ES2019で追加された新機能

配列のフラット化 - `flat()`, `flatMap()`

配列をフラット化するためのメソッド。フラット化とは、多階層の配列を一階層にすることです。

`flat()` メソッドの挙動は次のとおりです。

flat()メソッド

```
[[1, 2], 3, 4].flat();
// 結果：[1, 2, 3, 4]
```

`flatMap()` メソッドは、配列をマッピングしたあとにフラット化します。たとえば、次のような二次元の配列から、「 `favorite` 」だけからなる配列を作りたいケースを考えてみましょう。

flatMap()メソッド

```
const MyData = [
  {
    name: "菅原さん",
    favorite: [ "大豆", "プロテイン", "つけ麺"]
  },
  {
    name: "山根さん",
    favorite: ["プロテイン", "ビタミンC"]
  },
  {
    name: "鹿野さん",
    favorite: [ "ラーメン", "つけ麺", "軟骨の唐揚げ"]
  }
];
```

`flatMap()` メソッドを使わない場合は、次のような記述が必要です。

```
MyData
  .map(data => data.favorite)
  .flat();
// 結果：["大豆", "プロテイン", "つけ麺", /*中略*/ "軟骨の唐揚げ"]
```

flatMap() メソッドを使うと、次のように記述できます。**フラット化したあとにマッピングではないこと**に注意しましょう。

```
MyData
  .flatMap(data => data.favorite);
// 結果：["大豆", "プロテイン", "つけ麺", /*中略*/ "軟骨の唐揚げ"]
```

公式ドキュメント

- Array.prototype.flatMap & Array.prototype.flat

キー・値のペアからオブジェクトを生成 - Object.fromEntries()

キー・値のペアからオブジェクトを生成するスタティックメソッド。挙動は次のとおりです。

Objec.fromEntries()

```
Object.fromEntries([["id", 16], ["name", "鈴木"]]);
// 結果：{id: 16, name: "鈴木"}
```

Map オブジェクトでもキー・値のペアを渡して初期化できましたが、それと似た挙動です。

Mapの初期化の参考

```
new Map([[1, "高橋"], [2, "後藤"]]);
// 結果： {1 => "高橋", 2 => "後藤"}
```

公式ドキュメント

- TC39 proposal for Object.fromEntries

文字列の先頭または末尾の空白を除去 - trimStart() , trimEnd()

文字列の先頭または末尾の空白を除去するメソッドです。

`trimStart()` メソッドは、文字列の先頭の空白を除去します。

trimStart()メソッド

```
"　寿司ざんまい　　".trimStart();
// 結果："寿司ざんまい　　"
```

`trimEnd()` メソッドは、文字列の末尾の空白を除去します。

trimEnd()メソッド

```
"　寿司ざんまい　　".trimEnd();
// 結果："　寿司ざんまい"
```

なお、文字列の両端を取り除く `trim()` メソッドはES2015で追加済みです。

trim()メソッド（ES2015で追加済）

```
"　寿司ざんまい　　".trim();
// 結果："寿司ざんまい"
```

公式ドキュメント

- String.prototype.trimStart/trimEnd

Symbol の説明を返す - description

`Symbol` の説明（ `description` ）を返すプロパティーです。説明とは、デバッグ用に `Symbol` を区別するためのものです。

Symbolのdescription

```
Symbol("矢部").description;
// 結果："矢部"
```

公式ドキュメント

- Symbol description accessor

try catch の catch ブロックで引数指定箇所を省略可能に

try catch の catch ブロックでは、引数指定の (error) 部分を記載しないとシンタックスエラーになっていました。

従来のtry-catch

```
try {
  throw new Error("☠");
} catch(error) {    // (error)の指定は必須
  console.warn("エラーをキャッチしました")
}
```

try catch の catch ブロックで、引数指定の箇所の (error) を記載しなくても良くなりました。

ES2019のtry-catch

```
try {
  throw new Error("☠");
} catch {   // (error)の記載はしなくても良い
  console.warn("エラーをキャッチしました")
}
```

公式ドキュメント

- proposal for ECMAScript to allow omission of the catch binding

関数を toString() で文字列に変換した際の仕様が改訂

関数を文字列化した場合、従来はこのような挙動でした。

従来

```
function /* こんにちは */ myFunction  () {}

myFunction.toString();
// 結果："function myFunction() {}"
// コメントや文字列が削除される
```

ES2019からは以下の挙動が正式な仕様となります。

ES2019

```
function /* こんにちは */ myFunction  () {}

myFunction.toString();
// 結果："function /* こんにちは */ myFunction  () {}"
// コメントや文字列が保持される
```

公式ドキュメント

- Function.prototype.toString revision

行区切り文字や段落切り文字をエスケープなしに扱えるように - ECMA-262構文をJSONのスーパーセットに拡張

JSONの元々の仕様として、行区切り文字（U+2028、Line Separator）や段落区切り文字（U+2029、Paragraph Separator）をエスケープなしに扱えます。

JSONでは元々エラーなし

```
JSON.parse('"\u2029"');  // エラーなし
```

※ \u番号 はユニコード文字を示す

一方で、JavaScriptで行区切り文字や段落区切り文字を扱う場合には、エスケープをしないとシンタックスエラーになっていました。

従来のJavaScriptのコードではシンタックスエラー

```
const PS = eval("'\u2029'"); // エラー
```

ES2019からは、シンタックスエラーに**ならない**のが仕様となり、JSONの挙動と同じになります。

ES2019ではシンタックスエラーにならないのが仕様

```
const PS = eval("'\u2029'"); // エラーなし
```

公式ドキュメント

- Proposal to make all JSON text valid ECMA-262

※ 2019/01/31 @mandel59 さんのコメントを受け、説明を改善しました

`JSON.stringify()` の取り扱い改善

文字列の  （U+D834: INVALID CHARACTER、参考リンク）などをエスケープし、
`JSON.stringify()` でJSON文字列に変換するケースを考えます。

このJavaScriptコードで解説します

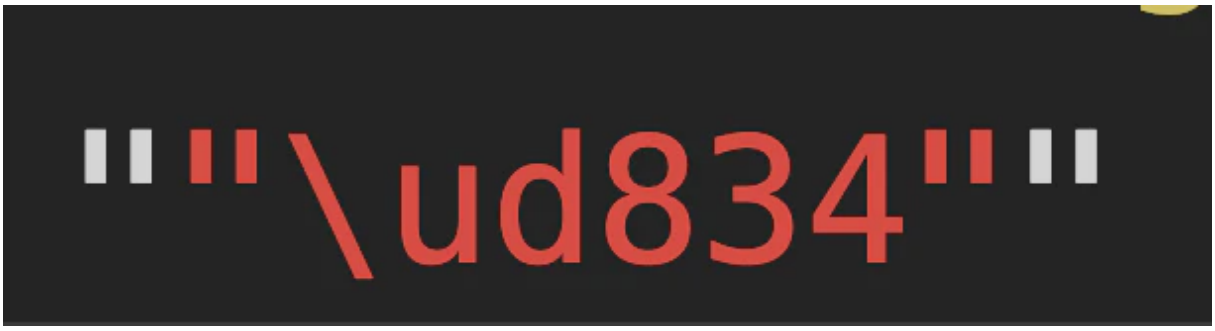
```
JSON.stringify('\uD834')
```

従来は次のようになっていました。次に示すのは、Google Chrome 71での実行結果です。

▼ Google Chrome 71でのコンソール表示結果

ES2019では、エスケープされたまま出力されるのが仕様となります。次に示すのは、Google Chrome **74**での実行結果です。

▼ Google Chrome 74でのコンソール表示結果



公式ドキュメント

- Proposal to prevent JSON.stringify from returning ill-formed strings

Google Chrome 73よりES2019の新機能はすべて使える

Google Chrome 73より、ES2019の新機能はすべて使えるとのことです。

Google Chrome 74（Canary）のコンソールで、今回紹介したES2019用のコードを実行した際のキャプチャーです。すべて仕様通りに動作していることがわかります。

▼ Google Chrome 74（Canary）の動作結果

```
> [[1, 2], 3, 4].flat();
◂ ▶(4) [1, 2, 3, 4]
> ["牛", "豚", "マグロ"].flatMap(food => [food, `${food}丼`]);
◂ ▶(6) ["牛", "牛丼", "豚", "豚丼", "マグロ", "マグロ丼"]
> Object.fromEntries([["id", 16], ["name", "鈴木"]]);
◂ ▶{id: 16, name: "鈴木"}
> "  寿司ざんまい     ".trimStart();
◂ "寿司ざんまい     "
> "  寿司ざんまい     ".trimEnd();
◂ "  寿司ざんまい"
> Symbol("矢部").description;
◂ "矢部"
> try {
    throw new Error("🙅‍♀️");
  } catch {
    console.warn("エラーをキャッチしました")
  }
⚠ ▶エラーをキャッチしました
◂ undefined
> function /* こんにちは */ myFunction  () {}

  myFunction.toString();
◂ "function /* こんにちは */ myFunction  () {}"
> const PS = eval("'\u2029'");
◂ undefined
> JSON.stringify('\uD834');
◂ ""\ud834""
```

ES2019への追加を期待していたが、まだステージ3の機能

次の機能がES2019への追加されることを個人的に期待していましたが、現時点ではまだステージ3です。早く正式仕様になってほしいと思いつつ、議論の進捗を見守っています。

- Dynamic Import
- BigInt
- クラスフィールド
- プライベートフィールド

最後に

配列のフラット化とOptional Catch Bindingが正式仕様になることが嬉しく思いました。

本記事で紹介したES2019の仕様は、Chrome 73ですべて対応済みです。その他のブラウザーについては、TypeScript・Babelとポリフィルで対応するとよいでしょう。

策定が終了した仕様の一覧は、次のURLで確認できます。

- Finished Proposals - tc39

ICS MEDIAで発信しているJavaScriptの資料もあわせてどうぞ。

- 「JavaScriptコードレシピ集」を執筆しました！ - ICS MEDIA
- 2019年までに見直しておきたいCSS・JavaScriptの手法 - Speaker Deck
- 2018年に見直した現代的なJavaScriptの記法を紹介するぜ - ICS MEDIA