Engineer

# プロトコル

プログラム同士が通信するために合意した規則

## どのようなものか

プロトコルは、プログラム同士のやり取りについて書き記した取り決めです。各メッセージの形と意味、どちらが先に話すか、何の後に何を送れるか、相手が予想外のことを言ったときにそれぞれがどう動くかを定めます。別々の人が異なる言語で書いた2つのプログラムでも連携できるのは、相手の実装に合わせたのではなく、同じ文書に従って作られているからです。

NISTの用語集は、各刊行物からいくつかの定義を集めています。共通する核は、共同の目的を達成するために2者以上が従う、メッセージの形式と関連手順についての規則です。その1つは、見落としやすい側面も示しています。規則が定めるのはメッセージの構造だけでなく、順序でもあるという点です。形式が正しくても送るタイミングを間違えれば、形式が壊れている場合と同じようにやり取りが成立しません。

RFCとして公開される仕様は、大文字のキーワードで規則の強さを示します。MUSTは絶対的な要件です。SHOULDから外れてよいのは、その影響をすべて理解し、十分に検討した場合だけです。MAYは、完全に任意であることを示します。2017年以降、これらの意味を持つのは大文字の場合だけなので、同じ文書内の小文字の「must」は普通の英語です。SHOULDとMUSTの違いは、通常とは異なるプログラムと、誤ったプログラムの違いにあたります。

プロトコルは層を重ねて使います。ページ、メニュー、メールなどを扱うアプリケーションプロトコルは、通常、2台のマシン間でバイトを運ぶことだけを仕事とするトランスポートプロトコルの上に載ります。多くが使うTCPは、仕様が「信頼性があり、順序を保つバイトストリームのサービス」と呼ぶものを提供します。バイトは欠けずに、送った順序で届きます。一方、送信側が1回に書き込むまとまりと受信側が1回に読み取るまとまりの間に、対応関係は保証しません。ストリーム自体にはメッセージの区切りがないため、その上の各プロトコルが、改行、先に送る長さ、接続の終了、その他の合意した印などで終端を示す必

要があります。UDPが選ぶのは逆の方針です。最小限の仕組みで個々のデータグラムを運びますが、到達も重複の防止も保証しません。どちらのトランスポートも、1台のマシン上のサービスをポート番号で区別します。その番号はTCPとUDPがそれぞれ別に持っています。

プロトコルのそのほかの特徴は、意図するかどうかにかかわらず、設計の際に行われるいくつかの選択で決まります。

- **どちらが先に話すか。** 接続したクライアントに挨拶するサーバーも、リクエストを黙って待つサーバーもあります。両方が待てば、タイマーが切れるまで何も起きません。
- **サーバーが何を覚えるか。** ステートレスなプロトコルでは、各リクエストに応答に必要なものがすべて含まれます。ステートフルなプロトコルでは、メッセージの意味が同じ接続でそれ以前に何が起きたかに依存します。
- **どう拡張するか。** バージョン番号、オプションコード、未知なら無視しなければならないフィールドは、プロトコルが将来のために残しておく余地です。

概念を判断に生かす

「プロトコルとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 Wi-Fi、ファイアウォール、リモートアクセスを整え、どこからでも働け、権限のある人だけが接続できるようにします。

**サービスを見る** お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 Ubuntu Linux VPSインスタンス30台を管理 — 30 復旧計画とともに管理したLinuxサーバー

**Ubuntu Linux VPSインスタンス30台を管理** お問い合わせ →

## なぜ重要なのか

プロトコルがあることで、通信のどちら側も交換できます。文書に従って書かれたクライアントは、同じ文書に従うどのサーバーとも通信できます。そのため、利用者がソフトウェアを替えたり、サーバーを別の言語で書き直したりしても、相手側は気づきません。一緒に作ったからこそ一致しているだけのプログラムは、別々には変更できません。

プロトコルは、コードが存在する前からセキュリティ方針の多くも決めます。どちらが接続を開くかによって、何を外から受け入れる必要があるかが決まります。クライアントを待つサーバーは外部に開いたポートを必要としますが、外向きの接続しかしないプログラムには不要です。メッセージが何を運ぶかによって、何を記録し、数え、漏らし得るかも変わります。パスを指定するリクエスト

はパス別に記録でき、読者のネットワークアドレスを隠しても、サーバーによるリクエストの計数は妨げられません。暗号化せずに動くプロトコルは、通過するすべてのネットワークにすべてのリクエストを見せます。

最後に、プロトコルはそれを使うプログラムより長生きします。実装は数年ごとに入れ替わっても、やり取りは何十年も続きます。稼働中のプログラムが依存する動作は、文書に何と書いてあっても、そのやり取りの一部になります。Internet Architecture Boardは、この問題の両面について論じています。許容された誤りが標準になることと、使われないまま放置された拡張の余地が、いざ使おうとしたときには機能しなくなっていることです。

## 実際の活用例

このサイトは、1台のサーバーの1つのフォルダー内のファイルから、4つのプロトコルに応答します。HTTP、Gemini、Gopher、そしてTor上のonionサービスです。案内ページはGopherを、暗号化もスクリプトもない平文TCPを使う1991年のメニュープロトコルと説明し、手で入力するリクエスト全体を示しています。

```
printf '/contact\r\n' | nc engineer.company 70
```

そのページは、この1行がプロトコルのすべてだと説明しています。仕組みとしては、1回のデプロイで1つのフォルダーを書き出し、4つの入口がそれを読みます。別々に保守するコピーではないため、どれかだけがずれることもありません。

ファイアウォールルールは、有無ではなく位置で検証しています。パケットフィルターは順序付きのリストを調べ、最初に一致したルールで止まるからです。検査規則の1つはプロトコルに関するものです。各ポートのルールは正確に1つで、必ずプロトコルを指定します。指定のないルールは、気づかないうちに公開範囲を広げるからです。宣言した公開範囲は5つのポートの短いリストで、全体として照合します。したがって、宣言していないポートが現れれば検査は失敗します。

4つ目の入口はonionミラーで、そのプロトコルがファイアウォールと集計方法の両方を形作っています。匿名化デーモンは外向きの回線を作ってネットワークに接続するため、外部からの接続はなく、受信用ポートを開く必要もなく、2層のファイアウォールのどちらにも同期するものがありません。Geminiのリクエス

トとGopherの接続は、アドレスもパスも含めず日別に数えます。onionにはカウンターも、それを追加するフラグもありません。このサイトが、その読者を意図的に数えない方針を取っているからです。

## 問題が起きるところ

**許容した誤りが標準になる。** 「送るものには慎重に、受け入れるものには寛容に」という堅牢性の原則は、プログラム同士の連携を助けるためのものでした。しかし限度なく適用すると、欠陥のある実装が公開され、広まってしまいます。次の実装もそれと連携するために同じ欠陥を受け入れざるを得ず、その欠陥が事実上の標準になります。新しい実装は、バグまで同じように再現しなければ互換性を持てません。RFC 9413は、稼働中の実装が仕様を上書きしかねない適合性テストになると明言しています。仕様が禁じるものは、プロトコルの歴史のできるだけ早い段階で拒否し、直せる人にエラーを見える形で知らせます。

**2つのプログラムがメッセージの終端について一致しない。** バイトストリームの境界は、プロトコルが定めた位置にあります。終端の示し方を2通り許すプロトコルや、曖昧な書き方を受け入れるパーサーでは、プロキシとその背後のサーバーが同じバイトを別々のメッセージとして解釈できます。一方がリクエストの末尾とみなす部分を、他方はプロキシが一度も検査していない新しいリクエストとみなします。区切り方の規則は1つにし、曖昧なものを拒否し、経由するすべての箇所で同じように解析させます。

**ルールにポートはあるが、プロトコルがない。** ポート番号は、トランスポートを指定して初めて意味を持ちます。ツールによっては、番号だけのファイアウォールルールがTCPとUDPの両方でそのポートを開きます。片方だけを必要としたサービスは使えますが、もう片方も、後でそこで待ち受ける何かに対して開いたままです。すべてのルールにプロトコルを記し、公開範囲全体を宣言したリストと比較します。

**拡張のための余地が使えなくなる。** 稼働中の送信側が拡張ポイントを一度も使わないと、見たことのないものを拒否する受信側が作られ、最初の本当の拡張がそこで失敗します。その受信側が長く稼働するほど、修正は難しくなります。意味を持たない値も含め、最初から拡張ポイントを使って、許容性を想定するだけでなく、常に試します。

## 参考資料


- NIST CSRC用語集：プロトコル — NISTのセキュリティ刊行物で使われる定義を、それぞれの出典文書とともに示します。

- RFC 9293：伝送制御プロトコル（TCP） — メッセージ境界を持たないバイトストリームを含む、現行のTCP標準です。
- RFC 768：ユーザーデータグラムプロトコル（UDP） — 1980年に書かれた、3ページでUDP全体を説明する文書です。
- RFC 1436：インターネットGopherプロトコル — セッション例を含む、アプリケーションプロトコル全体を説明する参考情報（Informational）文書です。
- RFC 9110：HTTPセマンティクス — 通信上のバージョンに依存しない、リクエスト、レスポンス、メソッド、ステータスコードの意味を説明します。
- RFC 9413：堅牢なプロトコルの維持 — エラーを許容するとそれが定着する理由についてのInternet Architecture Boardの論考です。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

## 用語を実務に結び付ける

「プロトコルとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く 連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/protocol/