**Локальные переменные для объекта**

Замыкания можно использовать сотнями способов. Иногда люди сами не замечают, что использовали замыкания – настолько это просто и естественно.

В этой главе мы рассмотрим дополнительные примеры использования замыканий и задачи на эту тему.

**[Счётчик-объект](http://learn.javascript.ru/closures-usage" \l "schyotchik-obekt)**

Ранее мы сделали счётчик.

Напомню, как он выглядел:

function makeCounter() {

var currentCount = 1;

return function() {

return currentCount++;

};

}

var counter = makeCounter();

// каждый вызов возвращает результат, увеличивая счётчик

alert( counter() ); // 1

alert( counter() ); // 2

alert( counter() ); // 3

Счётчик получился вполне рабочий, но вот только возможностей ему не хватает. Хорошо бы, чтобы можно было сбрасывать значение счётчика или начинать отсчёт с другого значения вместо 1 или… Да много чего можно захотеть от простого счётчика и, тем более, в более сложных проектах.

**Чтобы добавить счётчику возможностей – перейдём с функции на полноценный объект:**

function makeCounter() {

var currentCount = 1;

return { // возвратим объект вместо функции

getNext: function() {

return currentCount++;

},

set: function(value) {

currentCount = value;

},

reset: function() {

currentCount = 1;

}

};

}

var counter = makeCounter();

alert( counter.getNext() ); // 1

alert( counter.getNext() ); // 2

counter.set(5);

alert( counter.getNext() ); // 5

Теперь функция makeCounter возвращает не одну функцию, а объект с несколькими методами:

* getNext() – получить следующее значение, то, что раньше делал вызов counter().
* set(value) – поставить значение.
* reset() – обнулить счётчик.

Все они получают ссылку [[Scope]] на текущий (внешний) объект переменных. Поэтому вызов любого из этих методов будет получать или модифицировать одно и то же внешнее значение currentCount.

## [Объект счётчика + функция](http://learn.javascript.ru/closures-usage" \l "obekt-schyotchika-funktsiya)

Изначально, счётчик делался функцией во многом ради красивого вызова: counter(), который увеличивал значение и возвращал результат.

К сожалению, при переходе на объект короткий вызов пропал, вместо него теперь counter.getNext(). Но он ведь был таким простым и удобным…

Поэтому давайте вернём его!

function makeCounter() {

var currentCount = 1;

// возвращаемся к функции

function counter() {

return currentCount++;

}

// ...и добавляем ей методы!

counter.set = function(value) {

currentCount = value;

};

counter.reset = function() {

currentCount = 1;

};

return counter;

}

var counter = makeCounter();

alert( counter() ); // 1

alert( counter() ); // 2

counter.set(5);

alert( counter() ); // 5

Красиво, не правда ли? Получился полноценный объект, который можно вдобавок ещё и вызывать.

Этот трюк часто используется при разработке JavaScript-библиотек. Например, популярная библиотека [jQuery](http://jquery.com/) предоставляет глобальную переменную с именем [jQuery](http://api.jquery.com/jQuery/) (доступна также под коротким именем $), которая с одной стороны является функцией и может вызываться как jQuery(...), а с другой – у неё есть различные методы, например jQuery.type(123) возвращает тип аргумента.

Далее вы найдёте различные задачи на понимание замыканий. Рекомендуется их сделать самостоятельно.