JavaScript
<<Closure, Context>>

Замыкание (closure)

Замыкание (**closure**) в программировании — функция первого класса, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся ее параметрами. Говоря другим языком, замыкание — функция, которая ссылается на свободные переменные в своей области видимости.

Замыкание, так же как и экземпляр объекта, есть способ представления функциональности и данных, связанных и упакованных вместе.

Замыкание — это особый вид функции. Она определена в теле другой функции и создаётся каждый раз во время ее выполнения. Синтаксически это выглядит как функция, находящаяся целиком в теле другой функции. При этом вложенная внутренняя функция содержит ссылки на локальные переменные внешней функции. Каждый раз при выполнении внешней функции происходит создание нового экземпляра внутренней функции, с новыми ссылками на переменные внешней функции.

```
function testClosure(){
    var div = document.querySelector('#text');
    div.onclick = function (){
        div.style.color = 'red'; // <-- closure
    }
}
```

Контекст выполнения (this)

Контекст выполнения (**execution context**) — концепция, описывающая окружение, в котором производится выполнение кода на **JavaScript**. Код всегда выполняется внутри некоего контекста.

Типы контекста:
– **Глобальный контекст выполнения**. Это базовый, используемый по умолчанию контекст выполнения. Если некий код находится не внутри какой-нибудь функции, значит этот код принадлежит глобальному контексту. Глобальный контекст характеризуется наличием глобального объекта, которым, в случае с браузером, является объект **window**, и тем, что ключевое слово **this** указывает на этот глобальный объект. В программе может быть лишь один глобальный контекст.

- **Контекст выполнения функции**. Каждый раз, когда вызывается функция, для неё создается новый контекст. Каждая функция имеет собственный контекст выполнения. В программе может одновременно присутствовать множество контекстов выполнения функций.

- **Контекст выполнения функции eval**. Код, выполняемый внутри функции eval, также имеет собственный контекст выполнения.

Пример контекста исполнения

Предположим что у нас есть функция задача которой добавить или удалить класс у любого **DOM элемента**. Чтобы написать такую универсальную функцию нам необходимо обратиться к ключевому слову **this**

```
function addActive(){
    console.log(this);
    this.classList.toggle('item-active');
}
```

Если мы вызовем эту функцию просто так то по умолчанию эта функция вызовется в рамках глобального контекста и ключевое слово **this** будет указывать на **window**

```
addActive();
```

Но если мы повесим эту функцию на какое либо событие элемента то **this** будет указывать на элемент в рамках которого она будет вызываться (в нашем примере на элемент с классом **item**)

```
var itemList = document.querySelectorAll('.items .item');
for(var i = 0; i < itemList.length; i++){
    itemList[i].onclick = addActive;
}
```

Изменение контекста исполнения

Часто нам приходится изменять контекст исполнения когда мы используем функции которые были написаны не нами допустим функции которые встроены в объект **window**, они всегда будут вызываться в глобальном контексте.

Для изменения контекста исполнения у каждой функции **JavaScript** существует две функции **call(context, arg1,arg2,...)** и **apply(context,[arg1,arg2,..])**.

Обе функции первым параметром принимают контекст в рамках которого должна вызываться (на что будет указывать **this** внутри функции), а отличаются они вторым параметром, **call** - принимает список аргументов перечисленных через запятую, **apply** - принимает массив аргументов

```
var items = document.querySelectorAll('.items .item');
function activeItem() {
   this.classList.toggle('item-active');
}
setInterval(function () {
   var rnd = Math.floor(Math.random()*items.length);
   activeItem.call(items[rnd]);//<-- будет вызвана в контексте items[rnd]
},500);
```