

IT 資格 道 場

HTML5プロフェッショナル 認定試験 レベル1 学習用テ キスト

予備知識ゼロから合格ラインへ —— 読んで理解する1冊目

無料サンプル(第1章まで)

LPI-Japan HTML5プロフェッショナル認定試験 レベル1 出題範囲(Ver2.5)準拠
2026年10月版

ご利用にあたって

- このテキストは「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルも著作物です。転載・改変しての再配布・二次販売はできません。
- AI モデルの学習データとしてのご利用(学習目的の入力・データセット化を含む)はできません。
- 製品版(学習用テキスト・頻出度別ノート・暗記用ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

目次

この試験の範囲 [overview]

第1章 HTTP, HTTPSプロトコル [1.1.1]

HTTPの基本(クライアント／サーバー・リクエスト／レスポンス・通信の手順)

URIとURL(書式・利用可能文字・構成要素)

リクエストメソッド(GET/POST/HEAD/PUT/DELETE)

HTTPヘッダフィールド(リクエスト／レスポンス／エンティティ)

ステータスコード(2xx/3xx/4xx/5xx)

リダイレクト(転送の仕組み・3xx・Location)

Cookie(Set-Cookie・属性・容量)

認証方式(Basic認証・Digest認証)

SSL/TLSとHTTPS

HTTPバージョンの違い(HTTP/1.0・1.1・2)

Webブラウザサーバー間の通信手順(接続・持続的接続)

サンプルはここまでです

この試験の範囲 [overview]

HTML5プロフェッショナル認定試験 レベル1 は、HTML5 と CSS3 を使って Web ページを作る基礎知識を問う試験です。運営は特定非営利活動法人エルピーアイジャパン (LPI-Japan) で、出題範囲は Ver2.5 に従います[^1]。試験の形は次の表のとおりです。

項目	内容
問題数	約60問
試験時間	90分 (守秘義務への同意5分、試験80分、アンケート5分)
出題形式	選択式がほとんどで、キーボード入力問題も多少
合格点	非公表
受験方式	コンピュータ試験 (CBT)

出題範囲は5つの分野で、この教材も同じ順に章を立てます。

分野	章	内容
Webの基礎知識	第1～3章	HTTP と HTTPS、HTML の書式、Web に関わる技術と攻撃への対策
CSS	第4～6章	スタイルシートの基本、CSS によるデザイン、カスケード
要素	第7～9章	要素と属性の意味、メディア要素、インタラクティブ要素

分野	章	内容
レスポンシブWebデザイン	第10～11章	マルチデバイス対応、メディアクエリ
APIの基礎知識	第12～15章	マルチメディアとグラフィックス、デバイスアクセス、オフラインとストレージ、通信

公式は小項目ごとに重要度（おおよその問題の比率を表す相対値）を公表していて、合計は92です。

HTML と CSS の例は、手元のブラウザで一度開いて表示を確かめてください。

この教材の狙いは、コードを読んで頭の中で動かす力をつけること。

第1章 HTTP, HTTPSプロトコル [1.1.1]

HTTPの基本(クライアント／サーバー・リクエスト／レスポンス・通信の手順)

HTTP は、Web ブラウザと Web サーバーが文書や画像をやり取りする時の約束事(プロトコル)です。通信を始める側を「クライアント」、要求を受けて返す側を「サーバー」と呼び、ブラウザはクライアントの代表です。

やり取りは 1 回ごとに、クライアントが送る HTTPリクエストと、サーバーが返す HTTPレスポンスの 1 組で終わります。向きは固定で、サーバーのほうから好きな時にリクエストを送ることはできません。名前は向きで覚えます。送る側のメッセージが HTTPリクエスト、返す側のメッセージが HTTPレスポンスです。ステータスコードはレスポンスの 1 行目に入る 3 桁の数字で、メッセージ全体の名前ではありません。

HTTP は前のやり取りを覚えていません。この性質はステートレスと呼ばれ、同じ利用者が続けて 2 回アクセスしても、サーバーにはそれぞれが初対面の要求に見えます。ログインの状態は、あとで見る Cookie とセッション管理で補います。サーバー側からの通知は HTTP だけでは送れません。WebSocket や Server-Sent Events といった別の仕組みを使います。

メッセージの形は、実際のやり取りで確かめられます。

```
GET /new.html HTTP/1.1
Host: 127.0.0.1:45678
User-Agent: curl/8.7.1
Accept: */*
```

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 11
Connection: keep-alive

<p>new</p>
```

上がリクエスト、下がレスポンスです。リクエストの 1 行目はリクエストラインで、メソッド (GET)、リクエスト対象 (/new.html)、HTTP のバージョンが並びます。レスポンスの 1 行目はステータスラインで、バージョン、ステータスコード (200)、理由句 (OK) が並び、メソッド名は入りません。2 行目以降は「名前: 値」の形のヘッダフィールドで、終わりを空行で示し、その後ろがボディになります。空行は省けません。省くと、どこからが中身か分からなくなります。Content-Length が表すのはヘッダではなくボディの大きさで、例のボディ `<p>new</p>` は改行を入れて 11 バイトなので、値の 11 と合います。

URIとURL (書式・利用可能文字・構成要素)

URI は、インターネット上のものを一意に指し示す識別子です。URL はそのうち「場所」で指し示すもので、URI の一種にあたります。Web ページの住所とし

て使うのはほとんど URL なので、アドレス欄に打つ文字列は URL と呼んで構いません。

URL は次の部品に分かれます。

```
https://user@www.example.com:8080/path/to/file.html?
id=10&mode=edit#section2
```

部品	上の例での値	役割
スキーム	https	使うプロトコル
ユーザー情報	user	認証に使う名前 (今はほとんど使わない)
ホスト名	www.example.com	接続先のサーバー
ポート番号	8080	サーバー内の窓口の番号
パス	/path/to/file.html	サーバー上のリソースの位置
クエリ	id=10&mode=edit	「名前=値」を & でつないだ追加情報
フラグメント	section2	ページ内の位置

ポート番号を省略すると、スキームごとの既定が使われ、http は 80 番、https は 443 番です。例のように `:8080` と書けば既定を上書きします。`@` の前の `user` はドメイン名の一部ではなくユーザー情報で、ホスト名は `@` の後ろの [www.example.com](https://www.example.com:8080/path/to/file.html?id=10&mode=edit#section2) です。フラグメントはブラウザがページ内の移動に使う部分で、サーバーには送られません。

ホスト名の大文字小文字は区別されないので、Ping-t.com と書いても ping-t.com と書いても同じサーバーを指します。

URL に書ける文字は、ASCII の英数字と一部の記号に限られます。そのままデータとして使える記号は `- . _ ~` の 4 つだけです。`:` `/` `?` `#` `[` `]` `@` や `!` `$` `&` `'` `(` `)` `*` `+` `,` `;` `=` は URL の区切りに使うので予約されていて、データとして書きたい時にはエンコードが必要になります。日本語などの文字も同じ扱いで、UTF-8 のバイト列を `%` と 16 進数 2 桁で表すパーセントエンコーディングに直さなければ使えません。「あ」は `%E3%81%82` になります。

`https://example.com/日本語/ページ.html?q=あ` を変換すると、次の形になります。

```
https://example.com/%E6%97%A5%E6%9C%AC%E8%AA%9E/%E3%83%9A%E3%83%BC%E3%82%B8.html?q=%E3%81%82
```

ブラウザのアドレス欄は読みやすいように日本語で表示しますが、通信に乗るのはエンコードした形です。

リクエストメソッド (GET/POST/HEAD/PUT/DELETE)

メソッドは、リクエストの 1 行目に書く「何をしてほしいか」の指定で、主なものは次の 9 つです。

メソッド	働き
GET	リソースを取得する
HEAD	GET と同じ応答のヘッダだけを求める (ボディは返らない)

メソッド	働き
POST	サーバーにデータを送信する。送ったデータをサーバーに処理させる
PUT	指定した URI にボディの内容でリソースを作る。あれば置き換える
DELETE	指定した URI のリソースを削除する
OPTIONS	そのリソースで使えるメソッドなどを問い合わせる
PATCH	リソースの一部だけを変更する
CONNECT	プロキシ経由のトンネルを作る
TRACE	経路をたどってリクエストを送り返させる

CREATE というメソッドはありません。データの送信は POST、URI を指定したリソースの作成は PUT で行います。

GET はパラメータを URL のクエリ文字列に載せます。次のリクエストでは、q と page が URL の ? の後ろに入り、ボディはありません。

```
GET /search?q=html&page=2 HTTP/1.1
Host: 127.0.0.1:45678
```

URL に入るので、ブックマークや履歴に残り、同じ条件のページを人に伝えやすい反面、パスワードのような値は載せられません。POST は同じデータをボディに入れて送ります。

```
POST /post HTTP/1.1
Host: 127.0.0.1:45678
Content-Length: 16
Content-Type: application/x-www-form-urlencoded

name=taro&age=30
```

HTML のフォームで method 属性に書けるのは `get` ・ `post` ・ `dialog` の 3 つで、`head` は書けません。`get` なら入力値がクエリ文字列として URL に付き、`post` ならボディに入ります。`get` から `post` に変えると、次の 3 つが起こります。

- 入力値がアドレス欄の URL に現れなくなる
- URL の長さの制限を受けなくなる
- `enctype="multipart/form-data"` と組み合わせて、ファイルを送れるようになる

`get` のままでは、ファイルを選んでも中身は送られず、ファイル名だけがクエリ文字列に付きます。通信の暗号化はメソッドの仕事ではないので、`post` にしても暗号化はされません。ステータスコードも、メソッドではなくサーバーの処理の結果で決まります。

HEAD は、大きなファイルを取る前に更新日時や大きさだけ確かめたい時や、リンク切れの確認に使います。同じ URL に HEAD を送ると、ヘッダは GET と同じで、ボディだけがありません。

```
HTTP/1.1 200 OK
```

```
Content-Type: text/html; charset=UTF-8
```

```
Content-Length: 11
```

```
Connection: keep-alive
```

メソッドには「安全」と「冪等」という性質があります[^2]。安全とは、サーバーのリソースの状態を変えないことで、GET・HEAD・OPTIONS・TRACE が当たります。冪等（べきとう）とは、同じリクエストを何度送っても結果が同じになることです。安全なメソッドに PUT と DELETE を加えたものが冪等で、GET・HEAD・PUT・DELETE・OPTIONS・TRACE の 6 つになります。PUT は同じ内容で何度置き換えても結果が変わらず、DELETE も消したあとに何度送っても「無い」状態が続きます。POST は送るたびに新しいリソースが作られうるので冪等ではなく、PATCH と CONNECT も冪等と定義されていません。通信が切れた時に自動で再送してよいかは、この性質で決まります。

REST 風の API では、URI でリソースを、メソッドで操作を表します。 `/users/42` に対して、GET は取得、PUT は送った内容での置き換え、DELETE は削除です。OPTIONS が返すのはメソッドの情報で、会員情報の取得には使えません。HEAD もボディを返しません。

HTTPヘッダフィールド(リクエスト／レスポンス／エンティティ)

ヘッダフィールドは、メッセージに添える「名前: 値」の付帯情報で、誰がどの向きで使うかによって 4 つに分けて覚えます。

分類	代表的なフィールド	内容
リクエスト	Accept・Accept-Language・Accept-Encoding・User-Agent・Referer・Cookie・Host・Authorization	クライアントからサーバーへの情報
レスポンス	Server・Set-Cookie・Location・WWW-Authenticate・Retry-After	サーバーからクライアントへの情報
エンティティ	Content-Type・Content-Length・Content-Encoding	ボディそのものの情報
両方で使う	Cache-Control・Date・Connection	通信全体の指示

Accept は、クライアントが受け取れるメディアタイプをサーバーに伝えます。メディアタイプは「種類/サブタイプ」の形で、`text/html` ・ `text/css` ・ `image/png` ・ `image/webp` ・ `application/json` などがあり、サーバーはこの値を見て、たとえば WebP に対応したブラウザには WebP の画像を返せます。リクエストヘッダのうち、取り違いやすい4つを挙げます。

- **Accept-Language** : 受け取りたい言語
- **Accept-Encoding** : 受け取れる圧縮方式 (`gzip` など)
- **User-Agent** : ブラウザなどの種類
- **Referer** : 直前に見ていたページの URL。流入元の分析に使う。綴りは仕様の初期の誤記がそのまま残ったもの

Accept 系のフィールドで相手に合わせて内容を選ぶ仕組みを、コンテンツネゴシエーションと呼びます。

`Content-Type` は、ボディのメディアタイプを伝えます。`Content-Type: text/html; charset=UTF-8` なら、ボディは HTML 文書で、文字エンコーディングは UTF-8 です。受け取れる種類を伝える `Accept` とは向きが逆で、送るボディの種類を表し、ボディのあるリクエストにも付きます。UTF-8 は文字エンコーディングであって、圧縮方式ではありません。圧縮は `Content-Encoding: gzip` で伝え、ブラウザは展開してから内容として扱います。この時の `Content-Length` は、圧縮された状態で実際に送られるボディのバイト数です。HTTP ヘッダに `charset` があれば、文書の meta による宣言は効きません(第2章)。

`Cache-Control` は、キャッシュ(ブラウザや中継サーバーに残す手元のコピー)の扱いを指示します。`max-age=600` は、600 秒間はコピーを再利用してよいという意味で、単位は秒です。`no-store` は保存そのものを禁じ、`no-cache` は保存しても使う前に必ずサーバーへ確認させます。`no-cache` という値を取るのには `Cache-Control` で、`Referer` や `User-Agent` にそのような値はありません。

次のレスポンスヘッダでは、サーバーのソフトウェアが Apache、この CSS を 600 秒間再利用でき、ボディは gzip で圧縮された 2048 バイトと読み取れます。

```
HTTP/1.1 200 OK
Server: Apache
Cache-Control: max-age=600
Content-Type: text/css; charset=UTF-8
Content-Length: 2048
Content-Encoding: gzip
```

`Server` が表すのはサーバー側のソフトウェア名で、ブラウザ名は入りません。

ステータスコード (2xx/3xx/4xx/5xx)

ステータスコードは、リクエストの処理結果を表す 3 桁の数字で、レスポンスのステータスラインに入ります。百の位で次のように分類します。

分類	意味	代表例
1xx	情報	100 Continue
2xx	成功	200 OK・201 Created・204 No Content
3xx	リダイレクト・追加の動作が必要	301・302・303・304・307・308
4xx	クライアントエラー (要求側の問題)	400・401・403・404・405
5xx	サーバーエラー (処理側の問題)	500・502・503・504

4xx がクライアントエラー、5xx がサーバーエラーなので、取り違えると監視の集計が逆になります。4xx の代表は次の 5 つです。

- 400 Bad Request: リクエストの書式が誤っていて解釈できない
- 401 Unauthorized: 認証が必要である
- 403 Forbidden: 権限が無く、アクセスを拒否した
- 404 Not Found: リソースがサーバー上に見つからない
- 405 Method Not Allowed: そのリソースに許可されていないメソッドが使われた

401 は「認証していない」、403 は「認証していても権限が無い」という違いです。405 のレスポンスには、使えるメソッドの一覧が `Allow` ヘッダフィールドで

付きます。GET と HEAD だけを許したリソースへ DELETE を送ると、次の応答が返ります。

```
HTTP/1.1 405 Method Not Allowed
```

```
Allow: GET, HEAD
```

```
Content-Length: 0
```

5xx の代表も 4 つあります。

- 500 Internal Server Error: サーバー内部の予期しない障害
- 502 Bad Gateway: 中継サーバーが上流から不正な応答を受けた
- 503 Service Unavailable: 過負荷やメンテナンスで、サービスを一時的に提供できない
- 504 Gateway Timeout: 上流の応答が間に合わなかった

メンテナンス中の案内には 503 を使い、再試行の目安を `Retry-After` で伝えます。

```
HTTP/1.1 503 Service Unavailable
```

```
Retry-After: 3600
```

```
Content-Length: 0
```

2xx は成功を表し、次の 3 つが代表です。

- 200 OK: 内容を返した成功
- 201 Created: リソースが作られた成功
- 204 No Content: 成功したがボディが無い。フォームの保存のように、画面を変えずに成功だけ伝えたい時に使う

304 Not Modified は 3xx に属し、キャッシュを使ってよいという合図です。ブラウザが画像を再取得する時は、手元のコピーの日時を `If-Modified-Since` に付けて送り、更新が無ければ、サーバーはボディの無い 304 だけを返します。

```
GET /logo.png HTTP/1.1
```

```
If-Modified-Since: Mon, 01 Sep 2025 10:00:00 GMT
```

```
HTTP/1.1 304 Not Modified
```

```
Last-Modified: Mon, 01 Sep 2025 10:00:00 GMT
```

```
Content-Length: 0
```

ブラウザはこの応答を受けて、保存してあるキャッシュの画像を表示します。変更があれば、200 と新しいボディが返り、304 にはなりません。`ETag` と `If-None-Match` の組でも、同じ条件付きリクエストを作れます。

リダイレクト(転送の仕組み・3xx・Location)

リダイレクトは、ある URL へのアクセスを別の URL へ転送する仕組みです。サーバーが 3xx のステータスコードと、転送先を示す `Location` ヘッダフィールドを返し、ブラウザがその URL に自動でアクセスし直します。サイトの URL を変えた時に、古い URL への訪問者を新しい URL へ案内するのに使います。

```
HTTP/1.1 301 Moved Permanently
```

```
Location: /new.html
```

```
Content-Length: 0
```

ブラウザと同じように `curl -L` で転送を追っても、`/new.html` から 200 が返るまで手動の操作は要りません。

3xx の使い分けは、恒久か一時か、メソッドを変えるかで決まります。

コード	意味	使う場面
301 Moved Permanently	恒久的な移動	URL を変更した時。検索エンジンも新しい URL を登録し直す
302 Found	一時的な移動	一時的に別の URL へ誘導する時
303 See Other	別の URL を GET で取得させる	POST の後に結果ページへ移す時
307 Temporary Redirect	一時的な移動 (メソッドを変えない)	POST を POST のまま転送したい時
308 Permanent Redirect	恒久的な移動 (メソッドを変えない)	同上で恒久の場合

転送先を伝えるのは Location です。似た名前の `Content-Location` は、返したボディの内容に対応する URL を示すもので、転送先の指定には使えません。
`Referer` は直前のページ、`Host` は接続先のホスト名を伝える、どちらもリクエスト側のフィールドです。HTML の中で転送する meta の `http-equiv="refresh"` もありますが (第2章)、サーバー側のリダイレクトが基本です。

Cookie (Set-Cookie・属性・容量)

Cookie は、サーバーがブラウザに預ける小さな「名前=値」のデータです。HTTP がステートレスなので、ログイン状態のような情報を次のアクセスへつ

なぐのに使います[^3]。仕組みは、サーバーが `Set-Cookie` ヘッダフィールドで預け、ブラウザが条件に合う次のリクエストの `Cookie` ヘッダフィールドで送り返す往復です。`Set-Cookie` はレスポンス側、`Cookie` はリクエスト側のフィールドで、取り違えられません。

```
HTTP/1.1 200 OK

Set-Cookie: sid=abc123; Max-Age=3600; Path=/; HttpOnly

Content-Type: text/plain; charset=UTF-8
```

このあと同じサーバーへ送ったリクエストには、次のように付きます。

```
GET /new.html HTTP/1.1

Host: 127.0.0.1:45678

Cookie: sid=abc123
```

`Cookie` の名前と値は、サーバーではなくブラウザ(クライアント)に保存されます。サーバー側のデータの入れ物は別にあり、`Cookie` に入るのは、その入れ物を指すセッション ID です。HTTPS で通信しても、保存された `Cookie` の内容はブラウザの中に平文で残り、暗号化されません。パスワードそのものを `Cookie` に入れる設計は避けます。

`Set-Cookie` は「名前=値」の後ろに、セミコロン区切りで属性を並べます。

属性	働き
Expires	有効期限を日時で指定する
Max-Age	有効期間を秒で指定する

属性	働き
Domain	Cookie を送る先を、指定したドメインとそのサブドメインに限る
Path	指定したパスとその配下へのリクエストにだけ送る
Secure	HTTPS の通信でだけ送る
HttpOnly	JavaScript の <code>document.cookie</code> から読めなくする
SameSite	別のサイトから発生したリクエストに付けるかを決める

Expires も Max-Age も無い Cookie は、セッション Cookie と呼ばれ、ブラウザを終了すると残りません。どちらかがあれば、終了後も期限まで残り、両方あれば Max-Age が優先されます。Max-Age=3600 は 3600 秒、つまり 1 時間で失効し、Max-Age=86400 なら 24 時間です。Path=/app の Cookie は、/app とその配下 (/app/menu など) には付きますが、/login には付きません。Domain に書けるのは Cookie を送るドメインで、ワイルドカードの * は使えません。

Secure は、通信路の盗聴への対策です。付けても、ブラウザ内の保存領域で値が暗号化されるわけではありません。HttpOnly は、XSS (第3章) で埋め込まれたスクリプトにセッション ID を読み取られる被害を減らします。

SameSite は Strict・Lax・None の 3 値を取ります。Strict は、別サイトから発生するリクエストに一切付けません。Lax は、別サイトのリンクから移動する GET のような安全なナビゲーションには付け、別サイト発の POST や画像の読み込みには付けません。None は別サイト発のリクエストにも付けますが、Secure が必須です。CSRF の対策の一部として使います。

Set-Cookie の例を 1 つ読みます。

```
Set-Cookie: token=xyz; Max-Age=86400; Secure; HttpOnly; SameSite=Lax;  
Path=/app
```

この Cookie は、https:// で始まる URL へのリクエストにだけ付き、24 時間後に失効します。`/app` 配下にだけ付き、JavaScript からは読めません。別サイトのフォームから送られる POST には付かず、ブラウザを終了しても期限まで残ります。

容量には目安があります。RFC 6265 は、ブラウザが 1 つの Cookie につき少なくとも 4096 バイト、1 つのドメインにつき少なくとも 50 個、全体で少なくとも 3000 個を保持できることを求めています。10 個までといった上限は定められていません。Cookie は該当するすべてのリクエストに付いて送られるので、大きなデータを入れると通信量が増えます。大きなデータは Web Storage (第14章) などに置きます。

認証方式 (Basic認証・Digest認証)

HTTP 自体にも、パスワードを使う認証の仕組みがあります^[^4]。流れは共通で、認証が必要なリソースへ最初にアクセスすると、サーバーが `401 Unauthorized` と `WWW-Authenticate` ヘッダフィールドを返します。

```
HTTP/1.1 401 Unauthorized  
  
WWW-Authenticate: Basic realm="Admin Area"  
  
Content-Type: text/plain; charset=UTF-8
```

ブラウザはユーザー名とパスワードの入力ダイアログを出し、入力を受けて資格情報を付けたリクエストを送り直します。`realm` は、保護された領域の名前

を示す文字列で、ダイアログに表示されることがあります。ハッシュ化に使う乱数ではありません。

Basic 認証は、「ユーザー名:パスワード」を Base64 でエンコードして、`Authorization: Basic` の後ろに付けて送ります。

```
GET /admin/ HTTP/1.1
Host: 127.0.0.1:45678
Authorization: Basic dGFybzpzZWNyZXQ=
```

`dGFybzpzZWNyZXQ=` は `taro:secret` を Base64 にした文字列です。Base64 は暗号ではなく、誰でも元に戻せるエンコードなので、盗聴されればパスワードは取り出せます。パスワードをハッシュ化して送る方式ではありません。HTTP のまま使わず、HTTPS と組み合わせます。

Digest 認証は、この弱点を補う方式です。401 で返す `WWW-Authenticate: Digest` に、サーバーが生成した使い捨ての値 `nonce` が付き、クライアントは、ユーザー名・パスワード・nonce などから計算したハッシュ値を送ります。パスワードそのものは通信路に流れず、使い捨ての値のおかげで、同じ値を再利用する攻撃も効きません。ただし、通信内容は暗号化されないのので、現在は HTTPS の上で認証するのが普通です。ハッシュ値を送るのは Digest 認証で、Basic 認証が送るのは Base64 の文字列です。Cookie や URL のクエリでパスワードを渡す方法は、HTTP の認証方式にありません。

SSL/TLSとHTTPS

HTTPS は、HTTP を SSL/TLS の上で行うプロトコルです。HTTP の通信路を暗号化して盗聴と改ざんを防ぎ、サーバーが本物であることを証明書で確かめます。

SSL は TLS の前身の名前で、現在使われているのは TLS です。慣習として SSL/TLS と並べて呼ぶだけで、SSL 自体はもう使われません。HTTPS の既定のポートは 443 番で、URL は `https://` で始まります。接続の最初にサーバーが認証局の発行した証明書を示し、ブラウザがそれを検証してから、暗号化した通信に入ります。

会員情報を入力するページのセキュリティを高めたい時は、サイト全体を HTTPS にします。HTTPS が行うのは、通信の暗号化・改ざんの検知・サーバーの認証の 3 つです。利用者のユーザー名とパスワードによる認証は別の仕組みで、gzip などの圧縮は `Content-Encoding` の役割なので、HTTPS には含まれません。UDP の上で動くのは HTTP/3 の特徴で、HTTPS の説明には入りません。

HTTPバージョンの違い(HTTP/1.0・1.1・2)

HTTP は版が上がるたびに、接続の使い方を改善してきました。

版	接続の既定	主な特徴
HTTP/1.0	リクエストごとに切断する	維持したい時は <code>Connection: keep-alive</code> を付けて要求する
HTTP/1.1	既定で接続を維持する(持続的接続)	<code>Host</code> ヘッダフィールドが必須。閉じたい時は <code>Connection: close</code>
HTTP/2	1 つの TCP 接続を使い続ける	バイナリのフレーム、多重化、ヘッダ圧縮

HTTP/1.1 で Host が必須になったので、1 つの IP アドレスで複数のドメインのサイトを動かせます。サーバーがリクエストの Host を見て返すサイトを選ぶ

ので、Host の無いリクエストは 400 のエラーになります。

HTTP/1.1 では、1 つの接続で同時に返せるレスポンスは 1 つです。パイプライン(複数のリクエストをまとめて送る方法)を使っても、レスポンスはリクエストを受けた順に返す決まりで、先頭の遅い応答が後ろを待たせます。そこでブラウザは、複数の接続を張って並列化していました。

HTTP/2 は、この制約を解消するために作られました。特徴は次の 3 つです。

- ストリームとフレーム: 1 つの TCP 接続の上に複数のストリームを作り、メッセージをバイナリのフレームに分けて多重化する
- HPACK: ヘッダフィールドを圧縮する専用の方式。以前に送ったヘッダを表で参照するなどして、通信量を減らす
- サーバープッシュ: クライアントが要求する前に、サーバーがリソースを先回りして送る機能

HTTP/2 のメッセージはバイナリのフレーム形式で、HTTP/1.1 のようなテキストの行単位の解析はしません。ヘッダ圧縮の方式名は HPACK です。HTTP/3 で使うのは別の QPACK で、HTTP/3 は UDP ベースの QUIC の上で動きます。HTTP/2 は TCP の上で動くので、QUIC は使いません。サーバープッシュは仕様にある機能ですが、ブラウザ側では 2022 年に Chrome が対応を外すなど、実際にはほとんど使われていません。

Webブラウザ—サーバー間の通信手順(接続・持続的接続)

アドレス欄に `https://www.example.com/` と打ってからページが出るまでには、順番の決まった手順があります。

1. DNS で、ホスト名から IP アドレスを調べる(名前解決)

2. その IP アドレスへ TCP 接続を作る (SYN・SYN/ACK・ACK の 3 ウェイハンドシェイク)
3. HTTPS なら、TLS のハンドシェイクで暗号化の準備をする
4. HTTP リクエストを送る
5. HTTP レスポンスを受け取る

接続先が分からないうちは TCP 接続を作れず、TCP 接続ができる前に TLS や HTTP のやり取りは始められません。HTTP リクエストが最初に出ていくことはありません。レスポンスの HTML を読むと、画像や CSS が必要になり、ブラウザは追加のリクエストを送ります。

追加のリクエストのたびに手順 2 や 3 を繰り返すのは無駄が多いので、1 つの TCP 接続を複数のリクエストに使い回します。これが持続的接続で、接続を作り直す回数が減るため、画像を多く含むページの表示が速くなります。HTTP/1.1 は既定で持続的接続で、冒頭のレスポンスにも `Connection: keep-alive` が付いています。HTTP/1.0 の既定はリクエストごとの切断なので、維持したい時はクライアントが `Connection: keep-alive` を指定しなければなりません。指定する値は `keep-alive` で、`persistent` という値はありません。接続を閉じたい時は `Connection: close` を付け、そのレスポンスの後に接続が閉じます。

持続的接続で省けるのは、接続を作る手間です。ホスト名の名前解決は、接続先の IP アドレスを知るための手順なので、持続的接続とは別の話で、なりません。

サンプルはここまでです

続き(第2章～巻末)は、HTML5プロフェッショナル認定試験 レベル1 問題集のご購入で、頻出度別ノート・暗記用ノートと合わせて3点すべてダウンロードできます。

SAMPLE

HTML5プロフェッショナル認定試験 レベル1 学習用テキスト(無料サンプル)

版

2026年10月版(LPI-Japan HTML5プロフェッショナル認定試験 レベル1 出題範囲(Ver2.5)準拠)

発行

IT資格道場(<https://www.shikaku-doujou.com>)

お問い合わせ

info@shikaku-doujou.com

HTML5プロフェッショナル認定試験は LPI-Japan の試験です。HTML・CSS・Web API などの名称や仕様の権利は、W3C・WHATWG などそれぞれの権利者に帰属します。記載されているその他の会社名・製品名・サービス名は、各社の商標または登録商標です。

本教材は公式の出題範囲(Ver2.5)と公開されている仕様を基に作成したもので、LPI-Japan との提携・承認関係はありません。

本文・図表・設問はすべてオリジナルの著作物です。無断での転載・複製・再配布・二次販売はできません。お気づきの点は info@shikaku-doujou.com までご連絡ください。