

Webエンジニアの 教科書

The textbook of the web engineer

株式会社ヒトメディア

佐々木達也／瀬川雄介／内藤賢司



NoSQLデータベースやフロントエンド、可視化、
ログの取り扱い、環境構築の自動化など、
**Webエンジニアとして知っておくべき技術を、
実際に試せるように、わかりやすく解説！**

**2～3年目までのエンジニア、最新の技術動向を
知っておきたいシステム関係者など必読の1冊！**

 C&R研究所

本書では、Webサービスの開発を行うエンジニアのことを「Webエンジニア」と表現しています。フロントエンドのリッチ化、サーバ側での新しい技術の登場、ログの肥大化、仮想環境構築技術の進化などの影響で、Webエンジニアが担当する技術領域は非常に幅広くなりました。しかし、そういった技術が気になりつつも、たとえば、フロントエンドエンジニアだとフロント側の技術に、サーバサイドエンジニアだとサーバ側の技術にばかり目が行きがちで、それ以外の領域までなかなかキャッチアップできないのが実際のところではないでしょうか。

NoSQLデータベース、フロントエンド、ログの取り扱い、可視化、環境構築の自動化といった技術や、便利な外部サービスについて紹介していきます。こういった技術を名前しか知らないのと、実際に試してみるところまでやっているのとでは雲泥の差なので、ぜひ手を動かしながら試してみてくださいと思います。

（「はじめに」より抜粋）

C&R研究所について

C&R研究所は新潟市にある出版社です。ユニークな社風や教育方針は新聞やテレビなどで紹介されたりします。詳細については、次のWebサイトでご覧いただくことができます。

www.c-r.com

また、新潟本社には2代目会社犬「ラッキー」がいます。名刺を持つ正式な社員として広報部に勤務しつつ、セラピードッグとして社内のメンタルヘルスにも貢献しています。

●会社犬「ラッキー」



Webエンジニアの 教科書

The textbook of the web engineer

株式会社ヒトメディア

佐々木達也／瀬川雄介／内藤賢司



■権利について

- 本書に記述されている社名・製品名などは、一般に各社の商標または登録商標です。
- 本書では™、©、®は割愛しています。

■本書の内容について

- 本書は著者・編集者が実際に操作した結果を慎重に検討し、著述・編集しています。ただし、本書の記述内容に関わる運用結果にまつわるあらゆる損害・障害につきましては、責任を負いませんのであらかじめご了承ください。
- 本書の内容は2015年2月現在の情報を基に記述しています。仕様の変更やバージョンアップなどにより、本書の内容のままでは動作しなくなったりする場合があります。また、紹介しているURLやWebサイトのデザインなども変更になる場合があります。あらかじめ、ご了承ください。
- 本書に記載したサンプルコードは、誌面の都合上、1つのサンプルコードがページをまたがって記載されていることがあります。その場合は▼の記号で、1つのコードであることを表しています。

●本書の内容についてのお問い合わせについて

この度はC&R研究所の書籍をお買い上げいただきましてありがとうございます。本書の内容に関するお問い合わせは、「書名」「該当するページ番号」「返信先」を必ず明記の上、C&R研究所のホームページ(<http://www.c-r.com/>)の右上の「お問い合わせ」をクリックし、専用フォームからお送りいただくか、FAXまたは郵送で次の宛先までお送りください。お電話でのお問い合わせや本書の内容とは直接的に関係のない事柄に関するご質問にはお答えできませんので、あらかじめご了承ください。

〒950-3122 新潟県新潟市北区西名目所4083-6 株式会社 C&R研究所 編集部
FAX 025-258-2801
『Webエンジニアの教科書』サポート係



本書では、Webサービスの開発を行うエンジニアのことを「Webエンジニア」と表現しています。フロントエンドのリッチ化、サーバ側での新しい技術の登場、ログの肥大化、仮想環境構築技術の進化などの影響で、Webエンジニアが担当する技術領域は非常に幅広くなってきました。しかし、そういった技術が気になりつつも、たとえば、フロントエンドエンジニアだとフロント側の技術に、サーバサイドエンジニアだとサーバ側の技術にばかり目が行きがちで、それ以外の領域までなかなかキャッチアップできないのが実際のところではないでしょうか。

近年、「フルスタックエンジニア」という言葉がバズワードとなっています。フルスタックエンジニアとは、インフラからミドルウェア、モバイル、デザイン、設計、プログラミング、デプロイまで、何でもこなせるエンジニアのことを指します。この言葉の是非はあると思いますが、Webエンジニアとして仕事をしていくのであれば、こういったさまざまな技術領域をある程度把握しつつ、自分の強みとなる技術領域を持つT型人才となる必要があると思います。ですが、なかなかそういった情報はまとまっていないので、キャッチアップしようにも時間が掛かってしまうでしょう。そこで本書では、Webエンジニアとして知っておいた方がよいであろう技術をまとめてみました。次のような構成となっています。

■本書の構成

- CHAPTER-01 Webエンジニアについて
- CHAPTER-02 Ruby on Railsでの開発
- CHAPTER-03 PHPでの開発
- CHAPTER-04 NoSQLデータベース
- CHAPTER-05 フロントエンドの実装
- CHAPTER-06 ログについて
- CHAPTER-07 データの可視化
- CHAPTER-08 環境構築の自動化
- CHAPTER-09 便利な外部サービス

NoSQLデータベース、フロントエンド、ログの取り扱い、可視化、環境構築の自動化といった技術や、便利な外部サービスについて紹介していきます。こういった技術を名前しか知らないのと、実際に試してみるところまでやっているのでは雲泥の差なので、ぜひ手を動かしながら試してみてくださいと思います。

対象読者としては、次のような読者を想定しています。

■本書の対象読者層

- 2年目、3年目のエンジニア
- フロントエンドをやっているがサーバサイドにも興味があるエンジニア
- サーバサイドをやっているがフロントエンドにも興味があるエンジニア
- フルスタックエンジニアになりたいエンジニア
- 最近新しい技術を目にするけど試せていないエンジニア

Webエンジニアの世界では常に新しい技術が登場するため、常に学習が必要です。最初は大変かもしれませんが、少しわかるようになってくると途端に楽しくなるくるものです。楽しさが感じられる状態まで到達できるように、本書がさまざまな技術に関して最初の一步を踏み出すきっかけとなることを願っています。

2015年2月

執筆陣を代表して
佐々木 達也



目次 contents

🌐 CHAPTER-01

Webエンジニアについて

□□1	Webエンジニアはどんなことをやっているのか	14
●	Webエンジニアとは	14
●	Webエンジニアがやっていること	14
□□2	必要とされる技術領域	16
●	HTML	17
●	CSS	21
●	フロントエンド	23
●	サーバサイド	23
●	データベース	24
●	Webサーバ	24
●	AWS	24
●	GitHub	25
□□3	次々に新しい技術が登場している	26
●	NoSQLデータベース	26
●	JavaScriptのフレームワーク	26
●	Fluentd	28
●	グラフによる可視化	28
●	仮想化技術	29
●	プロビジョニングツール	29
□□4	今後もWebエンジニアとしてやっていくために	30

🌐 CHAPTER-02

Ruby on Railsでの開発

□□5	Rubyの概要	32
□□6	Ruby on Railsの登場	34
●	Railsの基本理念	34
●	RESTfulな設計	35
□□7	MVC(Model-View-Controller)	36
●	Model	36
●	View	36

● Controller	37
● Concernsディレクトリ	37
□□8 Railsでアプリケーション開発	38
● Railsの開発環境を整える	38
● Railsを触ってみる	40
● OAuth認証	47
● OAuth認証を実装する	48
● Gistsの一覧を取得する	55
● 非同期処理	58
□□9 テストを書こう	61
● テストデータの一元管理	62
● 時間の関係するテスト	63
● モックやスタブを使いこなそう	65
□10 便利なgemの紹介	67
● Pry	67
● Better Errors	70
● MailCatcher	71
□11 gemの探し方	73
□12 まとめ	75

🌐 CHAPTER-03

PHPでの開発

□13 最近のPHP	78
□14 PHPのフレームワーク	79
● フルスタックなフレームワーク	79
● マイクロフレームワーク	81
□15 PHPの実行環境の構築	82
● PHPのインストール	82
● php.iniの設定	85
□16 Composerを使おう	87
● Composerをインストールする	88
□17 PSRを知ろう	89

□ 1 8	PHPアプリケーションの開発環境を構築する	90
●	プロジェクトの作成	91
●	APIの作成	94
□ 1 9	開発に便利なツール	97
●	REPL (Read-Eval-Print Loop)	97
●	コーディング規約のチェック・解析	101
●	その他のツール	103
□ 2 0	PHP開発の環境、エディタ	105
□ 2 1	まとめ	107

🌐 CHAPTER-04

NoSQLデータベース

□ 2 2	NoSQLデータベースとリレーショナルデータベースの違い	110
●	リレーショナルデータベースの特徴	110
□ 2 3	Redis	113
●	なぜRedisを使うのか	113
●	Redisのインストール	115
●	redis-cliを使ってみる	116
●	RubyからRedisを利用する	117
●	ランキング情報を扱う	118
□ 2 4	MongoDB	123
●	なぜMongoDBを使うのか	123
●	MongoDBのインストール	126
●	mongoシェルを使ってみる	127
●	RubyからMongoDBを利用する	128
●	ログを記録したい	129
□ 2 5	まとめ	132

CHAPTER-05

フロントエンドの実装

026	フロントエンド開発	134
●	jQueryの簡単な使い方	135
●	DOM要素の取得方法	136
●	jQueryオブジェクト	139
●	CoffeeScriptやTypeScriptの登場	140
027	CoffeeScript	141
●	公式サイト上で触ってみよう	142
●	CoffeeScriptの導入	142
●	CoffeeScriptを使ってみる	143
028	TypeScript	146
●	公式サイト上で触ってみよう	146
●	TypeScriptの導入	147
●	TypeScriptを使ってみる	147
029	Grunt	149
●	Gruntの導入	150
●	Gruntを使って処理を自動化する	151
030	JavaScriptのフレームワークの登場	154
●	jQueryはDOMの変更に弱い	154
●	フロントエンドの開発が大規模化している	154
031	AngularJSの特徴	156
●	HTMLをそのままテンプレートとして使える	157
●	双方向データバインディング	157
●	DI(Dependency Injection) コンテナ	158
032	AngularJSを使ってみる	160
●	AngularJSの導入	160
●	AngularJSの単純な例	161
●	DOMを操作する	162
●	Controllerを使ってみる	163
●	TODOリストを実装する	166
●	サーバー側と通信する	172
033	Service	176
●	\$location	176

● \$timeout	177
● \$cookieStore(ngCookies)	178
● 初期化处理	179
034 まとめ	180

🌐 CHAPTER-06

ログについて

035 ログはなぜ重要なのか	182
● バグの調査	182
● 不穏な兆候を知る	182
● 仮説に対して検証する	183
● 監査ログ	183
036 Fluentdが登場した背景	184
037 Fluentdとsyslogdの違い	186
● ログの集約方法や記録先など、柔軟にカスタマイズ可能	186
● ログにタグ付けが行えるので管理しやすい	187
● さまざまな言語向けのモジュールが提供されている	187
038 Fluentdを使ってみよう	188
● 入力プラグインと出力プラグイン	188
● Fluentdを導入する	188
● Fluentdの仕組み	189
● Fluentdに適当なメッセージを送信する	190
● ログを監視する	191
● ログを別のFluentdサーバに転送する	193
● ホスト名を付与する	195
● MongoDBに保存してみる	196
● Fluentdの推奨構成	199
039 ログを可視化したい	200
040 Elasticsearch	201
● Elasticsearchを導入する	201
● Elasticsearchを触ってみる	202
● FluentdからElasticsearchにデータを流し込む	204
● インデックスの削除	206

□4.1	Kibana	208
	● Kibanaを導入する	208
	● Kibanaに触ってみる	211
□4.2	まとめ	213

🌐 CHAPTER-07

データの可視化について

□4.3	なぜGoogle Chartなのか	216
□4.4	棒グラフ	218
	● ユースケース	218
	● サンプルを表示する	218
	● グラフをカスタマイズする	220
□4.5	積み上げ棒グラフ	223
	● ユースケース	223
	● サンプルを表示する	223
	● グラフをカスタマイズする	224
□4.6	折れ線グラフ	227
	● ユースケース	227
	● サンプルを表示する	227
	● グラフをカスタマイズする	228
□4.7	散布図	231
	● ユースケース	231
	● サンプルを表示する	231
	● グラフをカスタマイズする	232
□4.8	外部のJSONファイルやURLを読み込みたい	235
	● JSONデータを作成する	235
	● JSONデータの構造を確認する	236
	● JSONファイルを利用するHTMLファイルを作成する	236
	● HTMLファイルをブラウザで表示する	238
□4.9	まとめ	239
	● AWStats	239
	● Webalizer	239
	● Google Analytics	240

● New Relic	240
● Datadog	240
● Mackerel	240

🌐 CHAPTER-08

環境構築の自動化

050 手動での環境構築のリスク	242
● 本番サーバを新たに追加したい	242
● すでに稼働している本番サーバにライブラリやミドルウェアを追加したい	243
● 開発環境を気軽に構築したい	243
051 Vagrant	244
● Vagrantを導入する	244
● Vagrantで仮想マシンを起動する	245
● コマンドやシェルスクリプトの実行	247
● 複数台の仮想マシンをまとめて起動する	250
● Amazon EC2のインスタンスを起動してみよう	251
● その他便利なプラグインの紹介	255
● Vagrantの仮想化イメージ	257
052 Ansible	258
● Ansibleを導入する	259
● インベントリファイルを用意する	261
● 簡単な処理を実行してみる	261
● 「playbook」を使って複雑な処理を実行してみる	262
● 幂等性の確保	264
● ファイルの末尾に追記する	265
● その他の便利モジュール	267
053 Serverspec	270
● Serverspecを導入する	270
● Serverspecの初期設定	270
● テストドリブンなプロビジョニング	271
054 Docker	274
● Dockerの仮想化イメージ	274
● Dockerを導入する	275
● Dockerを試してみる	277
● DockerfileからDockerイメージを作成する	282

● Dockerイメージを共有する	284
● Dockerを使ってCIテストを実行する	286
055 まとめ	288

🌐 CHAPTER-09

便利な外部サービス

056 外部サービスを利用する理由	290
● どうやって見つけるのか	290
● 導入時の注意点	291
057 外部サービスの紹介	292
● Mixpanel - アクセス解析ツール	292
● Slack - コミュニケーションツール	294
● Qiita:Team - 情報共有サービス	295
● Circle CI - 継続的インテグレーションサービス	296
058 まとめ	298

🌐 COLUMN

■ フルスタックエンジニアについて	15
■ PHPを学ぶには	108
■ リレーショナルデータベースはもう必要ない?	112
■ Mongo DB Is Web Scale	131
■ NoSQLの最近の動向	132
■ 「==」と「===」の違い	180
■ 「td-agent」と「Fluentd」の違いは?	213
■ その他のグラフツール	217
■ Ansibleの便利なオプション	269
■ プロトタイピングサービス	298
● 索引	300



CHAPTER

01

Webエンジニア について

▶▶▶ 本章の概要

Webエンジニアとはどのような仕事で、どのような技術を知っている必要があるのでしょうか。要件をまとめたり、設計をしたり、プログラムを書いたりが中心ですが、それ以外にも日々さまざまな作業があるので担当範囲は多岐にわたります。それだけでも大変ではありますが、日進月歩で新しい技術も登場してきており、それら新しい技術のキャッチアップも避けては通れないでしょう。常に学習し続ける必要のある職業といえるかと思います。

本章では、Webエンジニアにとって必要とされる技術領域や、一部ではありますが知っておいた方がよいと思われる技術について紹介していきます。

Webエンジニアは どんなことをやっているのか

🌐 Webエンジニアとは

本書のタイトルにもなっている「Webエンジニア」ですが、Webエンジニアとはどのようなエンジニアなのでしょう。一言でエンジニアといっても、「フロントエンドエンジニア」や「インフラエンジニア」など、さまざまなタイプが存在します。

本書では、RubyやPHPを使ってWebアプリケーションを開発する(フロントエンドなどもやりますが、主にサーバサイドを担当している)エンジニアのことをWebエンジニアと定義します。

🌐 Webエンジニアがやっていること

Webエンジニアは普段、どんなことをやっているのでしょうか。いつもTwitterをやっているイメージがあるかもしれませんが(もちろん仕事上の情報収集のためです)、Webアプリケーションの要件定義や設計、開発、テストといったサービスのリリースや改善に関わる作業がメインとなります。

しかし、それだけではありません。バグのないシステムはありえないので、何かエラーが発生すればその原因を突き止めるために調査をしたり、バグの修正をしたりします。WebアプリケーションのKPI(Key Performance Indicators)やその他の指標の測定のためにログを分析することもあるでしょう。新しい技術の導入のために技術調査をすることもあります。また、自分でコードを書くだけでなく、GitHub上などで他のエンジニアのコードレビューをすることもあります。

- Twitter
- 要件定義
- 設計
- 開発
- テスト
- エラーの原因調査
- バグ修正
- ログ分析

- 技術調査
- コードレビュー

人によっては専業で開発だけをやる、テストだけをやる、という場合もありますが、基本的にはWebアプリケーションを作るために上記の作業はすべてやることになるでしょう。もちろん、すべてをあなたひとりでこなす必要はなく、あなたのチームで分担してこれらの作業をこなすことになります。

大人数のチームでこれらの作業を完全に分業して担当できるならよいかもしれませんが、少人数のチームで仕事を進めていく上ではそうもいきません。各自がある程度さまざまな作業をできる状態が望ましいでしょう。



フルスタックエンジニアについて

近年、「フルスタックエンジニア」というキーワードが話題になっています。これは、フロントエンドだったりサーバサイドだったり、インフラだったり幅広くこなせるエンジニアのことを指しています。

特に、比較的スタートアップ企業の求人によく見かけます。スタートアップ企業ではエンジニアが少数精鋭ですが、やることは山のようにあり、どうしても一人二役、三役が求められるのではないのでしょうか。そういった背景からフルスタックエンジニアのような何でもできるエンジニアが求められてきています。

また、最近は後述するAWSの登場によって非インフラエンジニアでもある程度インフラの運用をすることができるようになってきたことなども、この流れを後押ししていると思います。

必要とされる技術領域

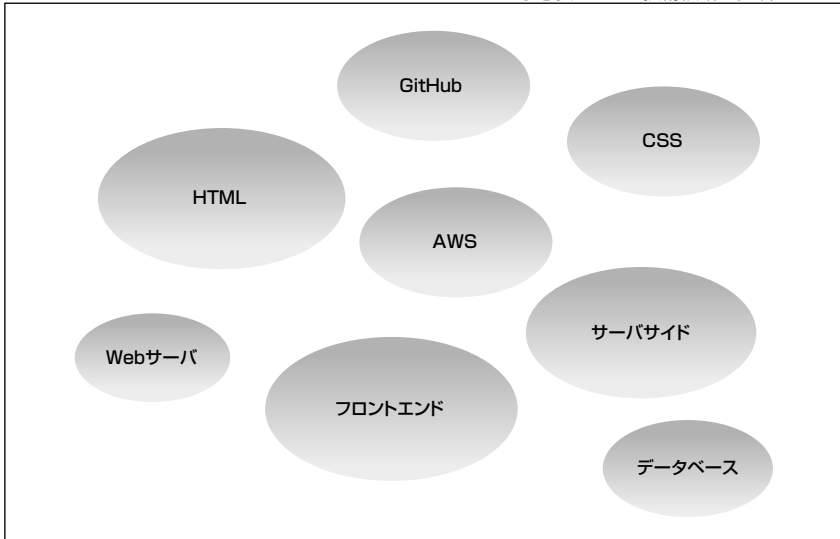
ここまで見てきたように、基本的にWebエンジニアがやる内容は多岐にわたります。技術的な移り変わりも激しいので、3年後、5年後というスパンで見ると、「今と同じ技術を使って同じことをやっている」と自信を持って答えられる人は少ないでしょう。常に新しい技術や手法を学習することが必要になってくるのがWebエンジニアという職業だと筆者は考えます。

Webエンジニアの作業内容について紹介してきましたが、これらの作業内容をこなすために必要とされる技術とは何でしょうか。技術を知らずに開発をしたり、バグの修正をしたり、コードレビューをしたりといったことはできません。

Webエンジニアが必要とされる技術領域も作業内容と同様に多岐にわたります。挙げるとキリがないですが、一般的には次のような技術が挙げられるかと思います。

- HTML
- CSS
- フロントエンド
- サーバサイド
- データベース
- Webサーバ
- AWS
- GitHub

● 必要とされる技術領域は多岐にわたる



それぞれについて少し詳しく見てみます。

🌐 HTML

HTML (HyperText Markup Language) は、Web上の文書を記述するためのマークアップ言語です。Webサイトを作成するにあたって最低限必要とってくる技術であり、皆さんもよく慣れ親しんでいるのではないのでしょうか。

HTMLは最初のバージョンとなるHTML 1.0が誕生してから何度も仕様変更を行い、HTML5が登場してきました。現在、HTML5の基本的な機能はIE9以降およびその他の主要ブラウザでは対応しているので、何らかの事情でIE8以下への対応が必要となるケースを除き、原則、HTML5を使用すべきです。なお、IE8以下へもJavaScriptを用いれば対応可能ではありません。

HTML5なのか、それ以前(HTML4.01もしくはXHTML1.0)なのかは、HTMLの先頭に記述する「doctype」で宣言します。HTML5の指定が最も簡単ですし、基本的にこれを「doctype」として宣言すればよいでしょう。

HTML4.01

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
```

XHTML1.0

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

HTML5

```
<!DOCTYPE html>
```

HTML5には、このような特徴があります。

- 動画や音声、グラフの描画が可能
- 新しく追加されたAPI
- フォーム周りの機能が強化
- 長かった記述が省略可能に
- より明確に文書構造を示すことができる

それぞれについて、少し詳しく見ていきましょう。

◆ 動画や音声、グラフの描画が可能

たとえば、動画を描画したい場合、HTML5を用いれば<video>タグを使って次のように指定できます。HTML5登場以前は、Webサイトで動画を貼り付けるにはFlashやQuickTimeなどを使うのが一般的でした。HTML5であれば画像を描画するのと同じような感覚で、簡単に動画を描画することができます。

```
<video controls>
  <source src="carp.mp4" type="video/mp4">
  <source src="carp.ogv" type="video/ogg">
  <source src="carp.webm" type="video/webm">
</video>
```

●videoタグで動画を描写する



3つの形式で動画を指定していますが、このように指定することで上から順にチェックし、各ブラウザが対応している動画形式で表示してくれます。「type」属性を指定し、ブラウザがDLと再生が可能な形式を判断できるようにします。

他にも<audio>タグで音声を描写したり、<canvas>タグでグラフを描写したりすることも可能です。これらのタグの登場により、表現の幅が増えました。

●動画や音声、グラフの描写

種別	タグ
動画	video
音声	audio
グラフ	canvas

◆新しく追加されたAPI

HTML5では新たに、位置情報を取得したり(Geolocation API)、大量のデータをブラウザに保持させたり(Web Storage)といったAPIが提供されています。

たとえば、Geolocation APIは、GPSを使わなくても無線LAN・WiFi・携帯電話基地局・GPS・IPアドレスなどから位置情報を取得できるHTML5の新機能です。Geolocation APIの登場以前にも携帯端末などで位置情報

を扱うことはできましたが、各携帯キャリア独自の規格であったためキャリアごとの仕様に合わせて開発する必要がありました。また、Geolocation APIの登場により、PCサイトであってもユーザの位置情報を取得することが可能となりました。

◆ フォーム周りの機能が強化

入力項目のバリデーションやプレースホルダーの機能が追加されています。たとえば、次のような機能が利用できます。

```
# 入力された値がemail形式かどうかのチェック
<input name="email" type="email">

# この項目を必須項目とする
<input name="text" type="text" require>

# プレースホルダーに文字を表示させる
<input name="text" type="text" placeholder="例)東京都">
```

◆ 長かった記述が省略可能に

文字コードの宣言が次のように短く書くことができるようになりました。ちょっとしたことですが、楽になります。

```
# HTML4.01 or XHTML1.0
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

# HTML5
<meta charset="utf-8">
```

また、<script>タグや<style>タグの「type」属性は省略可能となり、次のように短く書くことができるようになりました。

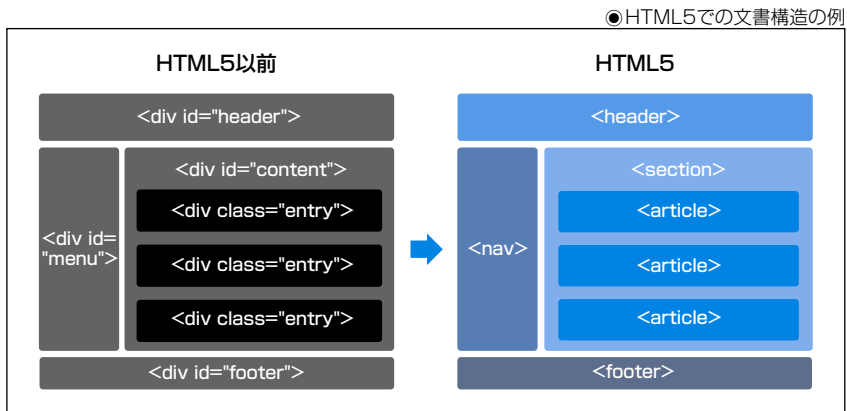
```
<style>
  .main {
    font-size: 15px;
  }
</style>

<script>
  alert("foobar123");
</script>
```

◆ より明確に文書構造を示すことができる

HTMLはマークアップ言語であり、以前から文書構造を示すことは可能でした。ただ、今まではdiv要素にIDやクラスを指定することで、どういう構造なのかを表現していたのではないのでしょうか。HTML5では、文書構造を表す新しい要素が加わることで、ブラウザや検索エンジンに対してより明確に文書構造を伝えられるようになります。

たとえば、ヘッダを示す<header>、フッタを示す<footer>、1つのセクションであることを示す<section>、記事であることを示す<article>、ナビゲーションであることを示す<nav>などの要素が追加されており、それぞれの役割に応じて適切な要素に割り当てることができるようになりました。



HTML5の特徴をざっと見てきましたが、他にもさまざまな特徴があります。興味のある方は、ぜひ調べてみてください。

🌐 CSS

CSSはCascading Style Sheetsの略で、Webアプリケーションのスタイルを指定するための言語です。HTMLを使って文書構造を定義し、CSSを使ってその見栄えを制御するという使い分けになります。

最近ではSCSSと呼ばれるメタ言語も登場してきています。SCSSを使うと、次のように書くことができます。ネストさせることで同じようなタグを何度も指定しなくてよかったり、何度も登場する値は変数として宣言することができ、後からその値を変更したい場合には宣言した1カ所だけの変更で済ませ

ることが可能となります。本書では詳しくは取り上げませんが、mixinという機能も大変強力です。

● sample.scss

```
$basecolor: #3c3c3c;
$graycolor: gray;

@mixin logofont {
  font: {
    family: "lucida grande", "lucida sans", sans-serif;
    size: 300%;
    weight: bold;
  }
}

.contents {
  color: $basecolor;

  h1 {
    @include logofont;
    border-bottom: 1px dotted $graycolor;
  }
}
```

このSCSSをコンパイルをすることで、次のようなCSSとなって出力されます。

● sample.css

```
.contents {
  color: #3c3c3c;
}

.contents h1 {
  font-family: "lucida grande", "lucida sans", sans-serif;
  font-size: 300%;
  font-weight: bold;
  border-bottom: 1px dotted gray;
}
```

SassMeister(<http://sassmeister.com/>) という サイト でSCSSをCSSにその場で変換することができるので、興味のある方は試してみるとよいでしょう。



🌐 フロントエンド

JavaScriptやjQueryを使ったフロントエンドの開発です。たとえば、特定の要素が最初は隠れていてあるボタンをクリックすると表示されるとか、特定の要素が動的に追加されるなどの処理を行います。最近ではWebエンジニアがフロントエンドの処理もある程度は対応するというのが当たり前となっており、Webエンジニアであってもフロントエンドの知識は必須といってもよいでしょう。

🌐 サーバサイド

RubyやPHP、Perl、Pythonなどを使ったサーバサイドの開発です。Webエンジニアとしては最も基本となる技術領域ではないでしょうか。フロントエンド側で表示を制御するにしても、そもそもサーバサイド側でデータを取得してくる必要があります。基本的に何らかのフレームワークを利用して開発を進めることになります。

Rubyの場合、フルスタックなフレームワークであるRuby on Railsが主に利用されています。一方、軽量なフレームワークとしてはSinatraやPadrinoがあります。PHPの場合、群雄割拠ですが、CakePHPやFuelPHP、Symfonyなどが有名でしょうか。最近だとLaravelというフレームワークが人気があります。Perlだと以前はCatalystが使われていましたが、最近だとMojoliciousやMojolicious::Liteがよく使われていそうです。Pythonの場合はフルスタックなDjango、Facebookが開発したTornado、軽量なフレームワークとしてFlaskやBottleなどがあります。

まとめると、次のような状態でしょうか。

●言語ごとの主なフレームワーク

言語	フレームワーク
Ruby	Rails, Sinatra, Padrino
PHP	CakePHP, FuelPHP, Symfony, Laravel
Perl	Catalyst, Mojolicious, Mojolicious::Lite
Python	Django, Tornado, Flask, Bottle

1 データベース

MySQLやPostgreSQL、OracleなどのリレーショナルデータベースもWebアプリケーションを作る上で避けては通れません。極端な言い方をすれば、データベースからデータを取得してそれを表示するのがWebアプリケーションです。当然、データベースを操作するためのSQLを扱える必要があります。

また、Webアプリケーションの場合、データベースからのデータの読み出しをいかに早くするかが重要です。Amazonの調査では、ページの表示速度が0.1秒遅くなると、売り上げが1%低下することが明らかになったといわれています。また、Googleではページの反応が0.5秒遅くなると、アクセス数が20%低下すると発表されています。さらに、米Aberdeen Groupの2008年の調査では、スピードが1秒遅くなるとページビューが11%、コンバージョンが7%、顧客満足度が16%低下することが報告されているなど、ページの表示速度は大きな影響を及ぼしていることがわかります。

このようにページの表示速度が売り上げやアクセス数に影響を及ぼすことが知られています。ページを素早く表示するためには、インデックスをきちんと設定するなど、データベース周りの理解は欠かせないでしょう。

2 Webサーバ

Webアプリケーションを公開するために、ApacheやNginxなどのWebサーバを扱うことも避けては通れません。近年はNginxがよく使われるようになりました。

3 AWS

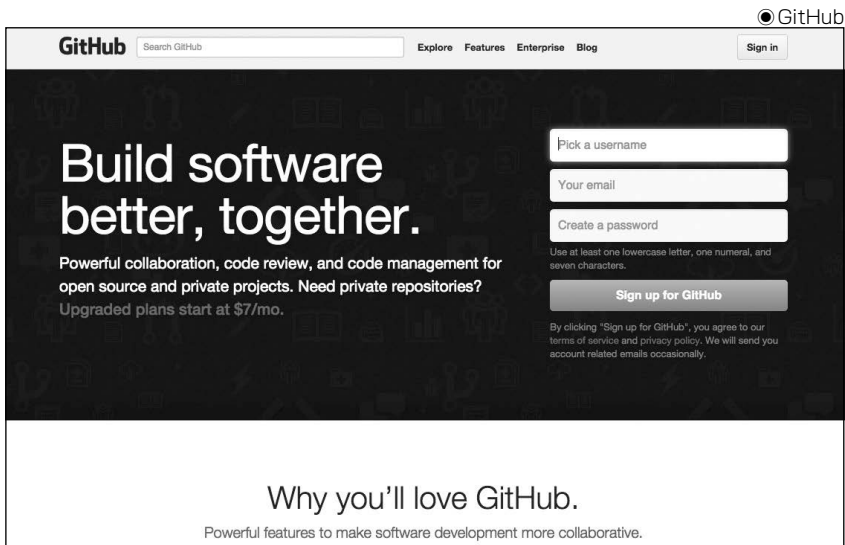
最近ではクラウドサービス、中でも特にAWS(Amazon Web Service)を利用してWebサーバを立てたり、データベースを構築したりといった事例が増えてきています。AWSでは、Amazon EC2、Amazon S3など非常に多くのサービスが提供されています。本書では触れませんが、AWSを利用する

企業はどんどん増えてきているので、今後はAWSを扱う技術や経験も必要になってくるでしょう。少しずつでも実際に手を動かして試してみてください。



GitHub

コードのホスティングサービスとしてGitHubを利用するケースも増えてきました。GitHub上でPRを出したり、コードレビューをしたりしている方も多いのではないのでしょうか。また、GitHubを使うにもコードのバージョン管理を行うにも、そもそもGit操作ができないと困ると思います。



次々に新しい技術が登場している

Webエンジニアを取り巻く技術は日進月歩で変化しており、先に挙げたものの以外にも次々と新しい技術が登場してきています。すべてを紹介することはできませんが、Webアプリケーションを開発・運用する上で知っておいた方がよさそうな周辺技術のいくつかは本書で取り上げたいと思います。こういった新しい技術の登場を楽しめるかどうかはWebエンジニアとしてやっていく上で大切なかもしれません。

- NoSQLデータベース
- JavaScriptのフレームワーク
- Fluentd
- グラフによる可視化
- 仮想化技術
- プロビジョニングツール

NoSQLデータベース

NoSQLデータベースとしてはMemcachedやRedis、MongoDBなどがあります。リレーショナルデータベースのあまり得意じゃない部分に対して強みを持っていることが多く、適材適所で利用するとよいでしょう。

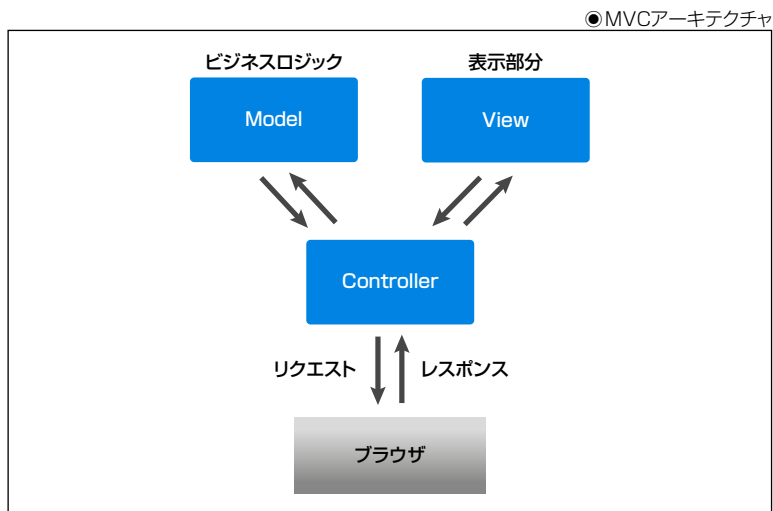
NoSQLデータベースについては、CHAPTER-04で詳しく扱います。

JavaScriptのフレームワーク

フロントエンドの実装といえばjQueryを使ってDOMを操作して行うかと思っています。jQueryの登場により、面倒だったJavaScriptでのDOM操作などがとても簡単に記述できるようになりました。しかし、近年はフロントエンドの開発が大規模化してきたこともあり、jQueryだけでは大規模なJavaScriptのプログラムを管理するのが難しくなってきているのが現状です。

jQueryはあくまでもDOM操作やAjaxなどをクロスブラウザで使うことを簡単にするという目的で作られたものであり、大規模なJavaScriptのプログラムを管理しやすくするという機能は持っていません。大規模なJavaScriptのプログラムを管理するために、フロントエンド側でもきちんと設計する必要があります。

良い設計の手法として代表的なものがMVCです。MVCというのは、Model(ビジネスロジックを担当する部分)、View(表示を担当する部分)、Controller(入力を受け取りModelとViewをつなぐ部分)という、大きく3つにコードを分けてプログラムを記述する設計手法です。



MVCはもともとサーバサイドでよく使われる設計手法ですが、フロントエンドでもこのMVCの考え方を設計に取り入れることが多くなってきています。最近では、MVCの考え方を設計に取り入れたJavaScriptのフレームワークとして、Backbone.jsやAngularJSが台頭してきました。

jQueryで開発していくのは大変わかりやすく便利なのですが、jQueryは対象の要素選択時にDOMの探索が発生するため、どうしてもDOMの変更に弱く、デザイン変更などでDOM構造を変えると動かなくなってしまうことも多々あります。Backbone.jsやAngularJSのようなフレームワークを使えば、DOMの探索が少なくなり、DOM構造の変更に強くなるというメリットもあります。

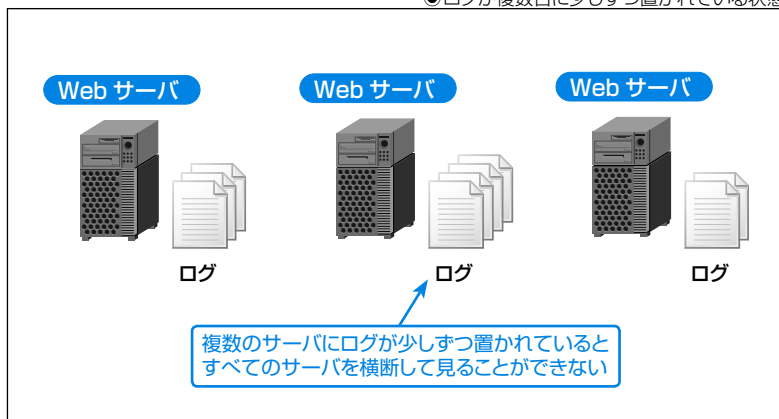
JavaScriptのフレームワークについては、CHAPTER-05で詳しく扱います。

Fluentd

ログをどのように扱うかは大切です。何か問題があったときに原因を探るにも、Webアプリケーションの改善を行っていくにも、ログがなければわからないことは多いと思います。

近年、サーバを複数台並べて(スケールアウト)負荷を分散させるのが当たり前になってきていますが、その場合困るのがログの扱いです。複数のサーバにログが少しずつ置かれている状態では、いざ見たいと思ったときにすべてのログを簡単に見ることができません。

●ログが複数台に少しずつ置かれている状態



この問題を解決してくれるのが、ログ収集ツールであるFluentdです。Fluentdを使うと、複数のサーバに点在するログをほぼリアルタイムに特定のサーバへ集めることが可能です。ログは出力しているだけではあまり意味がなく、必要となったときにすぐに調査できる状態にしておくことが大切です。

Fluentdについては、CHAPTER-06で詳しく扱います。

グラフによる可視化

ログを分析して、KPIなどさまざまなデータを測定している方も多いでしょう。管理画面などでそういった値を表示することも多いかと思います。そのような場合、得られたデータをグラフにして可視化する技術も必要になります。可視化することで、数字に強い一部の担当者だけではなくチーム全体で共有しやすくなります。

可視化については、CHAPTER-07で詳しく扱います。