**Статические и фабричные методы**

Методы и свойства, которые не привязаны к конкретному экземпляру объекта, называют «статическими». Их записывают прямо в саму функцию-конструктор.

**[Статические свойства](http://learn.javascript.ru/static-properties-and-methods" \l "staticheskie-svoystva)**

В коде ниже используются статические свойства Article.count и Article.DEFAULT\_FORMAT:

function Article() {

Article.count++;

}

Article.count = 0; // статическое свойство-переменная

Article.DEFAULT\_FORMAT = "html"; // статическое свойство-константа

Они хранят данные, специфичные не для одного объекта, а для всех статей целиком.

Как правило, это чаще константы, такие как формат «по умолчанию» Article.DEFAULT\_FORMAT.

**[Статические методы](http://learn.javascript.ru/static-properties-and-methods" \l "staticheskie-metody)**

С примерами статических методов мы уже знакомы: это встроенные методы [String.fromCharCode](http://javascript.ru/String.fromCharCode), [Date.parse](http://javascript.ru/Date.parse).

Создадим для Article статический метод Article.showCount():

function Article() {

Article.count++;

//...

}

Article.count = 0;

Article.showCount = function() {

alert( this.count ); // (1)

}

// использование

new Article();

new Article();

Article.showCount(); // (2)

Здесь Article.count – статическое свойство, а Article.showCount – статический метод.

Обратим внимание на использование this в примере выше. Несмотря на то, что переменная и метод – статические, он всё ещё полезен. В строке (1) он равен Article.

**[Пример: сравнение объектов](http://learn.javascript.ru/static-properties-and-methods" \l "primer-sravnenie-obektov)**

Ещё один хороший способ применения – сравнение объектов.

Например, у нас есть объект Journal для журналов. Журналы можно сравнивать – по толщине, по весу, по другим параметрам.

Объявим «стандартную» функцию сравнения, которая будет сравнивать по дате издания. Эта функция сравнения, естественно, не привязана к конкретному журналу, но относится к журналам вообще.

Поэтому зададим её как статический метод Journal.compare:

function Journal(date) {

this.date = date;

// ...

}

// возвращает значение, большее 0, если A больше B, иначе меньшее 0

Journal.compare = function(journalA, journalB) {

return journalA.date - journalB.date;

};

В примере ниже эта функция используется для поиска самого раннего журнала из массива:

function Journal(date) {

this.date = date;

this.formatDate = function(date) {

return date.getDate() + '.' + (date.getMonth() + 1) + '.' + date.getFullYear();

};

this.getTitle = function() {

return "Выпуск от " + this.formatDate(this.date);

};

}

Journal.compare = function(journalA, journalB) {

return journalA.date - journalB.date;

};

// использование:

var journals = [

new Journal(new Date(2012, 1, 1)),

new Journal(new Date(2012, 0, 1)),

new Journal(new Date(2011, 11, 1))

];

function findMin(journals) {

var min = 0;

for (var i = 0; i < journals.length; i++) {

// используем статический метод

if (Journal.compare(journals[min], journals[i]) > 0) min = i;

}

return journals[min];

}

alert( findMin(journals).getTitle() );

**Статический метод также можно использовать для функций, которые вообще не требуют наличия объекта.**

Например, метод formatDate(date) можно сделать статическим. Он будет форматировать дату «как это принято в журналах», при этом его можно использовать в любом месте кода, не обязательно создавать журнал.

Например:

function Journal() { /\*...\*/ }

Journal.formatDate = function(date) {

return date.getDate() + '.' + (date.getMonth()+1) + '.' + date.getFullYear();

}

// ни одного объекта Journal нет, просто форматируем дату

alert( Journal.formatDate(new Date) );

**[Фабричные методы](http://learn.javascript.ru/static-properties-and-methods" \l "fabrichnye-metody)**

Рассмотрим ситуацию, когда объект нужно создавать различными способами. Например, это реализовано во встроенном объекте [Date](http://learn.javascript.ru/datetime). Он по-разному обрабатывает аргументы разных типов:

* new Date() – создаёт объект с текущей датой,
* new Date(milliseconds) – создаёт дату по количеству миллисекунд milliseconds,
* new Date(year, month, day ...) – создаёт дату по компонентам год, месяц, день…
* new Date(datestring) – читает дату из строки datestring

**"Фабричный статический метод" – удобная альтернатива такому конструктору. Так называется статический метод, который служит для создания новых объектов (поэтому и называется «фабричным»).**

Пример встроенного фабричного метода – [String.fromCharCode(code)](http://javascript.ru/String.fromCharCode). Этот метод создает строку из кода символа:

var str = String.fromCharCode(65);

alert( str ); // 'A'

Но строки – слишком простой пример, посмотрим что-нибудь посложнее.

Допустим, нам нужно создавать объекты User: анонимные new User() и с данными new User({name: 'Вася', age: 25}).

Можно, конечно, создать полиморфную функцию-конструктор User:

function User(userData) {

if (userData) { // если указаны данные -- одна ветка if

this.name = userData.name;

this.age = userData.age;

} else { // если не указаны -- другая

this.name = 'Аноним';

}

this.sayHi = function() {

alert(this.name)

};

// ...

}

// Использование

var guest = new User();

guest.sayHi(); // Аноним

var knownUser = new User({

name: 'Вася',

age: 25

});

knownUser.sayHi(); // Вася

Подход с использованием фабричных методов был бы другим. Вместо разбора параметров в конструкторе – делаем два метода: User.createAnonymous и User.createFromData.

Код:

function User() {

this.sayHi = function() {

alert(this.name)

};

}

User.createAnonymous = function() {

var user = new User;

user.name = 'Аноним';

return user;

}

User.createFromData = function(userData) {

var user = new User;

user.name = userData.name;

user.age = userData.age;

return user;

}

// Использование

var guest = User.createAnonymous();

guest.sayHi(); // Аноним

var knownUser = User.createFromData({

name: 'Вася',

age: 25

});

knownUser.sayHi(); // Вася

Преимущества использования фабричных методов:

* Лучшая читаемость кода. Как конструктора – вместо одной большой функции несколько маленьких, так и вызывающего кода – явно видно, что именно создаётся.
* Лучший контроль ошибок, т.к. если в createFromData ничего не передали, то будет ошибка, а полиморфный конструктор создал бы анонимного посетителя.
* Удобная расширяемость. Например, нужно добавить создание администратора, без аргументов. Фабричный метод сделать легко: User.createAdmin = function() { ... }. А для полиморфного конструктора вызов без аргумента создаст анонима, так что нужно добавить параметр – «тип посетителя» и усложнить этим код.

**Поэтому полиморфные конструкторы лучше использовать там, где нужен именно полиморфизм**, т.е. когда непонятно, какого типа аргумент передадут, и хочется в одном конструкторе охватить все варианты.

А в остальных случаях отличная альтернатива – фабричные методы.

**[Итого](http://learn.javascript.ru/static-properties-and-methods" \l "itogo)**

Статические свойства и методы объекта удобно применять в следующих случаях:

* Общие действия и подсчёты, имеющие отношения ко всем объектам данного типа. В примерах выше это подсчёт количества.
* Методы, не привязанные к конкретному объекту, например сравнение.
* Вспомогательные методы, которые полезны вне объекта, например для форматирования даты.
* Фабричные методы.