6.1 HTTPの重要性

HTTP - Webの基盤となるプロトコル

HTTPは名前こそハイパーテキストの転送用プロトコルだが、
コンピュータで扱えるデータであれば、何でも転送できる

HTTPは、RESTを実現する上で重要な以下の機能を実現している

・統一インターフェイス
・ステートレスサーバー
・キャッシュ

6.2 TCP/IPとは何か

TCP/IP - インターネットの基盤を構成するプロトコル

TCP/IPは「階層型プロトコル」になっており、4つの階層がある

層ごとに抽象化することで、下位層の具体的な部分に左右されない

1. ネットワークインターフェース層 (イーサネット)

物理的なケーブルやネットワークアダプタに相当する

2. インターネット層 (IP)

IPアドレスを送信先として、パケット単位でデータを通信する

自分のネットワークインターフェースでデータを送り出すことだけ保証する
それ以降のこと (送信先に届くかなど) は保証しない

3. トランスポート層 (TCP, UDP)

インターネット層で保証されないデータの転送を保証する (TCPのみ)

TCPは、接続先の相手に対してコネクションを張って、
データが到達したかを確認する

転送したデータが、どのアプリケーションに渡るかを決定するのが、
ポート番号

4. アプリケーション層 (HTTP, SSH, DNS など)

具体的なインターネットアプリケーションを実現する

TCPでプログラムを作る時は、ソケット(Socket)というライブラリを使う
ソケットは、ネットワークデータのやり取りを抽象化したAPIのこと

6.3 HTTPのバージョン

HTTP/1.1

シンプルさを保ちながら、機能拡張が行なわれている

HTTP/2

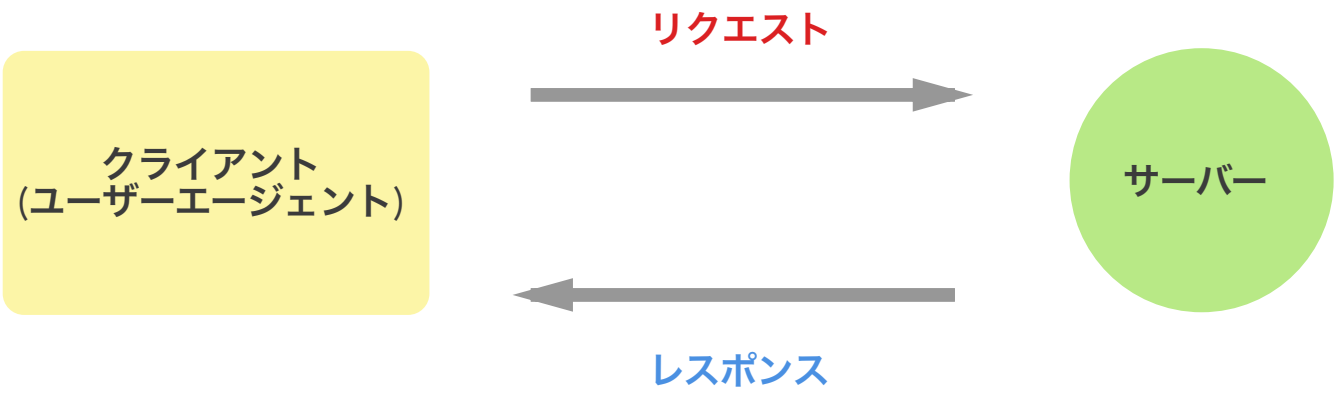
SPDY から派生したHTTP/1.1のパフォーマンスを改善した版

refs: https://codezine.jp/article/detail/8663

6.4 クライアントとサーバ

HTTPの具体的な仕組みを見ていく

Webはクライアント / サーバーを採用している

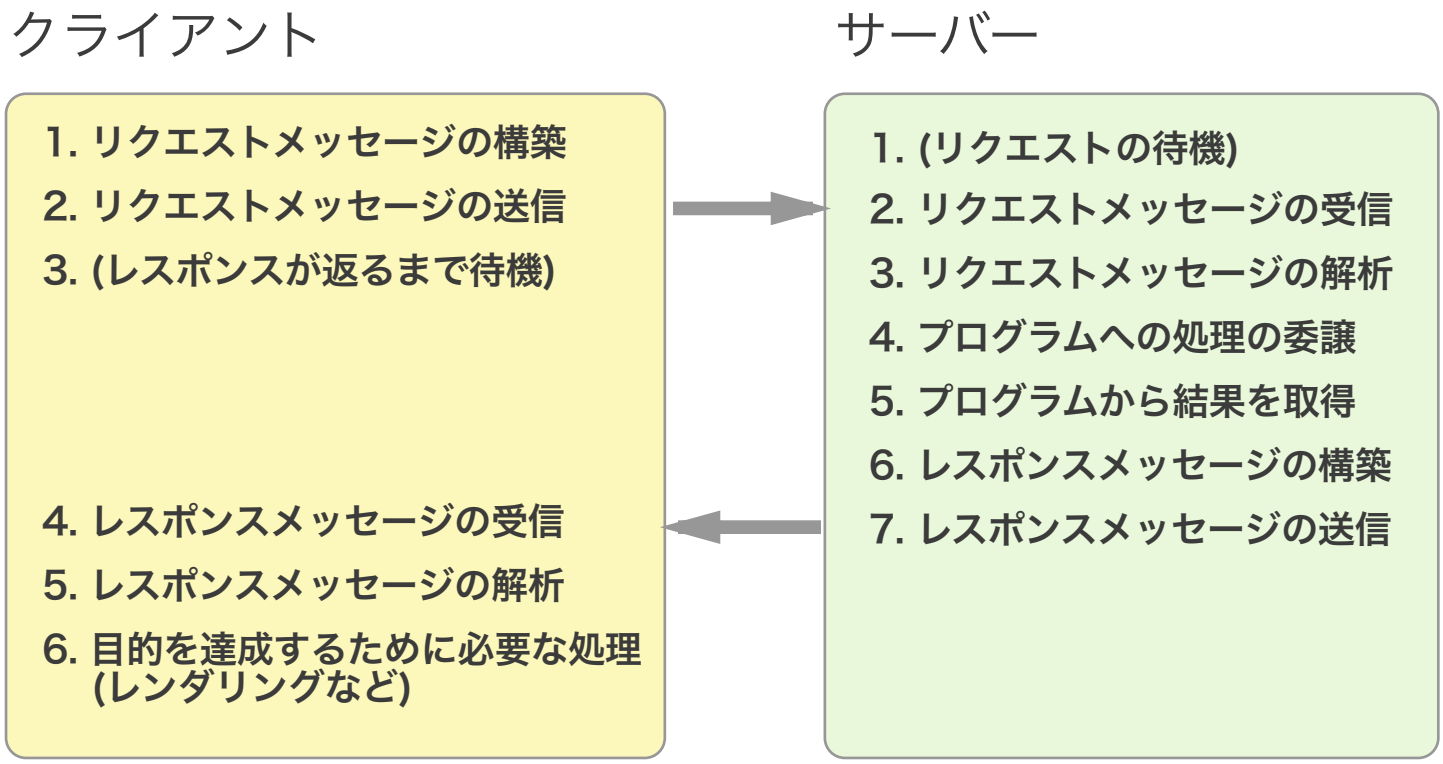


6.5 リクエストとレスポンス

HTTPは同期型 (Synchronous) のプロトコル

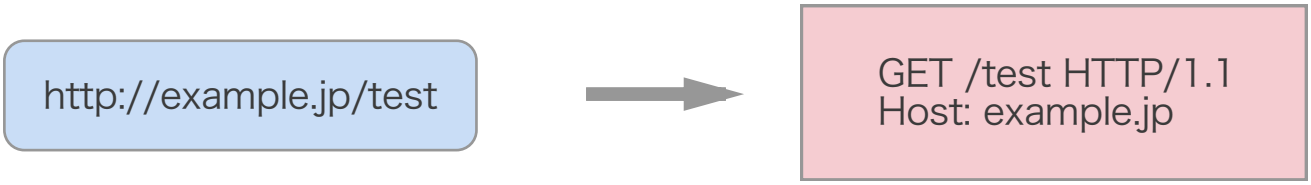
リクエストを出したクライアントは、レスポンスが返るまで待機する

クライアントとサーバーで行われていること



6.6 HTTPメッセージ

リクエストメッセージ



リクエストライン

「メソッド」 「リクエストURI 」「プロトコルバージョン」で構成

上記の例では、GET /test HTTP/1.1 の部分

リクエストURIは、パス以降の文字列か、絶対URIを指定する
URIフラグメント (#) は含めない

ヘッダ

メッセージのメタデータで、1つのメッセージに複数持てる。省略可能。

上記の例では、Host: example.jp の部分
このヘッダは、「Hostヘッダ」と呼ばれる
リクエストURIにポート番号が指定されている場合は、Hostヘッダに書く

「Key: value」 形式で記述する

ボディ

ヘッダの後に、そのメッセージを表す本質的な情報が入る。省略可能。

上記の例ではなかったが、リソースを新しく作成する場合に、
リソースの表現そのものが入る
ヘッダとボディの間には、CRLF (空行) が必要

レスポンスメッセージ

```
HTTP/1.1 200 OK
Content-Type: application/xhtml+xml; charset=utf-8

<html xmlns="http://www.w3.org/1999/xhtml">
…
</html>
```

ステータスライン

「プロトコルバージョン」 「ステータスコード」 「テキストフレーズ」
で構成される

上記の例では、HTTP/1.1 200 OK の部分

ヘッダ

リクエストメッセージと同様

上記の例では、Content-Type: application/xhtml+xml; charset=utf-8 の部分
HTMLのMIMEメディアタイプと、エンコーディング方式を指定している

ボディ

リクエストメッセージと同様

上記の例では、HTMLが記述されている の部分

レスポンスメッセージ

```
HTTP/1.1 200 OK
Content-Type: application/xhtml+xml; charset=utf-8

<html xmlns="http://www.w3.org/1999/xhtml">
…
</html>
```

6.7 HTTPのステートレス性

アプリケーション状態と、2つのアーキテクチャ

アプリケーション状態とは、クライアントの操作の状態のこと
別名「セッション状態 (Session State)」ともいう

セッションとは、
システムにログインしてからログアウトするまでの一連の操作のこと

ステートフルとは、アプリケーション状態をサーバーが管理する
アーキテクチャのこと

デメリットとしては、クライアントの数が増えた場合に
データを同期するオーバーヘッドが発生し、スケールアウトしにくい

ステートレスとは、アプリケーション状態をサーバーで管理しない
アーキテクチャのこと

アプリケーション状態は、クライアントのリクエストメッセージに格納する
このメッセージを、「自己記述的メッセージ (Self Descriptive Message)」
という

メリットとしては、サーバーはアプリケーション状態を管理しないでいいので、
常に新しく来るリクエストの処理に集中すれば良くなる。
クライアント数が増えた時も、サーバーを増やすだけでスケールアウトできる

デメリットとしては、
・クライアントのデータ送信量が増えるため、パフォーマンス劣化
・認証処理などで、DBアクセスが発生すると、サーバーのパフォーマンス劣化
・通信エラーへの対処が必要