-	-
Firefox 3.6	-	-	-	-
Chrome 8.0, Safari 5.0	-	-	-	-
Opera 11.0	-	-	-	-
Amaya 11.3	-	-	-	-

このように簡単なルビの表示は可能になってきている一方で，CSS で指定することができる詳細なスタイルにはほとんど対応していない。加えて，XHTML の複雑ルビを完全に実装したブラウザはまだない。このようなルビの表示では，書籍や書類のように詳細な組版が必要となる場合に使用することができない。Amaya は上下にルビをつけるための複雑ルビのみ表示することができる。しかし，Amaya の複雑ルビは一般的な対字ルビを表示することができず，全てグループルビとなる。また，上下で対字ルビとグループルビを組み合わせて使うための ruby-span に対応しているブラウザはない。

ルビの仕様に関する課題

ここでは，ルビに関する仕様の問題について述べる。まず一つ目の問題は，Web でルビを実現するための仕様が複数存在することである。特に，HTML5 と XHTML ではルビのマークアップの構造が異なっている。そのため，ルビの表示を実装する際に，これら二つのマークアップにどのように対応するかを考えなければならない。また，どちらのバージョンの HTML を用いるかによって，ユーザは異なるルビのマークアップをする必要がある。

二つ目は，複雑ルビの存在である。複雑ルビは HTML の Table と似たような構造をしており，レイアウトの計算が複雑である。そのため，実装のコストが大きく，現状では完全な実装が存在しない。

三つ目は，ルビを実装する際に必要となるマークアップが妥当でない (Invalid) 場合の対処を考慮しなければならないことである。Web では様々な経緯から，妥当でないマークアップの HTML が非常に多い。そのため，ページの製作者は妥当でないマークアップでも，ある程度は適切に表示されることを期待している。特に HTML5 と XHTML という二つの仕様が混在している状態では，妥当でないマークアップが現れやすいと考えられる。その対応をル

ビのマークアップ場合においても考慮する必要があるが，現状では仕様として明文化されていないため，実装する場合に個別に考えているのが現状である。

3.5.3 書字方向選択の活用と課題

書字方向選択の活用

Webで書字方向の指定によって縦書きが可能になることで，どのようにそれを活用ができるかを考えてみる。横書きが幅広く受け入れられている現在でも，新聞や書籍などは今でも縦書きが定着している。また，職業などの違いからメモやノートの書き方に，横書き用いる人と，縦書きを用いる人がおり，その好みや読みやすさが異なることが指摘されている[89]。このような今でも縦書きが非常に定着している文書，または縦書きを好む人のために，Webページであっても縦書きが可能になるのは大きな利点である。

序論で述べたように，青空文庫ではすでに多くの書籍のデータがWebページ化されている。これらを縦書きで読みたいという需要はすでに存在しており，専用のビューワなどが多数開発されている。一般的なWebブラウザでこれが可能になれば，専用のビューワを用いなくともスタイル指定をするだけでそれらを縦書きで読むことができる。また，モンゴル文字でモンゴル語を記述する場合には，基本的に縦書きしかできない。モンゴル文字は筆記体であるため，日本語のような横書き化がほぼ不可能である。このように縦書きでなくてはならない，また縦書きの方が定着しているという文書をWebページにできることが一つ目の活用である。

これに加えて，書字方向の選択が可能になることで，様々な書字方向が混在した新しいWebのレイアウトが可能になり，レイアウトの自由度を上げることができる。すでに画像や，動画，Flashを含んだ様々なレイアウトを持ったWebページがある。従来の書籍のように，全てを縦書きとして表示することがなくても，Webページの一部などに縦書きのレイアウトが可能になることは非常に有益である。例えば，何かの広告などでは縦書きが積極的に使われており，それらをテキストとして表示できるようになる。図3.12はモンゴル文字を使ったモンゴル語のサイト[10]である。このページ全体のブロック方向は上から下の横書きに分類されるが，個々の文字は縦書きされており，Webの新たなレイアウトの可能性を示す非常に興味深い例である。

[10] http://mng.ulaaq.com

図 3.12: モンゴル文字で書かれた縦書きレイアウトを持つ Web ページ

ブラウザの対応とその他の実装

ここでは書字方向の選択に関するブラウザの対応と，ブラウザ以外で縦書きを可能にする実装について述べる。現在のブラウザの書字方向指定の対応状況を表 3.3 に示す。書字方向の指定に関しては Internet Explorer のみ指定が可能である。もともと書字方向の選択機能が Internet Explorer 5.5 で独自に実装され，それを基に仕様を策定しようとしている。

レンダリングエンジンが書字方向選択に対応しているブラウザは Internet Explorer のみである。しかし，Web で縦書きを可能にする実装がいくつか存在している。それらの実装では，CSS の回転機能などを用いたり，Table レイアウトと JavaScript を用いて近似的に縦書

表 3.3: 書字方向指定のブラウザの対応 (シェア出典：文献 [88])

ブラウザ	シェア	書字方向指定の有無
Internet Explorer 8.0	60%	○
Firefox 3.6	23%	-
Chrome 8.0, Safari 5.0	13%	-
Opera 11.0	2%	-
Amaya 11.3	-	-

きを実現している。例えば，竹取 JS[11]は，CSS3 2D Transforms Module [90] の `transoform` というプロパティを用いて，指定されたボックス自体を 90 度回転するという方式で縦書きを実現している。nehan [12]では，指定された文章を全て Table レイアウトに変換し，縦書きのようなレイアウトを実現している。しかし，これらはあくまでも日本語で用いる右から左の縦書きのみの対応である。また，固定的なレイアウトであり動的なウインドウサイズの変更などには対応できていない。

現在，ブラウザのシェアを見ると 6 割ほどのブラウザが縦書きを表示できることになる。しかし，ほとんどの Web ページでは書字方向の指定による縦書きを利用していない。特に日本では非常にシェアが高い Internet Explorer が実装しているのにも関わらず，縦書きが用いられない。それは，日本語が横書きが可能な言語であるためだと考えられる。

書字方向選択の仕様に関する課題

ここでは，書字方向選択の仕様に関する問題点について述べる。まず，一番大きな問題点としては仕様自体が完成されていないということである。もともとこの仕様は Microsoft が International Layout として，テキストの様々な要素の国際化対応を目的として 1999 年に W3C に提案した [91]。その後，いくつかの仕様に分割されたが，2010 年現在でも書字方向選択の仕様である CSS3 Writing-Mode Module [84] は WorkingDraft 段階である。

これは，CSS のボックスモデルやフォーマッティングモデルが横書きを前提に規定されており，書字方向の選択はそれらに直接影響を与えるためである。また，Web 特有の動的なレイアウト決定の影響を一番うけるのは，この書字方向の指定である。そして，実際に実装しているブラウザが Internet Explorer というプロプライエタリなソフトウェアのみであり，

[11] `http://taketori.org/js.html`
[12] `https://code.google.com/p/nehan/`

ソースコードを参照できない。そのため，ソースコードを参考にして実装することが不可能であるし，実装時の問題点が見えにくい。このような状況で，具体的な問題点がはっきりと見えていないというのが一番の問題点であると考えられる。

第4章　実装

本章ではルビ表示と書字方向選択の実装について解説する。4.1節では本実装の概要とその目的について述べる。4.2節では実装対象となるMozilla Firefoxとそのレイアウトエンジンである Geckoの詳細やその他の技術について説明する。最後に4.3節と4.4節で，二つの実装の詳細について述べる。

4.1　目的と概要

本研究の実装の目的は，実際にルビと書字方向の選択を実装し，3.5節で指摘した問題点を解決することである。それに加えて，実装することで得られた知見や新たな機能を，仕様の一部として提案することである。本研究で注目しているルビの表示と書字方向の選択の共通の問題点として，実装例が少ない，または，まったく無いということがある。また，そのため仕様へのフィードバックがなく，仕様の策定が進まない。この二つは相互依存であり，実装があることで仕様の策定も進むし，仕様の策定が進めば実装も増える。そこで，実装を行うことで，ブラウザでその機能を利用できるようにすること，また仕様に関しての知見を獲得することを目標とする。

本研究では既存のオープンソースブラウザである Mozilla Firefoxを用いて，ルビ表示と書字方向の選択を実装する。この理由として，まず実装の無駄を省くことが挙げられる。また，Webの利点として既存のブラウザが高機能であることを挙げており，それらの機能を本研究の実装の中でそのまま利用するためである。加えて，Firefoxはオープンソースなソフトウェアであり，ソースコードを公開することで世界中の Firefoxユーザがこれらの機能を自由に利用することが可能である。他にも，現在，Firefoxの市場のシェアはおよそ22%であり，Microsoft Internet Explorerに続いて，二番目に多く利用されるブラウザである[88]。このようにフリーソフトウェアのブラウザとして一番シェアが高いことも本研究において採用した理由である。

4.2 Mozilla Firefox

4.2.1 概要

Mozilla Firefox は Mozilla 財団[1] が開発しているオープンソース Web ブラウザである。Netscape Communications 社が開発していた Netscape Communicator のソースコードを引き継いで開発が行われている。Windows，Mac OS X，Linux での動作が正式にサポートされているマルチプラットフォームソフトウェアである。Firefox は Web ページの描画のため，Gecko というレンダリングエンジンを搭載している。この Gecko のレンダリングエンジンは Google Chrome や Safari が採用する Webkit と似通っており，本研究において得られた実装は他レンダリングエンジンへの実装でも参考にもなると考えられる。以下ではこのレンダリングエンジン Gecko におけるレイアウト決定処理の詳細について説明する。また，Gecko のインタフェース作成言語である XUL とレンダリングライブラリである Thebes Gfx についても解説する。

4.2.2 レンダリングエンジン・Gecko

レンダリングエンジンとは Web ページを表示するための構文解析，レイアウトの計算などを行なうソフトウェアである。代表的なレンダリングエンジンとして，Internet Explorer の Trident や Safari，Google Chrome の Webkit などが挙げられる。Gecko は Firefox だけでなく，メーラーである Mozilla Thunderbird などにも組み込まれ，GUI を構築するためにも用いられている。以下では Gecko の内部処理について説明する。

図 4.1 は Gecko の内部処理の概要を示している。この図において，長方形は実行する処理自体を表し，平行四辺形は保持されるデータを表している。HTML の読み込みから実際の表示までの処理の概要ついて順に説明する。まず，HTML ファイルが読み込まれると Parser が字句解析と構文解析を行いながら Content Sink に結果を渡す。Gecko はネットワーク越しにファイルを取得しながら，Parser で字句解析と構文解析を行いつつ，Content Sink に構文解析結果を保存していく。同時に Content Sink 内では，HTML の構造を保持する Content Tree の生成と，スタイルを決定する Style Sheets に分離する処理を行う。Style Sheets は CSS Parser に渡され，カスケード処理を行ってスタイル情報を保持する Style Rules を構築する。

次に，HTML の構造を保持する Content Tree とスタイル情報を保持する Style Rules の二つを Frame Constructor に渡す。Frame Constructor は，渡されたこの二つの情報を基に Frame

[1] http://www.mozilla.org/foundation/

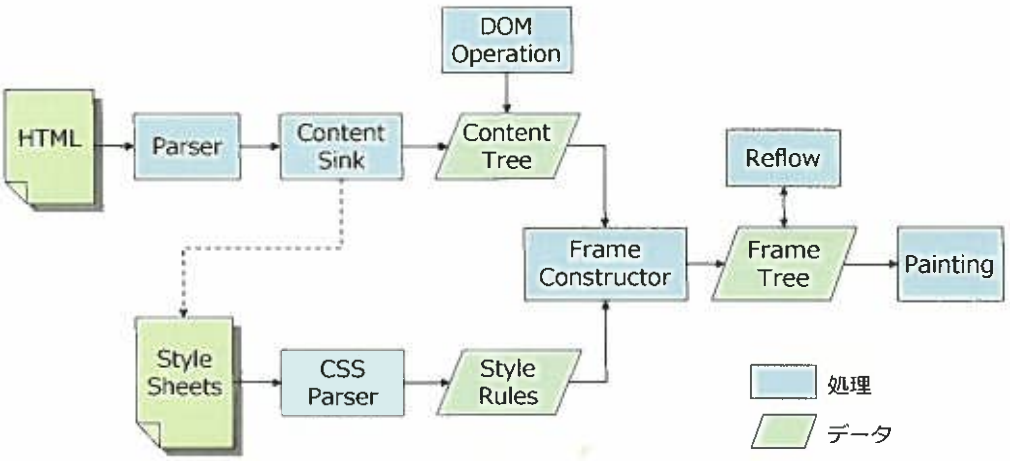


図 4.1: Gecko における処理とデータ (出典: 文献 [92])

Tree を構築する。Frame Tree とは Gecko においてページレイアウトに対応する構造であり，Web ページのスタイルを決めるために用いるものである。しかし，Frame Tree を構築した時点では，まだ具体的なレイアウトはまだ決定されていない。この時点では， Frame Tree は Frame 同士の関係のみを保持している。実際に，この Frame Tree に対してレイアウトの計算を行なうのは Reflow という処理である。Reflow では再帰的に Frame Tree をたどっていき，座標や幅，高さを計算する。その後，Reflow によって決定されたレイアウトを基に Painting 部分で描画を行う。

また，JavaScript などを用いて DOM 操作が行われた場合には，Content Tree に直接変更が加えられる。そうすると，FrameTree がそれに対応して再構築され，Reflow でレイアウトの再計算を行う。以下では，この中でも今回の実装と関係が特に深い，Frame Construction という処理と Reflow について以下で詳しく説明する。

Frame Construction

Frame Construction は Content Tree と Style Rules を基に，Frame Tree を構築するための処理である。図 4.2 は，構築された Frame Tree と Content Tree，Style Rules との関係を示している。基本的に Frame Tree は Content Tree の構造に対応して構築される。Content Tree を行きがけ深さ優先法でたどっていき，Content ノードに対応して同種類の Frame を生成する。例えば，divContent であれば，BlockFrame を生成する。Content Tree では HTML の構造の情報を保持するため，要素名に直結する div や p という情報を保持している。一方，Frame Tree の

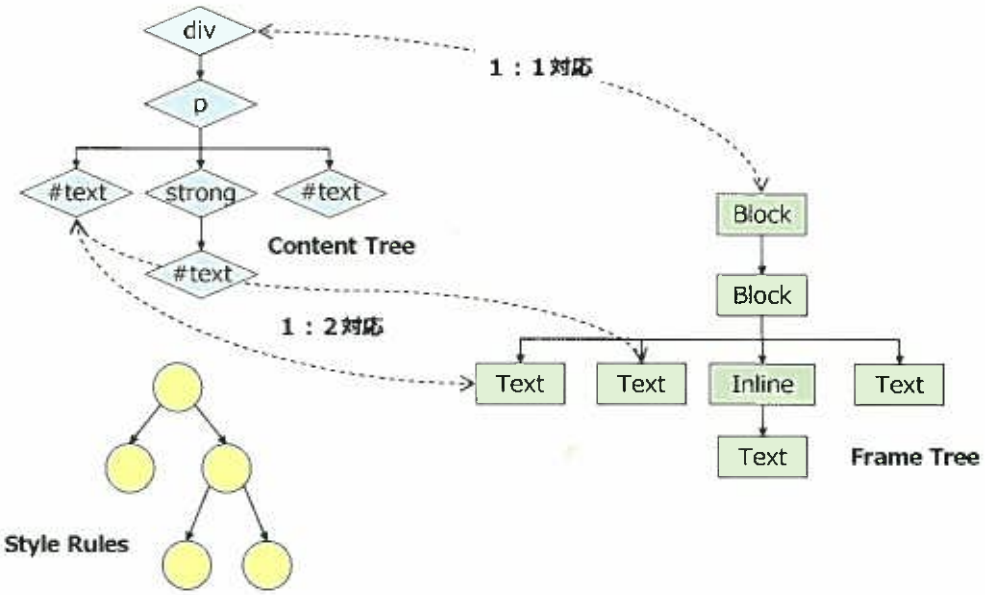


図 4.2: Content Tree, Frame Tree, Style Rules

場合はにはスタイルを決定するための情報だけが分かればよい。そのため，BlockFrame のようにスタイル上の情報のみを保持する。たとえ Content Tree において divCotent や pContent という Content ノードとしての違いがあっても，同じ BlockFrame という Frame を生成する。また，Content Tree 上では一つの#textContent であっても，表示できるブロックの幅によって，複数行へ分割される。そのため，図中左側の#textContent が二つの TextFrame に分割されていることが分かる。このように，一部のレイアウトでは特殊な処理を行って Frame Construction を実装しなければならない。

Reflow

Reflow は構築された Frame Tree に対してレイアウト計算を行う処理である。Reflow では帰りがけ深さ優先法で Frame をたどっていき，座標や幅，高さを決定する。つまり，子 Frame（より小さい部分）のレイアウトを決定してから，それを基に親 Frame（より大きい部分）のレイアウトを決定している。Reflow を行う際，各 Frame には ReflowState と ReflowMetrics という二つの情報が渡される。行きがけには ReflowState が，帰りがけには ReflowMetrics が各 Frame に渡される。

まず ReflowState について説明する。ReflowState は子 Frame にレイアウトの制限を与えるための情報である。例えば，幅や高さ，座標などの制限である。図 4.3 において，一番上位

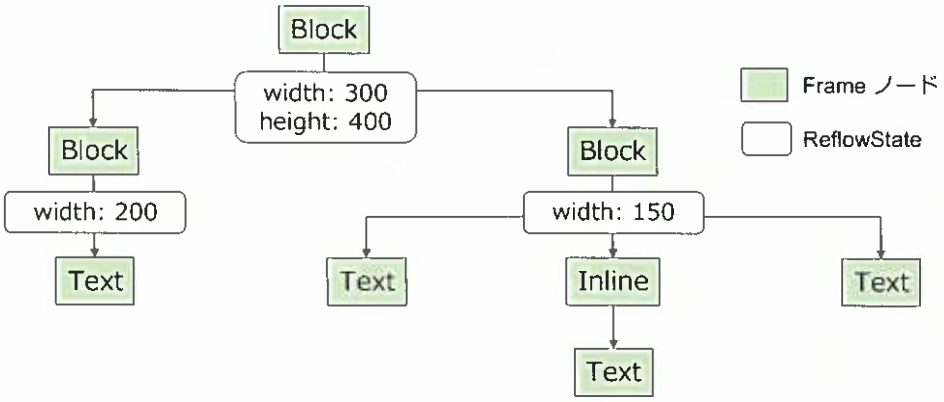


図 4.3: ReflowState の例

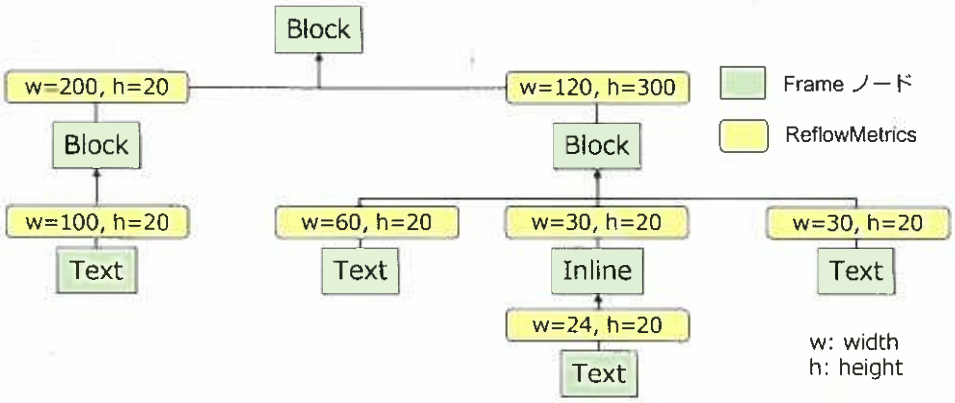


図 4.4: ReflowMetrics の例

になる BlockFrame には『幅は 300，高さは 400』というスタイル指定がある。その場合には，子 Frame も幅が 300 以下にならなければならない。そのため，ReflowState を用いて，その制限情報を子 Frame に渡す。同様に二つの Block Frame においても幅の指定（幅 200 と幅 150）があるので，それらの制限を子 Frame にを渡す。このようにして，ReflowState はレイアウト計算において親 Frame の制限を子 Frame に伝える役割を担っている。実際にはこれら以外にも ReflowState は様々な情報を保持する役目を担っている。より詳細な情報は Waterson による技術ノート [93] に記載されいている。

次に，ReflowMetrics について説明する。ReflowMetrics には子 Frame で実際に計算されたレイアウト情報が保持される。図 4.4 には，各 Frame が実際に計算したレイアウトの値を，ReflowMetrics として親 Frame に渡している例を示す。例えば一番左の TextFrame では，レイ

アウト計算によってFrameの幅が100, 高さが20であった。よって, その情報をReflowMetricsとして親Frameである BlockFrameに渡している。しかし, その親Frameである BlockFrameでは『幅は200』という制限情報がReflowStateで渡されているため, この BlockFrameの幅は200にする必要がある。そして, そのようにして決定されたレイアウト情報をReflowMetricsとして親Frameへと渡していく。このようにしてReflowでは, 子Frameから親Frameへとレイアウトを計算していき, 最終的に全てのレイアウトを計算する。

4.2.3 その他の技術

XUL

XUL (XML User Interface Language) は XMLベースで, GUIインタフェースを設計することができるユーザインタフェースマークアップ言語である。XMLベースの仕様になっていて習得が容易であり, クロスプラットフォームなアプリケーションインタフェースを開発することができる。XULではGUIアプリケーションを構成する様々なオブジェクトをXMLの要素として表現することができる。

図4.5はXULのソースコードの一例である。<window>はアプリケーションのウインドウ自体を表しており, <button>はウインドウ内のボタンオブジェクトを表している。このXULファイルを実行すると, 図4.6のようにウインドウとボタンを持ったGUIインタフェースを作成することができる。XULはFirefoxやThunderbirdのインタフェースは, この言語を用いて設計されている。また, Firefoxの多くのアドオンがこのXULを利用して新たな追加機能を実装している。

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window id="example-window" title="Example 2.2.1"
     xmlns:html="http://www.w3.org/1999/xhtml"
     xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <button label="Normal"/>
  <button label="Disabled" disabled="true"/>
</window>
```

図4.5: XULのソースコード例

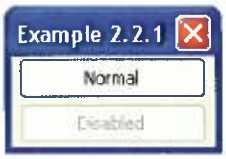


図 4.6: XUL で作成したアプリケーションウインドウの例

Thebes Gfx

Thebes Gfx はレンダリングのための API を提供するグラフィックライブラリである。レイアウトエンジンである Gecko は，最終的にこのライブラリの API を呼び出し，実際の図形や画像，テキストの描画を行う。図 4.7 は Thebes Gfx の概要であり，同図が示すように Thebes Gfx は二つの部分に分けることが出来る。Thebes Gfx は外部ライブラリである cairo の C++ ラッパーと部分である『Thebes GfxShape』，多言語テキスト描画処理部である『Thebes Gfx Text』の二つの部分に分けられる。

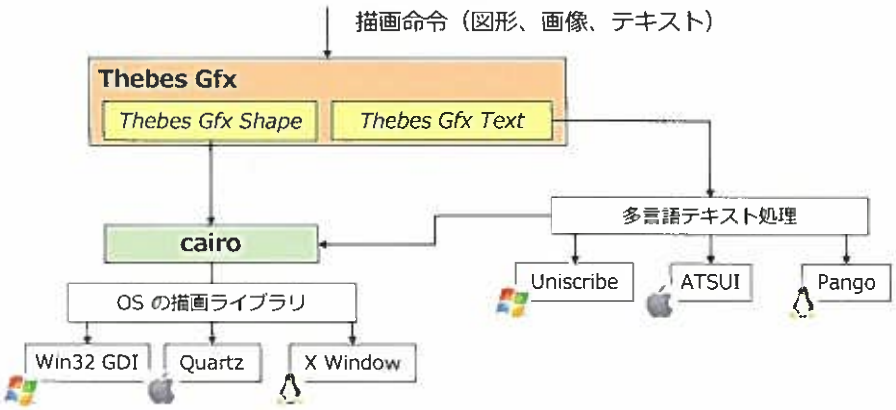


図 4.7: Thebes Gfx の概要図

まずはグラフィックエンジン cairo [2] について説明する。cairo はデバイスに依存しないベクトルグラフィックベースの API を提供するライブラリである。バックエンドとして OS 固有のレンダリングシステムを利用することができる。具体的には Linux の X Window System，Windows の Win32 GDI，Mac OS の Quartz などである。また，PostScript，SVG，PDF などのファイルへの出力も可能である。

[2] http://cairographics.org/

多言語テキスト処理は，ユニコードバイト列を適切にグリフ列へと変換する処理である。例えば，アラビア語の筆記体形などをグリフ列で表示するためには，グリフを置換するための特殊な処理が必要である。このような多言語テキスト処理について Thebes Gfx では各 OS に独自のシステムを利用している。Windows は Uniscribe，Mac OS X は ATSUI，Linux は Pango である。これらを利用することで複雑なグリフ処理が必要な多言語テキストを，単に Unicode バイト列を渡すだけで適切なレンダリングが可能である。Thebes Gfx についてのより詳細な説明は文献 [94] に記載されている。

4.3 ルビ表示の実装

4.3.1 要件と概要

ここでは，3.5.2 項で指摘した問題点を解決するための実装について述べる。この問題点を解決するため，ルビに表示の実装に必要な要件は以下のようなものである。

1. Firefox でのルビ表示の対応
2. XHTML 複雑ルビの完全な実装
3. HTML5 と XHTML の両方へのマークアップ対応
4. 妥当でないマークアップへの対応
5. CSS による詳細なスタイル指定への対応

まず，Firefox でルビの表示を可能にする。これによって，ほぼシェアの 100%を占めるブラウザで簡単なルビの表示が可能になる。次に他のブラウザでは実装されていない複雑ルビの完全な実装を行う。加えて，HTML5 と XHTML の二つのマークアップに対応することを目標とする。これを実現するため，マークアップではなく表示のみに注目したルビ用の Frame モデルを提案し実装する。そして，そのためのレイアウト計算処理を実装する。他にも，マークアップが妥当でない場合にも，提案した Frame が適切に構築されるような仕組みを実装する。また，詳細なスタイル指定を可能にするための CSS の各プロパティを実装する。特に `ruby-span` プロパティや `ruby-position` プロパティはブラウザ上で初めて実装される機能である。

4.3.2 ルビ用 Frame の提案と実装

本項では，ルビを表示するために必要な Frame について提案する。本研究において提案し，実装したルビ用の Frame は以下の三種類である。

1. nsRubyFrame (Ruby)
2. nsRubyContainerFrame (Container)
3. nsRubyCellFrame (Cell)

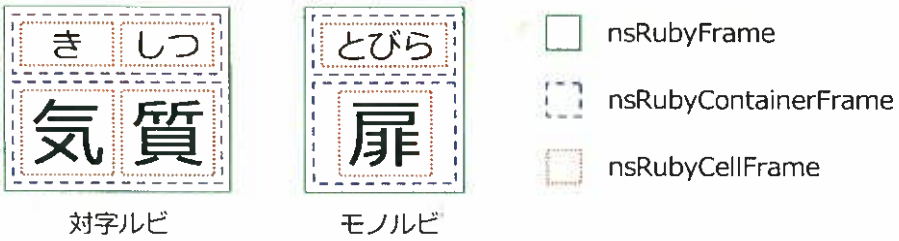


図 4.8: ルビ用 Frame の概要

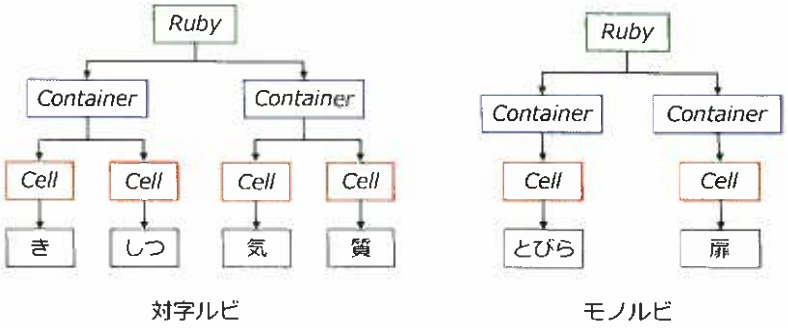


図 4.9: ルビ用の Frame に関する親子関係

図 4.8 は本研究において提案し，実装したルビ用 Frame の概要である。図 4.9 は，Frame の親子関係を示しており，同図のような Frame Tree を生成することを目標とする。nsRubyFrame はルビ全体のレイアウトに対応する Frame であり，0..n 個の nsRubyContainerFrame を子供に持つ。nsRubyContainerFrame は親文字列全体，ルビ文字列全体に対応する Frame であり，0..m 個の nsRubyCellFrame を子供に持つ。nsRubyCellFrame は部分文字列に分割されたル

ビ文字列や親文字列に対応し，1 個の BlockFrame を子供に持つ。以下では簡略のために，nsRubyFrame を Ruby，nsRubyContainerFrame を Container，nsRubyCellFrame を Cell と呼ぶ。

この Frame モデルは，CSS3 Ruby Module [83] のボックスモデルを参考にしている。しかし，本 Frame モデルの特徴として，<rtc> や <rtb> などの Container に対応するマークアップやスタイル指定がない場合の対処が挙げられる。CSS3 のボックスモデルでは簡単ルビの場合には Container に対応するボックスが存在しない。しかし，本モデルでは簡単ルビの場合でも Container を生成する。これによって，簡単ルビと複雑ルビを一つの Frame モデルで扱うことが出来るようになる。正確には，簡単ルビのボックスモデルを複雑ルビの一つのレイアウトに内包することができる。これによって，レイアウトを計算する Reflow の際に，簡単ルビと複雑ルビで処理を分ける必要がなくなる。

4.3.3 Frame 構築の実装

ここでは Content Tree から，前項で述べた Frame に準じたルビ用の Frame Tree を構築するための処理について説明する。Frame を構築する以前に，XHTML と HTML5 のマークアップの差，妥当でないマークアップなどが原因で様々な Content Tree が生成されていることが考えられる。そのため，どのような Content Tree であっても，提案に準じた Frame Tree を構築するための手法について提案し，実装する。

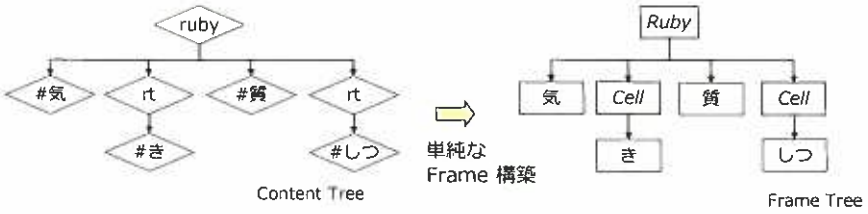


図 4.10: HTML5 マークアップ場合の Content Tree と，それを基に単純に生成した場合の FrameTree

ここでは，HTML5 のマークアップを例にして，適切な Frame を構築するための処理の詳細について述べる。まず，HTML5 のマークアップの場合には，図 4.10 の左図のような Content Tree が構築される。一般的な Frame 構築規則である『1 対 1 の生成』という規則に従うと，この Content Tree から図 4.10 の右のような Frame Tree が構築される。しかし，この Frame Tree は提案したモデルとは異なるため，以下で解説する補完処理が必要である。

まず，この場合，個々の親文字列（「気」や「質」）に対応する Cell が構築されていないことが分かる。それらの Frame を構築し，適切な位置に補完する処理が必要である。図 4.11 に示すように，Ruby 直下にあって Cell ではない Frame の親に Cell を補完する処理を行う。ここで，もし，その Cell でない Frame が CSS のインライン以外の場合には，その Frame をここで破棄する。これは，HTML5 の仕様においてインラインに対応するマークアップは認めているためである。

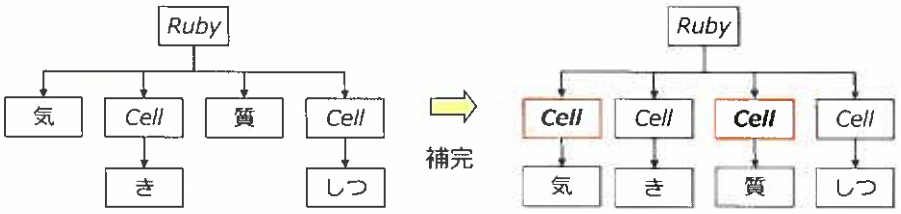


図 4.11: nsRubyCellFrame の生成と補完処理

次に，Container の補完処理について説明する。XHTML の簡単ルビと HTML5 のマークアップでは，明示的に Container に対応する Content ノードが生成されていない。そこで，図 4.12 に示すように Container の補完処理を行う必要がある。この際，単に補完処理を行うだけでなく各 Cell の順番の入れ替えも行う。なぜなら，親文字列に対応する Container の持つ全ての Cell は，部分親文字列に対応しなければならないからである。これはルビ文字列の場合も同様である。

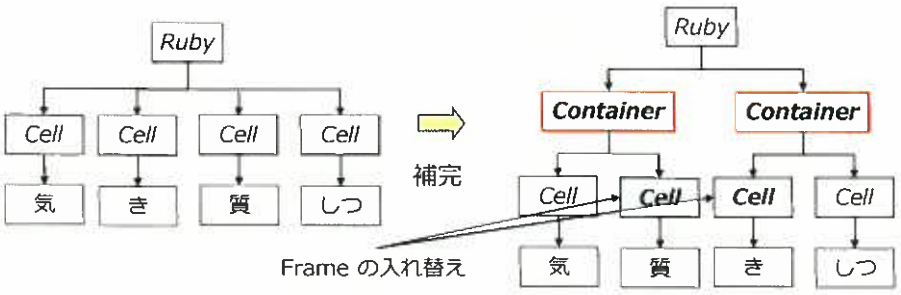


図 4.12: nsRubyContainerFrame を用いた補完と入れ替え処理

また，これ以外にも妥当でないマークアップへ対処するため，似たような補完処理を実装している。例えば，図 4.13 の上部のようなマークアップの場合には，同図の下部のようなマークアップだと解釈する。本研究では，このような妥当でないマークアップにする対応を

20 個程度考え，それらの意図するマークアップとの対応を考えた。これらの全ての対応は，本論文の付録に添付している。

```
妥当でないマークアップ：
  <ruby><rb>聡</rb>子<rt>さと</rt><rt>こ</rt></ruby>

解釈するマークアップ：
  <ruby>
    <rbc><rb>聡</rb><rb>子</rb></rbc>
    <rtc><rt>さと</rt><rt>こ</rt></rtc>
  </ruby>
```

図 4.13: 妥当でないマークアップと解釈するマークアップ

4.3.4 レイアウトの計算と実装

本節では，提案した Frame Tree のためのレイアウト計算処理の詳細ついて説明する。ルビ用の Reflow は以下のような手順で，全ての Frame のレイアウトを決定する。まず，Ruby の子孫となる全ての Cell の大きさ（幅と高さ）と座標を決定する。その大きさは，Cell が持つ BlockFrame の大きさと同じにする。例えば Cell が『扉』という文字を持つ TextFrame だった場合には，その TextFrame の大きさを決定し，その大きさと同じ値に Cell を設定する。その結果，図 4.14 の左側に示すように，全ての Cell の大きさと座標は，その子供の Frame のサイズに依存した未整列の状態になる。

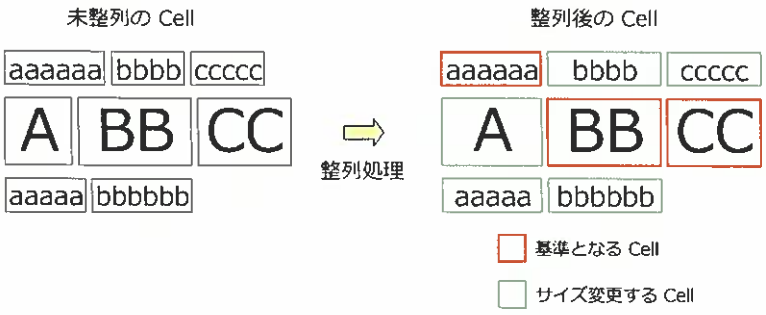


図 4.14: Cell の整列処理

次に各列で最大の大きさになるCellを探し，それを基準のCellとする。そして，同列にあるその他のCellを，基準としたCellの大きさにあわせる。また，各Cellのx座標はそれまでのCellの幅の和にする。例えば，3列目にある全てのCellのx座標の値は2列目までのCellの幅の和になる。このような処理をすることで同図の右側のようにCellが全て整列された状態になる。以下に，各Frameの大きさの決定についての詳細を記す。

1. Ruby
    - 幅 = Containerの幅
    - 高さ = 全てのContainerの高さの和
2. Container
    - 幅 = 子Cellの幅の和
    - 高さ = 子Cellの高さ
3. Cell
    - 幅 = 上述の方法で決められた幅
    - 高さ = 子Frameの高さ

4.3.5 各種CSSプロパティの実装

ruby-alignプロパティ

ここでは，`ruby-align`プロパティの実装について説明する。本研究の実装では，`start`，`center`，`end`のプロパティ値に対応している。`ruby-align`の実装は，Cell内のBlockFrameの位置を変えることで実現している。例えば，図4.15に示す`center`の場合では，Cellの幅から子BlockFrameの幅を引いて，その値を二で割ることでx座標の値を求める。前項の処理をすることで，同じ列のCellは全て同じ幅であるので，全てのCellにおいて同様な処理を行えば`ruby-align`の計算を適切に行うことができる。

図4.15: ruby-alignプロパティの実装

ruby-overhang プロパティ

ここでは ruby-overhang の実装について説明する。まず，ルビの Reflow 処理で Frame のレイアウトが計算された際に，それと同時にルビをかける量を計算し，内部にその値を保持しておく。次に，行のレイアウトの計算をする処理で，保持しているルビかけ量の分だけ Ruby とその次にある Frame の x 座標値をずらす。具体的には，図 4.16 に示すように Ruby とその後ろの TextFrame の x 座標値から計算したルビかけ量だけ引き算する。しかし，厳密な JIS X 4051 の規定では前後の文字クラスなどによってルビかけ量は異なるが，本実装ではそのような処理は行っていない。

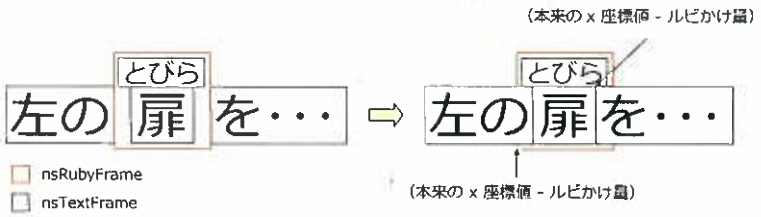


図 4.16: ruby-overhang プロパティの実装

ruby-span プロパティ

最後に ruby-span の実装について説明する。本実装では，HTML の Table に対して rowspan 指定された場合の処理のアルゴリズムを参考にしている。また，本項では ruby-span に 1 よりも大きい値が指定されている Cell のことを『span 指定された Cell』と記述する。

ruby-span の実装のため，Ruby の Frame モデルに新たに架空の Cell という概念を導入する。架空の Cell とは span 処理を実現するための Cell であり，図 4.17 に示すように，span が指定された Cell の後ろに指定の数だけ，架空の Cell があると考える。ただし，架空の Cell は実際には実装せず，各列の Cell を取得するメソッドを実装する際，もしその Cell が架空の Cell だった場合，幅を 0 として返すよう実装したのみである。

次に，全ての Cell に対して 4.3.4 項で述べた幅の計算処理を行う。この処理において，架空の Cell と，span 指定された Cell は幅を 0 とみなし，基準となる Cell を決定する。また，幅の大きさの変更に関して，span 指定がされている Cell であっても，span 指定がなされていない Cell と同様の処理を行う。

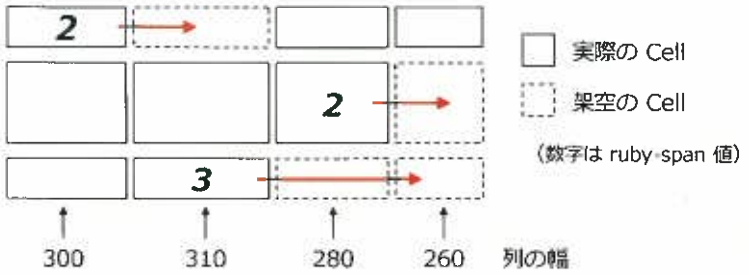


図 4.17: ruby-span 実装のための架空の Cell の概要

次に span 指定さた Cell のみに対して，span 指定されている数だけ列の幅を足した値を計算する。具体的には図 4.17 の赤矢印が示すように，span 指定された Cell の右側の列の幅，指定された数だけ足した値を計算する。例えば，同図の最下段にある span 値が 3 の Cell の場合は，$310 + 280 + 260 = 850$ となる。

ここで図 4.18 に示すように，span 指定された Cell の大きさによって二つのケースに処理を分ける。同図の (1) の場合のように，span 指定された Cell の中にある Frame の幅が span 指定された分の列の幅の和より小さい場合と，(2) のその逆の場合である。これら二つの場合で，異なる幅の変更処理を実行しなければならない。

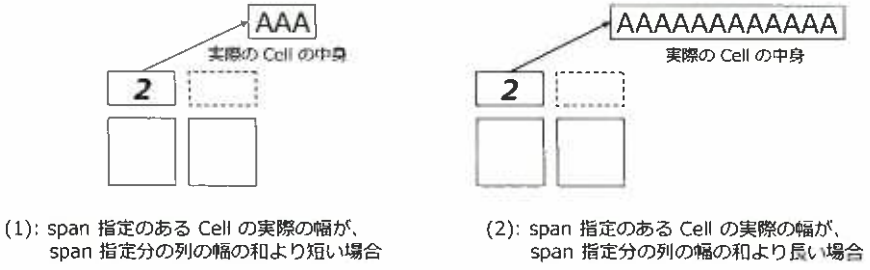


図 4.18: ruby-span 処理のためのサイズ変更に関する二つのケース

まず (1) の場合，図 4.19 の左図のように，span 指定された分の列の幅の和を，span 指定された Cell の幅に設定する。(2) の場合，(1) の処理を行うと文字列が収まりきらないので，同図の右側のように span 指定された列の幅自体を変更する必要がある。また，(2) の処理を行うと列の幅が変更され他の Cell の幅と座標もそれにあわせて変更する必要がある。

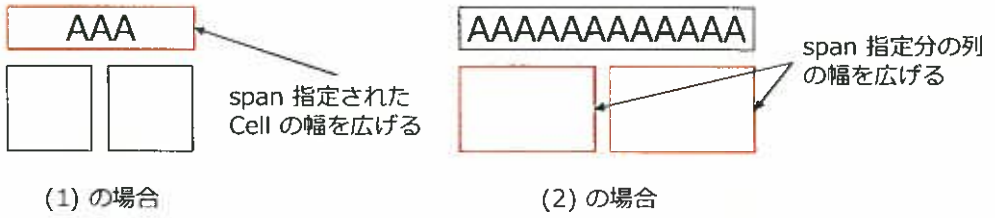


図 4.19: ruby-span 処理のための二つのサイズ変更方法

4.4 書字方向選択の実装

4.4.1 要件と概要

ここでは，3.5.3 項で指摘した問題点を解決するための実装について述べる。これらの問題点を解決するために必要な要件は以下のようなものである。

1. 既存のブラウザへの追加実装
2. 実装レイヤの提案
3. 全ての書字方向への対応
4. 動的なレイアウト変更への対応

今回の実装では，縦書き専用のブラウザを一から実装するのではなく，横書きのみが可能な既存のブラウザへ追加実装する。これは，既存のブラウザが様々な機能を持つ高度なレンダリングエンジンを持っているためである。そのため，既存の横書き用レイアウトエンジンに，縦書きの機能を追加するという形をとる。横書きをするレイアウトエンジンのコードを直接は変更せず，横書き用に計算されたレイアウトに対して，座標変換をすることで書字方向選択を実現する。この手法は，TEX などの文書作成ソフトウェアで縦書きをサポートする際に用いられる手法である。

また，書字方向指定の実装には広範囲に関する実装が必要である。そこで，実装をいくつかの段階に分けるため実装のレイヤを提案する。そして，各レイヤに対して個別に実装方法を提案し，実装する。こうすることで，広範囲に渡る実装を段階的に行うことができる。加えて，国際化という点から，単に日本語で用いられる右から左への縦書きのみではなく，モンゴル文字で書かれるモンゴル語の左から右への縦書きなどへも対応する。また，Web 特有の動的な変更に対応するように実装する。例えば，ウインドウの幅が変更された際には，その幅に対応して行幅が再計算されるようにする。

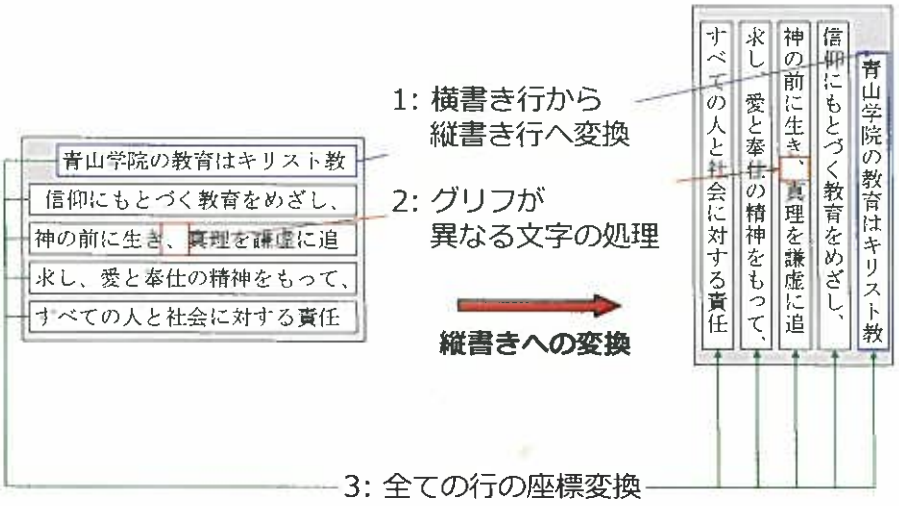


図 4.20: 横書きから縦書きへの変換

4.4.2 実装レイヤの提案

本節では書字方向の選択を実装するための実装レイヤの提案と，各レイヤの詳細について述べる。本実装では，図 4.20 に示すように，横書き用に計算されたレイアウトに対して，回転と座標変換を用いて指定された書字方向への変換を行う。この処理は，以下のような三つの部分に分割することができる。まず，縦と横でグリフが異なる文字の処理，一つ一つの行を横書きから縦書きへと変換する処理，全ての行の座標（順番）を入れ替える処理である。

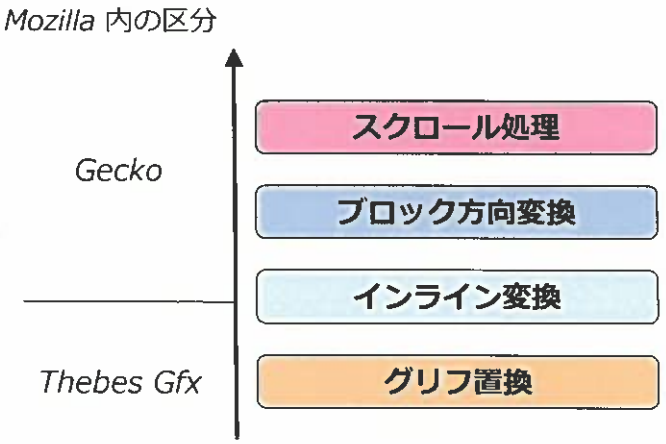


図 4.21: 縦書きの実装レイヤ

これらの処理を個別に実装するため，実装レイヤとして実装を分離して考える。本研究では，図 4.21 のような書字方向指定の実装のためのレイヤを定義した。このレイヤでは，下に位置するレイヤほど低いレベルの実装，より OS 側の処理に近い実装が必要であることを示している。このように各レイヤに分離する利点として，まず実装を独立して考えることができる。そして，本研究において全てのレイヤを 100%実装できなかったとしても，異なるレイヤの実装を他の実装に取り替えることも容易である。また，一部のレイヤの実装を変えた場合に，レイヤに分離した実装をしているため，変更が受け入れられやすい。以下では，各レイヤーで実装する処理の概要について述べる。

グリフ置換レイヤ

グリフ置換レイヤは，横書きと縦書きで異なるグリフを用いる文字の処理を実装するレイヤである。漢字圏においてほとんどの文字は横書きと縦書きで同じグリフを使うことができる。しかし，図 4.22 のように“「”や“。”などの約物に関しては，横書きと縦書きで違うグリフを使う必要がある。また，`text-orientation` プロパティに対応するためには，指定された部分のグリフを適切な角度に回転させる必要がある。

そこで，このレイヤではグリフの置換処理と，回転する文字列部分に対してグリフを回転させる処理が必要である。OpenType フォントは内部で Glyph Substitution Table という置換テーブルを持っている [95]。これらは一定のグリフをある条件によって別のグリフへと置換するためのテーブルである。例えば，アラビア文字では同じ文字であっても，単独系，頭字，中字，尾字でグリフが異なる。そのような場合には，この置換テーブルを用いて適切なグリフへと置換する。これには縦書き用のグリフへと置換するためのテーブルもあり，本レイヤの実装では，このテーブルにアクセスするための実装が必要になる。また，グリフの回転は Thebes Gfx へと回転フラグを渡して処理する必要がある。

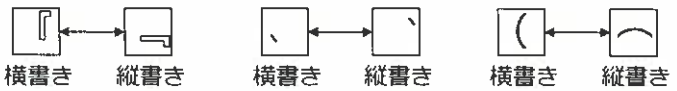


図 4.22: 横書きと縦書きでのグリフの差

インライン変換レイヤ

インライン変換レイヤでは横書きとして生成された行を，縦書きに変換する処理を実装する。今回は横書きレイアウトからの変換処理の実装であるため，縦書きの行を直接生成することはしない。図 4.23 に示すように，『グリフの反時計回りの回転』と『行全体の回転』という二つの処理を通して，横書き行から縦書き行へと変換する処理を行う。まず，全てのグリフを反時計回りに 90 度回転させた状態にし，その行全体を右に 90 度回転させることで縦書きの行を実現する。全ての個々のグリフの回転は，グリフ置換レイヤでの文字列の回転処理と同じように，Thebes Gfx 部分での処理が必要である。行全体を回転する処理は，一つの行に対応する Frame である nsTextFrame を実際に描画する際に，回転を行ってから描画をするように実装する。

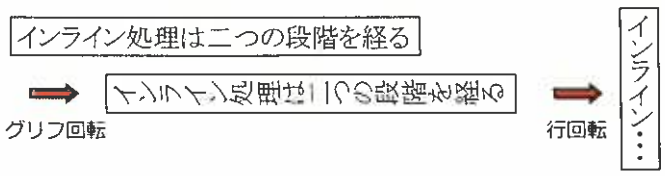


図 4.23: インライン処理の二段階

ブロック方向処理レイヤ

ブロック方向処理レイヤでは，上から下への横書きとして生成されたレイアウトを，指定されたブロック方向へと変換する処理を実装する。ブロック方向処理レイヤの実装には，座標変換処理を用いる。まず，既存のレイアウトエンジンを用いて，上から下への横書きの行を持つレイアウトを決定する。その結果を，各書字方向に応じて適切な座標変換をすることで，行の進む方向を変える。

このレイヤーでは単純な文字で構成される行以外の挙動も考慮する必要がある。Web では画像や動画，テーブルやフォーム，ボタンなどといった様々な要素がブロック内に入る。例えば，縦書きの場合，画像に関しては位置座標の変更が必要であるが，画像自体の回転は必要ない。このように，Web ページ上のどの要素を回転し，どの要素は回転しないのかをあらかじめ決めて実装しなければならない。

スクロール処理レイヤ

スクロール処理レイヤでは，書字方向指定の方向に応じてスクロールの挙動を変更するための処理を実装する。従来の横書きのページでは一般的に，上下にスクロールする。また，ページにおけるスクロールの初期位置はページの上部である。これらの挙動は書字方向の指定によって異なる。例えば，右から左への縦書き行を持つ日本語ページであれば，ページのスクロールの初期位置はページの右部分である。そして，この場合，ページは左右にスクロールしなければならない。また，モンゴル語の場合では左から右に縦書きの行が進むので初期位置はページの左部分であり，上の場合と同じように左右にスクロールする。しかし，Web ページ全体の書字方向が横書きであり，その一部のみに縦書きが指定されている場合にはそのような必要はない。これに加えて，インライン方向によってもスクロールの初期位置は異なる。例えば，英語のページの初期位置は左上部であるが，アラビア語のページの初期位置は右上部である。

4.4.3 書字方向変更のための Frame の導入

書字方向指定を実装するため，nsBlockFlowFrame (以下 BlockFlow) という Frame を新しく導入する。BlockFlow は指定された書字方向へ変換を行うための各処理を実行するための Frame である。この Frame を新たに導入することで，各変換処理のほとんどをこの Frame 内に実装することができる。そのため，既存のソースコードへの変更は必要最小限ですむ。

nsBlockFlowFrame は，図 4.24 のように Content Tree から Frame Tree を構築する際に，書字方向が指定された部分に自動的に挿入される。この挿入は，親の Content ノードが持つ `writing-mode` プロパティの値と，子の Content ノードが持つ `writing-mode` プロパティの値が変わった場合に行われる。たとえ，親 Content ノードの `writing-mode` 属性に同じ値が指定されている場合には生成と挿入はされない。しかし，`<body>` に `writing-mode` が指定され，ページ全体の書字方向が変更された場合には挿入位置が異なる。これの詳細はスクロール処理の項で述べる。

4.4.4 グリフ処理とインライン変換の実装

本節では，グリフ処理とインライン変換の実装について述べる。この二つの処理は，OS 側の処理に近いため Windows と Linux で異なる実装をする。本来であれば，4.4.2 項で述べたように，フォントから縦書きのグリフを取り出すような実装を，プラットフォームに依存

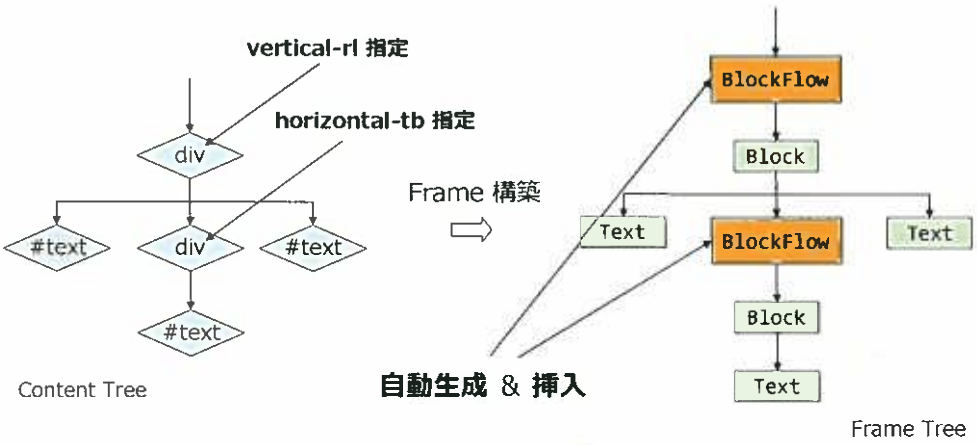


図 4.24: BlockFlow の挿入位置

しない形で行うべきである。図 4.25 は，グリフ処理とインライン変換の処理と，各 OS での実装の概要である。同図の数字は以下の処理を示している。

1. 縦書きグリフへの置換 (グリフ処理レイヤ)
2. グリフを左 90 度回転 (インライン変換レイヤ)
3. 行を 90 度回転 (インライン変換レイヤ)

ここで同図の ③ の処理は，Windows と Linux で共通の処理を行っている。以下では，① と ② の処理について，各 OS での実装の詳細について述べる。また，今回は Mac OS X のような，上記二つの OS 以外には対応していない。

Windows での実装

Windows における実装では，@マークの追加による縦書きフォントを利用する。具体的には，『@ＭＳ 明朝』のようにフォント名の頭に@マークを指定をすることで，グリフに関する以下処理を Windows 側で行うことができる。

1. 必要な場合は，縦書きグリフへ置換 (Windows GDI のみ)
2. 個々のグリフを 90 度左向きに回転

今回は，この二つの機能を用いてグリフ処理とインライン変換を実装した。まず，図 4.25 の ① に対応するグリフ処理は，上述リストの 1 の機能を用いる。次に，同図の ② に対応す

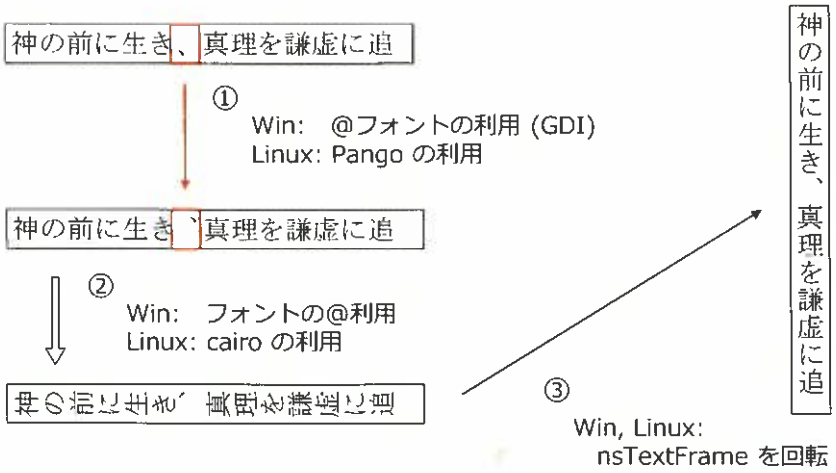


図 4.25: グリフ処理とインライン変換の実装概要

るインライン変換の最初の処理は，上述リストの 1 の機能を用いる。この二つの処理を利用することによって実装した。

Linux での実装

Linux の場合には，多言語テキストレンダリングを行う Pango[3] を利用する。Firefox では，すでに Pango を利用して他言語テキストのグリフ生成を行っている。Pango では縦書き用のフラグを与えることで，縦書き用のグリフの置換を行うことができる。そこで，図 4.25 の ① に関する処理は，Pango を用いることで実装できる。同図の処理 ② は，cairo に用意されている API を用いてグリフ回転を行う。具体的には，cairo でレンダリングを行う際にグリフの回転行列を与える。これらの処理を用いて，Linux においてグリフ処理レイヤとインライン変換レイヤの処理を実装した。

4.4.5 ブロック方向処理の実装

処理の概要と流れ

ブロック方向の変換処理を行うために実装した二つの処理について解説する。それは『指定方向の入替処理』と『座標変換処理』である。これら二つの処理を，図 4.26 のようにして Reflow 処理に組み込み，ブロック方向処理を実装する。

[3] http://www.pango.org/

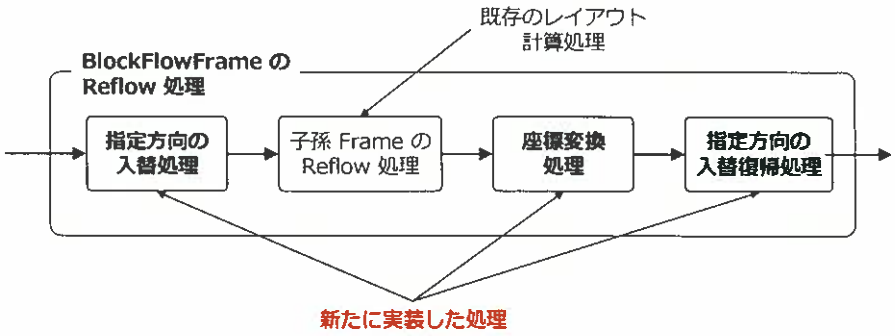


図 4.26: BlockFlowFrame の Reflow 処理の流れ

BlockFlow の Reflow 処理の流れについて説明する。まず，BlockFlow の Reflow 処理の最初に，座標変換処理のために指定方向の入替処理を行う。その後，BlockFlow が持つ子孫の Frame に対して Reflow 処理を実行する。これによって，BlockFlow が持つ子孫 Frame は全て横書きでのレイアウトが計算された状態になる。ここで，この子孫 Frame の Reflow 処理は，すでに Gecko で実装されている機能である。これには様々なレイアウト，Table や多段組，禁則処理を含んだ行幅の計算などが含まれている。そのため，horizontal-tb 以外の行幅の計算処理は追加する必要はない。その後に，Reflow によって計算された子孫 Frame に対して座標変換処理を行う。この処理で，実際に子孫 Frame のレイアウトが選択した書字方向に変換される。そして，最後に変換処理のために入れ替えていた指定方向の入替を元に戻す復帰処理を行う。

以下では，今回実装している『座標変換処理』と『指定方向の入替処理』という二つの処理の詳細について説明する。処理の順番とは前後するが，分かりやすさを優先するために，まず『座標変換処理』について述べ，その後『指定方向の入替処理』について解説する。

座標変換処理

座標変換処理は，子孫 Frame の全てに対して特定のブロック方向になるように座標を変換する処理のことである。前項で述べたように子孫 Frame の Reflow 処理によって図 4.27 のように上から下の横書きが得られる。その結果をレイアウトを各ブロック方向へ座標変換する。

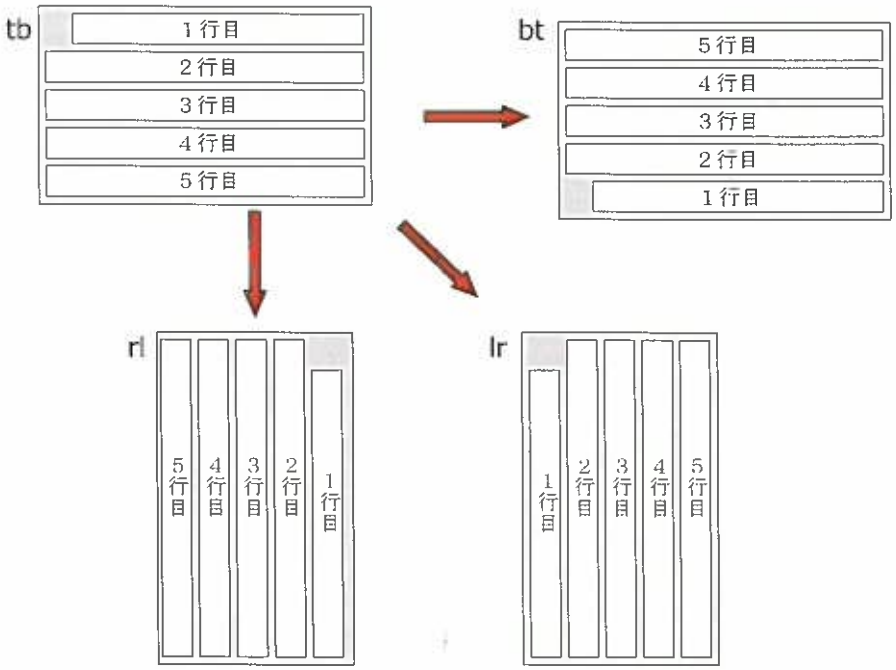


図 4.27: horizontal-tb からの各ブロック方向の変換

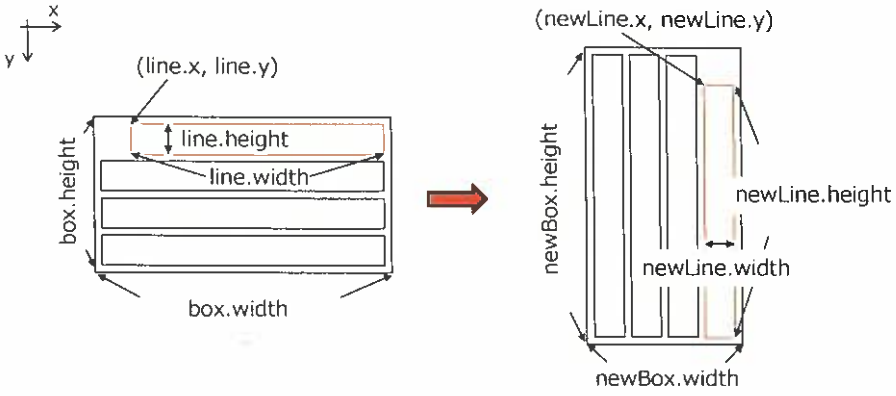


図 4.28: 座標変換処理における座標系と Frame の各座標

ここで各座標は図 4.28 のようになる。これは，ウインドウの左上を原点とする座標系であり，書字方向が指定された外側のボックスは box，行のボックスはそれぞれ line で表す。また，変換先の外側のボックスを newBox，行のボックスを newLine で表す。各ボックスは x，y，width，height の四つの値を保持しており，ボックスの座標は左上の点を

```
(1): vertical-rl への変換
  newLine.x = newBox.width - line.y - line.height
  newLine.y = line.x
  newBox.width  = box.height
  newBox.height = box.width

(2): vertical-lr への変換
  newLine.x = line.y
  newLine.y = line.x
  newBox.width  = box.height
  newBox.height = box.width

(3): horizontal-bt への変換
  newLine.x = line.x
  newLine.y = newBox.height - line.y - line.height
  newBox.width  = box.width
  newBox.height = box.height
```

図 4.29: 各ブロック方向への座標変換の計算式

示している。変換の式は図 4.29 のようになる。変換の式は，ボックスが保持する値である四つの値に対して存在する。また，各書字方向によって変換の式は異なる。

指定方向の入替処理

ここでは，座標変換処理の前処理として必要な，指定方向の入替処理について解説する。まず，座標変換処理に関する問題点について解説する。一つ目は，高さと幅の指定に関する問題である。今回は Frame の座標変換で書字方向の指定を実装している。この実装では，ユーザによって高さと幅が指定された場合，何もせずにそのまま座標変換すると，ユーザの指定した高さと幅が入れ替わってしまう。図 4.30 を例にして説明する。ユーザによって，幅：200px，高さ：400px で縦書きの書字方向が指定されたと仮定する。その場合，子孫 Frame の Reflow 処理によって，幅：200px，高さ：400px の横書きレイアウトが計算される。それに対して上述した座標変換処理を行うと，高さ：200px，幅：400px の縦書きレイアウトが計算されてしまう。

また，`margin` や `padding` などの方向の指定についても似たような現象が起きる。ユーザによって特定の方向に関する指定が行われた場合，座標変換処理によってその位置が指定した位置と異なってしまう。図 4.31 を例にして説明する。ユーザが縦書きの書字方向を指

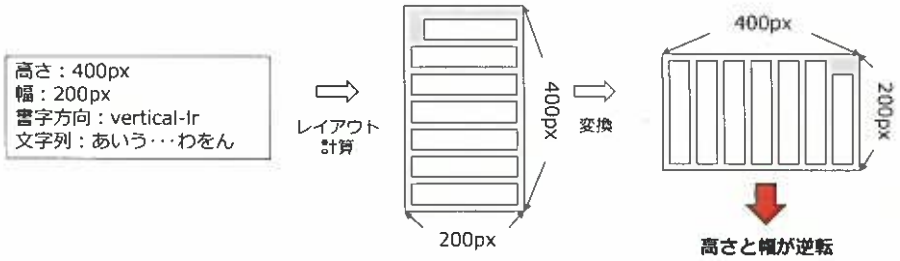


図 4.30: 座標変換処理による高さと幅の逆転

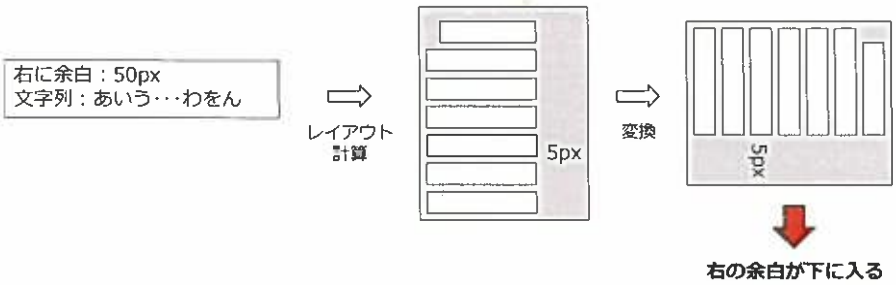


図 4.31: 座標変換処理による指定方向のずれ

定し，右側に余白を挿入したいと仮定する。これをそのまま，子係 Frame の Reflow 処理を行い，座標変換すると右側ではなく下側に余白ができてしまう。

そこで，これらの問題点を解決するために，変換処理を行う前に適切な幅や高さ，方向が指定されるような入替処理を行う。高さと幅の場合，図 4.31 のように幅と高さを入れ替える。こうすることで，座標変換後の最終的なレイアウトを適切に決定することができる。しかし，この処理を用いる必要があるのはこれは縦書きのブロック方向の場合のみである。

また，指定方向に関しても同様に適当な方向との入替処理を行う。図 4.33 を例にして説明する。書字方向の指定が vertical-rl の場合，右に余白を追加するためには，座標変換前にその値を，上側の余白指定の値と入れ替える。そうすることで，座標変換後，最終的には右側に余白が入る。これの対応は，ブロック方向で異なるので注意が必要である。これら，全ての入替の対応を表 4.1 に示す。

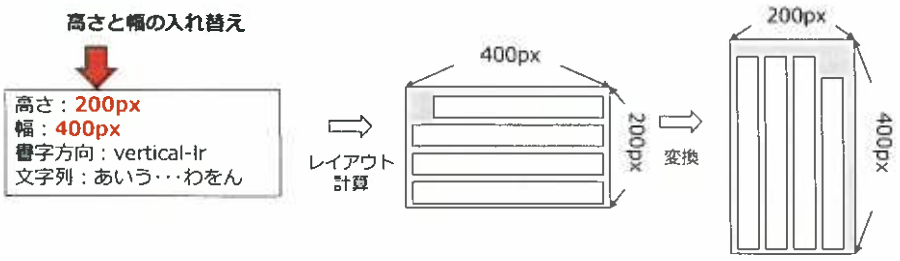


図 4.32: 幅と高さの入替処理の例

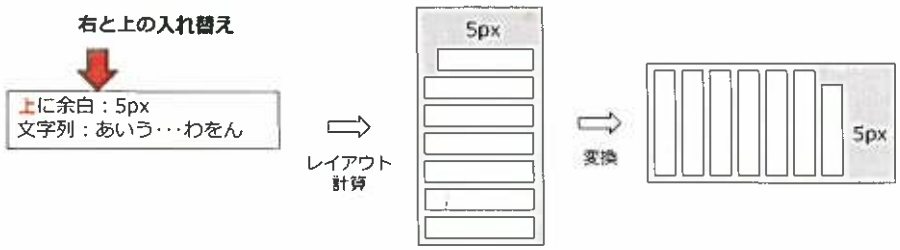


図 4.33: 指定方向の入れ替え処理の例

表 4.1: 各方向の入れ替えの対応

	vertical-rl	vertical-lr	horizontal-bt
width	height	height	width
height	width	width	height
top	left	left	bottom
right	top	bottom	right
bottom	right	right	top
left	bottom	top	left

4.4.6 スクロール処理の実装

ここでは，スクロール処理レイヤの実装の詳細について述べる。図 4.34 に示すように，body 要素に書字方向が指定されると，ページ全体の書字方向が変更される。そのため，スクロール処理では書字方向に応じて適切なスクロール位置を決定するために二つの実装が

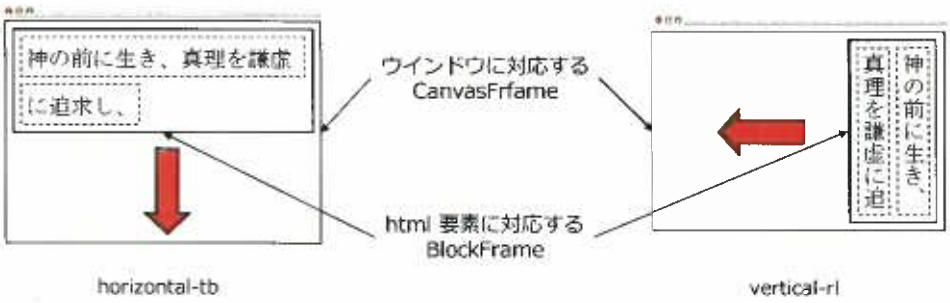


図 4.34: body 要素への書字方向の指定によるページが進む方向の違い

必要である。一つ目は CanvasFrame の直上に BlockFlowFrame を挿入しページの始まる位置を変更する処理，二つ目はスクロール処理を実装するための Overflow に関する実装である。以下ではこの二つの処理の実装の詳細について解説する。

CanvasFrame の直上への BlockFlow の挿入

ここでは，body 要素に書字指定が行われた場合に必要な，CanvasFrame の直上への BlockFlow の挿入処理について述べる。body 要素に書字方向の指定がされた場合には，図 4.34 のように，html 要素に対応する BlockFrame を，ウインドウ全体に対応する CanvasFrame の座標系で座標変換する必要がある。しかし，4.4.3 項で述べた処理では，書字指定された BlockFrame の直前に BlockFlow の挿入を行うため，body 要素に対応する BlockFrame の直前に Frame を挿入してしまう。そうすると，body 要素に対応する BlocKFrame のみに座標回転処理などが行われてしまうので，適切なレイアウトに配置されない。

そこで，html 要素に対応する BlockFrame に対して座標変換を行うため，ウインドウに対応する CanvasFrame の直前に BlockFlow を挿入する。そうすることで，ページ全体に対して座標変換を行うことができるので，目標とする適切なレイアウトが可能である。これによって例えば，body 要素に vertical-rl が指定されていれば，ページ全体が右側から始まるようになる。

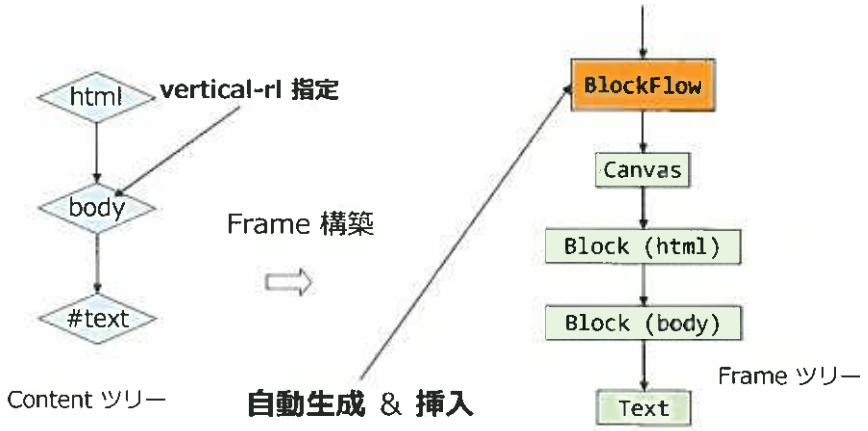


図 4.35: CanvasFrame の直上への BlockFlow の挿入

スクロールのための Overflow 処理

Oveflow の概要

ここでは body 要素に書字方向指定が行われた場合の，スクロールに関する処理の実装について述べる。Firefox ではスクロールを実現するために，CSS のボックスモデルの Overflow と同じ処理を使っている。そこで，まず一般的な Overflow の説明と Firefox での実装の詳細について説明する。

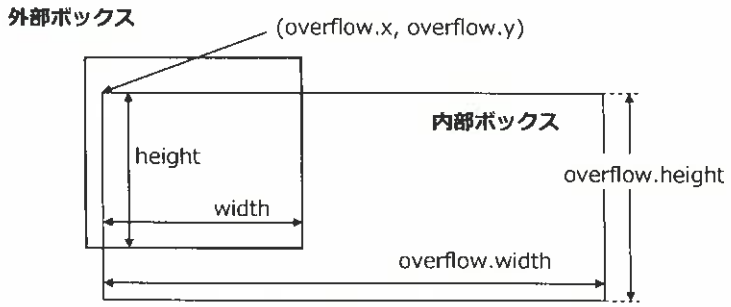


図 4.36: Overflow する場合の例と各座標値

Overflow は，あるボックスの範囲内に内容が収まりきらない場合に用いる処理である。図 4.36 を例にして説明する。この図では，外部ボックスの範囲内に内部ボックスが収まりきらない。この収まらない部分をどのように表示するかを，CSS の overflow プロパティで指定することができる。一般的なブラウザでは，外部ボックスをはみ出して内容を表示するが，

scrolled と指定すると，はみ出している部分をスクロールできる状態で表示することができる。Firefox のウインドウのスクロール処理においては，ウインドウ（表示域）が同図の外側ボックスに対応し，ページ自体が内部ボックスに対応している。そして，scrolled プロパティが常に指定されていると考え，スクロールの実装が行われている。

Overflow 処理の実装

次に Overflow に関する情報がどのように扱われているのかと，スクロール処理の実装について説明する。内部の Frame を Reflow した際に，スクロール処理を行うために図 4.36 に示すような，width，overflow.width，height，overflow.height の各値が保持されている。また，overflow の位置を示すための座標値，overflow.x，overflow.y の二つも保持されている。これら 6 つの値を用いて，Overflow に関する計算と，スクロール位置を計算を行っている。

書字方向の指定によって，スクロールの初期位置を変えるには，oveflow.x と overflow.y の値を変えることによって実現できる。図 4.37 は，vertical-rl の場合の処理を示している。vertical-rl の場合ではスクロールの初期位置を左に移動するため，初期位置の overflow.x に，width - overflow.width を計算する。これによって，はみ出している内部のボックスの右側部分が，初期位置に表示されるようになる。

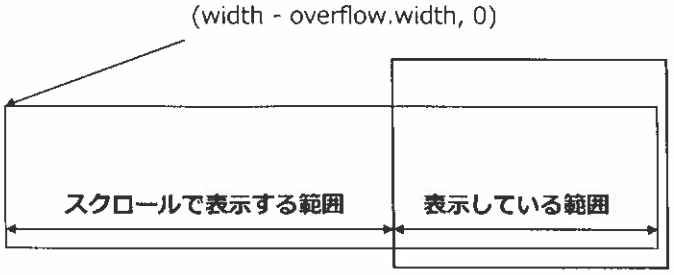


図 4.37: スクロールの初期位置の移動

第5章　評価

本章では前章において実装した各項目について，その対応と他のブラウザとの比較による評価を行う。

5.1　ルビ表示

5.1.1　対応状況

ここでは，本研究において実装したルビ表示において，どのようなマークアップやスタイル指定に対応しているかについて述べる。まず，マークアップの対応について述べる。実装では，HTML5 のルビ，XHTML の簡単ルビ，複雑ルビの各種マークアップに対応している。

これら三つのマークアップを実際に表示すると，図 5.1 のようになる。ここで，各マークアップと display を図 5.2 のように対応付けている。ここで，HTML5 の対字ルビと XHTML の簡単ルビ，HTML5 の対語ルビと XHTML の複雑ルビがそれぞれの表示が一致していることが分かる。HTML5 と XHTML の異なるマークアップに対して，同じルビの種類であれば同じ表示を行うことができるようになった。

図 5.1: ルビのマークアップの対応

```
ruby { display: ruby; }
rbc  { display: ruby-base-container; }
rtc  { display: ruby-text-container; }
rt   { display: ruby-text;
       font-size: 50%; }
rb   { display: ruby-base; }
```

図 5.2: ルビに関するタグの display プロパティ指定

他にも，多くの妥当でないマークアップにも対応している。ルビに関するいくつか妥当でないマークアップの例を，図 5.3 に示す。これらを実際に表示すると，図 5.4 のようにルビとして適切に表示される。以下では提示した妥当でないマークアップについての対応について解説する。同図の 1 は <ruby> がない場合である。この場合は <ruby> が外側にあるとみなし，Frame Tree を構築している。同図の 2 は HTML5 のマークアップに <rb> を用いている場合である。また，同図の 3 では <rbc> のマークアップがない場合である。これらの場合は，<ruby> タグ内の <rb> タグが一つにまとめられて一つの <rbc> タグに含まれているとみなす。実際には，他にも多くの妥当でないマークアップに対応している。想定した全ての場合への対応は付録 A に掲載している。

```
1: <ruby>タグがない場合
   <rb>心</rb><rt>こころ</rt><rb>心</rb>

2: HTML5 で <rb> タグを利用した場合
   <ruby>
     <rb>塩</rb><rt>しお</rt>
     <rb>澤</rb><rt>ざわ</rt>
   </ruby>

3: <rbc>のマークアップがない場合
   <ruby>
     <rb>塩</rb><rb>澤</rb>
     <rtc><rt>しお</rt><rt>ざわ</rt></rtc>
   </ruby>
```

図 5.3: Invalid なルビのマークアップの例

図 5.4: Invalid なルビのマークアップの表示

表 5.1: CSS3 Ruby Module の対応

プロパティ名	対応しているプロパティ値	未対応なプロパティ値
display	ruby, ruby-base, ruby-text, ruby-base-container, ruby-textcontainer	
ruby-position	before, after	bopomofo
ruby-align	start, center, end	auto, distribute-letter,distribute-space
ruby-overhang	auto, start, end, none	
ruby-span	任意の数	

次にルビのスタイル指定の対応について述べる。指定可能な各種プロパティとプロパティ値については表 5.1 に示す。まず，ルビのための `display` プロパティに対応しているため，`display:ruby;` のように指定することで，任意のタグにおいてルビを表示することができる。

ルビを表示する位置を指定する `ruby-position` プロパティでは上側にルビをつける `before` と，下側にルビをつける `after` に対応している。これを応用し，複雑ルビを用いて両側にルビを表示することも可能である。本実装では注音符号のための `bopomofo` を実装していない。しかし，図 5.6 に示すように，横書きの注音符号ならば利用することが出来る。

`ruby-align` では，`left`, `center`, `right` をそれぞれ実装した。`distribute-letter`, `distribute-space`, これらを自動的に選択する `auto` の三つのプロパティ値については，今回は実装していない。本来の初期値は `auto` であるが，今回の実装では `center` を初期値としている。

`ruby-overhang` に関しては一通りのプロパティ値へ対応している。これによってほと

図 5.5: ルビに関するスタイル指定の対応

んどのルビの場合に，文と親文字列が途切れずに文章を読むことができるようになった。しかし，ルビかけ量はどのような場合も一定になっており，JIS X 4051 で規定されている文字クラスごとのルビかけ量の計算は行っていない。最後に，ruby-span を実装した。これによって，対字ルビとグループルビを両側にふることができるようになった。

5.1.2 他の実装との比較

ここでは他のルビ表示の実装と比較を行う。対応状況であるが，他のブラウザとの対応状況の比較は表 5.2 に示す。まず，マークアップされたルビについては，Firefox 以外の主要なブラウザで基本的な表示が可能になっている。本実装においては HTML5 のマークアップと XHTML の簡単ルビには対応しており，加えて XHTML の複雑ルビのマークアップにも対応している。今回の実装が公式の Firefox に組み込まれることで，ほとんど全てのブラウザで基本的なルビは表示ができるようになる。実際に，本研究において実装した部分をパッチと

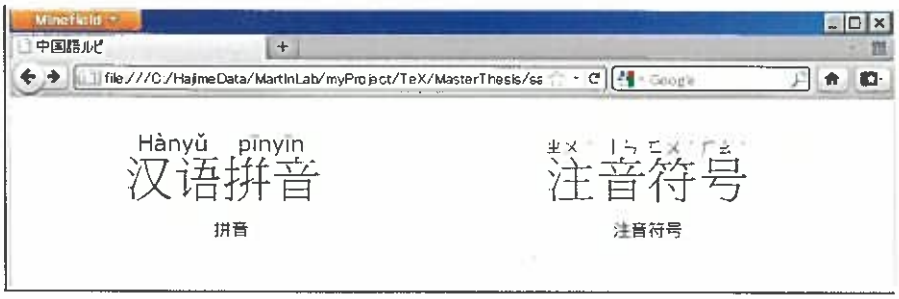


図 5.6: 中国語のルビの表示

表 5.2: ルビのマークアップのブラウザの対応 (シェア出典：文献 [88])

ブラウザ	シェア	HTML5	XHTML (簡単)	XHTML (両側)	XHTML (複雑)
Internet Explorer 8.0	60%	○	○	-	-
Firefox 3.6	23%	-	-	-	-
Chrome 8.0, Safari 5.0	13%	○	○	-	-
Opera 11.0	2%	-	-	-	-
Amaya 11.3	-	-	○	○	-
My Firefox	-	○	○	○	○

して Web で公開し，Firefox への採用に向けて作業を行っている[1]。これが完了すると，Web においてルビの利用を可能にするという基本的な目標は達成したと考えられる。

次に，各種スタイル指定への対応について比較する。ブラウザごとの各種スタイル指定の対応は表 5.3 に示しているが，既存のブラウザはほぼ未実装である。本実装ではそれと比較して多くのプロパティに対応した。まず，ルビの `display` プロパティに対応している。また，`ruby-align` や `ruby-position`，`ruby-overhang` などにも一部のみであるが，対応している。加えて，複雑ルビとの組み合わせで `ruby-span` プロパティを指定できる実装はいままでまったくなく，本実装が初めての実装である。

他にも，本実装は 3.3.2 項で述べた CSS3 Ruby Module のボックスモデルを忠実に再現している。図 5.7 は他のブラウザとのルビに関するボックスの比較である。この図では `<ruby>` を実線，`<rb>` を長破線，`<rt>` を点線で表示し，ボックスモデルと対応付けている。Internet Explorer では，`<rb>` をないものとみなし親文字に対応するボックスが削除されてしまって

[1] `https://bugzilla.mozilla.org/show_bug.cgi?id=256274`

表 5.3: CSS3 Ruby Module の対応

ブラウザ	ruby-position	ruby-align	ruby-overhang	ruby-span
Internet Explorer 8.0	-	○	-	-
Firefox 3.6	-	-	-	-
Chrome 8.0, Safari 5.0	-	-	-	-
Opera 11.0	-	-	-	-
Amaya 11.3	-	-	-	-
My Firefox	○（一部未対応）	○（一部未対応）	○	○

図 5.7: 各ブラウザのボックスの対応

いる。そのため，既存の <rb> を用いてルビマークアップを行っているページではスタイルを指定しても反映されない。Google Chrome においてはそのような問題はないが，Internet Explorer と共通として <ruby> に対応するボックスがルビ文字列を含んでいない。これは，現行の仕様との大きな違いである。しかしながら，現状の Ruby のボックスモデルは不完全であり，display プロパティの指定の詳細が規定されておらず，Webkit の開発者である Hyatt は意図的に実装していないと述べている [96]。

5.1.3 パフォーマンス

本項では，本実装におけるルビの表示がどの程度のパフォーマンスを持っているかについて述べる。一般的にルビの表示はテーブルの似たようなレイアウトを持つことから，レンダリングに時間がかかるとされている。特に本実装では三種類の Frame の生成と，その Reflow を行うため，一般的な書籍のような大量のルビを表示する際には，ルビを表示しない場合よりも多くの時間がかかる可能性が考えられる。そこで，Mozilla のパフォーマンス測定ツー

表 5.4: 実験に用いた三つのサンプルの詳細

ラベル	作品名	文字数	ルビの数	ルビ密度 (%)
Gulliver	『ガリバー旅行記』[4]	120,327	136	0.1
Kokoro	『こころ』[5]	160,432	4,567	2.9
Setsurei	『雪霊記事』[6]	33,030	7,425	22.5

ルである Talos[2] を用いてルビのあるページ読み込みの測定を行った。

今回は，本実装を使ったルビの表示，`inline-table` のプロパティを用いてのルビ表示，ルビとしてではなくインラインとしての表示の三種類の場合の測定を行う。`inline-table` を用いたルビの表示は，ルビに対応していないブラウザで用いられるテーブルレイアウトを用いてルビらしく表示させる方法である。この手法を用いた XHTML ルビサポート[3] という Firefox のアドオンもある。

本測定ではルビの密度が異なる青空文庫の三つ作品（Web ページ）をサンプルとして用いる。これらサンプルの詳細については表 5.4 に示す。ここで，ルビ密度とは全文字数に対してどの程度ルビが振られているかを示す指標である。本文の全文字一つずつにルビが振られていればその値は 100 になる。今回の測定は以下のような条件で行った。

- OS: Microsoft Windows XP SP3
- CPU: Intel® Xeon® E5430 2.66GHz
- メモリ: 3.25GB RAM
- Firefox のバージョン: Minefield 4.0 Beta9 (Gecko20b9)

測定の結果を，表 5.5 に示す。まず，表示方法の差がどのように，読み込み時間に影響しているかについて述べる。結果から，インライン表示の場合が一番，読み込みに時間がかかっている。`inline-table` での表示と，本実装での表示ではわずかに本実装での表示が速いことが分かる。本実装での Frame 生成と Reflow は単にインライン表示するよりも時間がかかるように予期したが，実際にはインライン表示の方が時間がかかっている。これは，単なるインライン表示であっても内部で詳細な処理がされているためであると考えられる。

[2] `https://wiki.mozilla.org/Buildbot/Talos`

[3] `https://addons.mozilla.org/ja/firefox/addon/1935/`

[4] Jonathan Swift, 原民喜訳, 『ガリバー旅行記』, `http://www.aozora.gr.jp/cards/000912/files/4673_9768.html`, 2004.

[5] 夏目漱石, 『こころ』, `http://www.aozora.gr.jp/cards/000148/files/773_14560.html`, 2010.

[6] 泉鏡花, 『雪霊記事』, `http://www.aozora.gr.jp/cards/000050/files/1184_20151.html`, 2005.

表 5.5: 各サンプルのページの読み込み時間 [ms]

	本実装での表示	inline-table での表示	インラインでの表示
Gulliver	28	28	28
Kokoro	8,187	8,281	10,684
Setsurei	26,830	27,037	30,076

次に，ルビ密度が読み込み時間にどのように影響しているかを述べる。結果から，Gulliver の読み込み時間は全て 0.03 秒ほどであり，非常に高速であること分かる。その次に Kokoro の読み込みに時間が掛かっていて，一番時間がかかるのは Setsurei である。上で述べた表示種類の差よりも，ルビ密度の差による読み込み時間の差のほうが顕著である。この結果から，読み込み時間が影響をうけるのは，レイアウト決定の処理の差ではなくファイル中のマークアップの数であることが分かる。実際，ルビが多くマークアップされている Setsurei は非常に時間がかかっている。

結論として，本実装の表示がページの読み込み時間は一番短時間である。しかし，ルビのマークアップが多い場合には，それに応じて読み込み時間が影響を受ける。例えば複雑ルビのマークアップの場合には，より時間がかかることが考えられる。しかし，今回の実装したルビ表示のある Firefox はデバッグオプション付き，最適化オプションなしのビルドである。そのため，実際のアプリケーションにおいてもこれらの読み込み時間がそのまま反映されるとは限らない。実際に，最適化オプションの指定されたリリース版 Firefox (4.0b9pre) で Kokoro を読み込むと，読み込み時間は体感で 1 秒以下である。一般的な Web ページの読み込み時間 80%〜90%は，HTML 文書以外のコンポーネント（画像，Flash，動画など）のダウンロードに時間が消費されおり [97]，ブラウザ側のレイアウト処理の時間はほとんど気にしないでよい。そのため，本実装は長い書籍等の表示においても十分実用的であると考えられる。

5.2 書字方向選択

5.2.1 可能になった表示の例

本研究では，書字方向の選択を Web ブラウザに実装するためレイヤを提案し，全てのレイヤに関しての実装を行った。実装の結果，可能になった書字方向選択に関係する表示につ

いて述べる。

各書字方向への対応

まず，writing-mode プロパティによる書字方向の選択に対応した。CSS の仕様で提案されている三つの書字方向に対応した。また，CSS の仕様では `horizontal-bt` というプロパティ値は存在しないが，レイアウトの自由度を上げるため実装した。図 5.8 は各書字方向指定を実際に行い，表示した例である。同図に示すように，全ての書字方向に対応することができた。

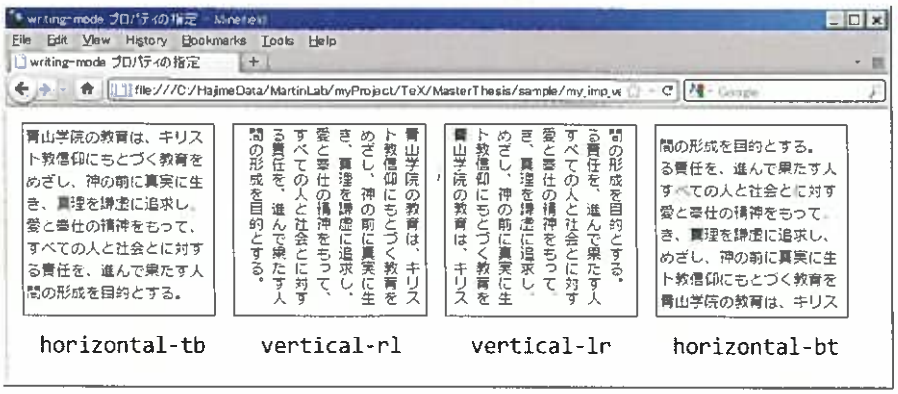


図 5.8: writing-mode プロパティの指定

また，さらに応用例として，書字方向選択による縦書きでのモンゴル文字の表示を挙げる。例えば，図 5.9 はの満州文字による翻訳に中国語が併記された『滿文三國志』（三国志の満州語翻訳）の冒頭の文章である。満州文字はモンゴル文字を利用して表記するため，モンゴル語と同じように左から右への行方向になる。これは，Unicode を用いて他言語表記が可能である点を利用しているために可能である。このように，日本語以外の文章においても本実装は有効である。

多言語テキストの表示

また，2.3.3 項で述べたような，縦組みの中に他の言語のテキストがある場合の表示が可能である。図 5.10 は，ラテンアルファベットとアラビア文字が混在した縦組みを表示した例である。Mozilla Firefox はすでに Unicode による Bidi Algorithm を実装しており，右から左へ書き進めるテキストをこのアルゴリズムに準じて表示させることができる。そのため，同図

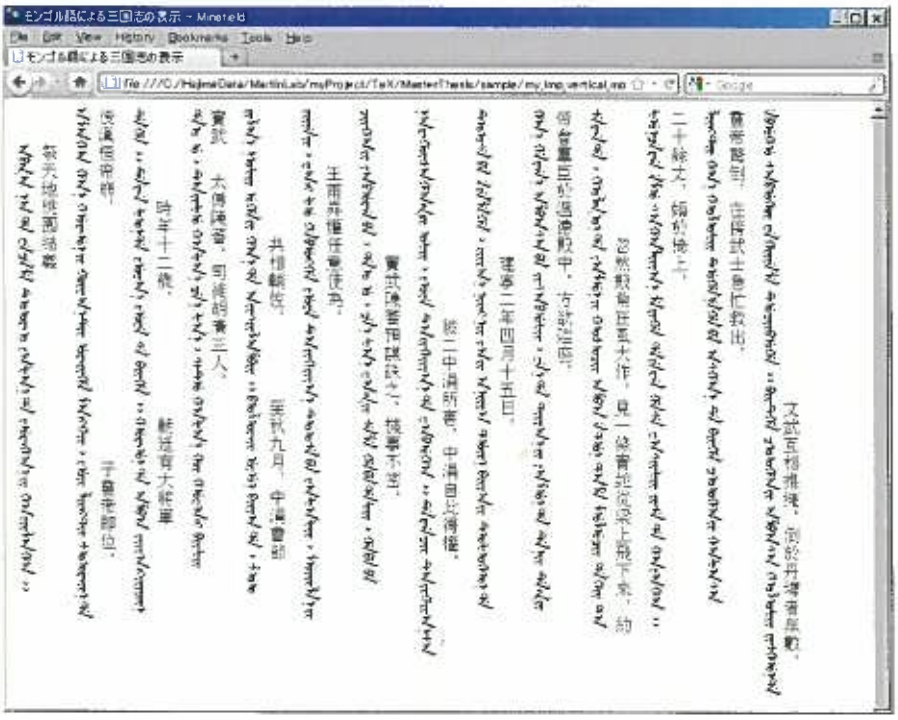


図 5.9: モンゴル文字による文章の表示 (文章出典: http://www.babelstone.co.uk/Test/Manchu.html)

の例のように縦書きとして表示した場合にも，これを利用して表示ができる。これは，縦書き専用ビューワなどを製作すると個別に実装する必要があるが，既存のブラウザに追加実装したためにそのような手間を省くことができた。しかし，本実装では `text-orientation` プロパティ指定を用いた，図 2.25 のような縦注横の組版を実現できない。略字の表記に関しては，全角の英数字を用いることで可能になっている。

動的なレイアウト変更への対応

本実装では，ウインドウの幅の変更などによって表示領域が変化した場合に，それに応じたレイアウトを自動的に行うことができる。図 5.12 は，実際に青空文庫にある夏目漱石（そうせき）の『坊ちゃん』のページを表示した例である。同図において，行幅はウインドウの幅に一致するように計算されている。ウインドウの幅が変更された場合には，既存のレイアウトエンジンがその変更を検知し，レイアウトを計算しなおす命令を発行する。その部分に関しては既存の実装を利用する。縦書きレイアウトの計算の際に，ウインドウの幅と同じような Frame を生成するように設定することでこのような実装を実現している。

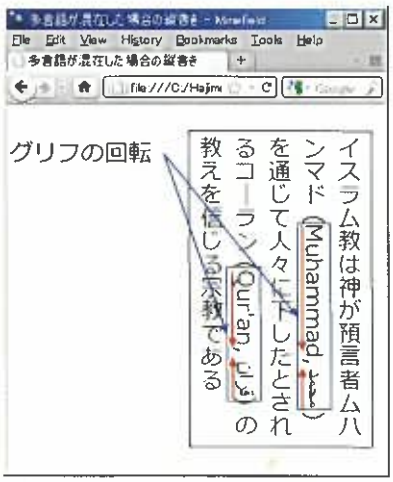


図 5.10: 縦組みに他言語のテキストが混在した場合の表示

また Firefox は禁則処理に対応している。同図の丸で囲った部分では，禁則処理が行われているため句読点が行頭にこないようになっている。また，この禁則処理はレイアウトの変更を行うたびに実行される。そのため，動的に変更した場合にも適切な禁則処理が行われることになる。

横組みと縦組みの組み合わせ

本実装では，それぞれの書字方向を単独で用いるのみではなく，それらを組み合わせて用いることも可能である。図 5.12 は，四つの書字方向の組み合わせたレイアウトの表示例である。同図は『horizontal-tb → vertical-rl → horizontal-bt → vertical-lr』という組み合わせを表示している。

これは，BlockFlowFrame を導入したことで，既存のレイアウト計算部分と書字方向選択のためのレイアウト計算が分離しているために可能になっている。もともと，Reflow という処理は Frame ツリーに対して再帰的に処理を行うため，各 Frame の計算において非常に独立性が高い。実際，子 Frame が親 Frame に与える情報は主に座標，幅，高さのみであり，それさえ分かっていれば，子供の Frame の中身がどのように配置されていようと影響を与えない今回の例で言えば，vertical-rl に含まれるブロックの内容がどのように配置されていようと，親 Frame である horizontal-tb 側からは，あるブロックのうちの一つにしか見えない。そのために，このような組み合わせを実現できた。

図 5.11: 動的なレイアウトの変更と禁則処理

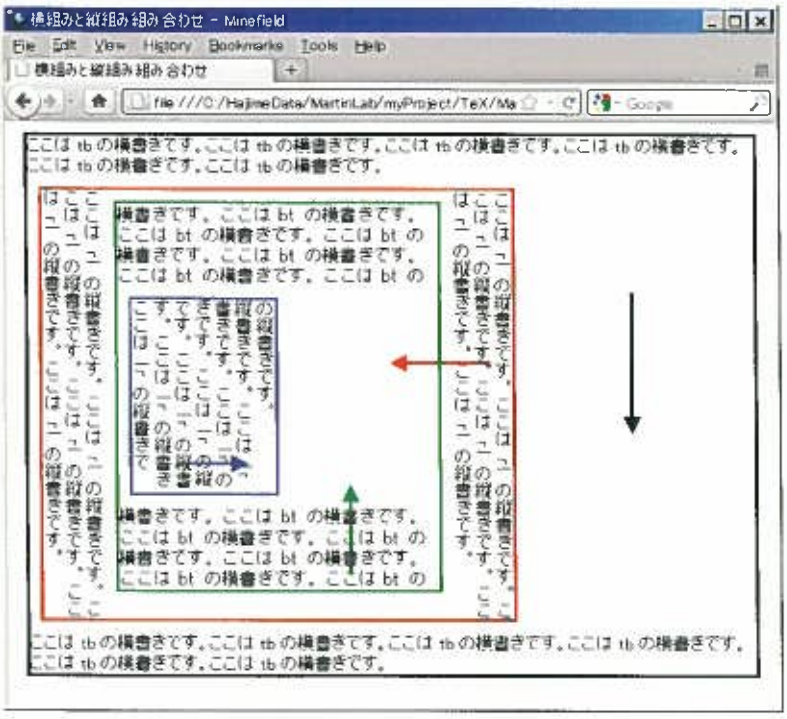


図 5.12: 横組みと縦組みを組み合わせたレイアウトの表示

縦組みの多段組

最後に，新たなレイアウトの可能性として，縦組みの多段組について述べる。図 5.13 は縦書きの多段組を用いた表示の例である。

Braganza らはコンピュータディスプレイで多段組の文書がユーザにとって読みやすいかどうかについて調査をしている [98]。その結果，多くのユーザが現在の Web でスクローリングで慣れてしまっているため，横書き多段組は左右スクロールが必要であるため，あまり好まれないということを指摘している。しかし，多段組のほう行幅適切でが読みやすいという意見も多く見うけられたということを述べており，慣れの問題であるという意見もあった。

一般的な，縦書きの場合は左右のスクロールが必要であり，ユーザにとってはなれない点がある。しかし，ここで縦組みを多段組にすることで，行幅は適切であり，なおかつスクロールの方向が上下になる。この方式は，Braganza らによる研究結果から類推すると，読みやすいものであることが推測できる。

5.2.2 得られた知見

物理プロパティ

今回の実装では，前処理として幅と高さの入れ替え，また各方向の入れ替えを行った。この入れ替えは論理プロパティと物理プロパティの入れ替えと一致する。すなわち，回転を用いたと場合の入れ替えは，論理プロパティの概念と同一であることが分かる。そのため，実装においても論理プロパティが実装されることが望ましいと考えられる。指定された物理プロパティを保持しておき，各 Reflow において，高さや幅，方向を指定する必要がある場合には，論理プロパティを用いて各値を算出する。実際の算出には，物理プロパティの値と書字方向の選択を用いて適切な物理プロパティ値を得る必要がある。こうすれば入れ替え処理を行うことなく座標変換を用いた実装が可能である。しかし，そのためには物理プロパティを用いている部分の実装を，全て論理プロパティに書き換える必要があるため，一定のコストがかかると考えられる。

縦組みと横組みの組み合わせにおける高さや幅の指定

縦組みと横組みの組み合わせにおいて問題になる幅や高さの規定について述べる。横組みの中に縦書きを入れた場合には，縦組みのボックスの高さをどのように決定するかを考慮しなければならない。これは，ページ全体が一方向の書字方向であれば問題にならない。なぜ

図 5.13: 縦組みを多段組に用いた例

すでに横書きであれば，ボックスモデルの視覚フォーマットモデルを用いればよいし，縦書きのみであれば縦と横を単に読み替えればいいからである。この，視覚フォーマットモデルでは横組みの場合には上から下に無限に伸びていくことが可能になっている。そのため，height が指定されていないボックスは行のスタックに応じて伸び縮みするように規定されている。しかし，横組みの中に縦組みが含まれた場合には，height が無限大になっているため，行がまったく折り返されない現象が起こる。

これを解決するため以下のような規則を用いた。まず，第一規則としてそれまでに height の値が指定されていればそれを用いる。指定されていない場合には，ウインドウの幅，正確には 100vh (viewport height) という指定を適用する。これは，横組みの場合の行幅と同じ挙動であり，きわめて直感的なものであると考えられる。この規定は CSS3 Writing-mode Module の中でも非常に重要であり，現在も議論が続いている [99]。

第6章　結論

6.1　本研究の成果

本研究では，テキストレイアウトの国際化という目的のため，ブラウザ側の実装を中心に研究を行った。特に，日本語と関連の深い『ルビの表示』と『書字方向の選択』という二つに注目した。ここでは本研究において，それらに関して得た成果について述べる。

まず，ルビの表示に関しての成果について述べる。本研究において，実際に Firefox 上でルビの表示ができるように実装を行った。その特徴として，まず HTML5 と XHTML の二つのマークアップへ対応した。そして，他のブラウザでの実装がない XHTML の複雑ルビに関して，`ruby-span` 指定も含めて完全に実装した。次に，仕様上で決められていない妥当でないマークアップに対する対応についても考慮し，適切な表示が行われるようにした。また，詳細な CSS プロパティの指定を可能にし，高品位な組版が必要な場合でも Web が用いることができるような可能性を見出した。本実装に関しては，ソースコードを Web で公開し，Mozilla によってリリースされる公式の Firefox に採用してもらうためにレビューを受けている。

次に書字方向の選択に関する成果について述べる。書字方向の選択は実装が広範囲にわたるため，実装を各レイヤに分離し，一通りの実装を行った。この分離によって，一部のレイヤの実装を，さらに高度な別の実装に取り替えることも可能である。本実装では日本語の右から左だけ縦書きに限らず全ての書字方向に対応することができた。そして，専用の縦書きブラウザを作成するのではなく，すでにあるブラウザへ追加機能として実装することで，無駄を省くだけでなく様々な相乗効果を得た。一つ目に，すでに Firefox において Bidi 問題を解決したブラウザであったため，アラビア語のような右か左へ書くような他言語を含んだテキストを縦書きで表示可能になっている。二つ目に，ウインドウの幅などによって動的なレイアウトが可能である。三つ目に，縦組みと横組みの組み合わせが可能である。四つ目に，縦書きの多段組が可能になっている。また，本研究で得られたいくつかの知見は W3C のメーリングリストに投稿しており，仕様の策定に生かされれば幸いである。

6.2 今後の課題

本研究ではルビの表示と書字方向の選択に関しての実装を行ったが，いくつか課題を残している。本節では，今後の課題について述べる。

まず，ルビの表示に関しての課題について述べる。一つ目に，対字ルビの場合には複数の行に分割することができる。しかし，本実装においては，ルビ自体が一つの大きなブロックで構成されており，その内部で行分割することができない。この挙動は，頻繁に行幅が変更されるブラウザの環境と親和性が高いと考えられるので，対応が必要だと考えられる。二つ目にルビとルビが含まれる行に関して，いくつかの点を考えなければならない。現時点の実装では，上側にルビがつく場合，下側にルビがつく場合，両側にルビがつく場合で，`vertical-align` プロパティを変更する必要がある。これはルビのボックスが行内でどの位置に寄るかが，ルビの位置によって異なるからである。これらはルビの種類に応じて適切に変更されることが理想である。これに加えて，ルビがある場合に行幅と行間をどのように決めるかを考えなければならない。この点に関しては，CSS3 Line Module [100] によって二種類のプロパティが規定されているが，より高度で便利な指定を追加する試みがある [101]。

次に，書字方向の指定に関しての課題について述べる。書字方向の指定に関しての課題として，まずグリフ置換処理の OS に依存しない実装が挙げられる。現在，Windows では@マークによる縦書きフォント指定，Linux においては Pango を用いた実装を行っている。@マークによる実装は，Windows GDI という古いレンダリングシステムを用いるため，複雑な多言語テキストを適切に表示できない場合がある。そこで，HarfBuzz というテキストレンダリングエンジンを用いることが一つの提案として考えられる。この HarfBuzz というライブラリは各 OS から利用することができるため，このライブラリに縦書きグリフ置換の機能を追加実装することが理想である。また，現在は主に書籍などのような文字中心のレイアウトに対して，座標回転を用いて書字方向の選択を実装した。画像や動画，さらに一般的な Web ページの書字方向を縦書きにした場合の更なるテストケースが必要であると考えられる。最後に，本実装の書字方向の選択でも，ルビの表示ができるような実装が可能になる必要もあるだろう。

謝辞

本研究の遂行と，本論文の執筆にあたり，多くの方々の指導と協力をいただきました。

指導教員である Martin J. Dürst 教授には学部時代から，多岐に渡り非常に多くのご指導をいただきました。ソフトウェアの国際化という非常に魅力溢れる世界を紹介していただいたことを含め，ここに深く感謝の意を申し上げます。また，研究と論文の校正にあたって丁寧な指導をしていただいた松原俊一助手に深謝いたします。お忙しい中，本論文の副査を担当してくださり適切なアドバイスをしていただいた佐久田博司教授，大原剛三准教授のお二人に感謝いたします。

W3C の Elika J. Etemad さんには，わざわざ本学までお越しいただき，ルビと縦書きの実装について議論させていただきました。また，同じく W3C の石井宏治さん，呂康豪さんにはルビの実装に関して協力していただきました。Mozilla Corporation の Boris Zbarsky さんと Robert O'Callahan さんのお二人には，多忙を極める中，レビューワーとしてルビの実装のソースコードをチェックしていただき，的確な指摘とアドバイスをいただきました。みなさんに，感謝いたします。

本学情報テクノロジー学科の矢吹太朗助教には，何度も私の雑談に付き合っていただき，本研究に限らず多くの鋭い意見をいただきました。また，『フィネガンズ・ウェイク』という素晴らしい文献を紹介していただきました。本当にありがとうございました。

同研究室の修士一年生である粟野君，地主君，松本君，水野君や学部四年生は，私にとって最後の学生生活を非常に楽しく有意義なものにしてくれました。ありがとう。

最後に，24 年間，常に私を支えてくれた両親，そして兄弟に心から感謝します。

2011 年 1 月

塩澤 元（しおざわ はじめ）

参考文献

[1] Jesse Alpert and Nissan Hajaj. *We knew the web was big...* `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`, 2008.

[2] Eric de Grolier, 大塚幸男訳. 『書物の歴史』. 白水社, 1992.

[3] 楽天リサーチ株式会社, 楽天株式会社. 『読書・図書に関するインターネット調査』. `http://research.rakuten.co.jp/report/20090914/`, 2009.

[4] 山口仲美. 『日本語の歴史』. 岩波書店, 2006.

[5] 内閣閣甲第 104 号. 『公用文作成の基準について』, 1949.

[6] 小宮山博史. 『日本語活字ものがたり』. 誠文堂新光社, 2009.

[7] The Unicode Consortium. *The Unicode Standard Version 6.0*. `http://www.unicode.org/versions/Unicode6.0.0/`, 2010.

[8] Douglas R. Hofstadter, 竹内郁雄・斉藤康巳・片桐恭弘訳. 『メタマジック・ゲーム』. 白揚社, 2005.

[9] Yannis Haralambous, P. Scott Horne 訳. *Fonts & Encodings*. O'Reilly, 2007.

[10] 安岡孝一, 安岡素子. 『文字符号の歴史 -欧米と日本編-』. 共立出版, 2006.

[11] American National Standards Institute. *ANSI/INCITS 4-1986: Coded Character Sets – 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)*, 1986.

[12] 日本規格協会. 『JIS X 0208: 7 ビット及び 8 ビットの 2 バイト情報交換用符号化漢字集合』, 1997.

[13] 中原康. 『日本語 EUC の定義と解説』. `http://euc.jp/i18n/euc-jp.txt`, 1991.

[14] 中国国家标准总局. 『GB 2312: 信息交换用汉字编码字符集・基本集』, 1980.

[15] Bureau of Indian Standard. *IS 13194: Indian Script Code for Information Interchange - ISCII*, 1991.

[16] International Organization for Standardization. *ISO/IEC 8859-6: Information Technology – 8-bit Single-Byte Coded Graphic Character Sets – Part 6: Latin/Arabic Alphabet*, 1999.

[17] Andrew Chernov. *RFC 1489: Registration of a Cyrillic Character Set*. `http://tools.ietf.org/html/rfc1489`, 1993.

[18] International Organization for Standardization. *ISO 10754: Information and Documentation – Extension of the Cyrillic Alphabet Coded Character Set for Non-Slavic Languages for Bibliographic Information Interchange*, 1996.

[19] 和田英一. SC2 (Character Sets and Information Coding) WG2 会議および総会. 『情報技術標準 NEWSLETTER』, Vol. 6, pp. 13–14, 1990.

[20] Mark Davis and Lee Collins. Unicode. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, pp. 499–504, 1990.

[21] 和田英一. SC2: Coded Character Sets. 『情報処理』, Vol. 33, No. 8, pp. 978–979, 1992.

[22] 情報規格調査会. 情報技術の国際標準化と日本の対応. 『情報処理』, Vol. 34, No. 9, pp. 1204–1212, 1993.

[23] François Yergeau. *RFC 3629: UTF-8, a transformation format of ISO 10646*. `http://tools.ietf.org/html/rfc3629`, 2003.

[24] Paul Hoffman and François Yergeau. *RFC 2781: UTF-16, an encoding of ISO 10646*. `http://tools.ietf.org/html/rfc2781`, 2000.

[25] Adobe Systems. *Glyph Bitmap Distribution Format (BDF) Specification*. `http://www.adobe.com/devnet/font/pdfs/5005.BDF_Spec.pdf`, 1993.

[26] Ken Lunde. *CJKV Information Processing - 2nd Edition* -. O'Reilly, 2009.

[27] Adobe Systems. *Adobe Type 1 Font Format*. `http://partners.adobe.com/public/developer/en/font/T1_SPEC.PDF`, 1990.

[28] Apple Computer. *TrueType Reference Manual*. `http://developer.apple.com/fonts/TTRefMan/index.html`, 2002.

[29] Microsoft Corporation. *OpenType Specification*. `http://www.microsoft.com/typography/otspec/default.htm`, 2009.

[30] Christopher Harvey. *Languagegeek: Full Aboriginal Unicode Fonts*. `http://www.languagegeek.com/font/fontdownload.html`, 2009.

[31] Microsoft Corporation. *Arial Unicode MS*. `http://www.microsoft.com/typography/fonts/family.aspx?FID=24`, 1998.

[32] Peter Constable and John H. Jenkins. *Re: Arial Uniocde MS (Unicode Mailing List)*. `http://www.unicode.org/mail-arch/unicode-ml/y2004-m12/0121.html` `http://www.unicode.org/mail-arch/unicode-ml/y2004-m12/0131.html`, 2004.

[33] Jonathan Kew, Tal Leming, and Erik van Blokland. *WOFF File Format 1.0 – W3C Working Draft*. `http://www.w3.org/TR/WOFF/`, 2010.

[34] Martin J. Dürst and Marc-Antoine Parent. Font Selection and Font Composition for Unicode. In *Proceedings of 7th International Unicode Conference*, 1995.

[35] Edward H. Trager. International Text Layout & Typography: The Big And Future Picture. In *Proceedings of Text Layout Summit*, 2006.

[36] Sharon Correll. *Graphite*. `http://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&cat_id=RenderingGraphite`, 2006.

[37] 株式会社モリサワ・日本エディタースクール編. 『文字組版入門』. 日本エディタースクール出版部, 2005.

[38] Robert Bringhurst. *The Elements of Typographic Style version 3.1*. HARTLEY & MARKS Publisher, 2005.

[39] Robert M. Ritter. *The Oxford Style Manual*. Oxford University Press, 2003.

[40] University of Chicago Press Staff. *The Chicago Manual of Style 16th Edition*. University of Chicago Press, 2010.

[41] 日本規格協会. 『JIS X 4051: 日本語文書の組版方法』, 2004.

[42] 逆井克己. 『基本 日本語文字組版』. 日本印刷新聞社, 1999.

[43] 由良君美. ルビの美学 (上). 『言語』, Vol. 2, No. 7, pp. 556–562, 1973.

[44] 白水社編. 『ふりがな廃止論とその批判』. 白水社, 1938.

[45] 内閣告示第 32 号. 『当用漢字表』, 1946.

[46] 由良君美. ルビの美学 (中). 『言語』, Vol. 2, No. 11, pp. 988–994, 1973.

[47] 紀田順一郎. 『日本語発掘図鑑』, 第四章 ルビという小さな虫, pp. 65–80. ジャストシステム, 1994.

[48] 小林龍生. ネットワーク社会でのルビの再評価 – HTML, Unicode に即して –. 『情報処理学会 情報メディア研究報告』, 1998.

[49] James Joyce, 柳瀬尚紀訳. 『フィネガンズ・ウェイク I・II』. 河出書房新社, 1991.

[50] International Organization for Standardization. *ISO 7098: Information and Documentation – Romanization of Chinese*, 1991.

[51] 阿南康宏, 千葉弘幸, 枝本順三郎, Richard Ishida, 石野恵一郎, 小林龍生, 小林敏, 小野澤賢三, Felix 佐々木. 『日本語組版処理の要件（日本語版） – W3C 技術ノート』. `http://www.w3.org/TR/jlreq/ja/`, 2009.

[52] 中国国家标准化管理委员会. 『GB/T 16159: 中文拼音正词法基本规则』, 1996.

[53] 國立臺灣師範大學. 『實用視聽華語 I』. 正中書局, 1997.

[54] 人民教育出版社小学語文室編. 『九年義務教育六年制小学教科書語文第四冊』. 人民教育出版社, 1995.

[55] 陳鐵君. 『遠流活用中文大辭典』. 遠流, 2008.

[56] Suvda Myagmar. *Mongolian Script*. http://www.suvda.com/personal.php?p=script, 2008.

[57] Lawrence Lo. *Ancient Scripts: Ogham*. http://www.ancientscripts.com/ogham.html, 2010.

[58] 夏目漱石. 『新装版・坊ちゃん』. PHP 研究所, 2007.

[59] 屋名池誠. 『横書き登場 —日本語表記の近代—』. 岩波書店, 2003.

[60] Mark Davis. *USA#9: Unicode Bidirectional Alogrithm*. http://www.unicode.org/reports/tr9/, 2010.

[61] 吉田秀夫. 『イタリア人口論研究』. 日伊協會, 1941.

[62] Elika J. Etemad. Robust Vertical Text Layout. In *Proceedings of 27th International Unicode Conference*, 2005.

[63] Tim Berners-Lee and Robert Cailliau. *WorldWideWeb: Proposal for a HyperText Project*. http://www.w3.org/Proposal, 1990.

[64] International Organization for Standardization. *ISO 8879: Information processing – Text and office systems – Standard Generalized Markup Language (SGML)*, 1986.

[65] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. *HTML 4.01 Specification – W3C Recommendation*. http://www.w3.org/TR/html4/, 1999.

[66] Steven Pemberton, Daniel Austin, Jonny Axelsson, Tantek Çelik, Doug Dominiak, Herman Elenbaas, Beth Epperson, Masayasu Ishikawa, Shin'ichi Matsui, Shane McCarron, Ann Navarro, Subramanian Peruvemba, Rob Relyea, Sebastian Schnitzenbaumer, and Peter Stark. *XHTML™1.0 The Extensible HyperText Markup Language (Second Edition) – W3C Recommendation*. http://www.w3.org/TR/xhtml1/, 2000.

[67] Ian Hickson. *HTML5 – A vocabulary and associated APIs for HTML and XHTML – W3C Working Draft*. http://www.w3.org/TR/html5/, 2010.

[68] Philippe Le Hégaret. *Document Object Model (DOM) Specifications*. http://www.w3.org/DOM/DOMTR, 2004.

[69] 社団法人・日本印刷技術協会. 『XML 化に取り組む出版社とサポートする印刷会社』. http://www.jagat.or.jp/story_memo_view.asp?StoryID=11608, 2008.

[70] Vannevar Bush. As We May Think. *The Atlantic Monthly*, July 1945.

[71] International Organization for Standardization. *ISO/IEC 8859-1: Information Technology – 8-bit Single-Byte Coded Graphic Character Sets – Part 1: Latin Alphabet No.1*, 1999.

[72] François Yergeau, Gavin Nicol, Glenn Adams, and Martin Dürst. *Internationalization of the Hypertext Markup Language*. http://www.ietf.org/rfc/rfc2070.txt, 1997.

[73] Martin J. Dürst, François Yergeau, Richard Ishida, Misha Wolf, and Tex Texin. *Character Model for the World Wide Web 1.0: Fundamentals – W3C Recommendation*. http://www.w3.org/TR/charmod/, 2005.

[74] Martin J. Dürst and Asmus Freytag. *Unicode in XML and other Markup Languages – W3C Working Group Note*. http://www.w3.org/TR/unicode-xml/, 2007.

[75] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification – W3C Working Draft*. http://www.w3.org/TR/CSS2/, 2010.

[76] Håkon Wium Lie. *CSS Multi-column Layout Module – W3C Candidate Recommendation*. http://www.w3.org/TR/css3-multicol/, 2009.

[77] John Daggett. *CSS Fonts Module Level 3 – W3C Working Draft*. http://www.w3.org/TR/css3-fonts/, 2009.

[78] Anders Berglund. *Extensible Stylesheet Language (XSL) Version 1.1 – W3C Recommendation*. http://www.w3.org/TR/xsl11/, 2006.

[79] International Digital Publishing Forum. *Open Publication Structure (OPS) 2.0 v1.0*. http://www.idpf.org/2007/ops/OPS_2.0_final_spec.html, 2007.

[80] International Organization for Standardization. *ISO/IEC 10179: Information technology – Processing languages – Document Style Semantics and Specification Language (DSSSL)*, 1996.

[81] Martin Dürst. *Ruby in the Hypertext Markup Language*. http://tools.ietf.org/html/draft-duerst-ruby-00, 1996.

[82] Michel Suignard, Masayasu Ishikawa, Martin Dürst, and Tex Texin. *Ruby Annotation – W3C Recommendation*. http://www.w3.org/TR/ruby/, 2001.

[83] Richard Ishida. *CSS3 Ruby Module – W3C Working Draft*. http://dev.w3.org/csswg/css3-ruby/, 2010.

[84] Elika J. Etemad, Koji Ishii, and Shinyu Murakami. *CSS Writing Modes Module Level 3 – W3C Working Draft*. http://www.w3.org/TR/css3-writing-modes/, 2010.

[85] インプレス R&D インターネットメディア総合研究所. 『インターネット白書 2010』. インプレスジャパン, 2010.

[86] Joseph T. Sinclair. *Typography on the Web*. Academic Press, 1999.

[87] National Information Standards Organization. *ANSI/NISO Z39.86: Specifications for the Digital Talking Book*, 2005.

[88] NetApplicatoins. *Browser Market Share*. http://marketshare.hitslink.com/browser-market-share.aspx?qprid=1, 2010.

[89] 佐藤泰正. 『現代作文講座 6・文字と表記』, 縦書きと横書き, pp. 187–209. 明治書院, 1977.

[90] Simon Fraser, Dean Jackson, David Hyatt, and Chris Marrin. *CSS 2D Transforms Module Level 3 – W3C Editor's Draft*. `http://www.w3.org/TR/css3-2d-transform/`, 2010.

[91] Marcin Sawicki. *International Layout – W3C Working Draft*. `http://www.w3.org/TR/1999/WD-i18n-format-19990910/`, 1999.

[92] David Baron. *Mozilla's Layout Engine*. `http://www.mozilla.org/newlayout/doc/layouto-2006-12-14/master.xhtml`, 2006.

[93] Chris Waterson. *Notes on HTML Reflow*. `http://www.mozilla.org/newlayout/doc/reflow.html`, 2008.

[94] Vladimir Vukićević. *Graphics in Mozilla - Part 1*. `http://blog.vladl.com/2007/12/11/graphics-in-mozilla/`, 2007.

[95] Microsoft Corporation. *GSUB - The Glyph Substitution Table*. `http://www.microsoft.com/typography/otspec/gsub.htm`, 2009.

[96] David Hyatt. *[css3-ruby] The ruby display types (W3C Mailing List)*. `http://lists.w3.org/Archives/Public/www-style/2010Dec/0140.html`, 2010.

[97] Steve Souders, 竹舎広幸・福地太郎・武舎るみ訳. 『ハイパフォーマンス Web サイト』. O'Reilly, 2008.

[98] Cameron Braganza, Kim Marriott, Peter Moulder, Michael Wybrow, and Tim Dwyer. Scrolling Behaviour with Single- and Multi-Column Layout. In *Proceedings of the 18th International World Wide Web Conference*, 2009.

[99] Koji Ishii. 『縦横混在時の auto 値の解釈 (W3C Mailing List)』. `http://lists.w3.org/Archives/Public/public-html-ig-jp/2010Dec/0036.html`, 2010.

[100] Michael Suignard and Eric A. Meyer. *CSS3 module: line – W3C Working Draft*. `http://www.w3.org/TR/css3-linebox`, 2002.

[101] Koji Ishii. *[css3-linebox] Proposal to add "auto" value for the line-stacking-ruby property (W3C Mailing List)*. `http://lists.w3.org/Archives/Public/www-style/2010Nov/0482.html`, 2010.

付録

A. 妥当でないルビのマークアップに対する対応
B. 質疑応答

付録 A

妥当でないルビのマークアップに対する対応

ここでは，妥当でないルビのマークアップのパターンと，その場合に構築する Frame，またレイアウトの対応について記載する。以下では，『妥当でないマークアップとそれに対応する Frame 構築』と，『構築された Frame とレイアウトの対応』の二つの部分に分けて記述する。

表記に関する説明

Markup の項目では，妥当でないマークアップの例を示す。
例：

```
Markup: <p><rb>base1</rb></p>
```

FrameTree の項目では，構築される FrameTree をテキストアートを使って示す。以下の例では，[] で囲まれた部分は Frame を表現している。この図は木構造を示しており，RubyFrame が二つの RubyContainerFrame を持っていることを示している。また，それらは各自一つの RubyCell を持っている。
例：

```
FrameTree: [Ruby]
             +--[RubyContainerFrame(base)]
             |    +--[RubyCell]---["base1"]
             +--[RubyContainerFrame(beforeText)]
                  +--[RubyCell]---["text1"]
```

Layout の項目では目標とするレイアウトをテキストアートを使って示す。xxxxxx はルビの周りの文字列を示し，"text1" は親文字列，"base1" はルビ文字列を示す。
例：

```
Layout:
                          +--------+
                          |"text1" |
                  +-------++-------++--------+
                  |xxxxxxx||"base1"||xxxxxxx|
                  +-------++-------++--------+
```

妥当でないマークアップとそれに対応する Frame 構築

1. <ruby> がなく，<rb> しかない場合

```
Markup: <p><rb>base1</rb></p>
FrameTree: [Ruby]
             +--[RubyContainer(base)]
                  +--[RubyCell]---["base1"]
```

2. <ruby> がなく，<rt> しかない場合

```
Markup: <p><rt>text1</rt></p>
FrameTree: [Ruby]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
```

3. <ruby> がなく，<rb> と<rt> がある場合

```
Markup: <p><rb>base1</rb><rt>text1</rt></p>
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
```

4. <ruby> がなく，<rbc> がある場合

```
Markup: <p><rbc><rb>base1</rb></rbc></p>
FrameTree: [Ruby]
             +--[RubyContainer(base)]
                  +--[RubyCell]---["base1"]
```

5. <ruby> がなく，<rtc> がある場合

```
Markup: <p><rtc><rt>text1</rt></rtc></p>
FrameTree: [Ruby]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
```

6. <ruby> 直下に，<rt> がない場合

```
Markup: <ruby><rb>base1</rb></ruby>
FrameTree: [Ruby]
             +--[RubyContainer(base)]
                  +--[RubyCell]---["base1"]
```

7. <ruby> 直下に，<rb> がない場合

```
Markup: <ruby><rt>text1</rt></ruby>
FrameTree: [Ruby]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
```

8. <rtc> や <rbc> が用いられない場合

```
Markup: <ruby>
          <rb>base1</rb><rb>base2</rb>
          <rt>text1</rt><rt>text2</rt>
        </ruby>
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             |    +--[RubyCell]---["base2"]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
                  +--[RubyCell]---["text2"]
```

9. <rtc> や <rbc> が用いられず，順番が交差している場合

```
Markup: <ruby>
          <rb>base1</rb><rt>text1</rt>
          <rt>text2</rt><rb>base2</rb>
        </ruby>
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             |    +--[RubyCell]---["base2"]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
                  +--[RubyCell]---["text2"]
```

10. <ruby> 直下にテキストノードのみがある場合

```
Markup: <ruby>base1</ruby>
FrameTree: [Ruby]
             +--[RubyContainer(base)]
                  +--[RubyCell]---["base1"]
```

11. <rtc> 直下にテキストノードがある場合

```
Markup: <ruby>
          <rbc><rb>base1</rb></rbc>
          <rtc>text1</rtc>
        </ruby>
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
```

12. <rbc> 直下にテキストノードがある場合

```
Markup: <ruby>
          <rbc>base1</rbc>
          <rtc><rt>text1</rt></rtc>
        </ruby>
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
```

13. <rtc> や <rbc> 内に，<rb> や<rt> が混在する場合

```
Markup: <ruby>
          <rbc><rt>base1</rt><rb>base2</rb></rbc>
          <rtc><rb>text1</rb><rt>text2</rt></rtc>
        </ruby>
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             |    +--[RubyCell]---["base2"]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
                  +--[RubyCell]---["text2"]
```

14. <rt> と <rb> の数が異なる場合

```
Markup: <ruby>
          <rbc><rb>base1</rb></rbc>
          <rtc><rt>text1</rt><rt>text2</rt></rtc>
        </ruby>
```

```
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
                  +--[RubyCell]---["text2"]
```

15. 複雑ルビにおいて，<ruby> 直下に <rt> がある場合

```
Markup: <ruby>
          <rbc><rb>base1</rb><rb>base2</rb></rbc>
          <rt>text1</rt>
          <rtc><rt>text2</rt></rtc>
        </ruby>

FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             +--[RubyContainer(beforeText)]
             |    +--[RubyCell]---["text1"]
             +--[RubyContainer(afterText)]
                  +--[RubyCell]---["text2"]
```

16. 複雑ルビにおいて，<ruby> 直下に <rt> と <rb> がある場合

```
Markup: <ruby>
          <rb>base1</rb><rt>text1</rt>
          <rbc><rb>base2</rb><rb>base3</rb></rbc>
          <rtc><rt>text2</rt><rt>text3</rt></rtc>
        </ruby>
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             +--[RubyContainer(beforeText)]
             |    +--[RubyCell]---["text1"]
             +--[RubyContainer(afterText)]
             |    +--[RubyCell]---["text2"]
             |    +--[RubyCell]---["text3"]
             +--[RubyContainer(base)]
                  +--[RubyCell]---["base2"]
                  +--[RubyCell]---["base3"]
```

構築された Frame に対するレイアウト対応

1. baseCell や textCell が一つの場合のレイアウト

```
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
Layout:
                            +-------+
                            |"text1"|
                       +-------++-------++-------+
                       |xxxxxxx||"base1"||xxxxxxx|
                       +-------++-------++-------+
```

2. baseCell や textCell が複数の場合のレイアウト

```
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             |    +--[RubyCell]---["base2"]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
                  +--[RubyCell]---["text2"]
Layout:
                            +-------++-------+
                            |"text1"||"text2"|
                       +-------++-------++-------+-------+
                       |xxxxxxx||"base1"||"base2"|xxxxxxx|
                       +-------++-------++-------+-------+
```

3. baseContainer がない場合のレイアウト

```
FrameTree: [Ruby]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
Layout:
                       +-------++-------++-------+
                       |xxxxxxx||"text1"||xxxxxxx|
                       +-------++-------++-------+
```

4. textContainer がない場合のレイアウト

```
FrameTree: [Ruby]
             +--[RubyContainer(base)]
                  +--[RubyCell]---["base"]
Layout:
                  +-------++-------++-------+
                  |xxxxxxx||"base1"||xxxxxxx|
                  +-------++-------++-------+
```

5. textCell が basecell より少ない場合のレイアウト

```
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             |    +--[RubyCell]---["base2"]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
Layout:
                           +-------+
                           |"text1"|
                  +-------++-------++-------+-------+
                  |xxxxxxx||"base1"||"base2"|xxxxxxx|
                  +-------++-------++-------+-------+
```

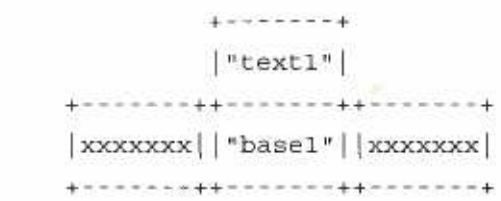
6. baseCell が textCell より少ない場合のレイアウト

```
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             +--[RubyContainer(beforeText)]
                  +--[RubyCell]---["text1"]
                  +--[RubyCell]---["text2"]
Layout:
                           +-------+
                           |"text1"|
                  +-------++-------++-------+-------+
                  |xxxxxxx||"base1"||"text2"|xxxxxxx|
                  +-------++-------++-------+-------+
```

7. baseContainer が二つ以上ある場合のレイアウト

```
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             +--[RubyContainer(beforeText)]
             |    +--[RubyCell]---["text1"]
             +--[RubyContainer(base)]
                  +--[RubyCell]---["base2"]
```

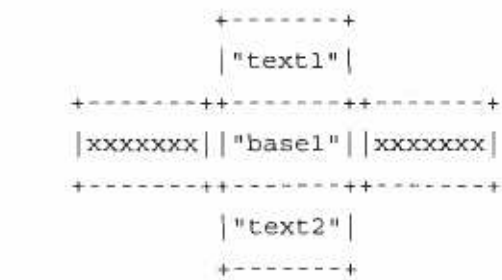
Layout:



8. textContainer が三つ以上ある場合のレイアウト

```
FrameTree: [Ruby]
             +--[RubyContainer(base)]
             |    +--[RubyCell]---["base1"]
             +--[RubyContainer(beforeText)]
             |    +--[RubyCell]---["text1"]
             +--[RubyContainer(afterText)]
             |    +--[RubyCell]---["text2"]
             +--[RubyContainer(text)]
                  +--[RubyCell]---["text3"]
```

Layout:



付録 B

質疑応答

発表後の質疑応答

松田先生の質問

新聞紙面のレイアウトを例に出していたが，あのような複雑なレイアウトを簡単に再現することは可能なのか。また，Word のような文書作成ソフトを用いた方が XML や HTML に親しくないユーザにとって簡単であり，より便利なのではないか。

当日の回答

新聞紙面のレイアウトは多段組の代表的な例であり，多段組そのものは可能になっていますが，全てを現在の Web 技術のみで再現することは困難です。しかし，将来的には新聞紙面のような複雑なレイアウトも可能になるべきだと考えています。

また，Word のような WYSWIG エディタなどに比べて，HTML と CSS を用いて文書のレイアウトを決定することが難しいのは事実です。しかし，新聞社や出版社においては高度で詳細な組版が要求されています。そのため，多少作成が困難であったとしても，高度で詳細な組版を実現するということが優先されると考えています。

後日の回答

Word のようなファイル形式では HTML と CSS のように，構造とスタイルの分離は困難であり，文書を作成するたびにレイアウトを決める必要があります。これに対して，CSS では構造が同じ文書に対して，同一のスタイルを適用することが可能です。現在の新聞は，レイアウトをそのつど決めていると推測されますが，CSS を用いることで，ある程度レイアウトの再利用が可能になると考えています。これは，毎日発行する必要がある新聞では大きな利点になると考えています。

狐崎先生の質問

私の Firefox で利用できるか。また，公式の Firefox にはいつごろ採用されるのか。

回答

ルビに関しては Web 上でソースコードを公開していますが，公式の Firefox には組み込まれておりません。現在，Firefox はバージョン 4.0 をリリースするための作業をすでに始めており，4.0 には採用されないと考えています。しかし，去年の夏にソースコードを公開し，Mozilla の専門家にレビューを受けていくつかの修正を行ってきました。そのため，バージョン 4.1 までには採用されると考えています。

矢吹先生の質問

組版のためにはすでに TEX があり，出版社は XML を基盤とした技術を独自に持っている。このような現状を考慮すると，この研究は同じことの繰り返しになってしまって，突き詰めたところでまた同じものが出来あがるだけではないか。

回答

確かに，TEX の組版はすでに非常に高度なものができていると考えています。しかし，Web は TEX にはない多くの利点を持っています。例えば，TEX はコンパイルされた時点でレイアウトが決定されるため，様々な媒体で閲覧することに対して有効ではありません。他にも多くの利点を持っている Web に従来の組版技術，TEX と同様の組版技術を組み込むことは十分意味のあることだと考えています。

小宮山先生の質問

横書きを縦書きに切り替えたときには半角英数字の扱いが問題になると思うが，その点はどのようになっているか。

当日の回答

書籍において，縦書き中の欧文はグリフを右に 90 度回転させて配置します。そのため，本研究の実装でもそのような回転を行い，配置するようになっています。しかし，略字 (JR や NTT) などに関しては，グリフを回転させず配置させる場合もあります。今回の実装では，それを指定するための実装は行っておりません。

後日の回答

これは，主言語の文章に他言語の文字列がある場合，どのように配置をするかを指定する問題になります。Webにおいて，この配置に関して指定するためには，`text-orientation`プロパティを用います。今回の実装では，このプロパティには対応しておらず，半角英数字を回転させずに配置することはできません。

しかし，半角英数字ではなく全角英数字を用いることで，グリフを回転せずに文字を配置することが可能です。これは，本実装で利用しているテキストレンダリングエンジン (Windows GDI, Pango) がすでにそのような機能を持っているためです。これは一般的な Word などの文書作成ソフトと同じ挙動になっています。

書面での質疑応答

佐久田先生

印刷した場合にはどうなるのか。

回答

Firefox では構築した Frame をもとにして印刷を行っているため，基本的には特に追加の実装を必要とせずに印刷が行えます。しかし，`body` 要素に対して `writing-mode` 指定を行った場合 (ページ全体に縦書き指定した場合)，二枚目以降の印刷が不可能となっています。

`body` 要素に対して `writing-mode` 指定を行った場合には，描画範囲 (Firefox ウインドウの範囲) を示す CanvasFrame に対して回転処理を行い，Web ページ全体の縦書きを実現します。しかし，印刷のためには，描画範囲のみを考慮するのではなく，Web ページ全体を指定された紙のサイズに分割してレイアウトを決定するという処理が必要になります。Firefox では印刷のレイアウトを決定する際，SimplePageSequenceFrame という Frame が自動挿入されます。その Frame で分割処理が行われますが，その部分にページ全体が縦書きの場合の分割処理を実装していません。そのため，二枚目以降の印刷が不可能になっていると考えられます。