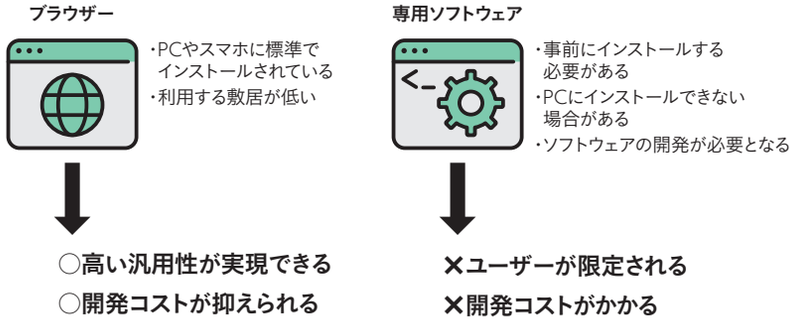
■ ブラウザーで汎用性の高いシステムを実現

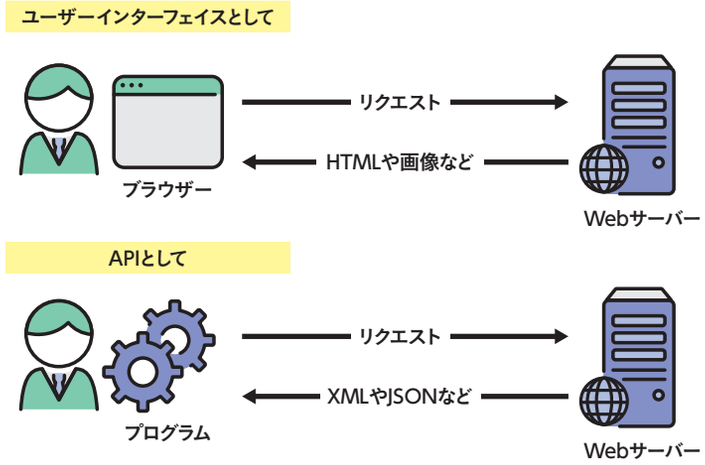


ブラウザーを利用しないWebシステム

Webというとブラウザーを思い浮かべる人が多いかもしれませんが、ブラウザーを利用しないWebシステムもたくさん存在します。

たとえば、ソーシャルゲームアプリやスマホアプリなどはブラウザーではなく専用アプリで利用しますが、裏側ではWeb技術の**Web API**（Application Programming Interface）を活用しています。Web APIについては8-3を参照してください。

■ ブラウザーとAPIによるWebシステム



02 インターネットの誕生

Web技術は、インターネットと切り離して考えることはできません。混同されがちなWebとインターネットですが、当初の開発目的は異なっていました。ここでは、インターネットが誕生した経緯をかんたんに見ていきましょう。

インターネット以前のネットワーク

インターネットが普及する前は、コンピューター同士の通信にさまざまな方式が用いられていました。

たとえば企業内で販売管理や発注管理を行うシステムや、銀行の預貯金管理システムやATMなどのシステムは、インターネットを利用することなく専用回線と専用通信方式でシステムが構成されていました。

インターネットが存在していない時代はこれが当たり前であり、コンピューター同士を接続する場合は、専用端末や専用回線、また専用の通信方式で行っていました。

コンピューターが普及するにつれて、どんなコンピューターも接続でき、かつさまざまなサービスに対応した、汎用性の高いコンピューターネットワークが求められるようになりました。

コラム インターネットの軍事利用に関する誤解

「インターネット」のルーツとなったARPANETは、米国の国防総省傘下にあったARPA（Advanced Research Projects Agency）の資金援助により開発されました。そのため、インターネットが軍事利用を目的に開発されたと誤解されがちですが、実際は汎用性の高い新しいネットワークの開発が目的でした。

DARPAと改称された現在でも、さまざまな研究への資金提供を行っており、対象となる研究はすべて一般から公募されています。

07 レスポンスメッセージ

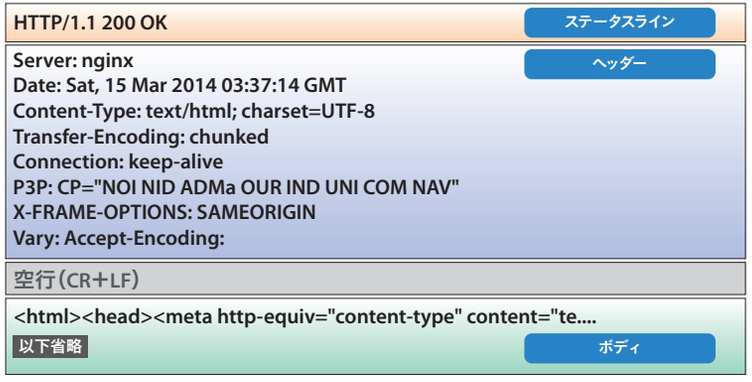
HTTPメッセージのうち、サーバーからクライアントへのレスポンスで使用されるものをレスポンスメッセージと呼びます。このレスポンスメッセージの構造や中身について解説します。

レスポンスメッセージの構造

レスポンスメッセージは1行目がステータスライン、2行目から空行までがヘッダー、空行の次から末尾までがボディで構成されています。レスポンスメッセージには、HTMLテキストや画像などのリソースとともに、成功や失敗といったリクエスト処理の結果も返送されます。

HTTPのようなリクエスト・レスポンス型のプロトコルでは、レスポンスに含まれるステータス（処理結果）が重要になります。成功した場合だけでなく、失敗した場合は何が原因となっているのか、サーバーからクライアントに知らせるようになっています。

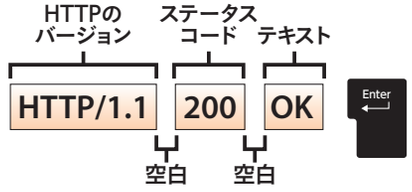
■ レスポンスメッセージの構造



ステータスライン

レスポンスメッセージの1行目にあたるのが**ステータスライン**です。以下の図にあるような書式を用いります。

■ ステータスラインの書式



先頭の**HTTPバージョン**には、使用するHTTPのバージョンがセットされます。次の**ステータスコード**には100〜500番台で値がセットされます。最後の**テキスト**には、ステータスコードの概要を説明する短い文章がセットされます。

■ ステータスコード

ステータスコード	概要	説明
100番台	情報（Informational）	処理が継続されていることを通知する。ほとんど使用されていない
200番台	成功（Success）	リクエストの処理に成功したことを通知する
300番台	リダイレクト（Redirection）	リクエストを完遂するには、さらに新たな動作が必要であることを通知する
400番台	クライアントエラー（Client Error）	リクエストの内容に問題があるためリクエスト処理に失敗したことを通知する
500番台	サーバーエラー（Server Error）	リクエスト処理中にサーバーにエラーが発生したことを通知する

ステータスコードを確認すればリクエストが成功したのか、それとも失敗したのかがわかります。また失敗した場合も値によって何が原因だったかを探ることができます。

TLS 1.2を使用したHTTPSがサポートされています。またブラウザーでも、古いSSL/TLSが使用された場合に警告を表示したり、アクセスをブロックしたりするようになっています。

2021年8月現在、TLSの最新バージョンは、2018年8月にリリースされたTLS 1.3です。TLS 1.2以前のバージョンに存在した古い暗号アルゴリズムが削除されたり、新たなハンドシェイクのしくみが導入されるなど、通信パフォーマンスに関する改善が主に行われています。

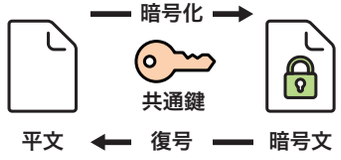
よりセキュアな暗号化通信を実現するために、Webサーバーのソフトウェアやブラウザーを随時アップデートして、常に最新のTLSを使用するようにしてください。

共通鍵暗号方式と公開鍵暗号方式

SSL/TLSによる暗号化通信では、**共通鍵暗号方式**と**公開鍵暗号方式**の2つの暗号化方式を使用します。共通鍵暗号方式では、暗号化と復号に同じ鍵を使用するのに対し、公開鍵暗号方式では、暗号化と復号に別々の鍵を使用します。

共通鍵暗号方式では、**送信元での通信内容の暗号化と、受信側での復号に同じ鍵**を使用します。そのため、鍵を送付しておくなどして、あらかじめ共有しておく必要があります。鍵を送付する際に第三者に盗聴されてしまうと、通信内容が復号され解読されてしまいます。

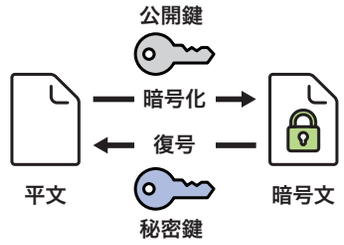
■ 共通鍵暗号方式



公開鍵暗号方式では、**暗号化に公開鍵、復号に秘密鍵**を使用します。公開鍵だけを送信側の暗号鍵として公開し、その公開鍵で暗号化された通信内容を秘密鍵を使用して復号します。

公開鍵によって暗号化されたメッセージは、ペアとなる秘密鍵を使用しないと復号することができません。秘密鍵は他人に公開する必要がない鍵であるため、適切に保管しておけば本人以外はメッセージを解読できず、盗聴される危険性もありません。

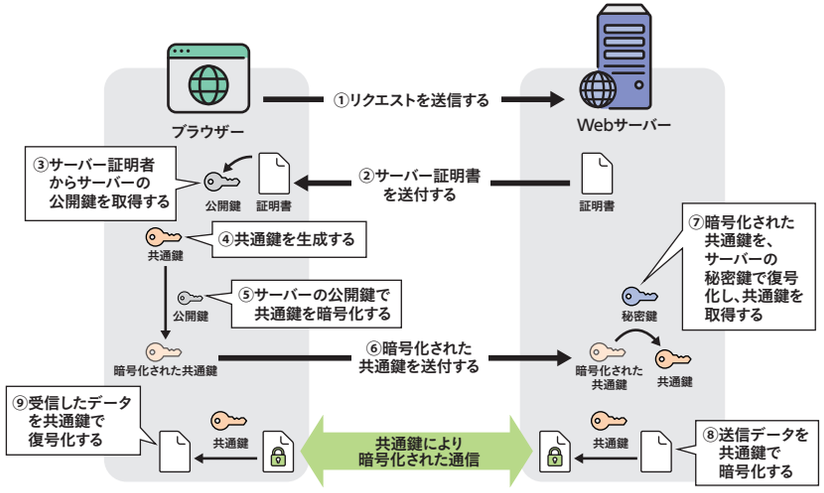
■ 公開鍵暗号方式



よりセキュアな暗号化通信の手順

通信データの暗号化に共通鍵を使用し、共通鍵の交換手順に公開鍵暗号方式を使用し、ブラウザーとWebサーバー間で、よりセキュアな暗号化通信を実現するための手順を以下に示します。

■ SSL/TLS暗号化通信の手順



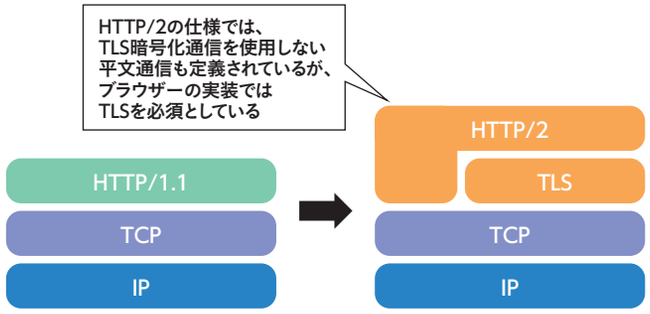
08 HTTP/2の特徴

HTTP/2は、パフォーマンスとセキュリティの改善を大きな目標としています。多くの技術仕様が追加され、より効率的なネットワークリソースの活用が可能となっています。

HTTP/2の特徴

HTTP/2では、パフォーマンスとセキュリティの改善が重要視されたことは4-7で述べました。多くの機能が追加され、より効率的にネットワークリソースの活用が可能となりましたが、その中でも**HTTPセッションを張る際にTLSによる暗号化通信を行う**ことが大きな特徴です。

■HTTP/2のプロトコルスタック



その他HTTP/2の主な特徴は以下の通りです。

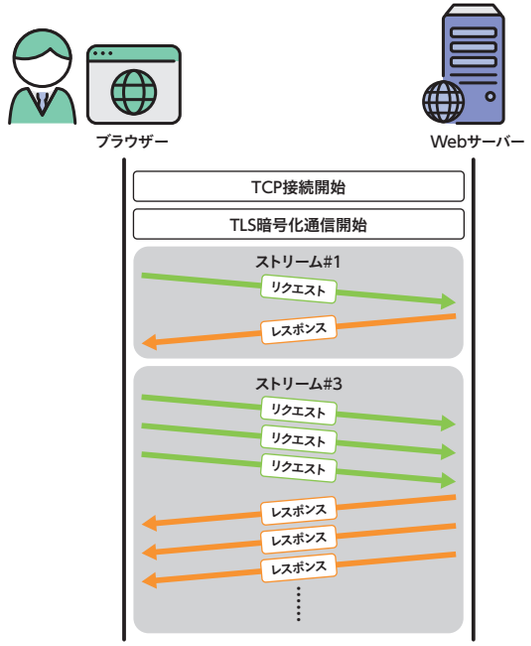
- ストリームによる多重化とTCPコネクションの再利用
- バイナリーフレーム
- プロトコルネゴシエーション
- HTTPヘッダーの圧縮
- 優先度によるストリーム制御
- サーバープッシュ
- 常時TLS暗号化通信によるセキュリティの向上

ストリームによる多重化

HTTP/1.1では、HTTPリクエストが発生した際にTCPコネクションを開始し、レスポンスの完了を待って切断します。TCPコネクションを開始する際、TCPの3ウェイハンドシェイクを行う必要がありますが、さらにHTTPSによるSSL/TLS通信を開始するには、SSL/TLSハンドシェイクを行う必要もあります。これらのやり取りが大量に発生することで、ネットワークやWebサーバーに大きな負担がかかります。

HTTP/2では、1つのTCPコネクションを有効に使い回せるよう、**ストリーム**と呼ばれる仮想的な接続を生成します。ストリームの中で複数のリクエストとレスポンスを並列に処理することで多重化を可能にしています。

■ストリームによるリクエストとレスポンスの多重化



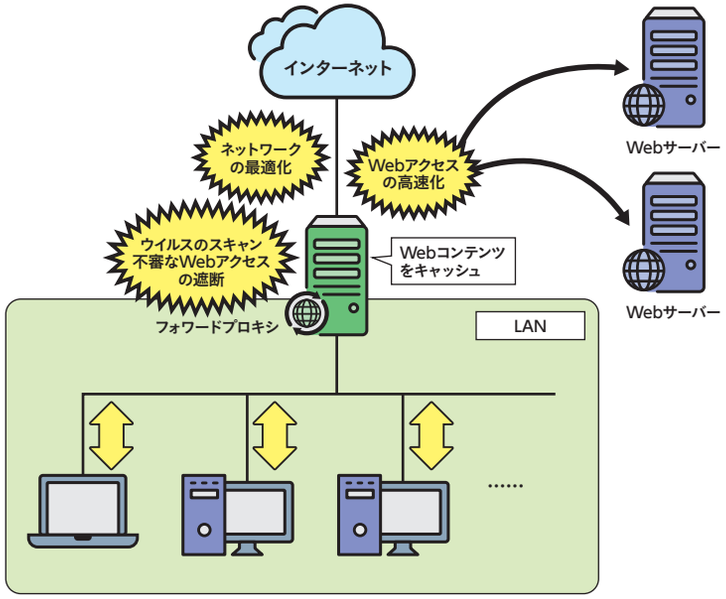
しやすくなり、同時にWebアクセスの最適化が可能となります。

また最近では、セキュリティ向上を目的として、プロキシサーバーを導入するケースも増えています。

プロキシサーバーでWebサーバーとのやり取りをチェックし、危険であると判断した場合は通信を遮断するという、**WAF**（Web Application Firewall）のような機能を果たします。また**Webコンテンツのウィルスチェックやスキャン**、**Webサーバーの URL チェック**などを行うことも可能です。

さらに**レスポンスの高速化**というメリットもあります。プロキシサーバーがWebアクセスを中継する際、オリジンサーバーのリソースをキャッシュして、次に同じリクエストが発生した際は、このキャッシュデータを再利用してレスポンスの高速化が可能になります。この機能を**フォワードプロキシ**機能（6-4参照）と呼びます。

■ フォワードプロキシ



Webサーバー側のメリット

Webサーバー側でプロキシサーバーを導入した場合は、ブラウザーからのリクエストをオリジンサーバーに代わって一旦プロキシサーバーが受けて、オリジンサーバーに転送します。

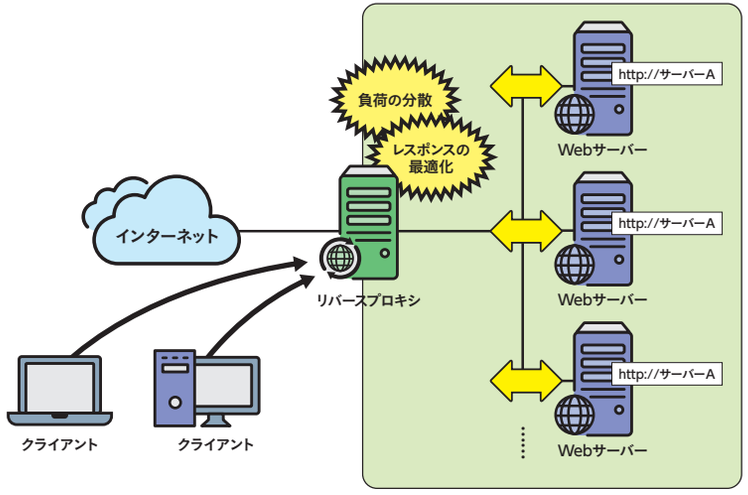
オリジンサーバーからのレスポンスもプロキシサーバーを経由して、ブラウザーに送信されます。またオリジンサーバーのコンテンツをプロキシサーバーにキャッシュすれば、レスポンスの高速化も実現できます。

オリジンサーバーに処理を分散させることを**ロードバランシング**（6-5参照）と呼び、**リバースプロキシ**機能（6-4参照）が利用されます。

リバースプロキシ機能では、ブラウザーからのリクエストを一旦中継してからバックエンドのWebサーバーへ振り分けます。複数のWebサーバーに処理を分散させることが可能な他、Webサーバーの構成を外部から隠蔽するといった効果もあります。

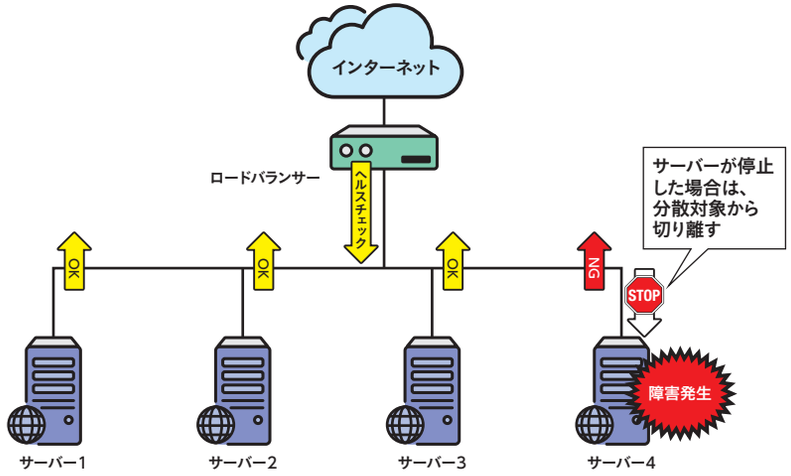
バックエンドのオリジンサーバーへの振り分けには、**URL マッピング**という手法が用いられます。特定のURLにマッチした場合は、指定したオリジンサーバーにリクエストを転送するというしくみです。振り分けの比率や優先順位を設定することで、耐障害性を高め、レスポンスの高速化が可能になります。

■ リバースプロキシ



動的な負荷分散では、**ヘルスチェック機能**で各オリジンサーバーをモニタリングし、モニターデータをロードバランサー内部に保持する必要があります。そのためロードバランサーのしくみが複雑になり、高いレスポンスを実現するには、より高性能なロードバランサーが必要になります。

■ ヘルスチェック



ロードバランサーによっては、複数の方式を組み合わせることも可能です。たとえば、最速応答時間方式と最少コネクション方式を組み合わせることで、よりサーバーの負荷状況を反映させた設定を行えます。

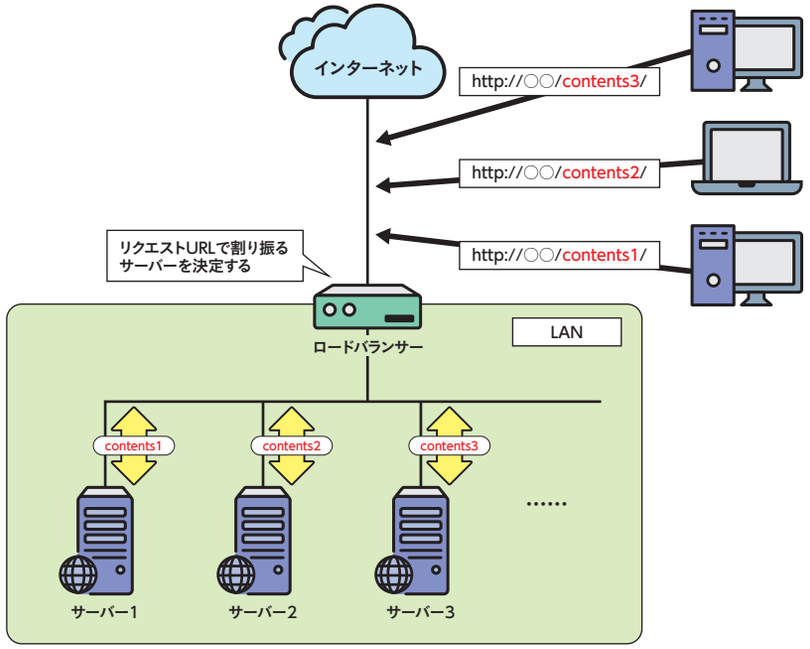
またラウンドロビン方式と優先順位方式を組み合わせて、ある一定量まではプールされたオリジンサーバーをラウンドロビンで分散し、しきい値を越えた場合のみ、オフロード用のオリジンサーバーに振り分けるなどを行うことも可能です。

コンテンツスイッチング

コンテンツスイッチング方式では、HTTPヘッダーやリクエストメッセージの中身によって割り振りを変更することができます。たとえばURLに「img」が含まれている場合はサーバー1、拡張子が「.php」の場合はサーバー2に割り振る、などが可能です。

またコンテンツスイッチング方式によって、画像やHTMLなどの静的コンテンツ専用のオリジンサーバーと、Webアプリケーション専用のオリジンサーバーに分けて運用することも可能です。

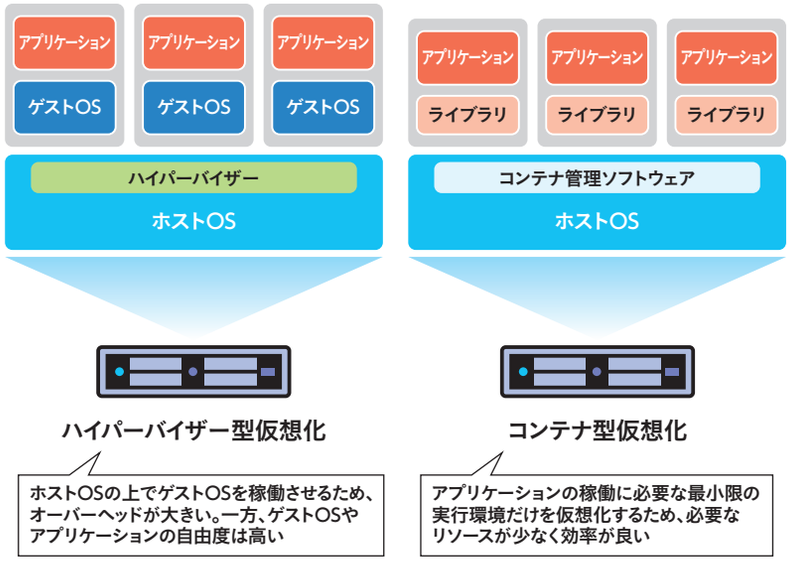
■ コンテンツスイッチング方式による負荷分散



コンテンツスイッチング方式では、HTTPリクエストに含まれる、URL、言語、クッキーなど、さまざまな属性を割り振るための条件として指定できます。

ラウンドロビン方式やその他の動的な負荷分散方式では、TCPレベルでパケットを解析しますが、コンテンツスイッチング方式では、HTTPレベルでパケットを分析するため、**L7スイッチ**とも呼ばれています。

■ 仮想化技術の手法



ハイパーバイザー型では、ホストOS上でハードウェアをエミュレートし、ゲストOSを起動します。二重でOSが起動するため起動が遅く、ハードウェアリソースを大量に消費します。

一方コンテナ型では、アプリケーションの実行に必要なプロセスやリソースをコンテナ化し、他のプロセスから切り離して実行します。そのため必要なリソースが少なく起動も高速です。

クラウドサービス

ここ数年で**クラウドサービス**の利用が急速に増えています。「クラウド」と呼ばれるインターネット上のサービスを利用して、Webサーバーをかんたんに立ち上げるのがWebシステム構築の主流になりつつあります。

クラウドでは、サーバー・ストレージ・ネットワークなどのハードウェアリソースを抽象化することで、これらの詳細を知らなくてもかんたんに利用できるようになります。

またクラウドアプリケーションを通して運用管理できるようにAPIが提供されています。

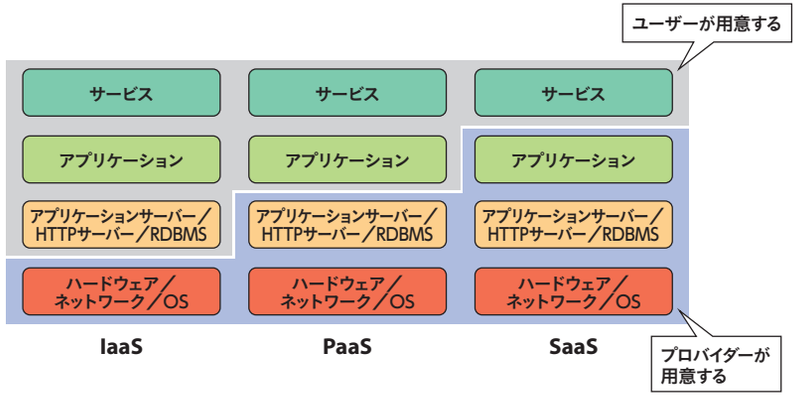
クラウドサービスには、さまざま形態があります。

ハードウェアリソースやインフラを提供する形態の**IaaS（Infrastructure as a Service）**、アプリケーションのためのプラットフォームを提供する**PaaS（Platform as a Service）**、そしてサービスやアプリケーションを提供する形態の**SaaS（Software as a Service）**などの種類があります。

開発や運用の自由度は「IaaS ＞ PaaS ＞ SaaS」と順になります。ただし自由度が高い分、開発や運用管理にかかるコストも大きくなります。

たとえば、自社で機器を用意して運用管理もすべて主体的に行う**オンプレミス**環境上で動いているWebアプリケーションをクラウドサービスに移行する場合は、一般的にはIaaSかPaaSを選択して移行作業を行います。

■ さまざまな形態のクラウドサービス



IaaS

ネットワークやサーバーなどのインフラを好きなときに好きなだけオンデマンドに利用できるのが、**IaaS（Infrastructure as a Service）**です。APIを通してリソースを操作できるため、CPUやメモリーのスケールアップや、サーバーのスケールアウトなどをプログラムに代行させて、大規模なインフラを瞬時に

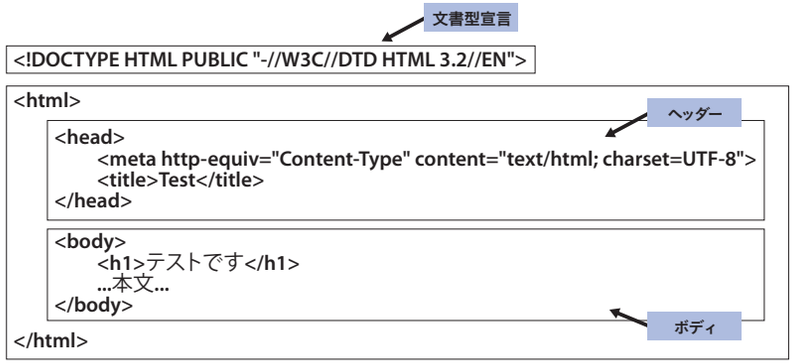
03 HTMLの基本構造

HTMLはさまざまな要素で構成されています。HTMLを記述する上で欠かせない要素について解説します。

文書型宣言・ヘッダー・ボディ

HTMLドキュメントは、大きく**文書型宣言**、**ヘッダー**、**ボディ**で構成されています。HTTPメッセージ（3-5参照）にもヘッダーとボディが含まれているため、単にヘッダー・ボディといっただけでは、どちらを指しているのかわかならないため、注意が必要です。

■ HTMLドキュメントの基本構造



文書型宣言

HTMLドキュメントの先頭には、その文書がどのような**文書型定義**（**DTD**：Document Type Definition）に基づいて記述されているかを「**<!DOCTYPE ...>**」を使って宣言します。HTMLには複数のバージョンが存在し、バージョンごとに使えるタグや属性が異なっていたり、指定方法が異なるためです。そこでどのバージョンに基づいて作られたHTMLドキュメントなのかをここで明確にしています。

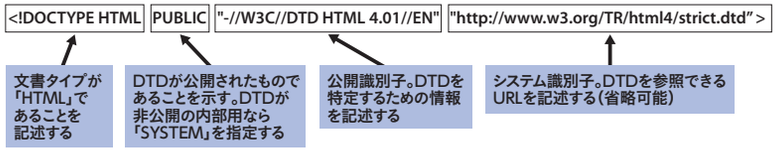
同じバージョンであっても、新バージョンへの移行のための互換性を配慮したものや、逆にそうでないものなど、DTDが異なるものが複数存在します。

HTMLドキュメントで使用される主なDTDは以下の通りです。

- HTML 3.2 DTD
- HTML 4.01 Transitional DTD（移行用）
- HTML 4.01 Frameset DTD（フレーム使用可能）
- HTML 4.01 Strict DTD（厳格）
- HTML5
- ISO-HTML

「<!DOCTYPE ...>」は以下のように記述します。DTDを参照できるURLを示す**システム識別子**は省略可能です。

■ 文書型宣言（DTD）の指定方法



「<!DOCTYPE ...>」はブラウザーで解釈され、レンダリングの方法を決定します。最近のブラウザーは**DOCTYPEスイッチ**と呼ばれる機能で、表示モードを切り替えることができます。詳細は7-4で解説します。

ヘッダー

ヘッダーには、ページタイトル、使用言語（文字コード）、キャッシュの有効期限、検索サイトのためのキーワード、サイトの説明など、HTMLドキュメントに対するメタデータを挿入します。

07 WebSocket

ブラウザーとWebサーバーが双方向通信を可能にする、軽量なネットワークプロトコルがWebSocketです。このWebSocketのメリットとしくみについて見ていきましょう。

WebSocketとは

WebSocketは、クライアントとWebサーバーとの間で双方向通信を行うための軽量なネットワークプロトコルです。当初はHTML5の機能の1つとして策定されましたが、現在は単独のプロトコルとして規定されています。ゲームやSNSなど、リアルタイム性が要求されるアプリケーションなどで採用されています。

通常のWebサービスでは、ブラウザーからWebサーバーに対してリクエストを送信できても、逆にWebサーバーからブラウザーに対して接続を試みることはできません。

WebSocketでは、一度Webサーバーとブラウザー間で接続が確立すれば、Webサーバーとブラウザーのどちらからでも接続を開始できます。そのため、Webサーバー側からデータのプッシュ配信を行う**サーバープッシュ**が可能になります。

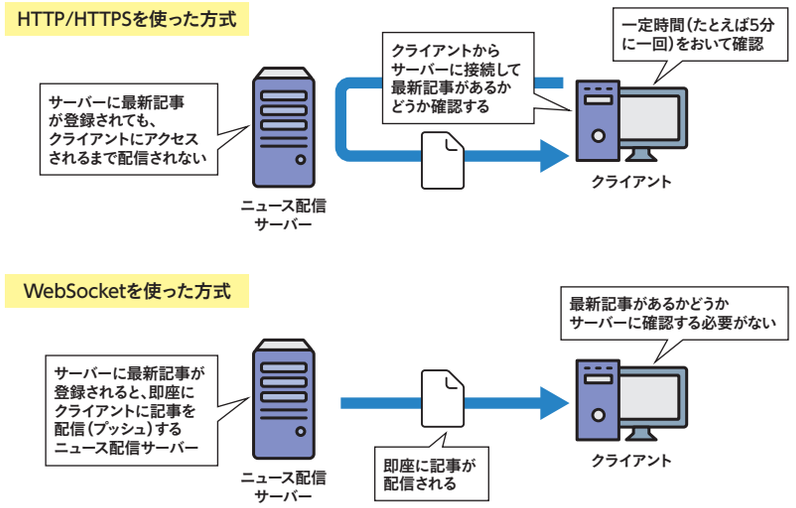
WebSocketのメリット

Webサーバーの情報が更新されたかどうか、ブラウザーから定期的に確認を行わなくても情報の更新と同時に、Webサーバーからブラウザーに更新情報を送信できるため、即時性の高い情報にとても有効です。

たとえばニュース配信システムの場合、HTTP/HTTPSを使った方式では、クライアントが決まったタイミングで最新ニュースの有無をサーバーに確認する必要があります。この方式だとタイミングがずれた場合は、最新ニュースの受信に時間差が発生する可能性があります。

しかしWebSocketを利用すれば、最新ニュースがサーバーに登録されると同時に、クライアントへの通知や配信が可能になるため、よりリアルタイムでニュースの配信が可能になります。

■即時性の高い情報に有効なWebSocket



WebSocketは、一度確立した接続を維持し続けます。接続と切断を繰り返すHTTP/HTTPSと比べてオーバーヘッドが少なく、またヘッダーのサイズもHTTPと比べて極めて小さくなります。そのため通信量の削減が可能で、サイズの小さいメッセージをひんぱんにやり取りするようなケースで効果的です。

WebSocketのしくみ

WebSocketによるクライアントとサーバー間の通信手順は以下の通りです。

①最初にHTTPでクライアント側がサーバーに対してハンドシェイクリクエストを送ります。
②次にサーバー側はハンドシェイクレスポンスを返して、コネクションが確

04 Webアプリケーションフレームワーク

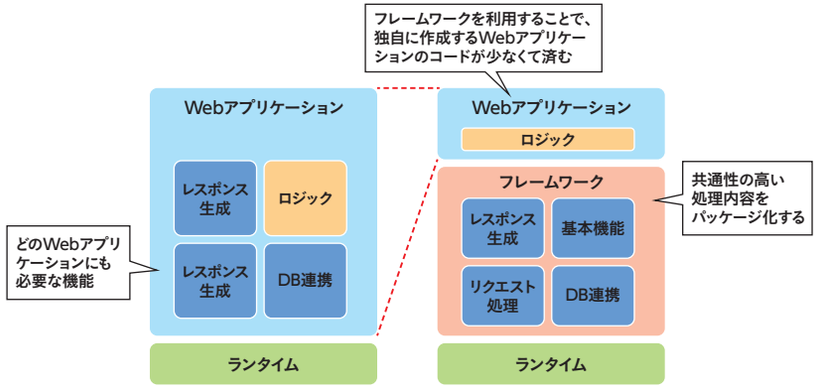
Webアプリケーション開発には多くの知識と技術が必要です。開発規模が大きくなるほど開発工数も膨らみます。そこで開発のスピードや効率を向上させるためには、Webアプリケーションフレームワークを利用します。

Webアプリケーションフレームワークとは

Webアプリケーションフレームワーク（以下フレームワーク）とは、Webアプリケーション開発において、ひんぱんに使用する機能をライブラリーにしたものや、共通性の高い処理内容をパッケージ化してまとめたものです。

リクエスト処理、データベースとの連携、レスポンスの生成など、Webアプリケーション開発に必要な機能を容易に扱えるようになります。そのため、フレームワークを利用したほうがイチからコードを書くよりも、開発効率が格段に向上します。

■ Webアプリケーションフレームワークの役割



フレームワークの基本機能

Webアプリケーションの基本機能は、クライアントからリクエストを受け取り、データベースからデータを読み取ってWebコンテンツを作成し、それをレスポンスとしてクライアントに返すことです。この基本機能をベースに、個々のWebアプリケーションに必要な機能を加えていきます。

たとえばユーザー認証（サインアップ、サインイン、サインアウト）、管理者用画面、フォーム、ファイルのアップロードなど、Webアプリケーションでよく使われる機能も、フレームワークで用意されたものを利用することで、開発工数やコストを軽減することが可能です。

フレームワークの設定ファイルを編集するだけで、簡易なWebアプリケーションをすぐに開発できます。データベースサーバーに接続してSQLを発行するなどの連携も可能で、データベースとの接続・再接続・切断などのコネクション管理もフレームワークに任せることができます。

また開発工数やコストの軽減だけでなく、間違いが少なくなるため、Webアプリケーションの品質が向上するというメリットもあります。

主なフレームワーク

プログラミング言語の**Java**は、サーバーサイド向けの**Servlet**や**JSP**が開発され、フレームワークの**Struts**が登場したことで、エンタープライズ分野で大きなシェアを獲得しました。またスクリプト言語のRubyは**Ruby on rails**というフレームワークが登場したことで、さらに注目されるようになりました。

■ 主なフレームワーク