Objets et chaînes de <prototypes> en JavaScript

Philippe Genoud

Philippe.Genoud@univ-grenoble-alpes.fr

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

<prototype> d'un objet

```
protosObjetsConstructeur1.js
1  function Chat(nom,age,race,poids) {
2     this.nom = nom ;
3     this.age = age ;
4     this.race = race ;
5     this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10    };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13       this.poids -= deltaPoids;
14    };
15
16 let felicite = new Chat("Félicité",10,"chat de gouttière", 3.5);
17
18 console.log(felicite);
19
```

définition du constructeur d'objets **Chat**

méthodes associées aux objets **Chat**

création d'un objet **Chat** (instance)

Sur la console du navigateur

```
Éléments   Console   Sources
top ▾   Filtrer
1 problème : 1
object                                                     index.html:129
                                             protosObjetsConstructeur1.js:18
▶ Chat {nom: 'Félicité', age: 10, race: 'chat de gouttière', poids: 3.5}
>
```

L'objet référencé par **felicite** défini avec 4 propriétés : **nom**, **age**, **race**, **poids**

Voir le code

Dans le débogueur du navigateur

```
Suspendu sur point d'arrêt              Niveau d'accès   Contrôle
▶ Points d'arrêt                        ▼ Script
▼ Pile d'appel                            ▼ felicite: Chat
  (anonyme)  protosObjetsCon...ructeur1.js:18    age: 10
▶ Points d'arrêt XHR/Fetch                    nom: "Félicité"
▶ Points d'arrêt DOM                          poids: 3.5
▶ Écouteurs globaux                           race: "chat de gouttière"
                                            ▶ [[Prototype]]: Object
                                        ▶ Globales                 Window
```

```
Mis en pause par un point d'arrêt       ▶ Expressions espionnes
▶ Points d'arrêt                        ▼ Portées
▼ Pile d'exécution                        ▼ Bloc
  (global)  protosObjetsConstructeur1.js:18   ▶ <this>: Window
▶ Points d'arrêt XHR                        ▼ felicite: {…}
▶ Points ... □ Enregistrer dans le journal      age: 10
▶ Points d'arrêt de mutations DOM               nom: "Félicité"
                                                poids: 3.5
                                                race: "chat de gouttière"
                                              ▶ <prototype>: {…}
                                          ▶ Window: Global
```

En interne l'objet référencé par **felicite** possède une propriété supplémentaire que l'on peut observer à l'aide du débogueur : **[[Prototype]]** (Chrome ou Edge) ou **<prototype>** (Firefox)

L'attribut interne `<prototype> d'un objet` ≠ attribut `prototype` d'une fonction constructeur

Chaîne de <prototypes>

- Chaque objet possède un lien interne (masqué) nommé son `<prototype>` (Firefox) ou `[[Prototype]]` (Chrome, Edge, Node) qui le relie soit à un autre objet soit à `null`.

- Ce lien est unique : un objet possède un et un seul lien `<prototype>`
- Ces liens constituent ce que l'on nomme la **chaîne de <prototypes>** de l'objet

```
protosObjetsConstructeur1.js
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9         console.log("Miaou ! Miaou !");
10     };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13        this.poids -= deltaPoids;
14     };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

```
Portées
▼ Bloc
  ▶ <this>: Window
  ▶ felicite: {…}
▶ Window: Global
```

L'objet **`felicite`**

```
▼ felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  ▶ <prototype>: {…}
```

`<prototype>` de l'objet **`felicite`**

```
▶ constructor:Chat()
▶ maigrir:Chat.prototype.maigrir()
▶ miauler:Chat.prototype.miauler()
▶ <prototype>: {…}
```

<prototype> du <prototype> de l'objet **`felicite`**

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: »
▶ constructor:Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

chaîne de `<prototypes>` de l'objet référencé par **`felicite`**

dernier objet de la chaîne son `<prototype>` est **null**

Voir le code

Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10   };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13       this.poids -= deltaPoids;
14   };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
- Window: Global

L'objet **felicite**

```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

lors de l'accès à une propriété (attribut) ou lors de l'exécution d'une méthode d'un objet, l'interpréteur JavaScript, s'il ne trouve pas la propriété ou la méthode dans l'objet lui-même, effectue une recherche de celle-ci en parcourant la chaîne de <prototypes>.

Voir le code

Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9          console.log("Miaou ! Miaou !");
10     };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13         this.poids -= deltaPoids;
14     };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
- Window: Global

L'objet **felicite**

```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  <prototype>: {…}
```

Prototype de l'objet **felicite**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

Prototype du prototype de l'objet **felicite**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

```
console.log(felecite.nom);
```

Voir le code

Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9          console.log("Miaou ! Miaou !");
10     };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13         this.poids -= deltaPoids;
14     };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
- Window: Global

L'objet **felicite**

```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

```
console.log(felicite.nom);
```

Voir le code

Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10    };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13       this.poids -= deltaPoids;
14    };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

Portées

```
▼ Bloc
  ▶ <this>: Window
  ▶ felicite: {…}
▶ Window: Global
```

L'objet **felicite**

```
▼ felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  ▶ <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
▶ constructor:Chat()
▶ maigrir:Chat.prototype.maigrir()
▶ miauler:Chat.prototype.miauler()
▶ <prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: »
▶ constructor:Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

```
console.log(felicite.nom);
```

```
felicite.miauler();
```

Voir le code

© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9         console.log("Miaou ! Miaou !");
10     };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13        this.poids -= deltaPoids;
14     };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
- Window: Global

L'objet **felicite**

```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
    <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

```
console.log(felicite.nom);
```

```
felicite.miauler();
```

Voir le code

© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9          console.log("Miaou ! Miaou !");
10     };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13         this.poids -= deltaPoids;
14     };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
- Window: Global

L'objet **felicite**

```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

```
console.log(felicite.nom);
```

```
felicite.miauler();
```

Voir le code

© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

Chaîne de prototypes

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```javascript
function Chat(nom,age,race,poids) {
    this.nom = nom ;
    this.age = age ;
    this.race = race ;
    this.poids = poids ;
}

Chat.prototype.miauler = function () {
      console.log("Miaou ! Miaou !");
  };

Chat.prototype.maigrir  = function(deltaPoids) {
      this.poids -= deltaPoids;
  };

let felicite = new Chat("Félicité",10,"
console.log(felicite);
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
- Window: Global

L'objet **felicite**

```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
    <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

```javascript
console.log(felicite.nom);
felicite.miauler();
console.log(felicite.toString());
```

Voir le code

Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9          console.log("Miaou ! Miaou !");
10     };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13         this.poids -= deltaPoids;
14     };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
- Window: Global

L'objet **felicite**

```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

```
console.log(felicite.nom);
```

```
felicite.miauler();
```

```
console.log(felicite.toString());
```

Voir le code

© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

Chaîne de <prototypes> : fallback

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10   };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13       this.poids -= deltaPoids;
14   };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
- Window: Global

L'objet **felicite**

```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

```
console.log(felicite.nom);
```

```
felicite.miauler();
```

```
console.log(felicite.toString());
```

Voir le code

Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10     };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13       this.poids -= deltaPoids;
14     };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
- Window: Global

L'objet **felicite**

```
▼ felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  ▶ <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
▶ constructor:Chat()
▶ maigrir:Chat.prototype.maigrir()
▶ miauler:Chat.prototype.miauler()
▶ <prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: »
▶ constructor:Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

```
console.log(felicite.nom);
felicite.miauler();
console.log(felicite.toString());
```

Voir le code

© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

Chaîne de <prototypes> : fallback

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9          console.log("Miaou ! Miaou !");
10     };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13         this.poids -= deltaPoids;
14     };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
- Window: Global

L'objet **felicite**

```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

```
console.log(felicite.nom);
```

```
felicite.miauler();
```

```
console.log(felicite.toString());
```

```
felicite.grossir(1.0);
```

Voir le code

© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

Chaîne de <prototypes> : fallback

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9          console.log("Miaou ! Miaou !");
10     };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13         this.poids -= deltaPoids;
14     };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
- Window: Global

L'objet **felicite**

```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

```
console.log(felicite.nom);
felicite.miauler();
console.log(felicite.toString());
felicite.grossir(1.0);
```

Voir le code

Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

protosObjetsConstructeur1.js

```
1  function Chat(nom,age,race,poids) {
2      this.nom = nom ;
3      this.age = age ;
4      this.race = race ;
5      this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10     };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13       this.poids -= deltaPoids;
14     };
15
16 let felicite = new Chat("Félicité",10,"
17
18 console.log(felicite);
19
20
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
- Window: Global

L'objet **felicite**

```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

```
console.log(felicite.nom);
felicite.miauler();
console.log(felicite.toString());
felicite.grossir(1.0);
```

```
TypeError: felicite.grossir is not a function
```

Voir le code

<prototype> et constructeur

- Les objets créés à partir d'un même constructeur partagent le même <prototype>

protosObjetsConstructeur1_1.js

```
1  function Chat(nom,age,race,poids) {
2     this.nom = nom ;
3     this.age = age ;
4     this.race = race ;
5     this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10    };
11
12 Chat.prototype.maigrir  = function(deltaPoids) {
13       this.poids -= deltaPoids;
14    };
15
16 let felicite = new Chat("Félic
17 let felix    = new Chat("Felix
18
19 console.log(felicite);
20
```

```
Portées
  Bloc
    <this>: Window
    felicite: {…}
    felix: {…}
  Window: Global
```

Voir le code

L'objet **felicite**

```
felicite: {…}
   age: 10
   nom: "Félicité"
   poids: 3.5
   race: "chat de gouttière"
   <prototype>: {…}
```

<prototype> de l'objet **felicite**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

<prototype> et constructeur

- Les objets créés à partir d'un même constructeur partagent le même <prototype>

protosObjetsConstructeur1_1.js

```
1  function Chat(nom,age,race,poids) {
2     this.nom = nom ;
3     this.age = age ;
4     this.race = race ;
5     this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10    };
11
12 Chat.prototype.maigrir  = function(del
13       this.poids -= deltaPoids;
14    };
15
16 let felicite = new Chat("Félic
17 let felix    = new Chat("Felix
18
19 console.log(felicite);
20
```

Voir le code

L'objet **felicite**

```
▼ felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  ▶ <prototype>: {…}
```

```
▼ felix: {…}
    age: 6
    nom: "Felix"
    poids: 3
    race: "siamois"
  ▶ <prototype>: {…}
```

<prototype> de l'objet **felicite** et de l'objet **felix**

```
▶ constructor:Chat()
▶ maigrir:Chat.prototype.maigrir()
▶ miauler:Chat.prototype.miauler()
▶ <prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: »
▶ constructor:Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

```
▼ Portées
  ▼ Bloc
    ▶ <this>: Window
    ▶ felicite: {…}
    ▶ felix: {…}
  ▶ Window: Global
```

<prototype> et constructeur

- Les objets créés à partir d'un même constructeur partagent le même <prototype>

protosObjetsConstructeur1_1.js

```
1  function Chat(nom,age,race,poids) {
2     this.nom = nom ;
3     this.age = age ;
4     this.race = race ;
5     this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10    };
11
12 Chat.prototype.maigrir  = function(del
13       this.poids -= deltaPoids;
14    };
15
16 let felicite = new Chat("Félic
17 let felix    = new Chat("Felix
18
19 console.log(felicite);
20
```

Voir le code

L'objet **felicite**

```
▼ felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  ▶ <prototype>: {…}
```

```
▼ felix: {…}
    age: 6
    nom: "Felix"
    poids: 3
    race: "siamois"
  ▶ <prototype>: {…}
```

```
▼ Portées
  ▼ Bloc
    ▶ <this>: Window
    ▶ felicite: {…}
    ▶ felix: {…}
  ▶ Window: Global
```

```
▼ constructor:Chat()
    arguments: null
    caller: null
    length: 4
    name: "Chat"
  ▶ prototype: {…}
  ▶ <prototype>()
```

La fonction constructeur **Chat** est aussi un objet

Le **<prototype>** des objets créés avec le constructeur **Chat** est le **prototype** de l'objet constructeur **Chat**

<prototype> de l'objet **felicite** et de l'objet **felix**

```
▶ constructor:Chat()
▶ maigrir:Chat.prototype.maigrir()
▶ miauler:Chat.prototype.miauler()
▶ <prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: »
▶ constructor:Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

<prototype> et constructeur

- Les objets créés à partir d'un même constructeur partagent le même <prototype>

protosObjetsConstructeur1_1.js

```
1  function Chat(nom,age,race,poids) {
2     this.nom = nom ;
3     this.age = age ;
4     this.race = race ;
5     this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10    };
11
12 Chat.prototype.maigrir  = function(del
13       this.poids -= deltaPoids;
14    };
15
16 let felicite = new Chat("Félic
17 let felix    = new Chat("Felix
18
19 console.log(felicite);
20
```

L'objet **felicite**

```
▼ felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  ▶ <prototype>: {…}
```

```
▼ felix: {…}
    age: 6
    nom: "Felix"
    poids: 3
    race: "siamois"
  ▶ <prototype>: {…}
```

```
▼ Portées
  ▼ Bloc
    ▶ <this>: Window
    ▶ felicite: {…}
    ▶ felix: {…}
  ▶ Window: Global
```

Voir le code

```
▼ constructor:Chat()
    arguments: null
    caller: null
    length: 4
    name: "Chat"
  ▶ prototype: {…}
  ▶ <prototype>()
```

<prototype> de l'objet **felicite** et de l'objet **felix**

```
▶ constructor:Chat()
▶ maigrir:Chat.prototype.maigrir()
▶ miauler:Chat.prototype.miauler()
▶ <prototype>: {…}
```

```
▶ apply()
  arguments: »
▶ bind()
▶ call()
  caller: »
▶ constructor:Function()
  length: 0
  name: ""
▶ toSource()
▶ toString()
▶ Symbol(Symbol.hasInstance):[Symb
▶ <get arguments()>: arguments()
▶ <set arguments()>: arguments()
▶ <get caller()>: caller()
▶ <set caller()>: caller()
▶ <prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
▼ <prototype>: {…}
  ▶ __defineGetter__()
  ▶ __defineSetter__()
  ▶ __lookupGetter__()
  ▶ __lookupSetter__()
    __proto__: »
  ▶ constructor:Object()
  ▶ hasOwnProperty()
  ▶ isPrototypeOf()
  ▶ propertyIsEnumerable()
  ▶ toLocaleString()
  ▶ toSource()
  ▶ toString()
  ▶ valueOf()
```

Comme tout objet le constructeur **Chat**[1] possède une **chaîne de <prototypes>**

Toutes les chaînes de <prototypes> partagent le même objet terminal

[1] **Chat** est une fonction et en JavaScript les fonctions sont des objets

<prototype> vs. Constructeur.prototype

Ne pas confondre

- la propriété (attribut) **prototype** d'un objet constructeur
- le lien prototype (**<prototype>**) de cet objet constructeur

protosObjetsConstructeur1_1.js

```
1  function Chat(nom,age,race,poids) {
2     this.nom = nom ;
3     this.age = age ;
4     this.race = race ;
5     this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10    };
11
12 Chat.prototype.maigrir  = function(del
13       this.poids -= deltaPoids;
14    };
15
16 let felicite = new Chat("Félic
17 let felix    = new Chat("Felix
18
19 console.log(felicite);
20
```

Voir le code

L'objet **felicite**

```
felicite: {…}
   age: 10
   nom: "Félicité"
   poids: 3.5
   race: "chat de gouttière"
   <prototype>: {…}
```

```
felix: {…}
   age: 6
   nom: "Felix"
   poids: 3
   race: "siamois"
   <prototype>: {…}
```

```
Portées
  Bloc
    <this>: Window
    felicite: {…}
    felix: {…}
  Window: Global
```

```
constructor:Chat()
   arguments: null
   caller: null
   length: 4
   name: "Chat"
   prototype: {…}
   <prototype>()
```

<prototype> de l'objet **felicite** et de l'objet **felix**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

```
apply()
arguments: »
bind()
call()
caller: »
constructor:Function()
length: 0
name: ""
toSource()
toString()
Symbol(Symbol.hasInstance):[Symb
<get arguments()>: arguments()
<set arguments()>: arguments()
<get caller()>: caller()
<set caller()>: caller()
<prototype>: {…}
```

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

<prototype> vs. Constructeur.prototype

Ne pas confondre

1. la propriété (attribut) **prototype** d'un objet constructeur
- le lien prototype (**<prototype>**) de cet objet constructeur

```
protosObjetsConstructeur1_1.js

1  function Chat(nom,age,race,poids) {
2     this.nom = nom ;
3     this.age = age ;
4     this.race = race ;
5     this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10    };
11
12 Chat.prototype.maigrir  = function(del
13       this.poids -= deltaPoids;
14    };
15
16 let felicite = new Chat("Félic
17 let felix    = new Chat("Felix
18
19 console.log(felicite);
20
```

Portées
- Bloc
  - <this>: Window
  - felicite: {…}
  - felix: {…}
- Window: Global

Voir le code

L'objet **felicite**

```
felicite: {…}
   age: 10
   nom: "Félicité"
   poids: 3.5
   race: "chat de gouttière"
   <prototype>: {…}
```

```
felix: {…}
   age: 6
   nom: "Felix"
   poids: 3
   race: "siamois"
   <prototype>: {…}
```

```
constructor:Chat()
   arguments: null
   caller: null
   length: 4
   name: "Chat"
   prototype: {…}
   <prototype>()
```

```
apply()
arguments: »
bind()
call()
caller: »
constructor:Function()
length: 0
name: ""
toSource()
toString()
Symbol(Symbol.hasInstance):[Symb
<get arguments()>: arguments()
<set arguments()>: arguments()
<get caller()>: caller()
<set caller()>: caller()
<prototype>: {…}
```

<prototype> de l'objet **felicite** et de l'objet **felix**

```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

1. le **prototype** de l'objet constructeur **Chat** est le **<prototype>** des objets créés avec le constructeur **Chat**

<prototype> vs. Constructeur.prototype

Ne pas confondre

1. la propriété (attribut) **prototype** d'un objet constructeur
2. le lien prototype (**<prototype>**) de cet objet constructeur

protosObjetsConstructeur1_1.js

```
1  function Chat(nom,age,race,poids) {
2     this.nom = nom ;
3     this.age = age ;
4     this.race = race ;
5     this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10    };
11
12 Chat.prototype.maigrir  = function(del
13       this.poids -= deltaPoids;
14    };
15
16 let felicite = new Chat("Félic
17 let felix    = new Chat("Felix
18
19 console.log(felicite);
20
```

Voir le code

L'objet **felicite**

```
▼ felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  ▶ <prototype>: {…}
```

```
▼ felix: {…}
    age: 6
    nom: "Felix"
    poids: 3
    race: "siamois"
  ▶ <prototype>: {…}
```

```
▼ Portées
  ▼ Bloc
    ▶ <this>: Window
    ▶ felicite: {…}
    ▶ felix: {…}
  ▶ Window: Global
```

```
▼ constructor:Chat()
    arguments: null
    caller: null
    length: 4
    name: "Chat"
  ▶ prototype: {…}      (1)
  ▶ <prototype>()       (2)
```

```
▶ apply()
  arguments: »
▶ bind()
▶ call()
  caller: »
▶ constructor:Function()
  length: 0
  name: ""
▶ toSource()
▶ toString()
▶ Symbol(Symbol.hasInstance):[Symb
▶ <get arguments()>: arguments()
▶ <set arguments()>: arguments()
▶ <get caller()>: caller()
▶ <set caller()>: caller()
▶ <prototype>: {…}
```

<prototype> de l'objet **felicite** et de l'objet **felix**

```
▶ constructor:Chat()
▶ maigrir:Chat.prototype.maigrir()
▶ miauler:Chat.prototype.miauler()
▶ <prototype>: {…}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: »
▶ constructor:Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

1. le **prototype** de l'objet constructeur **Chat** est le **<prototype>** des objets créés avec le constructeur **Chat**
2. **<prototype>** du constructeur est la propriété **prototype** de la fonction qui a servi à créer l'objet constructeur **Chat()**

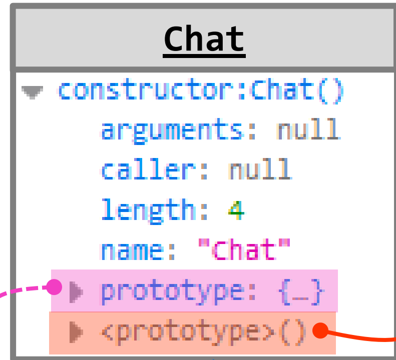
© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

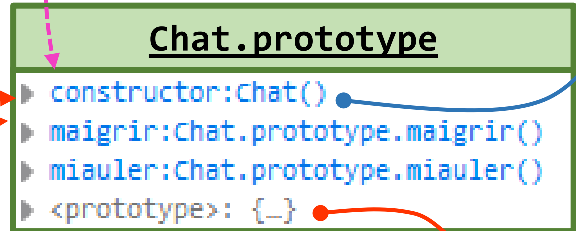
Built-in objects

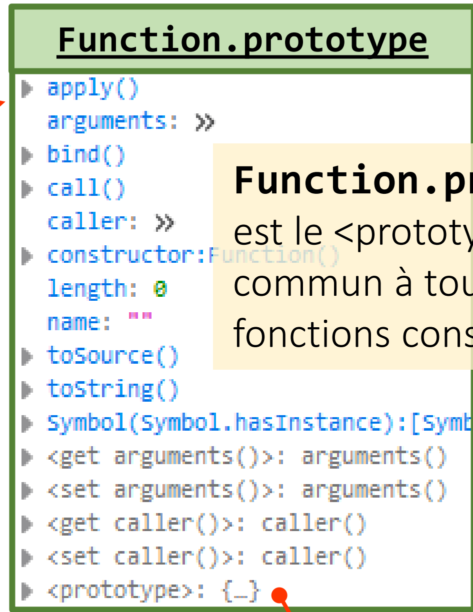
`Object` et **`Function`** sont des objets (fonctions constructeur) prédéfinis (*built-in*)

protosObjetsConstructeur1_1.js

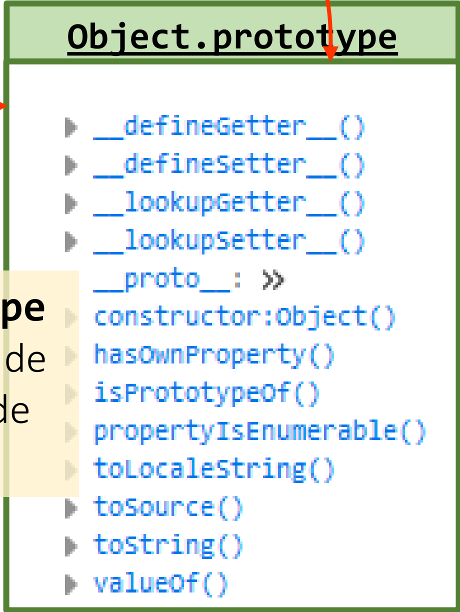
```
1  function Chat(nom,age,race,poids) {
2     this.nom = nom ;
3     this.age = age ;
4     this.race = race ;
5     this.poids = poids ;
6  }
7
8  Chat.prototype.miauler = function () {
9        console.log("Miaou ! Miaou !");
10    };
11
12 Chat.prototype.maigrir  = function(del
13       this.poids -= deltaPoids;
14    };
15
16 let felicite = new Chat("Félicité",10,"chat de gouttière", 3.5);
17 let felix    = new Chat("Felix",6,"siamois",3);
18
19 console.log(felicite);
20
```







`Function.prototype` est le <prototype> commun à toutes les fonctions constructeur

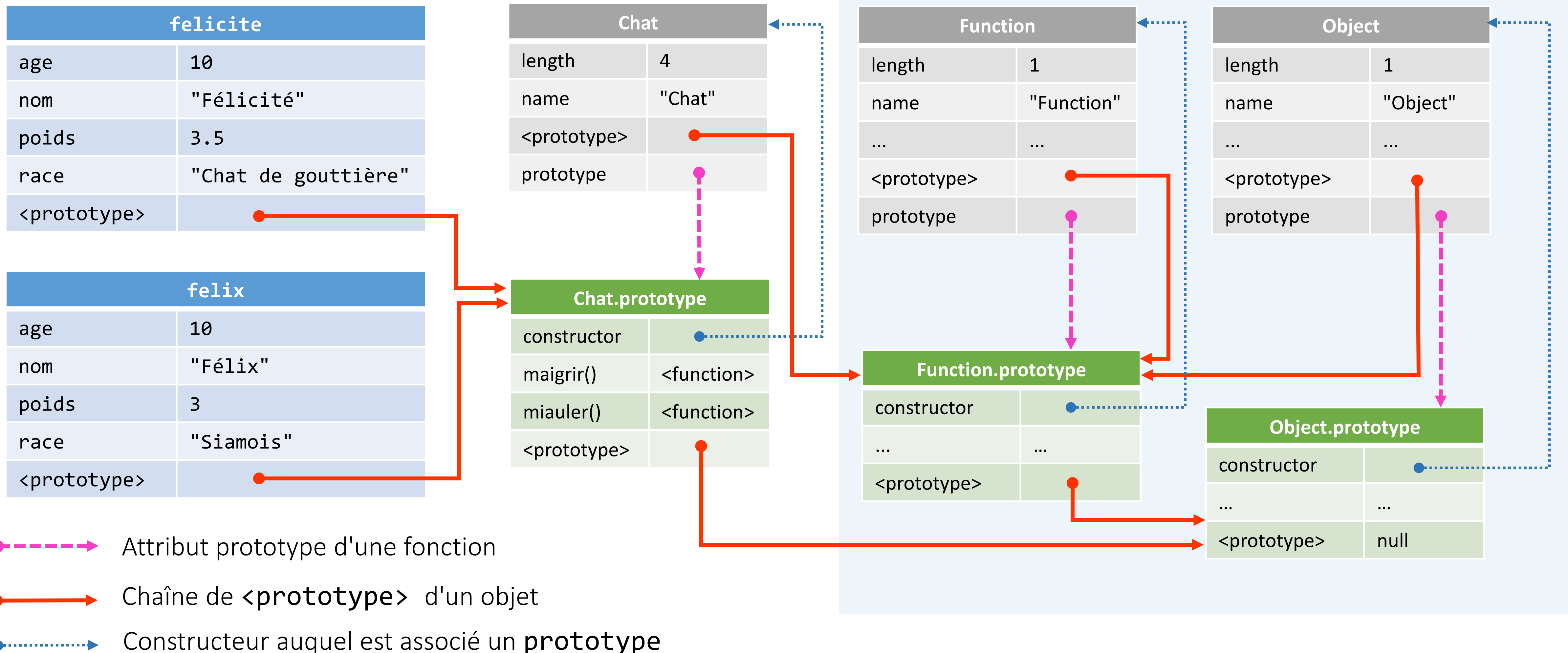


`Object.prototype` est l'objet terminal de toutes les chaînes de <prototypes>

Voir le code

Built-in objects

- Les fonctions **`Function()`** et **`Object()`** et leur chaîne de **`<prototypes>`**



<prototype> objet Littéral

- Le <prototype> d'un objet littéral est `Object.prototype`

```
1  function Chat(nom, age, race, poids) {
2      this.nom = nom;
3      this.age = age;
4      this.race = race;
5      this.poids = poids;
6  }
7
8  Chat.prototype.miauler = function () {
9      console.log("Miaou ! Miaou !");
10 };
11
12 Chat.prototype.maigrir = function (deltaP
13     this.poids -= deltaPoids;
14 };
15
16 let felicite = new Chat("Félicité", 10, "
17 let felix = new Chat("Felix", 6, "siamois
18
19 let fritz = {
20     nom: "Fritz le chat",
21     age: 14,
22     race: "chat tigré",
23     poids: 4.0
24 };
```

felicite
```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  <prototype>: {…}
```

felix
```
felix: {…}
    age: 6
    nom: "Felix"
    poids: 3
    race: "siamois"
  <prototype>: {…}
```

fritz
```
fritz: {…}
    age: 14
    nom: "Fritz le chat"
    poids: 4
    race: "chat tigré"
  <prototype>: {…}
```

Chat
```
constructor:Chat()
    arguments: null
    caller: null
    length: 4
    name: "Chat"
  prototype: {…}
  <prototype>()
```

Chat.prototype
```
constructor:Chat()
maigrir:Chat.prototype.maigrir()
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

Function.prototype
```
apply()
arguments: »
bind()
call()
caller: »
constructor:Function()
length: 0
name: ""
toSource()
toString()
Symbol(Symbol.hasInstance):[Symb
<get arguments()>: arguments()
<set arguments()>: arguments()
<get caller()>: caller()
<set caller()>: caller()
<prototype>: {…}
```

Object.prototype
```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

Voir le code

Fallback de propriétés

- Une propriété définie au niveau du `prototype` d'un constructeur est 'héritée' par tous les objets ayant ce `<prototype>`

```
1  function Chat(nom, age, race, poids) {
2      this.nom = nom;
3      this.age = age;
4      this.race = race;
5      this.poids = poids;
6  }
7
8  Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11     console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maigrir = function (deltaPoids) {
15     this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19     console.log("cri de félicité " + felicite.cri);
20     console.log("cri de félix " + felix.cri);
21     felicite.miauler();
22     felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

???

felicite

```
▼ felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  ▶ <prototype>: {…}
```

felix

```
▼ felix: {…}
    age: 6
    nom: "Felix"
    poids: 3
    race: "siamois"
  ▶ <prototype>: {…}
```

Chat.prototype

```
▶ constructor:Chat(nom, age, race, poids)
  cri: "Miaou Miaou"
▶ maigrir:Chat.prototype.maigrir(deltaPoids)
▶ miauler:Chat.prototype.miauler()
▶ <prototype>: {…}
```

Object.prototype

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: »
▶ constructor:Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

Voir le code

Fallback de propriétés

- Une propriété définie au niveau du **prototype** d'un constructeur est 'héritée' par tous les objets ayant ce **<prototype>**

```
1  function Chat(nom, age, race, poids) {
2      this.nom = nom;
3      this.age = age;
4      this.race = race;
5      this.poids = poids;
6  }
7
8  Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11     console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maigrir = function (deltaPoids) {
15     this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19     console.log("cri de félicité " + felicite.cri);
20     console.log("cri de félix " + felix.cri);
21     felicite.miauler();
22     felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

???

felicite

```
▼ felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  ▶ <prototype>: {…}
```

propriété cri ?

felix

```
▼ felix: {…}
    age: 6
    nom: "Felix"
    poids: 3
    race: "siamois"
  ▶ <prototype>: {…}
```

propriété cri ?

Chat.prototype

```
▶ constructor:Chat(nom, age, race, poids)
  cri: "Miaou Miaou"
▶ maigrir:Chat.prototype.maigrir(deltaPoids)
▶ miauler:Chat.prototype.miauler()
▶ <prototype>: {…}
```

Object.prototype

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: »
▶ constructor:Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

évaluation de **felicite.cri**

recherche de la propriété **cri** dans l'objet **felicite** (propriété propre)

elle n'est pas trouvée

recherche de la **cri** dans le **<prototype>** de **felicite**

→ "Miaou Miaou"

Voir le code

© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

Fallback de propriétés

- Une propriété définie au niveau du **prototype** d'un constructeur est 'héritée' par tous les objets ayant ce **<prototype>**

```
1  function Chat(nom, age, race, poids) {
2      this.nom = nom;
3      this.age = age;
4      this.race = race;
5      this.poids = poids;
6  }
7
8  Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11     console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maigrir = function (deltaPoids) {
15     this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19     console.log("cri de félicité " + felicite.cri);
20     console.log("cri de félix " + felix.cri);
21     felicite.miauler();
22     felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

felicite

```
▼ felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  ▶ <prototype>: {…}
```

propriété cri ?

felix

```
▼ felix: {…}
    age: 6
    nom: "Felix"
    poids: 3
    race: "siamois"
  ▶ <prototype>: {…}
```

```
cri de félicité Miaou Miaou
cri de félix Miaou Miaou
Félicité : Miaou Miaou
Felix : Miaou Miaou
```

les cris des objets sont *'hérités'* de leur objet <prototype>

propriété cri ?

Chat.prototype

```
▶ constructor:Chat(nom, age, race, poids)
  cri: "Miaou Miaou"
▶ maigrir:Chat.prototype.maigrir(deltaPoids)
▶ miauler:Chat.prototype.miauler()
▶ <prototype>: {…}
```

Object.prototype

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: »
▶ constructor:Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

évaluation de **felicite.cri**

recherche de la propriété **cri** dans l'objet **felicite** (propriété propre)

elle n'est pas trouvée

recherche de la **cri** dans le **<prototype>** de **felicite**

→ "Miaou Miaou"

idem pour **felix.cri** et **this.cri** dans la méthode **miauler**

Voir le code

Fallback de propriétés

- Une propriété définie au niveau du **`prototype`** d'un constructeur est 'héritée' par tous les objets ayant ce **`<prototype>`**

```
1  function Chat(nom, age, race, poids) {
2      this.nom = nom;
3      this.age = age;
4      this.race = race;
5      this.poids = poids;
6  }
7
8  Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11     console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maigrir = function (deltaPoids) {
15     this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19     console.log("cri de félicité " + felicite.cri);
20     console.log("cri de félix " + felix.cri);
21     felicite.miauler();
22     felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

felicite

```
▼ felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  ▶ <prototype>: {…}
```

propriété cri ?

felix

```
▼ felix: {…}
    age: 6
    nom: "Felix"
    poids: 3
    race: "siamois"
  ▶ <prototype>: {…}
```

propriété cri ?

Chat.prototype

```
▶ constructor:Chat(nom, age, race, poids)
  cri: "Miaou Miaou"
▶ maigrir:Chat.prototype.maigrir(deltaPoids)
▶ miauler:Chat.prototype.miauler()
▶ <prototype>: {…}
```

Object.prototype

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: »
▶ constructor:Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

```
cri de félicité Miaou Miaou
cri de félix Miaou Miaou
Félicité : Miaou Miaou
Felix : Miaou Miaou
```

les cris des objets sont *'hérités'* de leur objet <prototype>

de manière générale
évaluation de `obj.prop`

recherche de la propriété `prop` dans l'objet `obj` (propriété propre)

si elle n'est pas trouvée
    recherche de la `prop` dans la chaîne de <prototype> de `obj`
    si elle n'est pas trouvé → `undefined`

```
valeur(obj, prop)
    tantque (obj !== null && ! prop in obj)
        obj = <prototype> de obj
    fin tantque

    si (obj === null)
      → undefined

    sinon
      → valeur de prop pour obj
```

Voir le code

Fallback de propriétés

- Modification d'une propriété du **`prototype`** d'un constructeur s'applique à toutes les instances

```
1  function Chat(nom, age, race, poids) {
2      this.nom = nom;
3      this.age = age;
4      this.race = race;
5      this.poids = poids;
6  }
7
8  Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11     console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maigrir = function (deltaPoids) {
15     this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19     console.log("cri de félicité " + felicite.cri);
20     console.log("cri de félix " + felix.cri);
21     felicite.miauler();
22     felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

felicite

```
felicite: {…}
    age: 10
    nom: "Félicité"
    poids: 3.5
    race: "chat de gouttière"
  <prototype>: {…}
```

felix

```
felix: {…}
    age: 6
    nom: "Felix"
    poids: 3
    race: "siamois"
  <prototype>: {…}
```

Chat.prototype

```
constructor:Chat(nom, age, race, poids)
cri: "Meow !  Meow !"
maigrir:Chat.prototype.maigrir(deltaPoids)
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

Object.prototype

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

```
cri de félicité Miaou Miaou
cri de félix Miaou Miaou
Félicité : Miaou Miaou
Felix : Miaou Miaou
```

les cris des objets sont *'hérités'* de leur objet <prototype>

```
28
29 Chat.prototype.cri = "Meow !  Meow !";
30 afficherCris();
31
32 felicite.cri = "Miaou !  Miaou !";
33 afficherCris();
34
35 delete felicite.cri;
36 afficherCris();
37
38 delete felix.cri;
39 afficherCris();
```

```
cri de félicité Meow !  Meow !
cri de félix Meow !  Meow !
Félicité : Meow !  Meow !
Felix : Meow !  Meow !
```

les cris des objets sont *'hérités'* de leur objet <prototype>

Voir le code

Fallback de propriétés

- L'affectation d'une propriété héritée en passant par une instance modifie uniquement l'objet instance

```
1  function Chat(nom, age, race, poids) {
2      this.nom = nom;
3      this.age = age;
4      this.race = race;
5      this.poids = poids;
6  }
7
8  Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11     console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maigrir = function (deltaPoids) {
15     this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19     console.log("cri de félicité " + felicite.cri);
20     console.log("cri de félix " + felix.cri);
21     felicite.miauler();
22     felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

```
cri de félicité Miaou Miaou
cri de félix Miaou Miaou
Félicité : Miaou Miaou
Felix : Miaou Miaou
```

les cris des objets sont *'hérités'* de leur objet <prototype>

felicite

```
felicite: {…}
  age: 10
  cri: "Miaou !  Miaou !"
  nom: "Félicité"
  poids: 3.5
  race: "chat de gouttière"
  <prototype>: {…}
```

felix

```
felix: {…}
  age: 6
  nom: "Felix"
  poids: 3
  race: "siamois"
  <prototype>: {…}
```

Chat.prototype

```
constructor:Chat(nom, age, race, poids)
cri: "Meow !  Meow !"
maigrir:Chat.prototype.maigrir(deltaPoids)
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

Object.prototype

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

la propriété **cri** est redéfinie au niveau de l'instance **felicite** (ajout de la propriété si elle n'existe pas, modification si elle existe déjà)

```
28
29 Chat.prototype.cri = "Meow !  Meow !";
30 afficherCris();
31
32 felicite.cri = "Miaou !  Miaou !";
33 afficherCris();
34
35 delete felicite.cri;
36 afficherCris();
37
38 delete felix.cri;
39 afficherCris();
```

```
cri de félicité Meow !  Meow !
cri de félix Meow !  Meow !
Félicité : Meow !  Meow !
Felix : Meow !  Meow !
```

les cris des objets sont *'hérités'* de leur objet <prototype>

```
cri de félicité Miaou !  Miaou !
cri de félix Meow !  Meow !
Félicité : Miaou !  Miaou !
Felix : Meow !  Meow !
```

Les cris sont différents **felicite** : propriété propre (*own property*) **felix** : propriété héritée (*fallback*)

Voir le code

Fallback de propriétés

- La suppression d'une propriété en passant par une instance modifie uniquement l'objet instance

```
1  function Chat(nom, age, race, poids) {
2      this.nom = nom;
3      this.age = age;
4      this.race = race;
5      this.poids = poids;
6  }
7
8  Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11     console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maigrir = function (deltaPoids) {
15     this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19     console.log("cri de félicité " + felicite.cri);
20     console.log("cri de félix " + felix.cri);
21     felicite.miauler();
22     felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

```
cri de félicité Miaou Miaou
cri de félix Miaou Miaou
Félicité : Miaou Miaou
Felix : Miaou Miaou
```

les cris des objets sont *'hérités'* de leur objet <prototype>

felicite

```
felicite: {…}
  age: 10
  cri: "Miaou ! Miaou !"   (barré)
  nom: "Félicité"
  poids: 3.5
  race: "chat de gouttière"
  <prototype>: {…}
```

felix

```
felix: {…}
  age: 6
  nom: "Felix"
  poids: 3
  race: "siamois"
  <prototype>: {…}
```

Chat.prototype

```
constructor:Chat(nom, age, race, poids)
cri: "Miaou Miaou"
maigrir:Chat.prototype.maigrir(deltaPoids)
miauler:Chat.prototype.miauler()
<prototype>: {…}
```

Object.prototype

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toSource()
toString()
valueOf()
```

retrait de la propriété `cri` dans l'instance `felicite`

```
28
29 Chat.prototype.cri = "Meow !  Meow !";
30 afficherCris();
31
32 felicite.cri = "Miaou !  Miaou !";
33 afficherCris();
34
35 delete felicite.cri;
36 afficherCris();
37
38 delete felix.cri;
39 afficherCris();
```

```
cri de félicité Meow !  Meow !
cri de félix Meow !  Meow !
Félicité : Meow !  Meow !
Felix : Meow !  Meow !
```

les cris des objets sont *'hérités'* de leur objet <prototype>

```
cri de félicité Miaou !  Miaou !
cri de félix Meow !  Meow !
Félicité : Miaou !  Miaou !
Felix : Meow !  Meow !
```

Les cris sont différents
`felicite` : propriété propre (*own property*)
`felix` : propriété héritée (*fallback*)

```
cri de félicité Meow !  Meow !
cri de félix Meow !  Meow !
Félicité : Meow !  Meow !
Felix : Meow !  Meow !
```

La valeur de la propriété `cri` pour `felicite` est à nouveau la propriété héritée (*fallback*)

Voir le code

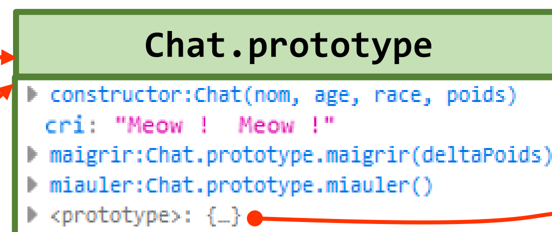
© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

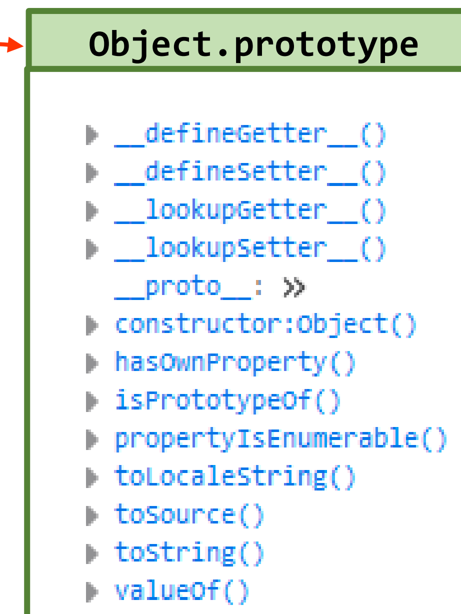
Chaîne de prototypes

- La suppression d'une propriété en passant par une instance modifie uniquement l'objet instance

```
1  function Chat(nom, age, race, poids) {
2      this.nom = nom;
3      this.age = age;
4      this.race = race;
5      this.poids = poids;
6  }
7
8  Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11     console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maigrir = function (deltaPoids) {
15     this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19     console.log("cri de félicité " + felicite.cri);
20     console.log("cri de félix " + felix.cri);
21     felicite.miauler();
22     felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

```
cri de félicité Miaou Miaou
cri de félix Miaou Miaou
Félicité : Miaou Miaou
Felix : Miaou Miaou
```






```
28
29 Chat.prototype.cri = "Meow !  Meow !";
30 afficherCris();
31
32 felicite.cri = "Miaou !  Miaou !";
33 afficherCris();
34
35 delete felicite.cri;
36 afficherCris();
37
38 delete felix.cri;
39 afficherCris();
```

```
cri de félicité Meow !  Meow !
cri de félix Meow !  Meow !
Félicité : Meow !  Meow !
Felix : Meow !  Meow !
```

```
cri de félicité Miaou !  Miaou !
cri de félix Meow !  Meow !
Félicité : Miaou !  Miaou !
Felix : Meow !  Meow !
```

```
cri de félicité Meow !  Meow !
cri de félix Meow !  Meow !
Félicité : Meow !  Meow !
Felix : Meow !  Meow !
```

```
cri de félicité Meow !  Meow !
cri de félix Meow !  Meow !
Félicité : Meow !  Meow !
Felix : Meow !  Meow !
```

suppression sans effet si l'instance n'a pas directement cette propriété (*own property*)

Voir le code

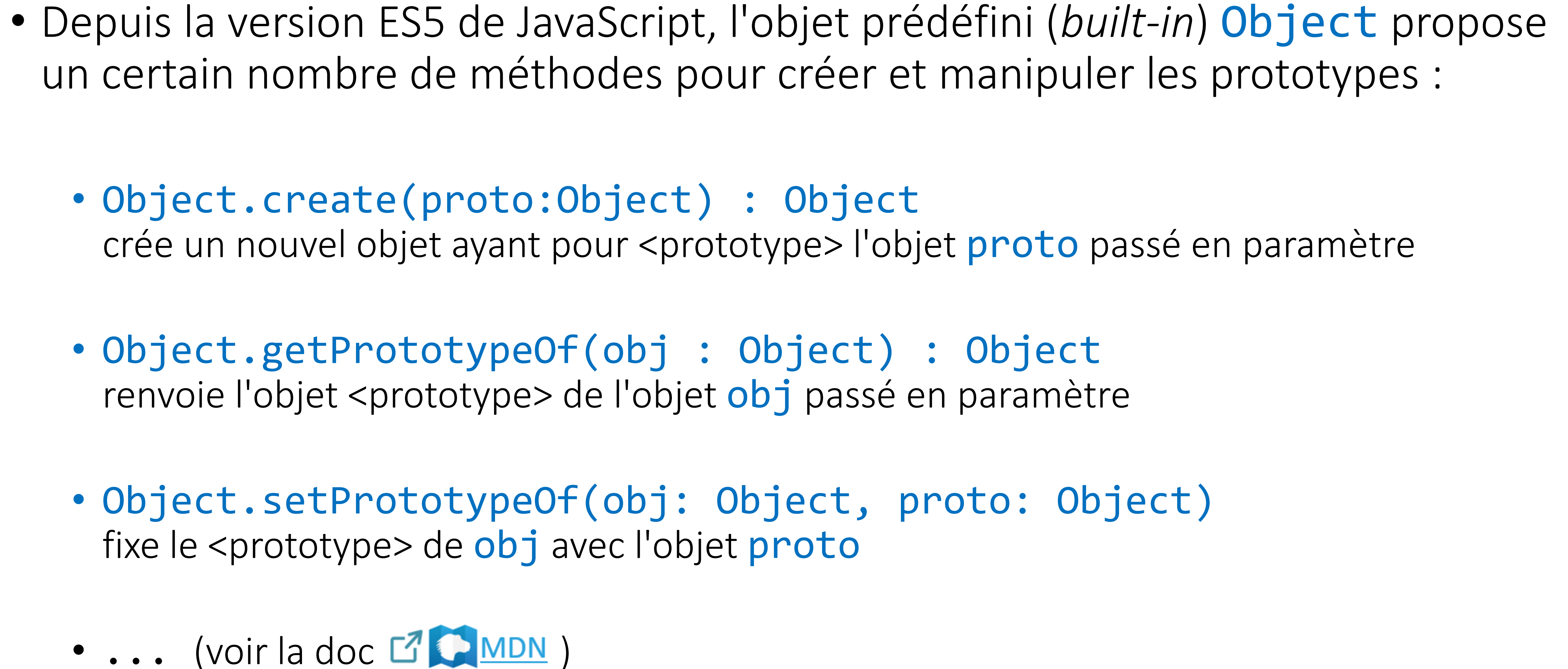
© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

Objets et ES5/ES6

- La création d'objets avec une fonction constructeur et l'opérateur `new` est historique, elle masque le mécanisme de prototypage souvent mal compris par les programmeurs.
- ES5 puis ES6 proposent de nouvelles manières de définir des objets (même si en interne le principe du prototypage est inchangé) afin de rendre l'utilisation des objets plus accessible aux développeurs:
  - `Object.create()` (ES5)
  - Classes (ES6)

© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

Objets et ES5 : `Object`



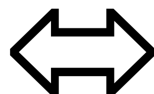
Objets et ES6 : Classes

- Avec l'introduction de `Class`, ES6 offre une syntaxe compacte pour définir des chaînes de prototypes et qui se rapproche de l'approche plus couramment utilisée dans les langages orientés objets (langages de classes comme Java)

avant ES6

```
function Chat(nom,age,race,poids) {
   this.nom = nom ;
   this.age = age ;
   this.race = race ;
   this.poids = poids;
}

Chat.prototype.miauler = function () {
    console.log(this.nom + "-> Miaou ! Miaou !");
};
Chat.prototype.maigrir = function(deltaPoids) {
      this.poids -= deltaPoids;
};
```

constructeur



méthodes

ES6 +

```
class Chat {
   constructor(nom,age,race,poids) {
      this.nom = nom ;
      this.age = age ;
      this.race = race ;
      this.poids = poids;
   }
   miauler() {
         console.log(this.nom + "-> Miaou ! Miaou !");
   }
   maigrir(deltaPoids) {
         this.poids -= deltaPoids;
   }
}
```

mot réservé pour identifier la fonction constructeur

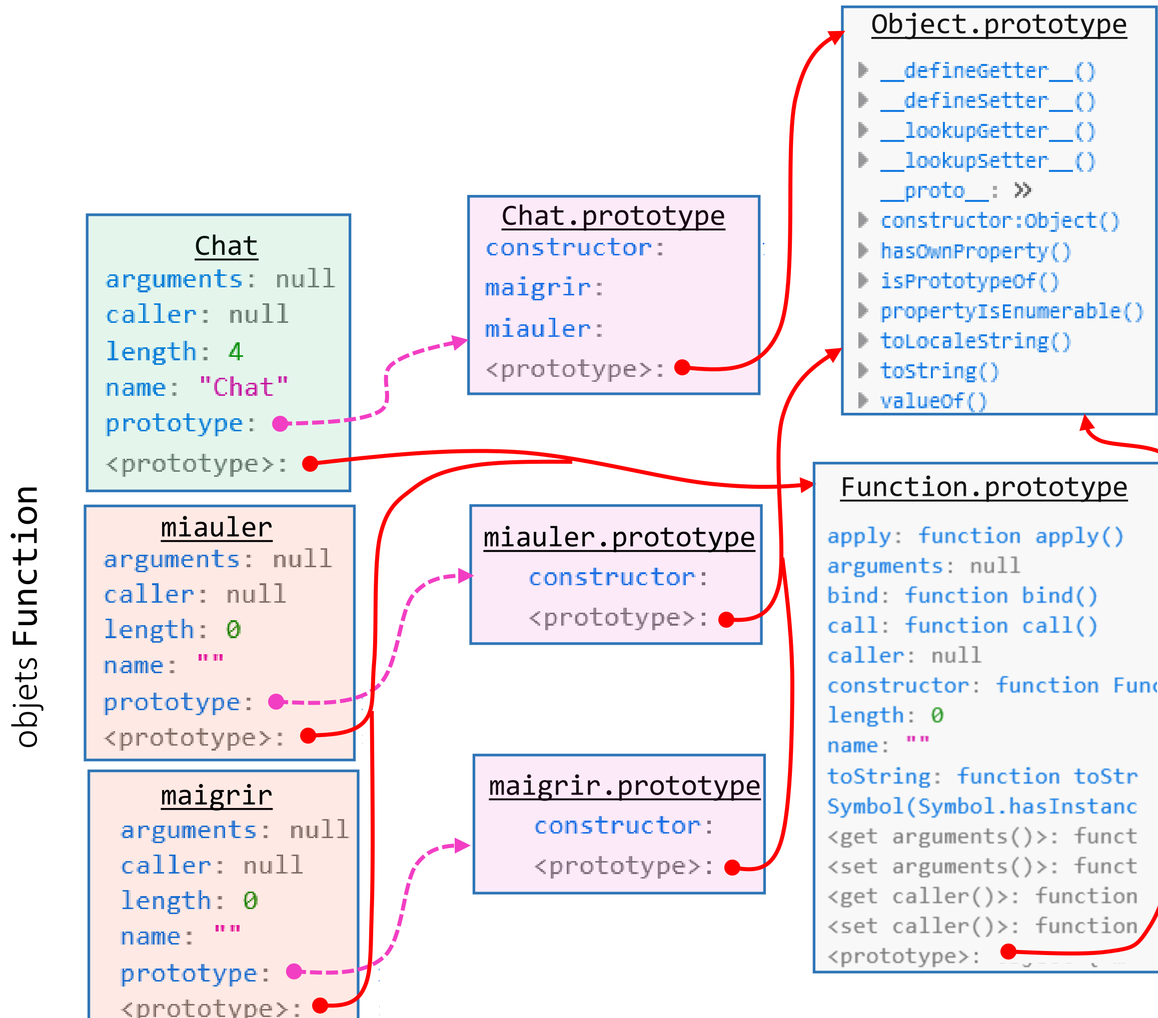
Une classe ne peut avoir qu'un seul constructeur

pas de **;** entre les déclarations de méthodes

[Voir le code]

Objets et ES6 : Classes

- Les objets créés par une déclaration de classe sont des objets fonction correspondant respectivement au constructeur et aux méthodes de la classe ainsi que les `prototypes` associés




```
class Chat {
    constructor(nom,age,race,poids) {
        this.nom = nom ;
        this.age = age ;
        this.race = race ;
        this.poids = poids;
    }
    miauler() {
        console.log(this.nom + "-> Miaou ! Miaou !");
    }
    maigrir(deltaPoids) {
        this.poids -= deltaPoids;
    }
}
```

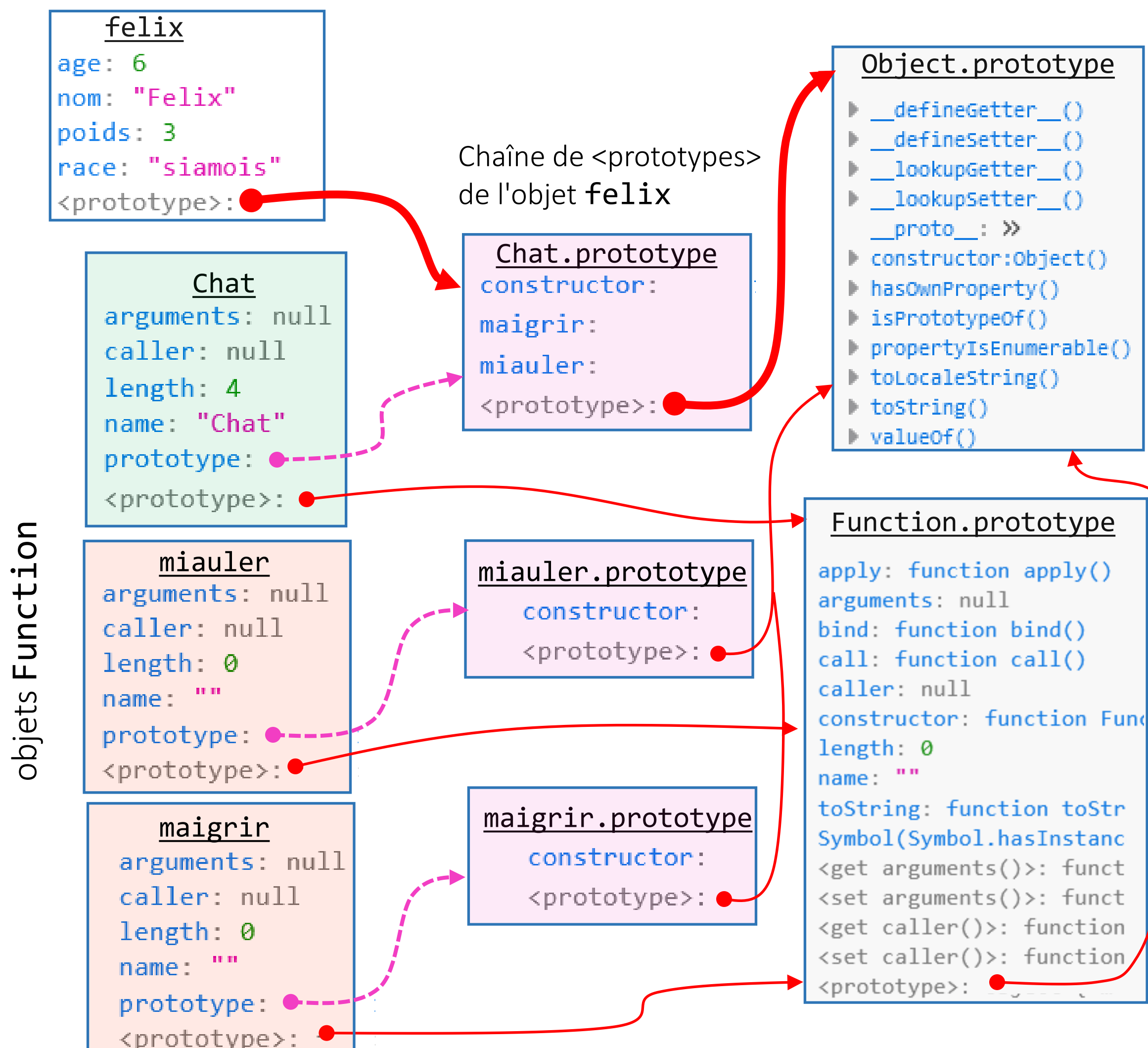
Voir le code

Objets et ES6 : Classes

- La création des objets se fait comme précédemment avec l'opérateur **new**

allocation mémoire — exécution de la fonction constructeur

```
let felix = new Chat("Felix",6,"siamois",3);
```




```
class Chat {
    constructor(nom,age,race,poids) {
        this.nom = nom ;
        this.age = age ;
        this.race = race ;
        this.poids = poids;
    }
    miauler() {
        console.log(this.nom + "-> Miaou ! Miaou !");
    }
    maigrir(deltaPoids) {
        this.poids -= deltaPoids;
    }
}
```

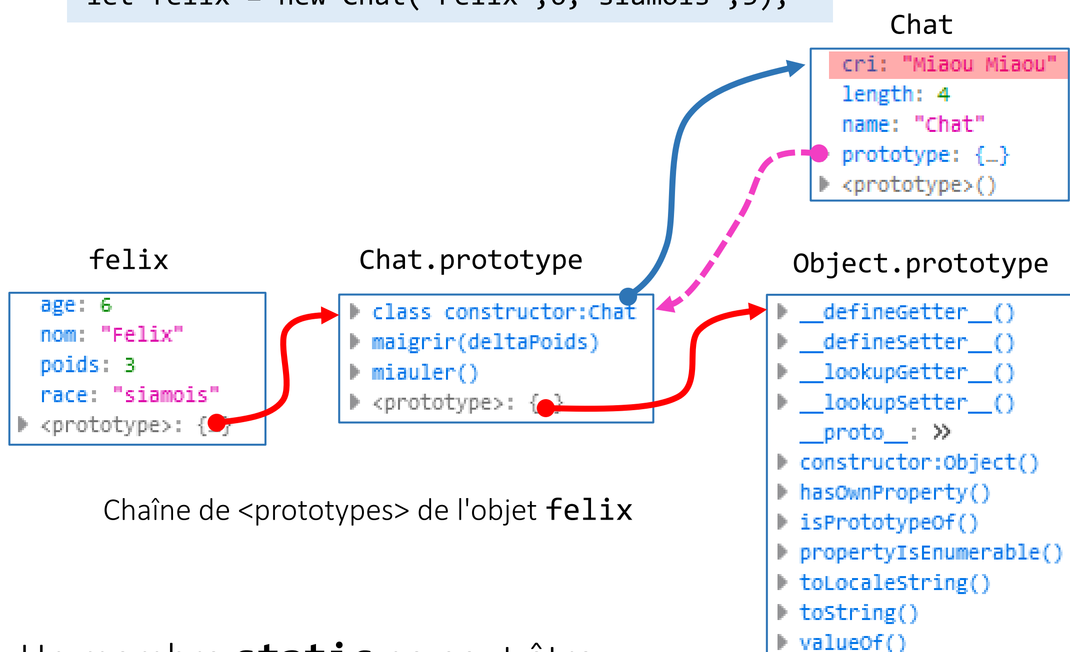
appelée immédiatement après la création d'une nouvelle instance pour l'initialiser. Cette fonction reçoit les arguments passés après le nom de la classe suivant l'opérateur **new**.

Voir le code

Objets et ES6 : Classes – membres statiques

- Propriétés **statiques**
  - Possibilité d'associer des **propriétés** à la classe directement
  - Déclaration de variable préfixée par le mot clé **`static`**
  - Rattaché à la fonction Constructeur et non pas au `prototype` de celle-ci

```
let felix = new Chat("Felix",6,"siamois",3);
```



Chaîne de <prototypes> de l'objet `felix`

```
class Chat {
  constructor(nom,age,race,poids) {
     this.nom = nom ;
     this.age = age ;
     this.race = race ;
     this.poids = poids;
  }
   static cri = "Miaou ! Miaou !";
   miauler() {
       console.log(this.nom + "-> " + Chat.cri);
   }
   maigrir(deltaPoids) {
      this.poids -= deltaPoids;
   }
}
```

[Voir le code]

- Un membre **`static`** ne peut être accédé via une instance de la classe (pas dans la chaîne de <prototypes>)

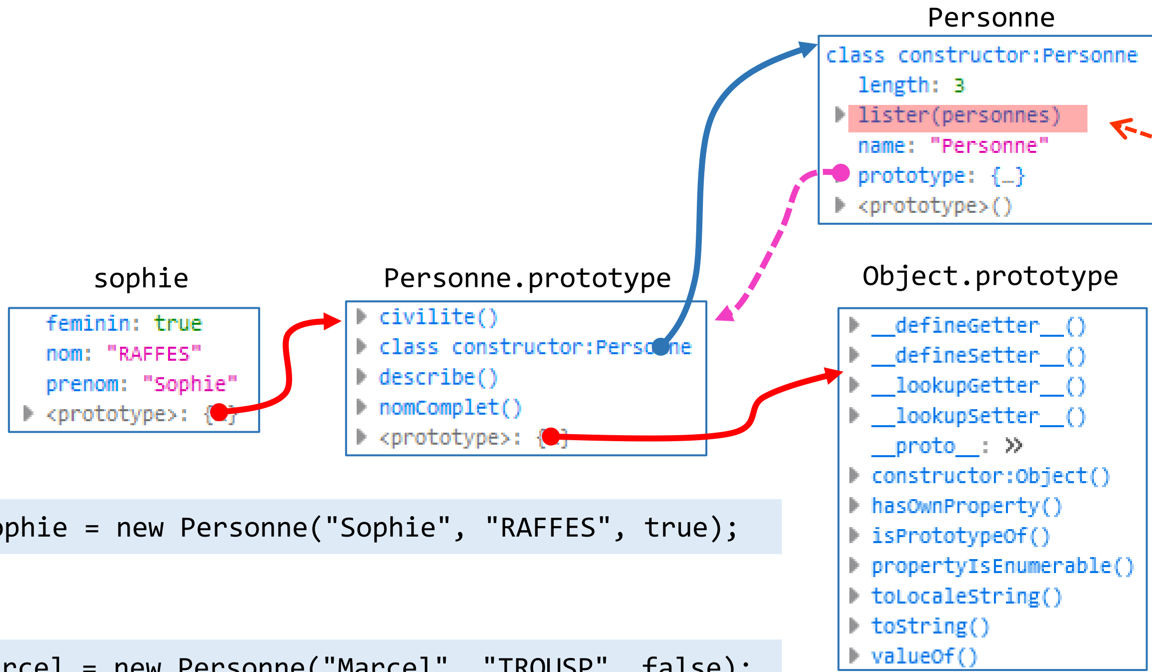
```
felix.cri  →  undefined
```

- Un membre **`static`** est accédé via l'identifiant de la classe

```
Chat.cri  →  "Miaou ! Miaou !
```

Objets et ES6 : Classes – membres statiques

- Méthodes **statiques**
  - déclaration de méthode préfixée par le mot clé **`static`**
  - fonction rattaché à la fonction constructeur et non pas au `prototype`




```
let marcel = new Personne("Marcel", "TROUSP", false);

Personne.lister([sophie, marcel]);
```

→
```
Personne
--------
Mme RAFFES Sophie
M. TROUSP Marcel
```

Pour invoquer la méthode statique il faut passer par l'objet classe (la fonction constructeur)

```
1
2  class Personne {
3      constructor(prenom, nom, feminin) {
4          this.prenom = prenom;
5          this.nom = nom;
6          this.feminin = feminin; // true si une femme, false sinon
7      }
8
9      nomComplet() {
10         return `${this.nom} ${this.prenom}`;
11     }
12
13     civilite() {
14         return (this.feminin) ? "Mme" : "M.";
15     }
16
17     describe() {
18         return `${this.civilite()} ${this.nomComplet()}`;
19     }
20
21     /**
22      * affiche la description des personnes contenues dans un tableau
23      * @param {Array[Personne]} personnes le tableau des personnes
24      */
25     static lister(personnes) {
26         console.log(this.name);
27         console.log("--------");
28         for (let i = 0; i < personnes.length; i++) {
29             console.log(personnes[i].describe());
30         }
31     }
32 }
```

Méthode statique se situe non pas au niveau du prototype pour les Personnes mais au niveau de la fonction constructeur **Personne**

Voir le code

© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

Objets et ES6 : Classes - *getters* et *setters*

- Accesseurs : fonctions *getters*
  - **get** permet de lier une propriété d'un objet à une fonction qui sera appelée lorsqu'on accédera à la propriété.

```
1
2  class Personne {
3      constructor(prenom, nom, feminin) {
4          this.prenom = prenom;
5          this.nom = nom;
6          this.feminin = feminin; // true si une femme, false sinon
7      }
8
9      nomComplet() {
10         return `${this.nom} ${this.prenom}`;
11     }
12
13     civilite() {
14         return (this.feminin) ? "Mme" : "M.";
15     }
16
17     describe() {
18         return `${this.civilite()} ${this.nomComplet()}`;
19     }
20
21     /**
22      * affiche la description des personnes contenues dans un tableau
23      * @param {Array[Personne]} personnes le tableau des personnes
24      */
25     static lister(personnes) {
26         console.log(this.name);
27         console.log("--------");
28         for (let i = 0; i < personnes.length; i++) {
29             console.log(personnes[i].describe());
30         }
31     }
32 }
```

```
let sophie = new Personne("Sophie", "RAFFES", true);
console.log(sophie.civilite()); → 'Mme'
console.log(sophie.nomComplet()); → 'RAFFES Sophie'
```

Méthodes transformées en getters

```
1
2  class Personne {
3      constructor(prenom, nom, feminin) {
4          this.prenom = prenom;
5          this.nom = nom;
6          this.feminin = feminin;
7      }
8
9      get nomComplet() {
10         return `${this.nom} ${this.prenom}`;
11     }
12
13     get civilite() {
14         return (this.feminin) ? "Mme" : "M.";
15     }
16
17     describe() {
18         return `${this.civilite} ${this.nomComplet}`;
19     }
20
21     /**
22      * affiche la description des personnes contenues dans un tableau
23      * @param {Array[Personne]} personnes le tableau des personnes
24      */
25     static lister(personnes) {
26         console.log(this.name);
27         console.log("--------");
28         for (let i = 0; i < personnes.length; i++) {
29             console.log(personnes[i].describe());
30         }
31     }
32 }
```

Voir le code

```
console.log(sophie.civilite);   → 'Mme'
console.log(sophie.nomComplet); → 'RAFFES Sophie'
```

Les getters peuvent être utilisés comme des propriétés

Objets et ES6 : Classes - *getters* et *setters*

- Mutateurs (fonctions *setters*)
  - **set** permet de lier une propriété d'un objet à une fonction qui sera appelée lorsqu'on affectera la propriété.

```
let p1 = new Personne("Sophie", "RAFFES", true);

console.log(p1.nomComplet);      → 'RAFFES Sophie'

p1.nomComplet = "  Maeva   MERLIN";

console.log(p1.prenom);          → 'Maeva'

console.log(p1.nom);             → 'MERLIN'
```

Appel du *setter*

```
1
2  class Personne {
3      constructor(prenom, nom, feminin) {
4          this.prenom = prenom;
5          this.nom = nom;
6          this.feminin = feminin;
7      }
8
9      get nomComplet() {
10         return `${this.nom} ${this.prenom}`;
11     }
12
13     set nomComplet(prenomNom) {
14         const tokens = prenomNom.split(/\b\s+(?!$)/);  // expression régulière*
15         this.prenom = tokens[0];
16         this.nom = tokens[1];
17     }
18
19     get civilite() {
20         return (this.feminin) ? "Mme" : "M.";
21     }
22
23     describe() {
24         return `${this.civilite} ${this.nomComplet}`;
25     }
26
27     /**
28      * affiche la description des personnes contenues dans un tableau
29      * @param {Array[Personne]} personnes le tableau des personnes
30      */
31     static lister(personnes) {
32         console.log(this.name);
33         console.log("--------");
34         for (let i = 0; i < personnes.length; i++) {
35             console.log(personnes[i].describe());
36         }
37     }
38 }
```

On peut aussi définir getters et setters dans des objets littéraux

```
let date =  {
    jour : '17',
    mois : 'Novembre',
    annee : '2020',
    get dateComplete() {
        return `${this.jour} ${this.mois} ${this.annee}`;
    },
    set dateComplete(dateString) {
        const tokens = dateString.split(/\b\s+(?!$)/);
        this.jour = parseInt(tokens[0]);
        this.mois = tokens[1];
        this.annee = parseInt(tokens[2]);
    }
};
```

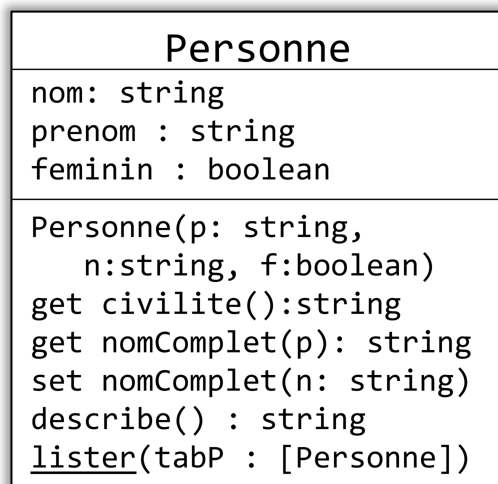
Voir le code

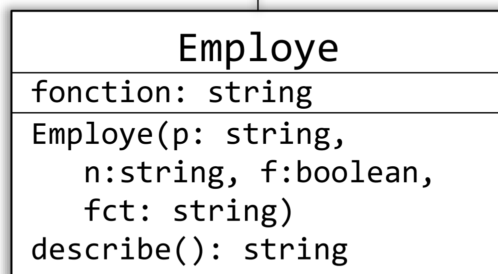
Voir le code

* https://blog.abelotech.com/posts/split-string-into-tokens-javascript/

Objets et ES6 : Classes - Héritage

- Héritage – possibilité de définir une classe comme étendant une classe existante (sous classe)





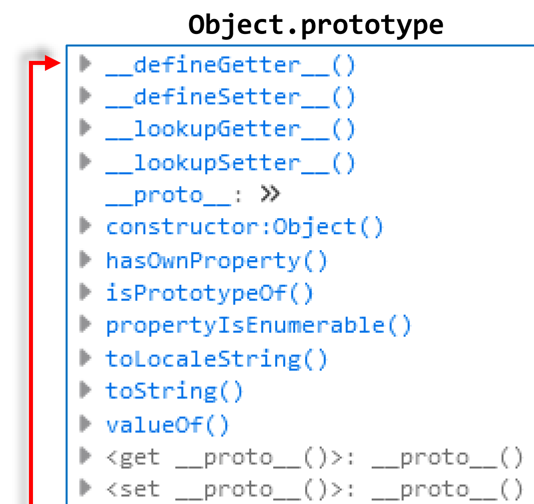
On veut exprimer le fait qu'un employé c'est une personne mais qui en plus a une fonction.

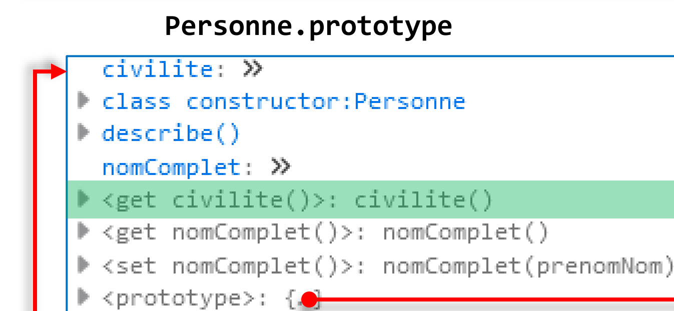
Possibilité d'obtenir ce comportement avec les chaînes de prototypes en JavaScript

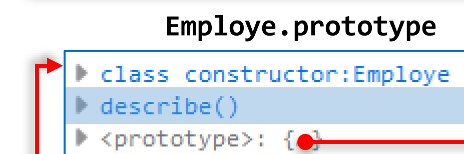
```
let p1 = new Employe("Sophie", "RAFFES", true, "Directrice");
```

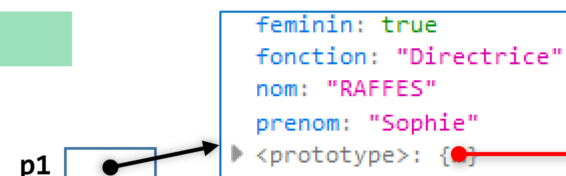
```
console.log(p1.describe()); → 'Mme RAFFES Sophie (Directrice)'
```

```
console.log(p1.civilite);   → 'Mme'
```









© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

Objets et ES6 : Classes - Héritage

- Héritage – **`extends`** permet d'exprimer cette relation d'héritage (et de mettre en place la chaîne de prototypes)

```
class Personne {
    constructor(prenom, nom, feminin) {
        this.prenom = prenom;
        this.nom = nom;
        this.feminin = feminin;
    }

    get nomComplet() {
        return `${this.nom} ${this.prenom}`;
    }

    set nomComplet(prenomNom) {
        const tokens = prenomNom.split(/\b\s+(?!$)/);
        this.prenom = tokens[0];
        this.nom = tokens[1];
    }

    get civilite() {
        return (this.feminin) ? "Mme" : "M.";
    }

    describe() {
        return `${this.civilite} ${this.nomComplet}`;
    }

    static lister(personnes) {
        console.log(this.name);
        console.log("--------");
        for (let i = 0; i < personnes.length; i++) {
            console.log(personnes[i].describe());
        }
    }
}
```

`super()` appelle le constructeur de la super classe.
Cet appel doit être fait avant d'accéder à `this`

```
class Employe extends Personne {

    constructor(prenom, nom, feminin, fonction) {
        super(prenom, nom, feminin);
        this.fonction = fonction;
    }

    describe() {
        return super.describe() + ` (${this.fonction})`;
    }
}
```

`super.nomMethode()` invoque une méthode héritée

```
let p1 = new Employe("Sophie", "RAFFES", true, "Directrice");

console.log(p1.describe()); → 'Mme RAFFES Sophie (Directrice)'

console.log(p1.civilite);   → 'Mme'
```

Voir le code

Object.prototype

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toString()
valueOf()
<get __proto__()>: __proto__()
<set __proto__()>: __proto__()
```

Personne.prototype

```
civilite: »
class constructor:Personne
describe()
nomComplet: »
<get civilite()>: civilite()
<get nomComplet()>: nomComplet()
<set nomComplet()>: nomComplet(prenomNom)
<prototype>: {…}
```

Employe.prototype

```
class constructor:Employe
describe()
<prototype>: {…}
```

```
feminin: true
fonction: "Directrice"
nom: "RAFFES"
prenom: "Sophie"
<prototype>: {…}
```

Objets et ES6 : Classes - Héritage

- Héritage – **`extends`** défini aussi une chaîne de <prototype> au niveau des classes (fonctions constructeurs) → héritage des membres statiques

```
class Personne {
    // ...
    static lister(personnes) {
        console.log(this.name);
        console.log("--------");
        for (let i = 0; i < personnes.length; i++) {
            console.log(personnes[i].describe());
        }
    }
}

class Employe extends Personne {
    // ...
}
```

```
let tabEmployes = [
  new Employe("Sophie", "RAFFES", true, "Directrice") ,
  new Employe("Marcel", "TROUSP", false, "Sous directeur")
];
```

`Personne.lister(tabEmployes);` →
```
Personne
--------
Mme RAFFES Sophie (Directrice)
M. TROUSP Marcel (Sous directeur)
```

`Employe.lister(tabEmployes);` →
```
Employe
--------
Mme RAFFES Sophie (Directrice)
M. TROUSP Marcel (Sous directeur)
```

Function.prototype
```
apply()
arguments: »
bind()
call()
caller: »
constructor:Function()
length: 0
<set caller()>: caller()
<prototype>: {…}
```

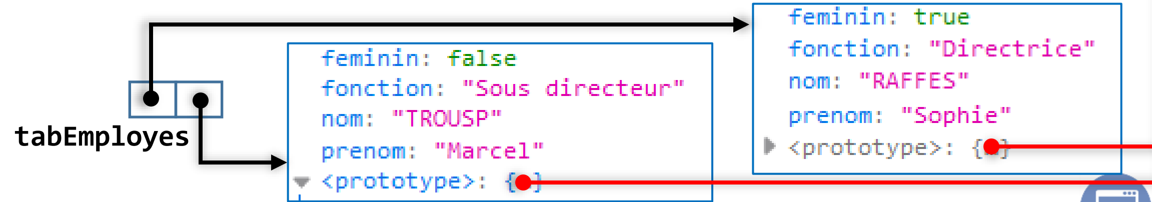
Object.prototype
```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: »
constructor:Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toString()
valueOf()
<get __proto__()>: __proto__()
<set __proto__()>: __proto__()
```

Personne (class)
```
length: 3
lister(personnes)
name: "Personne"
prototype: {…}
<prototype>()
```

Personne.prototype
```
civilite: »
class constructor:Personne
describe()
nomComplet: »
<get civilite()>: civilite()
<get nomComplet()>: nomComplet()
<set nomComplet()>: nomComplet(prenomNom)
<prototype>: {…}
```

Employe (class)
```
length: 4
name: "Employe"
prototype: {…}
class <prototype>:Personne
```

Employe.prototype
```
class constructor:Employe
describe()
<prototype>: {…}
```



Voir le code

© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2024/2025

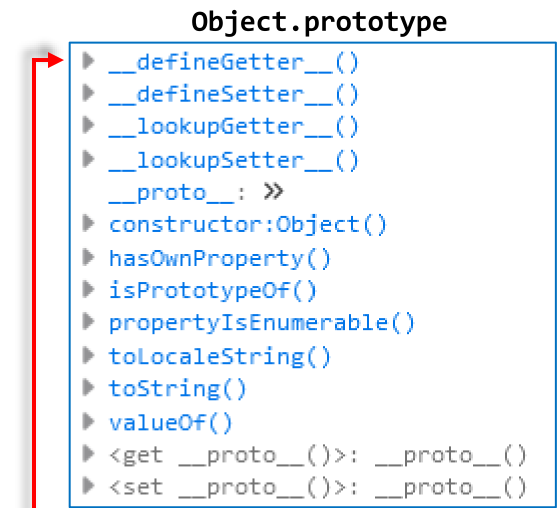
Objets : opérateur `instanceof`

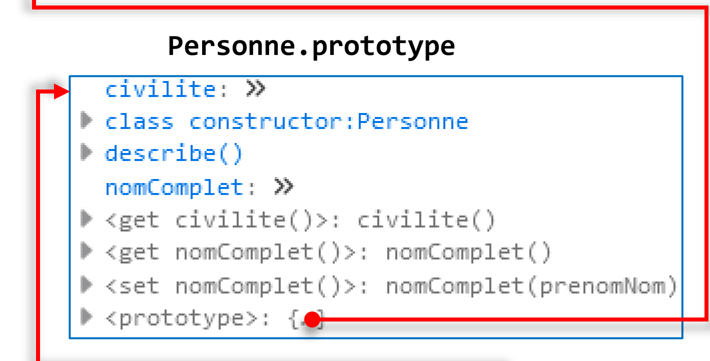
- `x instanceof C` renvoie `true` si l'objet référencé par `x` est instance de la classe `C` ou d'une sous classe de `C` (plus précisément si `C.prototype` se trouve dans la chaîne de `<prototypes>` de `x`), `false` sinon

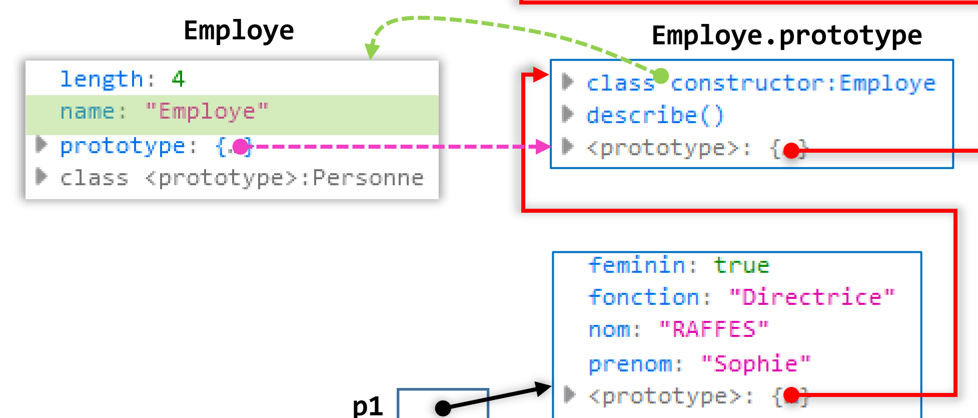
```
let p1 = new Employe("Sophie", "RAFFES", true, "Directrice");

p1 instanceof Employe     →  true

p1 instanceof Personne    →  true

p1 instanceof Object      →  true

let p2 = new Personne("Maeva", "FASFER", true);

p2 instanceof Employe     →  false

typeof p1 →  object
```

Nom de la classe qui a permis de créer p1

```
p1.constructor.name     →  Employe
```







Objets JavaScript

- Quelques liens

  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

  - https://www.w3schools.com/js/js_object_definition.asp
  - https://www.w3schools.com/js/js_class_intro.asp

  - http://blog.xebia.fr/2013/06/10/javascript-retour-aux-bases-constructeur-prototype-et-heritage/