this, контекст, каррирование

Ключевое слово this

Пусть у нас есть какая-то функция func, внутри которой используется this:

```
function func() {
	alert(this.value);
}
```

На что указывает this в этой функции? А мы не знаем. И JavaScript не знает. И сама функция не знает. То есть: в момент создания функции на что именно указывает this не определено. И определится это только тогда, когда эта функция будет вызвана.

Давайте теперь сделаем инпут и привяжем к нему нашу функцию func. Теперь this указывает на наш инпут:

```
<input id="elem" value="привет">
var elem = document.getElementById('elem');
elem.addEventListener('blur', func);
function func() {
	alert(this.value); //выведет 'привет'
}
```

Но ведь у нас может быть не один инпут, а несколько, и каждому мы можем привязать нашу функцию func. В этом случае для первого элемента this в функции будет указывать на него, а для второго - на него:

```
<input id="elem1">
<input id="elem2">
var elem1 = document.getElementById('elem1');
elem1.addEventListener('blur', func); //тут this - это первый элемент

var elem2 = document.getElementById('elem2');
elem2.addEventListener('blur', func); //тут this - это второй элемент

function func() {
	alert(this.value); //а тут this не еще определен
}
```

А что будет, если в функции указать this, но не привязать ее ни к какому элементу, вот так:

```
function func() {
	console.log(this);
}

func();
```

В этом случае в новом стандарте JavaScript там будет лежать **undefined**, а в старом - объект **window**.

Из сказанного выше может показаться, что в строгом режиме this всегда будет undefined. Как бы не так! Если просто вывести this вне функции, то в нем будет ссылка на window независимо от режима:

```
"use strict";
console.log(this); // в this ссылка на window
```

Если у нас **функция в функции** или используется **setInterval**, то тогда нужно ввести переменную, чтобы правильно использовать this во внутренней функции

```
var self = this;
```

http://old.code.mu/books/javascript/context/prodvinutaya-rabota-s-kontekstom-v-javascript.html

Метод call

Итак, мы разобрали, как на самом деле работает this. Давайте теперь рассмотрим методы, которые позволяют принудительно указать, в каком контексте вызывается функция (*то есть принудительно сказать, чему равен this*).

```
input id="elem" value="привет">
var elem = document.getElementById('elem');

function func() {
	alert(this.value);
}
func.call(elem); //выведет value инпута
```

Пусть теперь функция func принимает некоторые параметры, назовем их param1 и param2:

```
function func(param1, param2) {
	alert(this.value + param1 + param2);
}
```

При вызове функции через call можно передать эти параметры вот так:

```
func.call(elem, param1, param2);
```

Метод apply

Кроме метода call, существует очень похожий метод apply - разница в способе передачи параметров. Следующие две записи эквивалентны:

```
func.call(elem, param1, param2);
func.apply(elem, [param1, param2]);
```

То есть в apply параметры передаются в виде массива, а не перечисляются через запятую, как в методе call. Вот и все отличие. В зависимости от задачи бывает удобен то один, то другой метод.

Метод bind

Следующий метод bind позволяет привязать контекст к функции навсегда. Он вызывается вот так: func.bind(elem), но результатом работы будет не результат функции func, а новая функция, которая такая же, как и func, но у нее this всегда указывает на elem.

```
<input id="elem" value="привет">
var elem = document.getElementById('elem');

function func(param1, param2) {
	alert(this.value + param1 + param2);
}

var newFunc = func.bind(elem);
newFunc('!', '?'); //выведет 'привет!?'
```

Не обязательно записывать результат работы bind в новую функцию newFunc, можно просто перезаписать func:

```
var func = func.bind(elem);
```

Теперь func - такая же функция, как и была, но с жестко связанным this.