

Раздел

[Продвинутая работа с функциями](#)

Навигация по уроку


Синтаксис

Замыкание

Итого

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠 → Язык программирования JavaScript](#)
[→ Продвинутая работа с функциями](#) 29-го августа 2019

Синтаксис "new Function"

Существует ещё один вариант объявлять функции. Он используется крайне редко, но иногда другого решения не найти.

Синтаксис

Синтаксис для объявления функции:

```
1 let func = new Function([arg1, arg2, ...argN], function()
```

Функция создаётся с заданными аргументами `arg1...argN` и телом `functionBody`.

Это проще понять на конкретном примере. Здесь объявлена функция с двумя аргументами:

```
1 let sum = new Function('a', 'b', 'return a + b');  
2  
3 alert( sum(1, 2) ); // 3
```

А вот функция без аргументов, в этом случае достаточно указать только тело:



```
1 let sayHi = new Function('alert("Hello")');  
2  
3 sayHi(); // Hello
```



Главное отличие от других способов объявления функции, которые были рассмотрены ранее, заключается в том, что функция создаётся полностью «на лету» из строки, переданной во время выполнения.

Все предыдущие объявления требовали от нас, программистов, писать объявление функции в скрипте.

Но `new Function` позволяет превратить любую строку в функцию. Например, можно получить новую функцию с сервера и затем выполнить её:

```
1 let str = ... код, полученный с сервера динамически ...  
2  
3 let func = new Function(str);  
4 func();
```

Это используется в очень специфических случаях, например, когда мы получаем код с сервера для динамической компиляции функции из шаблона, в сложных веб-приложениях.

Замыкание

Обычно функция запоминает, где родилась, в специальном свойстве `[[Environment]]`. Это ссылка на лексическое окружение (Lexical Environment), в котором она создана (мы разбирали это в главе [Замыкание](#)).

Но когда функция создаётся с использованием `new Function`, в её `[[Environment]]` записывается ссылка не на внешнее лексическое окружение, в котором она была создана, а на глобальное. Поэтому такая функция имеет доступ только к глобальным переменным.

Раздел

[Продвинутая работа с функциями](#)

Навигация по уроку

Синтаксис

Замыкание

Итого

Комментарии

Поделиться



Редактировать на GitHub



```
1 function getFunc() {
2   let value = "test";
3
4   let func = new Function('alert(value)');
5
6   return func;
7 }
8
9 getFunc(); // ошибка: value не определено
```

Сравним это с обычным объявлением:

```
1 function getFunc() {
2   let value = "test";
3
4   let func = function() { alert(value); };
5
6   return func;
7 }
8
9 getFunc(); // "test", из лексического окружения функц
```

Эта особенность `new Function` выглядит странно, но оказывается очень полезной на практике.

Представьте, что нужно создать функцию из строки. Код этой функции неизвестен во время написания скрипта (поэтому не используем обычные функции), а будет определён только в процессе выполнения. Мы можем получить код с сервера или с другого ресурса.

Наша новая функция должна взаимодействовать с основным скриптом.

Что если бы она имела доступ к внешним переменным?

Проблема в том, что перед отправкой JavaScript-кода на реальные работающие проекты код сжимается с помощью *минификатора* – специальной программы, которая уменьшает размер кода, удаляя комментарии, лишние пробелы, и, что самое главное, локальным переменным даются укороченные имена.

Например, если в функции объявляется переменная `let userName`, то минификатор изменяет её на `let a` (или другую букву, если она не занята) и изменяет её везде. Обычно так делать безопасно, потому что переменная является локальной, и никто снаружи не имеет к ней доступ. И внутри функции минификатор заменяет каждое её упоминание. Минификаторы достаточно умные. Они не просто осуществляют «тупой» поиск-замену, они анализируют структуру кода, и поэтому ничего не ломается.

Так что если бы даже `new Function` и имела доступ к внешним переменным, она не смогла бы найти переименованную `userName`.

Если бы `new Function` имела доступ к внешним переменным, при этом были бы проблемы с минификаторами.

Кроме того, такой код был бы архитектурно хуже и более подвержен ошибкам.

Чтобы передать что-то в функцию, созданную как `new Function`, можно использовать её аргументы.

Итого

Синтаксис:

```
1 let func = new Function ([arg1, arg2, ...argN], functio
```

По историческим причинам аргументы также могут быть объявлены через запятую в одной строке.

Эти 3 объявления ниже эквивалентны:

Раздел

[Продвинутая работа с функциями](#)

Навигация по уроку

Синтаксис

Замыкание

Итого

Комментарии

Поделиться



Редактировать на GitHub



```
1 new Function('a', 'b', 'return a + b'); // стандартный
2 new Function('a,b', 'return a + b'); // через запятую в
3 new Function('a , b', 'return a + b'); // через запятую
```

Функции, объявленные через `new Function`, имеют `[[Environment]]`, ссылающийся на глобальное лексическое окружение, а не на родительское. Поэтому они не могут использовать внешние локальные переменные. Но это очень хорошо, потому что страхует нас от ошибок. Переданные явно параметры – гораздо лучшее архитектурное решение, которое не вызывает проблем у минификаторов.

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

Комментарии

перед тем как писать...

