

🏠

🕒

📖

📄

🔍

✎

🔊

教材

🔍 検索

所属チーム ▼ 🔔<sup>1</sup> 👤

本文 目次 質問一覧 23件

ホーム 教材 JavaScriptの基礎を学ぼう 繰り返し処理を理解しよう

8章 繰り返し処理を理解しよう

同じ処理を続けて実行するには繰り返し処理を活用しましょう。

🕒 120分 🏆 - 📖 読了

8.1 本章の目標

本章では以下を目標にして学習します。

▪ 繰り返し処理とは何か、概要をつかむこと

▪ while文の書き方を知り、実際にコードを書いてみること

▪ for文の書き方を知り、実際にコードを書いてみること

▪ 繰り返し処理が止まらず永遠に実行され続ける無限ループの危険性を知ること

例えば1〜10までの数値を順番に出力したいとき、以下のように同じようなコードを何度も書くのは手間がかかりますし、コードが無駄に長くなってしまいます。

```
1 console.log(1);
2 console.log(2);
3 console.log(3);
4 console.log(4);
5 console.log(5);
6 console.log(6);
7 console.log(7);
8 console.log(8);
9 console.log(9);
10 console.log(10);
11
```

そこで使うのが**繰り返し処理**です。繰り返し処理を使えば、以下のようにコードをスッキリさせられます（コードの意味は本章で詳しく解説します）。

```
1 for (let i = 1; i <= 10; i += 1) {
2   console.log(i);
3 }
4
```

JavaScriptをはじめとするプログラミング言語では、繰り返し処理を条件分岐と同じくらいよく使います。

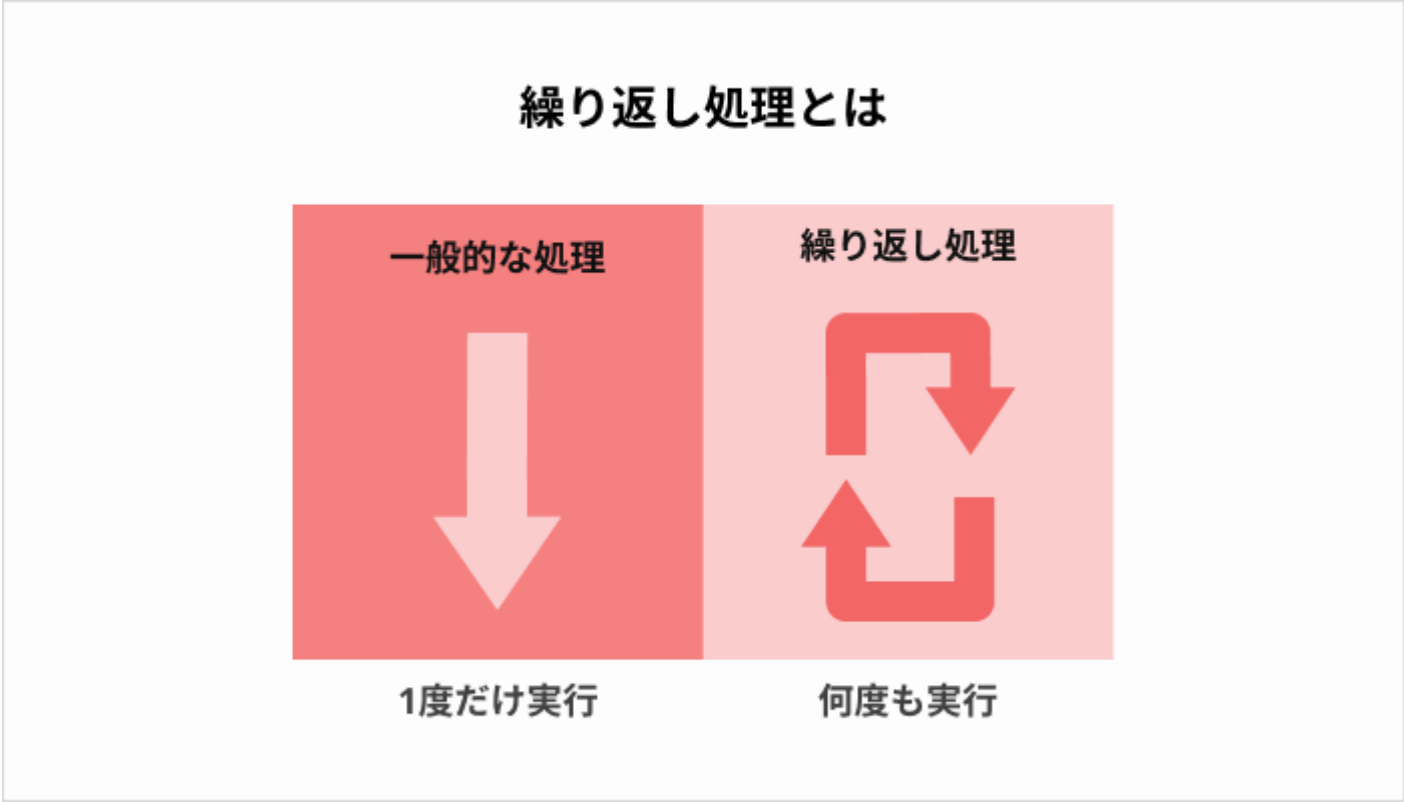
本章では繰り返し処理について学び、より短く読みやすいコードを書けるようになります。

+ 質問する



## 8.2 繰り返し処理とは

繰り返し処理とは、「決まった回数」または「条件を満たしている間」**同じ処理を繰り返し行う**ことです。



条件分岐ではif文とswitch文を使いましたが、繰り返し処理で使うのは**while文**と**for文**です。

なお、while文とfor文の使い分け方は以下のとおりです。

構文	使うケース	例
while文	繰り返す回数があらかじめ <b>わからない</b> 場合	サイコロで6の目が出るまで繰り返す
for文	繰り返す回数があらかじめ <b>わかっている</b> 場合	1～10までの数字を順番に表示する

では順番に学んでいきましょう。

## 8.3 while文の書き方

while文を使えば、「**条件を満たしている間**」同じ処理を繰り返し行うことができます。

while文の書き方は以下のとおりです。while文では、if文と同じように条件式を使います。

```
1 while (条件式) {
2   条件を満たしている間、繰り返す処理
3 }
4
```

### 条件式に使う比較演算子について復習しよう

ここで、条件式に使う比較演算子について復習しておきましょう。ざっと確認するだけで十分です。



比較演算子	処理の内容
==	2つの値が等しい場合は true を返す（等価演算子）。
===	2つの値とデータ型が等しい場合は true を返す（厳密等価演算子）。
!=	2つの値が等しくない場合は true を返す。
!==	2つの値とデータ型が等しくない場合は true を返す。
>	左辺の値が右辺の値よりも大きい場合は true を返す。
>=	左辺の値が右辺の値以上の場合は true を返す。
<	左辺の値が右辺の値よりも小さい場合は true を返す。
<=	左辺の値が右辺の値以下の場合は true を返す。

## while文の使用例

では、while文の使用例を見てみましょう。以下の `num !== 0` の部分が条件式で、変数 `num` の値が `0` 以外のときに `true` を返します。つまり以下の例では、変数 `num` の値が `0` 以外である間、同じ処理が繰り返し行われます。

JSファイル（見本）

```
1 // 変数numに0〜4までのランダムな整数を代入する
2 let num = Math.floor(Math.random() * 5);
3
4 // 変数numの値が0以外である間、変数numの値を出力し続ける
5 while (num !== 0) {
6   num = Math.floor(Math.random() * 5);
7   console.log(num);
8 }
9
```

## while文を書いてみよう

while文を実際を書いてみましょう。まずはVisual Studio Codeを開き、`js` フォルダ内に新しく `loop.js` というファイルを作成してください。

続いて、`loop.js` を以下のように編集してください。先ほどの使用例と同じように、変数 `num` の値が `0` 以外である間、変数 `num` の値を出力し続けるプログラムを作成します。

loop.js



```
1 + // 変数numに0〜4までのランダムな整数を代入する
2 + let num = Math.floor(Math.random() * 5);
3 +
4 + // 変数numの最初の値を出力する（確認用）
5 + console.log('最初の値は' + num + 'です');
6 +
7 + // 変数numの値が0以外である間、変数numの値を出力し続ける
8 + while (num !== 0) {
9 +   // 変数numに0〜4までのランダムな整数を再代入する
10 +   num = Math.floor(Math.random() * 5);
11 +
12 +   // 次の条件式で比較される、変数numの現在の値を出力する
13 +   console.log('現在の値は' + num + 'です');
14 + }
15
```

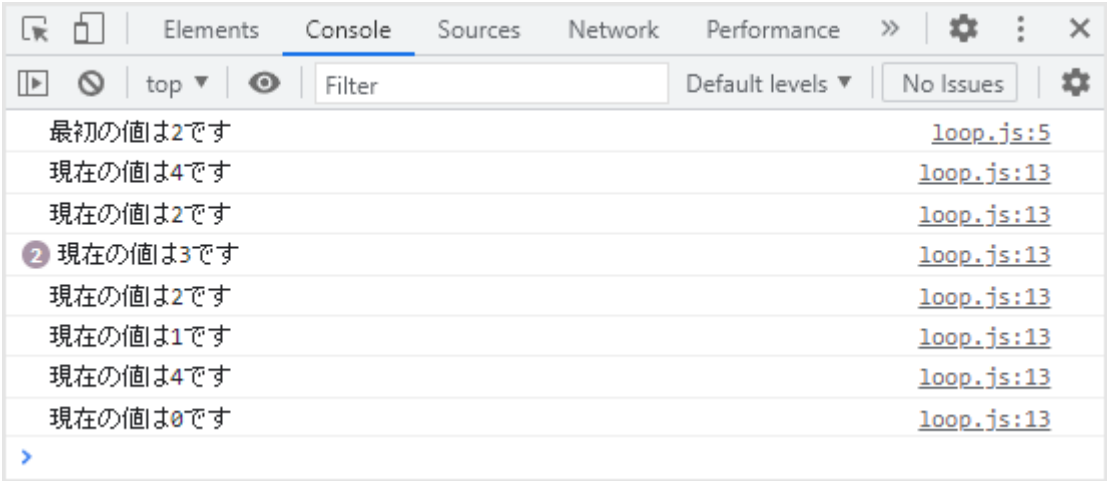
次に index.html を以下のように編集し、読み込むJSファイルを loop.js に変更してください。

index.html

```
1 <!DOCTYPE html>
2 <html lang="ja">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>JavaScript基礎編</title>
7 </head>
8
9 <body>
10 -   <script src="js/if-switch.js"></script>
11 +   <script src="js/loop.js"></script>
12 </body>
13
14 </html>
15
```

では index.html をブラウザで開き、デベロッパーツールのコンソールを確認してみましょう。


以下のように、変数 num の値が 0 になった時点で繰り返し処理が終了していればOKです。なお、同じ出力内容が連続する場合はまとめられ、左側にまとめられた数が表示されます。









このように、while文を使えば条件を満たしている間、同じ処理を繰り返す行うことができます。

## 無限ループの危険性

繰り返し処理を行うときに必ず注意しなければならないのが、**無限ループ**です。







JavaScript基礎編

← → ↺ ⓘ ファイル | C:/Users/g0932/Desktop/javascr... ☆ New 🔒 📁 🌐 ⚙️ 悠介 ⋮

Elements

Console

Sources

Network

Performance

⋮

html

Styles

Computed

Layout

Event listeners

DOM breakpoints

Properties

Accessibility

Filter

:hov .cls +

⏏

⋮

Console

Issues

What's New

⋮

⏏

🔇

top

👁

Filter

Default levels

No issues

⚙️

>

無限ループは、以下のように条件式が `true` を返し続けるとき（`false` を返す可能性がないとき）に発生します。

JSファイル（見本）

```
1 // 定数numに5を代入する
2 const num = 5;
3
4 // 条件式が常にtrueを返すので、無限ループになる
5 while (num === 5) {
6   console.log(num);
7 }
8
```

上記の例では、最初に定数 `num` に `5` を代入したきりそのままなので、定数 `num` の値は常に `5` です。よって、条件式 `num === 5` は常に `true` を返し、無限ループが発生してしまいます。

繰り返し処理を行うときは、「**条件式が `false` を返す可能性はあるか**」を必ず確認するようにしましょう。上記の例であれば、条件式 `num === 5` が `false` を返す、つまり定数 `num` の値が `5` 以外になる可能性があるかどうかを確認します。

## 無限ループが発生したときの対処法

万が一、無限ループが発生する繰り返し処理をブラウザで実行してしまった場合は、すぐにタブを閉じてください。ブラウザがフリーズした場合は強制終了しましょう。

ブラウザを強制終了する方法は以下のとおりです。

- Windows
  - 1. 「Ctrl + Shift + Esc」でタスクマネージャーを開く
  - 2. フリーズしたブラウザ（Google Chrome）を選択し、「タスクの終了」ボタンをクリックする
- Mac
  - 1. 「option + command + esc」で強制終了ウィンドウを開く
  - 2. フリーズしたブラウザ（Google Chrome）を選択し、「強制終了」ボタンをクリックする

無事に無限ループから抜け出せたら、原因となっている繰り返し処理をすぐに修正してください。具体的には条件式が `false` を返すようにしましょう。修正せずにブラウザで開くと、再び無限ループが始まってしまいます。

>

https://terakoya.sejuku.net/programs/60/chapters/674

5/10



## 8.4 for文の書き方

続いて、もう1つの繰り返し処理であるfor文について学びましょう。for文は、「**決まった回数**」同じ処理を繰り返し行いたいときによく使われます。

for文の書き方は以下のとおりです。丸括弧 ( ) 内の記述量が多いので戸惑うかもしれませんが、現時点ではイメージだけつかめればOKです。なお、カウンタ変数については後述します。

```
1 for (カウンタ変数の初期値; 条件式; カウンタ変数の増減値) {
2     カウンタ変数が決まった値になるまで繰り返す処理
3 }
4
```

丸括弧 ( ) 内のコードはセミコロン ; で区切ります。ただし、丸括弧 ( ) 内3つ目の「カウンタ変数の増減値」のあとには不要です。

### カウンタ変数とは

カウンタ変数とは、「繰り返し処理の回数を数えるために使われる変数」のことです。一般的な変数と機能は全く同じですが、「繰り返し処理の回数を数える」という用途を強調するためにカウンタ変数という言葉が使われます。

カウンタ変数には、慣習的に `i` という変数名がつけられることが多いです。なお、`i` は「索引」を意味するindexの頭文字です。

### for文の使用例

では、for文の使用例を見てみましょう。以下は本章の冒頭にも掲載したコードで、1～10までの数値を順番に出力します。

JSファイル（見本）

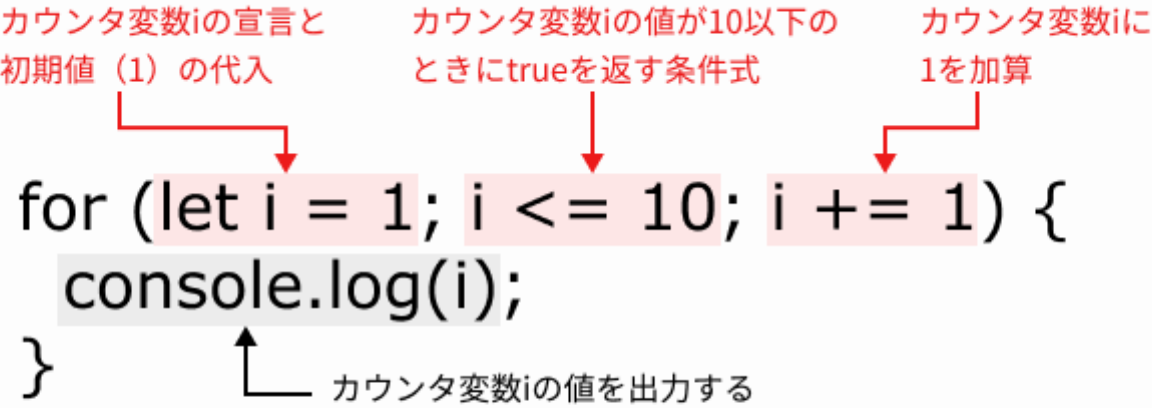
```
1 for (let i = 1; i <= 10; i += 1) {
2     console.log(i);
3 }
4
```

丸括弧 ( ) 内のコードを分解すると、以下のとおりです。

- `let i = 1` : カウンタ変数の初期値（カウンタ変数 `i` の宣言と値の代入を行う）
- `i <= 10` : 条件式（カウンタ変数 `i` の値が `10` 以下のときに `true` を返す）
- `i += 1` : カウンタ変数の増減値（処理を繰り返すごとに、カウンタ変数 `i` に `1` を加算する）

つまり、上記のコードはカウンタ変数 `i` が `1` からスタートし、処理を繰り返すごとに `1` ずつ加算されていきます。そして、カウンタ変数 `i` が `10` より大きくなる（条件式 `i <= 10` が `false` になる）と、繰り返し処理が終了します。





カウンタ変数*i*が1からスタートし、処理を繰り返すごとに1ずつ加算。  
カウンタ変数*i*が10より大きくなると、繰り返し処理が終了。

### 代入演算子について

カウンタ変数 *i* に 1 を加算するときに記述した += は、**代入演算子**といいます。

代入演算子とは、変数に値を代入するときに使う演算子のことです。これまでに何度も記述した = も代入演算子の1つです。参考程度に以下の一覧を見ておきましょう。

代入演算子	処理の内容
=	変数に右項の値を代入する（代入）。
+=	変数の値に右項の値を <b>足して</b> 代入する（加算代入）。
-=	変数の値から右項の値を <b>引いて</b> 代入する（減算代入）。
*=	変数の値に右項の値を <b>掛けて</b> 代入する（乗算代入）。
/=	変数の値を右項の値で <b>割って</b> 代入する（除算代入）。
%=	変数の値を右項の値で割った <b>余り</b> を代入する（剰余代入）。

### インクリメント演算子とデクリメント演算子について

*i* += 1 や *i* -= 1 のように単純に 1 増やしたり 1 減らしたりしたい場合、以下のように書き換えることもできます。

- *i* += 1 → *i*++
- *i* -= 1 → *i*--

この ++ のことを**インクリメント演算子**、 -- のことを**デクリメント演算子**といいます。

- インクリメント=増加
- デクリメント=減少

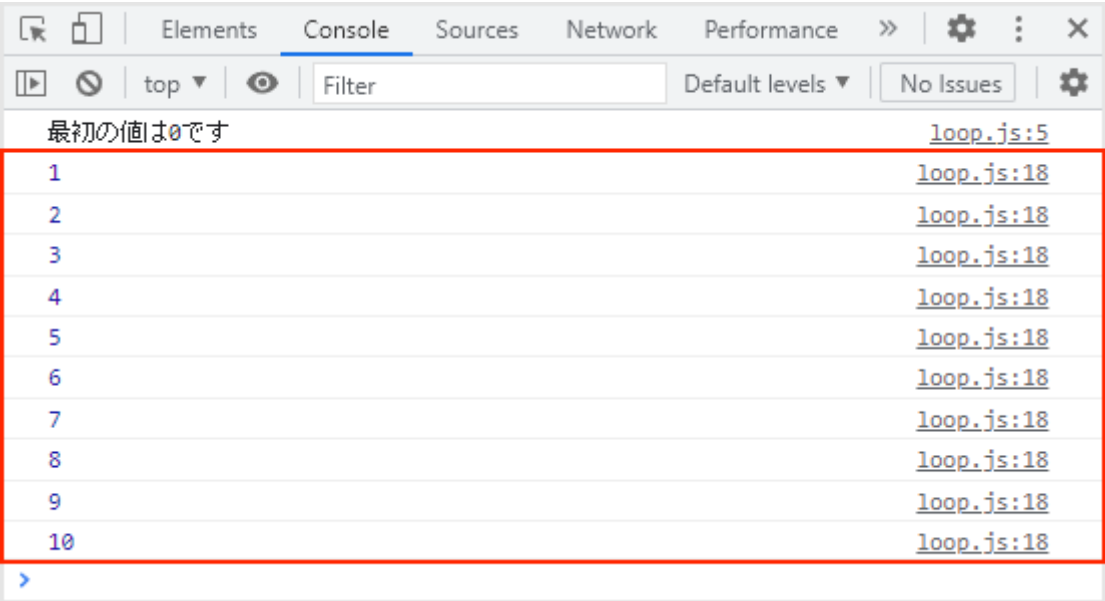
より簡潔に記述できるので、繰り返し処理ではよく使われます。覚えておきましょう。本教材でも、今後はインクリメント演算子またはデクリメント演算子を使って記述します。

では実際に、先ほどのfor文を書いてみましょう。 loop.js を以下のように編集してください。

```
loop.js

1 // 変数numに0〜4までのランダムな整数を代入する
2 let num = Math.floor(Math.random() * 5);
3
4 // 変数numの最初の値を出力する（確認用）
5 console.log('最初の値は' + num + 'です');
6
7 // 変数numの値が0以外である間、変数numの値を出力し続ける
8 while (num !== 0) {
9   // 変数numに0〜4までのランダムな整数を再代入する
10  num = Math.floor(Math.random() * 5);
11
12  // 次の条件式で比較される、変数numの現在の値を出力する
13  console.log('現在の値は' + num + 'です');
14 }
15
16 + // 1〜10までの数値を順番に出力する
17 + for (let i = 1; i <= 10; i++) {
18 +   console.log(i);
19 + }
20
```

続いて index.html をブラウザで開き、デベロッパーツールのコンソールを確認してみましょう。以下のように、1〜10までの数値が順番に表示されていればOKです。



なお、for文もwhile文と同様、無限ループが発生しないように十分注意する必要があります。

例えば先ほどのコードにおいて、増減値の部分を下のように減算代入（ i-- ）にしたとします。この場合、カウンタ変数 i が 10 より大きくなることが永遠にないため、無限ループが発生してしまいます。

```
1 // 条件式が常にtrueを返すので、無限ループになる
2 for (let i = 1; i <= 10; i--) {
3   console.log(num);
4 }
5
```

for文を使うときも、「条件式が false を返す可能性はあるか」を必ず確認するようにしましょう。上記の例であれば、条件式 i <= 10 が false になる、つまりカウンタ変数 i の値が 10 より大きくなる可能性があるかどうかを確認します。



🏠

🕒

📖

📄

🔍

✎

🔊

上記の例ではカウンタ変数 `i` が `1` から始まっているので、カウンタ変数の増減値が `0` より大きければいつかは必ず `10` よりも大きくなります。よって、カウンタ変数の増減値を `0` より大きくすれば無限ループは防げます。

本章の学習は以上です。お疲れさまでした。

## まとめ

本章では以下の内容を学習しました。

- 繰り返し処理とは、「決まった回数」または「条件を満たしている間」**同じ処理を繰り返し行うこと**である
- `while`文を使えば、「**条件を満たしている間**」同じ処理を繰り返し行うことができる
- `for`文は、「**決まった回数**」同じ処理を繰り返し行いたいときによく使われる
- 条件式が常に `true` を返す場合、**無限ループ**が発生してしまう
- 無限ループを防ぐために、繰り返し処理を行うときは「**条件式が `false` を返す可能性はあるか**」を必ず確認する
- `i += 1` や `i -= 1` のように単純に `1` 増やしたり `1` 減らしたりしたい場合は、**インクリメント演算子 `++`** や**デクリメント演算子 `--`** を使うことで簡潔に記述できる

構文	使うケース	例
<code>while</code> 文	繰り返す回数があらかじめ <b>わからない</b> 場合	サイコロで6の目が出るまで繰り返す
<code>for</code> 文	繰り返す回数があらかじめ <b>わかっている</b> 場合	1〜10までの数字を順番に表示する

代入演算子	処理の内容
<code>=</code>	変数に右項の値を代入する（代入）。
<code>+=</code>	変数の値に右項の値を <b>足して</b> 代入する（加算代入）。
<code>-=</code>	変数の値から右項の値を <b>引いて</b> 代入する（減算代入）。
<code>*=</code>	変数の値に右項の値を <b>掛けて</b> 代入する（乗算代入）。
<code>/=</code>	変数の値を右項の値で <b>割って</b> 代入する（除算代入）。
<code>%=</code>	変数の値を右項の値で割った <b>余り</b> を代入する（剰余代入）。

次章では、配列について学びます。

## 理解度を選択して次に進みましょう

ボタンを押していただくと次の章に進むことができます

✓

～50%

50～80%

80～100%



## 最後に確認テストを行いましょう

下のボタンを押すとテストが始まります。

教材をみなおす

テストをはじめる

前に戻る

10 / 26 ページ

次に進む

◀ 一覧に戻る

**!** 改善点のご指摘、誤字脱字、その他ご要望はこちらからご連絡ください。

© SAMURAI Inc.

[利用規約](#)

[法人会員利用規約](#)

[プライバシーポリシー](#)

[運営会社](#)