ECMAScript 2015

Les proxys

Proxy et metaprogrammation

La metaprogrammation

- Il existe deux niveau de code :
  - Celui qui s'exécute à un niveau classique (base level)
  - Le code qui process le code qui s'exécute (meta level)

- Dès que vous introspectez un élément, il s'agit de metaprogrammation
  - Itération sur les clefs d'un objet
  - Analyse des paramètres d'une fonction avec arguments

Les proxys

- Un proxy est un objet intermédiaire qui va intercepter les appels et de les modifier à la vollée
- Un proxy est constitué d'un **handler** qui définit son comportement et d'une **target** dont les appels sont interceptés

Exemple

```
const target = {};
const handler = {
  get(target, propKey, receiver) {
    console.log(`GET ${propKey}`);
    return '';
  }
};
const proxy = new Proxy(target, handler);

proxy.test; // affiche "GET test" et retourne ''
```

Intercepter les propriétés

- Intercepter l'appel :

```
get(target, propertyKey, receiver) {}
```

- Intercepter le test d'existance via **in**

```
has(target, propKey) {}
```

- Intercepter la suppression

```
deleteProperty(target, propKey) {}
```

Intercepter les méthodes

- Il est possible d'intercepter les méthodes avec **apply**

```
// Proxying a function object
var target = function () { return "I am the target";
};
var handler = {
  apply: function (receiver, ...args) {
    return "I am the proxy";
  }
};

var p = new Proxy(target, handler);
p() === "I am the proxy";
```

Proxy temporaires

- Il est possible de définir un proxy révocable :

```
const target = {}; // Start with an empty object
const handler = {}; // Don't intercept anything
const {proxy, revoke} = Proxy.revocable(target,
handler);

proxy.foo = 123;
console.log(proxy.foo); // 123

revoke();

console.log(proxy.foo); // TypeError: Revoked
```

Proxy passe-plat

- Il est possible de reprendre l'execution classique après l'execution du handler :

```
const handler = {
  deleteProperty(target, propKey) {
    console.log('DELETE ' + propKey);
    return delete target[propKey];
  },
  has(target, propKey) {
    console.log('HAS ' + propKey);
    return propKey in target;
  },
  // Other traps: similar
}
```

Proxy passe-plat avec Reflect

- Il existe un objet Reflect qui possède les même méthodes que le handler et fait passe-plat :

```
const handler = {
  deleteProperty(target, propKey) {
    console.log('DELETE ' + propKey);
    return Reflect.deleteProperty(target, propKey);
  },
  has(target, propKey) {
    console.log('HAS ' + propKey);
    return Reflect.has(target, propKey);
  },
  // Other traps: similar
}
```

Liste exhaustive des interceptions

```
handler.getPrototypeOf()
handler.setPrototypeOf()
handler.isExtensible()
handler.preventExtensions()
handler.getOwnPropertyDescriptor()
handler.defineProperty()
handler.has()
handler.get()
handler.set()
handler.deleteProperty()
handler.ownKeys()
handler.apply()
handler.construct()
```