ES6 : la déstructuration

Quand on utilise JavaScript on fait beaucoup usage des objets et tableaux. Il arrive également souvent qu'on ait besoin de certaines informations d'un objet ou d'un tableau dans des variables. On peut évidemment le faire avec ES5, mais on va voir dans ce chapitre que ES6 nous offre un outil bien pratique : la destructuration.

Déstructurons un objet

Vous avez un objet et vous voulez extraire ses données. Avec ES5 vous procédez ainsi :

```
var identite = { nom: 'Durand', prenom: 'Pierre' };
var nom = identite.nom;
var prenom = identite.prenom;
console.log(nom);  // Durand
console.log(prenom);  // Pierre
```

On a un code simple mais il pourrait rapidement s'alourdir avec un objet complexe ou chargé.

Avec ES6 vous pouvez utiliser cette syntaxe :

```
let identite = { nom: 'Durand', prenom: 'Pierre' };
let {nom, prenom} = identite;
console.log(nom);  // Durand
console.log(prenom);  // Pierre
```

- La valeur de la propriété **nom** de l'objet **identite** est copiée dans la variable **nom**.
- La valeur de la propriété **prenom** de l'objet **identite** est copiée dans la variable **prenom**.

Creusons

Vous pouvez aller aussi profondément dans l'objet que vous le désirez :

```
let personne = {
  identite: { nom: 'Durand', prenom: 'Pierre' },
  age: 20
};
let { identite: { nom, prenom }, age } = personne;
console.log(nom)  // Durand
console.log(prenom)  // Pierre
console.log(age)  // 20
```

Alias

Pour éviter des conflits de noms vous pouvez assigner des alias :

```
let personne = {
  identite: { nom: 'Durand', prenom: 'Pierre' },
  age: 20
};
let { identite: { nom: identiteNom, prenom: identitePrenom }, age:
identiteAge } = personne;
console.log(identiteNom)  // Durand
console.log(identitePrenom)  // Pierre
console.log(identiteAge)  // 20
```

Valeur par défaut

Que se passe-t-il pour une propriété non trouvée ?

Le comportement est exactement le même que lorsqu'on utilise l'extraction d'une propriété d'un objet avec la syntaxe classique, la valeur est **undefined** :

```
let personne = {nom: 'Durand', prenom: 'Pierre'};
let { nom, prenom, age } = personne;
console.log(nom)  // Durand
console.log(prenom)  // Pierre
console.log(age)  // undefined
```

Mais vous pouvez prévoir une valeur par défaut :

```
let personne = {nom: 'Durand', prenom: 'Pierre'};
let { nom, prenom, age = 20 } = personne;
console.log(nom)  // Durand
console.log(prenom)  // Pierre
```

```
console.log(age)  // 20
```

Déstructurons un tableau

Pour un tableau ça fonctionne de la même manière mais avec des crochets :

```
let prenoms = [ 'Pierre', 'Jacques', 'Paul' ];
let [ prenomUn, prenomDeux, prenomTrois ] = prenoms;
console.log(prenomUn); // Pierre
console.log(prenomDeux); // Jacques
console.log(prenomTrois); // Paul
```

Avec un tableau on doit choisir des noms pour les variables puisque à la base on a juste des index.

Mais rien n'empêche de déclarer auparavant les variables :

```
let prenoms = [ 'Pierre', 'Jacques'];
let prenomUn = 'Paul';
let prenomDeux = 'Martin';
[ prenomUn, prenomDeux ] = prenoms;
console.log(prenomUn); // Pierre
console.log(prenomDeux); // Jacques
```

On peut récupérer un nombre limité d'éléments si on veut :

```
let prenoms = [ 'Pierre', 'Jacques', 'Paul' ];
let [ prenomUn, prenomDeux ] = prenoms;
console.log(prenomUn); // Pierre
console.log(prenomDeux); // Jacques
```

Pour le cas où on veut sauter des éléments on utilise cette syntaxe :

```
let prenoms = [ 'Pierre', 'Jacques', 'Paul' ];
let [ , , prenomTrois ] = prenoms;
console.log(prenomTrois); // Paul
```

Creusons

Comme pour les objets on peut aller chercher des données imbriquées :

```
let prenoms = [ 'Pierre', ['Jacques']];
let [ prenomUn, [ prenomDeux ] ] = prenoms;
console.log(prenomUn); // Pierre
console.log(prenomDeux); // Jacques
```

Valeur par défaut

Comme pour les objets on peut prévoir une valeur par défaut :

```
let prenoms = [ 'Pierre' ];
let [ prenomUn, prenomDeux = 'Paul' ] = prenoms;
console.log(prenomUn); // Pierre
console.log(prenomDeux); // Paul
```

Inversion de variables

Il arrive qu'on ait besoin de permuter les valeurs de deux variables. Avec ES5 il faut passer par une variable temporaire :

```
var prenomUn = 'Pierre';
var prenomDeux = 'Paul';
var temp = prenomUn;
prenomUn = prenomDeux;
prenomDeux = temp;
console.log(prenomUn); // Paul
console.log(prenomDeux); // Pierre
```

Avec la déstructuration la syntaxe est plus légère et lisible :

```
let prenomUn = 'Pierre';
let prenomDeux = 'Paul';
[ prenomUn, prenomDeux ] = [ prenomDeux, prenomUn ];
console.log(prenomUn); // Paul
console.log(prenomDeux); // Pierre
```

Paramètre du reste

Je vous ai déjà parlé du paramètre du reste pour les fonctions. On peut aussi l'utiliser dans la déstructuration. Voici un exemple :

```
let prenoms = [ 'Pierre', 'Jacques', 'Paul' ];
let [ prenomUn, ...autres ] = prenoms;
```

```
console.log(prenomUn); // Pierre
console.log(autres[0]); // Jacques
console.log(autres[1]); // Paul
```

Des cas d’utilisation

Si vous avez une fonction qui retourne un objet il peut être pratique de déstructurer l’objet :

```
function getIdentite () {
  return { nom: 'Dupont', prenom: 'Pierre' };
}
var {nom, prenom} = getIdentite();
console.log(nom);  // Dupont
console.log(prenom);  // Pierre
```

Avec ES5 ça serait plus laborieux…

On peut considérer le problème inverse :

```
function dessineUnCercle ({ centre: { x = 10, y = 10 }, rayon = 20
}) {
  console.log(x, y, rayon);  // 5, 10, 20
}
dessineUnCercle({ centre: { x: 5 }});
```

Là aussi écrivez l’équivalent avec ES5 !

En résumé

- La déstructuration permet d’extraire les données d’un objet ou un tableau dans des variables.
- On peut prévoir des alias et des valeurs par défaut.
- La déstructuration permet de simplifier la syntaxe : inversion de variables, arguments de fonctions…