# 2-1 Webアプリケーションについて

コンピューターやスマートフォンで、Webブラウザ（ブラウザ）[注1]にURLを入力したり、画面のボタンやリンクを押したりすると、画面の表示が変わります。また、他の人のスマートフォンのブラウザでも同じURLを入力すると、同じ画面が表示されます。それはなぜでしょうか？

本節では、URLを入力したとき、画面のボタンやリンクを押したときに何が起こっているのかを説明しながら、Webアプリケーションについて学んでいきます。

## Webアプリケーションが表示される仕組み

Webアプリケーションは、「Amazon」（https://www.amazon.co.jp/）や「Instagram」（https://www.instagram.com/）などのように、ユーザーが「ログインする」「検索する」「投稿する」などの操作をすると、その操作に応じた結果がブラウザに表示されるものです[注2]。

Webアプリケーションの例として、「技術評論社 書籍案内のページ」（https://gihyo.jp/book/）を挙げます（以降、書籍案内のページといいます）。


注1　ブラウザはWebページを見るためのアプリケーションです。代表的なものに、Google Chrome・Safari・Microsoft Edgeがあります。

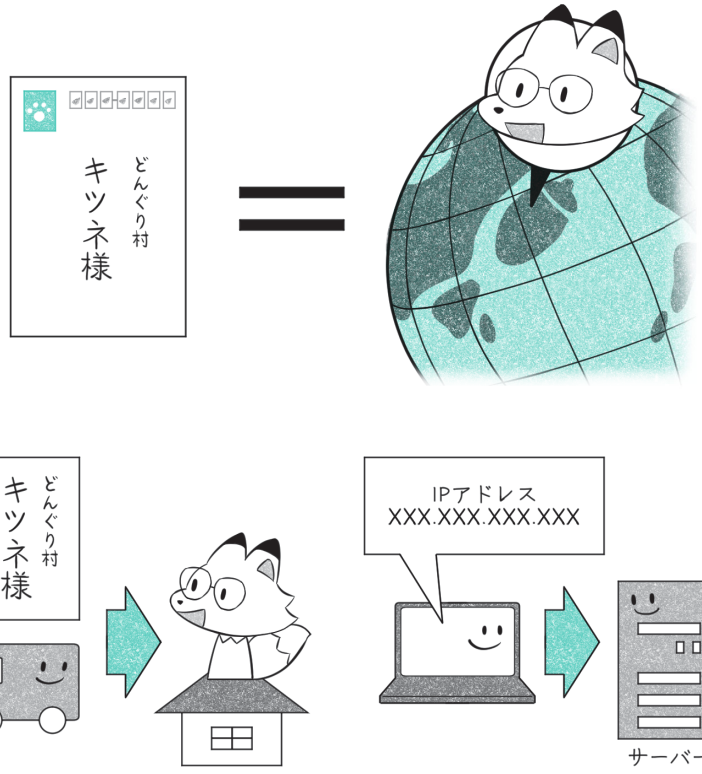
注2　Webアプリケーションによく似た言葉で、Webページというのがあります。Webページは、「Rubyリファレンスマニュアル Ruby 3.1版」（https://docs.ruby-lang.org/ja/3.1/doc/index.html）のように、ユーザーが操作する部分もなく、誰が見ても同じ情報を表示しているものになります。

COLUMN

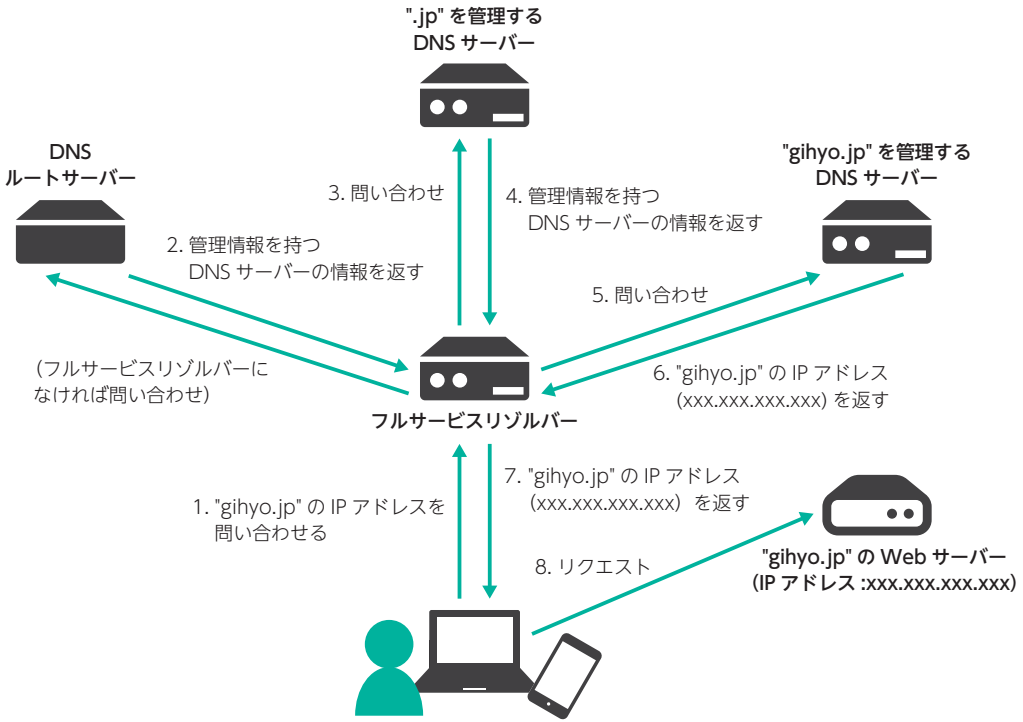
## 世界中のWebサーバーの中から1つのWebサーバーをどうやって見つけるの？

ブラウザにURLを入力するだけで、インターネットにつながっている世界中のWebサーバーの中から、リクエストを送るWebサーバーを見つけ出して、リクエストを送ります。その仕組みはどうなっているのでしょうか？

インターネットにつながっている世界中のサーバーはIPアドレスという数字の羅列で管理されています。わかりやすく例えると、手紙を送るときに、手紙を送りたい相手の都道府県、市町村の住所を書いて送ります。その住所は緯度経度でも表すことができます。サーバーも同じように人間にわかりやすい住所としてのURLの一部であるドメイン名[注a]と、管理するためのIPアドレスを持っているのです。



ではドメイン名からどのようにしてIPアドレスを導き出すのでしょうか？ドメイン名からIPアドレスへ変換するには、DNS（Domain Name System）というものを使います。




1. コンピューターやスマートフォンから、フルサービスリゾルバー（DNSキャッシュサーバーともいいます）というサーバーに"gihyo.jp"のIPアドレスを問い合わせします。フルサービスリゾルバーに、"gihyo.jp"のIPアドレスが保存されていれば、その結果を返します。保存されていない場合は、DNSルートサーバーに問い合わせします。
2. DNSルートサーバーは、".jp"を管理するDNSサーバーの情報を返します。
3. フルサービスリゾルバーは、返ってきた".jp"を管理するDNSサーバーに、"gihyo.jp"のIPアドレスを問い合わせします。
4. すると、"gihyo.jp"を管理するDNSサーバーの情報が返ってきます。
5. 返ってきた"gihyo.jp"を管理するDNSサーバーに、問い合わせします。
6. "gihyo.jp"のIPアドレスが、フルサービスリゾルバーに返ってきます。
7. フルサービスリゾルバーから、"gihyo.jp"のIPアドレスが、コンピューターやスマートフォンへ返されます。
8. 返ってきたIPアドレスをもとに、"gihyo.jp"のWebサーバーへリクエストを送ります。

# 3-1 Webアプリケーションづくりの第一歩

まずはじめに、Webアプリケーションとして最低限必要な機能、「URLをブラウザに入力すると、何かしらの画面が表示される」ものをRailsの機能を使って作っていきましょう。

## Railsの機能を使ってみよう

最初にこれから作っていくWebアプリケーションの名前を決めましょう。今回は、写真や自分で描いたイラストなどの画像（picture）で投稿する日記（diary）を作るので、「pdiary」という名前で作っていくことにします。

次に、Webアプリケーションを作るときの作業場所として、新しいディレクトリを作成します。ターミナル（Windowsの場合はコマンドプロンプト）を起動して、次のコマンドを実行してみましょう（コマンドの後ろにある#以降の文はコメントです。ターミナルに入力の必要はありません）。

▼ Windows の場合

```
cd %HOMEPATH% # ホームディレクトリに移動
mkdir myWebApp # myWebAppディレクトリを作成
cd myWebApp # myWebAppディレクトリに移動
```

▼ macOS の場合

```
cd ~ # ホームディレクトリに移動
mkdir myWebApp # myWebAppディレレクトリを作成
cd myWebApp # myWebAppディレクトリに移動
```

コマンドは1行書き終わるごとにEnterキーを押してから、次の行のコマンドを書きます。1行目のcdコマンドは、ディレクトリを移動するコマンドです。このコマンドでホームディレクトリ（Windowsの場合はC:¥Users¥アカウント名、macOSの場合は/Users/ユーザー名）に移動します。ホームディレクトリの指定には、Windowsの場合は%HOMEPATH%、macOSの場合は~（チルダ）を代わりに使うことができるので、コマンドではこちらを利用しています。2行目のmkdirコマンドは、ディレクトリを作成するコマンドです。このコマンドで、「myWebApp」というディレクトリを作成しています。3行目のcdコマンドで、先ほど作った「myWebApp」の中に移動します。

これから先、次のディレクトリのことを「作業ディレクトリ」と呼ぶようにします。

| OS | 作業ディレクトリ |
|---|---|
| Windows | C:¥Users¥アカウント名¥myWebApp |
| macOS | /Users/ユーザー名/myWebApp |

「pdiary」は、作業ディレクトリの中に作成します。

それでは、いよいよアプリケーションを作成します。今開いているターミナルで、次のコマンドを実行してみましょう（もしターミナルを閉じてしまっていたら、再度ターミナルを起動して、cdコマンドで作業ディレクトリに移動してから、コマンドを実行します）。

```
rails _7.0.4_ new pdiary
```

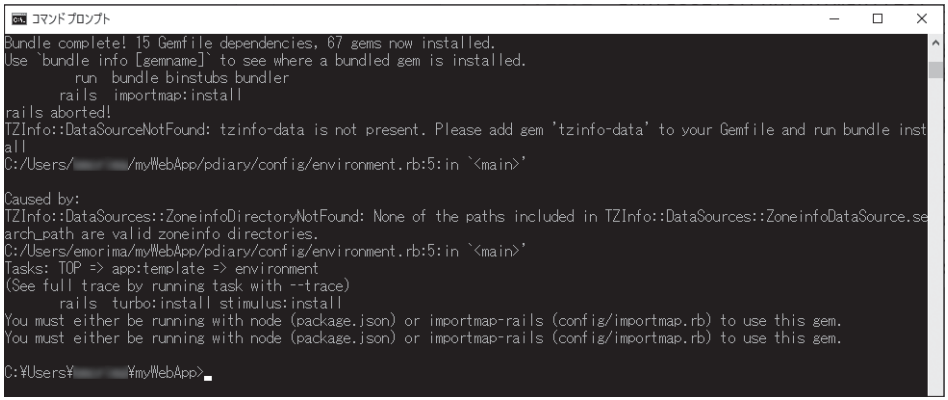
コマンドを実行すると、ターミナルに英語がたくさん表示されます。コマンドの処理は、Enterキーを押してプロンプト[注1]が表示されるようになったら終了しています。環境によってはターミナルに情報が表示されるまでにしばらく時間がかかる場合もありますので、少し気長に待ってみてください。


注1　プロンプトは、コマンドの入力を受け付けている状態を表す記号のことで、行末に>（Windowsの場合）、または$（macOSの場合）が表示されます。

COLUMN

## rails newコマンドでエラーになった

rails _7.0.4_ new pdiaryコマンドを実行したときに、Windowsの場合、次のようなエラーが表示される場合があります。




その場合は、次の作業を行ってください。

1. C:¥Users¥アカウント名¥myWebApp¥pdiary¥Gemfileをエディターで開きます（VS Codeでファイルを開きたい場合は、3-1にある「できたファイルを確認しよう」を参考にしてください）。
2. 40行目付近を次のように変更して保存します。

▼ 変更前

```
gem "tzinfo-data", platforms: %i[ mingw mswin x64_mingw jruby ]
```

この部分を削除します。

▼ 変更後

```
gem "tzinfo-data"
```

3. 変更したら、rails _7.0.4_ new pdiaryを実行したターミナルで、次のコマンドを実行します。これで先ほどのエラーが解消されます。

```
cd pdiary # 作業ディレクトリから、Railsルートディレクトリに移動
bundle install # Gemfileに記載したgemをインストール
```

# 6-1 プログラムを管理する仕組み

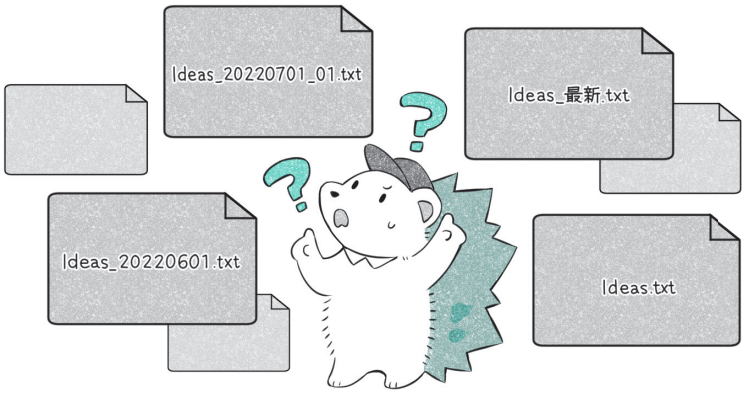
ここからは、pdiaryを実際に管理するために、まずはプログラムを管理するときの仕組みや流れを学習していきましょう。

## バージョン管理システム

ファイルを更新するときに「更新前の状態を残しておきたい」場合、次のようにファイル名に日付や番号などをつけて管理したことはあるでしょうか。

- ideas_20220601.txt
- ideas_20220701.txt
- ideas_20220701_01.txt
- ideas_20210701_02.txt
- ideas_最新.txt
- ideas.txt

この方法は手軽ですが、少しでも名前付けのルールが曖昧になると、どのファイルが最新の状態のファイルなのかわからなくなる問題があります。先ほど挙げたファイル名でも、ファイル名に日付がついているもの、日付と番号がついているもの、「最新」とついているもの、何もついていないものといろいろなものがあり、どのファイルが最新なのかがわかりにくくなっています。また、ある時点でのファイルを保存しているだけなので、どのような変更をしたかは、ファイルを見比べてみないとわかりません。間違えて本当はそのままにしておく必要があったファイルを上書きしてしまう可能性もあります。






こんなときに、ファイル名に日付や番号をつけて管理する代わりに、「いつ」「誰が」「どのファイルに」「どのような変更をしたか」を管理してくれるものが使えると便利です。「いつ」「誰が」「どのファイルに」「どのような変更をしたか」といった変更履歴を記録して管理するシステムのことを**バージョン管理システム**といいます。バージョン管理システムを利用すると、次のようなことが簡単にできます。

- どのような変更をファイルにしたのかを確認する
- 更新したファイルの変更を取り消す
- 間違えて削除してしまったファイルを元に戻す

バージョン管理システムは、ひとりでアプリケーションを作るときだけではなく、システム開発の現場など、1つのアプリケーションを複数人で開発するときにもよく利用されています。

また、バージョン管理システムには、いろいろな種類があります[注1]。本書では、RubyやRails以外にも多くのプロジェクトでプログラムを管理するときに多く利用されている、**Git**（https://git-scm.com/）について学習していきましょう。

注1　今回学習するGit以外にも、Mercurial・Subversion・CVSなどといったものがあります。

# 7-3 おみくじプログラムを拡張しよう

プログラムは少しずつ機能を追加したり、使いやすいように変更したりしながら、拡張していきます。おみくじプログラムも機能を追加して拡張していきましょう。

## 仕様を考えよう

**仕様1.** おみくじプログラムは、`ruby omikuji_ext.rb` で実行する

**仕様2.** 実行すると、大吉・中吉・吉・凶の結果と、結果別に次の説明をターミナルに表示する

| 結果 | 説明 |
|---|---|
| 大吉 | すいすいプログラムが書ける日。楽しく学んでいきましょう。 |
| 中吉 | 便利なメソッドが見つかるかも。Ruby公式ドキュメントを見てみよう。 |
| 吉 | プログラムに書く処理が難しい時には、処理を分解してみましょう。 |
| 凶 | エラーが表示された時は、エラーメッセージをよく見ると早く解決できるかも。 |

**仕様3.** 結果と説明を表示後、「もう一度おみくじを引きますか？」と表示し、q[注3]を入力したら、プログラムを終了する。q以外が入力されたら、もう1回おみくじを引く（仕様2以降を行う）

## 仕様から処理を考えよう

では、仕様を1つずつ確認してみましょう。

仕様1は、7-2にある「仕様から処理を考えよう」でファイルに書いたプログラムを実行する方法を学びましたね。今回は「omikuji_ext.rb」で実行するため、プログラムは「omikuji_ext.rb」というファイルに書いていきます。

仕様2はどうでしょうか？複数の処理になっているので、分解してみましょう。

注3 qは、quit（終了する）の先頭文字を意味します。同じ終了する意味でexitもありますが、edit（編集する）も先頭文字がeのため、1文字で終了を意味する場合、qを使うことが多いです。