

Раздел

[Прототипы, наследование](#)

Навигация по уроку


F.prototype по умолчанию,
свойство constructor

Итого

Задачи (2)

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠](#) → [Язык программирования JavaScript](#)
→ [Прототипы, наследование](#) 30-го ноября 2019

F.prototype

Как мы помним, новые объекты могут быть созданы с помощью функции-конструктора `new F()`.

Если в `F.prototype` содержится объект, оператор `new` устанавливает его в качестве `[[Prototype]]` для нового объекта.

На заметку:

JavaScript использовал прототипное наследование с момента своего появления. Это одна из основных особенностей языка.

Но раньше, в старые времена, прямого доступа к прототипу объекта не было. Надёжно работало только свойство `"prototype"` функции-конструктора, описанное в этой главе. Поэтому оно используется во многих скриптах.

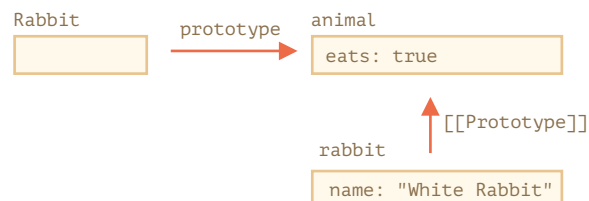
Обратите внимание, что `F.prototype` означает обычное свойство с именем `"prototype"` для `F`. Это ещё не «прототип объекта», а обычное свойство `F` с таким именем.

Приведём пример:

```
1 let animal = {
2   eats: true
3 };
4
5 function Rabbit(name) {
6   this.name = name;
7 }
8
9 Rabbit.prototype = animal;
10
11 let rabbit = new Rabbit("White Rabbit"); // rabbit.__p
12
13 alert( rabbit.eats ); // true
```

Установка `Rabbit.prototype = animal` буквально говорит интерпретатору следующее: "При создании объекта через `new Rabbit()` запиши ему `animal` в `[[Prototype]]`".

Результат будет выглядеть так:



На изображении: `"prototype"` – горизонтальная стрелка, обозначающая обычное свойство для `"F"`, а `[[Prototype]]` – вертикальная, обозначающая наследование `rabbit` от `animal`.

Раздел

[Прототипы, наследование](#)

Навигация по уроку

F.prototype по умолчанию,
свойство constructor

Итого

Задачи (2)

Комментарии

Поделиться



[Редактировать на GitHub](#)



F.prototype используется только в момент вызова new F()

F.prototype используется только при вызове new F() и присваивается в качестве свойства [[Prototype]] нового объекта. После этого F.prototype и новый объект ничего не связывает. Следует понимать это как «единоразовый подарок» объекту.

После создания F.prototype может измениться, и новые объекты, созданные с помощью new F(), будут иметь другой объект в качестве [[Prototype]], но уже существующие объекты сохраняют старый.

F.prototype по умолчанию, свойство constructor

У каждой функции по умолчанию уже есть свойство "prototype".

По умолчанию "prototype" – объект с единственным свойством constructor, которое ссылается на функцию-конструктор.

Вот такой:

```
1 function Rabbit() {}
2
3 /* прототип по умолчанию
4 Rabbit.prototype = { constructor: Rabbit };
5 */
```

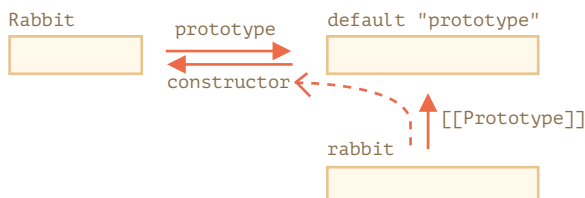


Проверим это:

```
1 function Rabbit() {}
2 // по умолчанию:
3 // Rabbit.prototype = { constructor: Rabbit }
4
5 alert( Rabbit.prototype.constructor == Rabbit ); // true
```

Соответственно, если мы ничего не меняем, то свойство constructor будет доступно всем кроликам через [[Prototype]]:

```
1 function Rabbit() {}
2 // по умолчанию:
3 // Rabbit.prototype = { constructor: Rabbit }
4
5 let rabbit = new Rabbit(); // наследует от {constructor
6
7 alert(rabbit.constructor == Rabbit); // true (свойство
```



Мы можем использовать свойство constructor существующего объекта для создания нового.

Пример:

```
1 function Rabbit(name) {
```

Раздел

[Прототипы, наследование](#)

Навигация по уроку

F.prototype по умолчанию,
свойство constructor

Итого

Задачи (2)

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
2   this.name = name;
3   alert(name);
4 }
5
6 let rabbit = new Rabbit("White Rabbit");
7
8 let rabbit2 = new rabbit.constructor("Black Rabbit");
```

Это удобно, когда у нас есть объект, но мы не знаем, какой конструктор использовался для его создания (например, он мог быть взят из сторонней библиотеки), а нам необходимо создать ещё один такой объект.

Но, пожалуй, самое важное о свойстве "constructor" это то, что...

...JavaScript сам по себе не гарантирует правильное значение свойства "constructor".

Да, оно является свойством по умолчанию в "prototype" у функций, но что случится с ним позже – зависит только от нас.

В частности, если мы заменим прототип по умолчанию на другой объект, то свойства "constructor" в нём не будет.

Например:

```
1 function Rabbit() {}
2 Rabbit.prototype = {
3   jumps: true
4 };
5
6 let rabbit = new Rabbit();
7 alert(rabbit.constructor === Rabbit); // false
```

Таким образом, чтобы сохранить верное свойство "constructor", мы должны добавлять/удалять/изменять свойства у прототипа по умолчанию вместо того, чтобы перезаписывать его целиком:

```
1 function Rabbit() {}
2
3 // Не перезаписываем Rabbit.prototype полностью,
4 // а добавляем к нему свойство
5 Rabbit.prototype.jumps = true
6 // Прототип по умолчанию сохраняется, и мы всё ещё имеем
```

Или мы можем заново создать свойство constructor:

```
1 Rabbit.prototype = {
2   jumps: true,
3   constructor: Rabbit
4 };
5
6 // теперь свойство constructor снова корректное, так как
```

Итого

В этой главе мы кратко описали способ задания [[Prototype]] для объектов, создаваемых с помощью функции-конструктора. Позже мы рассмотрим, как можно использовать эту возможность.

Всё достаточно просто. Выделим основные моменты:

- Свойство F.prototype (не путать с [[Prototype]]) устанавливает [[Prototype]] для новых объектов при вызове new F().
- Значение F.prototype должно быть либо объектом, либо null. Другие значения не будут работать.
- Свойство "prototype" является особым, только когда оно назначено функции-конструктору, которая вызывается оператором new.

В обычных объектах prototype не является чем-то особенным:

```
1 let user = {  
2   name: "John",  
3   prototype: "Bla-bla" // никакой магии нет - обычное с  
4 };
```

По умолчанию все функции имеют `F.prototype = { constructor: F }`, поэтому мы можем получить конструктор объекта через свойство `"constructor"`.

✓ Задачи

Изменяем "prototype"

важность: 5

В коде ниже мы создаём нового кролика `new Rabbit`, а потом пытаемся изменить его прототип.

Сначала у нас есть такой код:

```
1 function Rabbit() {}  
2 Rabbit.prototype = {  
3   eats: true  
4 };  
5  
6 let rabbit = new Rabbit();  
7  
8 alert( rabbit.eats ); // true
```

1.

Добавим одну строчку (выделенную в коде ниже). Что вызов `alert` покажет нам сейчас?

```
1 function Rabbit() {}  
2 Rabbit.prototype = {  
3   eats: true  
4 };  
5  
6 let rabbit = new Rabbit();  
7  
8 Rabbit.prototype = {};  
9  
10 alert( rabbit.eats ); // ?
```

2.

...А если код такой (заменяли одну строчку)?

```
1 function Rabbit() {}  
2 Rabbit.prototype = {  
3   eats: true  
4 };  
5  
6 let rabbit = new Rabbit();  
7  
8 Rabbit.prototype.eats = false;  
9  
10 alert( rabbit.eats ); // ?
```

3.

Или такой (заменяли одну строчку)?

Раздел

[Прототипы, наследование](#)

Навигация по уроку

F.prototype по умолчанию,
свойство constructor

Итого

Задачи (2)

Комментарии

Поделиться



[Редактировать на GitHub](#)

Раздел

[Прототипы, наследование](#)

Навигация по уроку

F.prototype по умолчанию,
свойство constructor

Итого

Задачи (2)

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
1 function Rabbit() {}
2 Rabbit.prototype = {
3   eats: true
4 };
5
6 let rabbit = new Rabbit();
7
8 delete rabbit.eats;
9
10 alert( rabbit.eats ); // ?
```

4.

Или, наконец, такой:

```
1 function Rabbit() {}
2 Rabbit.prototype = {
3   eats: true
4 };
5
6 let rabbit = new Rabbit();
7
8 delete Rabbit.prototype.eats;
9
10 alert( rabbit.eats ); // ?
```

решение

Создайте новый объект с помощью уже существующего

важность: 5



Представьте, что у нас имеется некий объект `obj`, созданный функцией-конструктором – мы не знаем какой именно, но хотелось бы создать ещё один объект такого же типа.



Можем ли мы сделать так?

```
1 let obj2 = new obj.constructor();
```

Приведите пример функции-конструктора для объекта `obj`, с которой такой вызов корректно сработает. И пример функции-конструктора, с которой такой код поведёт себя неправильно.

решение

Проводим [курсы по JavaScript и фреймворкам](#).



Комментарии

перед тем как писать...