Padrões JavaScript

Stoyan Stefanov

Novatec

Authorized Portuguese translation of the English edition of titled *JavaScript Patterns, First Edition*, ISBN: 978-0-596-80675-0 © 2010, Stoyan Stefanov. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Tradução em português autorizada da edição em inglês do título *JavaScript Patterns, First Edition*, ISBN: 978-0-596-80675-0 © 2010, Stoyan Stefanov. Esta tradução é publicada e vendida com a permissão da O'Reilly Media, Inc., detentora de todos os direitos para publicação e venda desta obra.

© Novatec Editora Ltda. 2011.

Todos os direitos reservados e protegidos pela Lei 9.610 de 19/02/1998. É proibida a reprodução desta obra, mesmo parcial, por qualquer processo, sem prévia autorização, por escrito, do autor e da Editora.

Editor: Rubens Prates
Tradução: Edgard Damiani
Revisão gramatical: Carla Mello Moreira
Editoração eletrônica: Camila Kuwabata e Carolina Kuwabata

ISBN: 978-85-7522-266-9

Histórico de impressões:

Janeiro/2011 Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Fax: +55 11 2950-8869
Email: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/novatec

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Stefanov, Stoyan
Padrões JavaScript / Stoyan Stefanov ; [tradução Edgard Damiani]. -- São Paulo : Novatec Editora ; Sebastopol, CA. : O´Reilly, 2010.

Título original: JavaScript patterns.
ISBN 978-85-7522-266-9

1. JavaScript (Linguagem de programação para computadores) I. Título.

10-13324 CDD-005.133

Índices para catálogo sistemático:

1. JavaScript : Linguagem de programação : Computadores : Processamento de dados 005.133

OGF_20101207

CAPÍTULO 1

Introdução

JavaScript é a linguagem da web. Ela começou como uma forma de manipular alguns tipos de elementos selecionados em uma página web (como imagens ou campos de formulários), mas acabou crescendo enormemente. Além de servir como script de navegador no lado do cliente, atualmente você pode usar o JavaScript para programar em uma variedade cada vez maior de plataformas. Você pode escrever código no lado do servidor (usando .NET ou Node.js), aplicações desktop (que funcionam em todos os sistemas operacionais) e extensões de aplicação (por exemplo, Firefox ou Photoshop), aplicações para dispositivos móveis e scripts de linha de comando.

O JavaScript também é uma linguagem incomum. Ela não possui classes, e funções são usadas como objetos de primeira classe em várias tarefas. Inicialmente, a linguagem foi considerada deficiente por vários desenvolvedores, mas nos últimos anos esse sentimento tem mudado. Curiosamente, linguagens como Java e PHP começaram a adicionar funcionalidades como closures e funções anônimas, que os desenvolvedores JavaScript vêm utilizando corriqueiramente há algum tempo.

O JavaScript é suficientemente dinâmico a ponto de ser possível fazê-lo parecer com qualquer outra linguagem com a qual você se sinta confortável. Mas a melhor abordagem é abraçar suas diferenças e estudar seus padrões específicos.

Padrões

Um padrão (pattern), no sentido mais amplo da palavra, é um "tema de eventos ou objetos recorrentes [...] ele pode ser um template ou modelo utilizado para gerar coisas" (*http://en.wikipedia.org/wiki/Pattern*).

Em desenvolvimento de software, um padrão é uma solução para um problema comum. Um padrão não é necessariamente uma solução de código pronta para ser copiada e colada, e sim mais uma prática melhor, uma abstração útil e um modelo de resolução de categorias de problemas.

É importante identificar padrões porque:

- Eles nos ajudam a escrever melhores códigos, utilizando práticas comprovadas e não reinventando a roda.

- Eles fornecem um nível de abstração – o cérebro pode armazenar apenas um tanto em dado momento, então, quando você pensa em um problema mais complexo, ajuda não ter de se preocupar com os detalhes de baixo nível, encapsulando-os em blocos de construção autocontidos (padrões).
- Eles melhoram a comunicação entre desenvolvedores e equipes, que costumam ficar em locais remotos e não se comunicar pessoalmente. O simples fato de rotular uma técnica ou abordagem de programação facilita saber se estamos falando do mesmo assunto. Por exemplo, é mais fácil dizer (e pensar) "função imediata" do que "esta coisa em que você encapsula a função em parênteses e, no final dela, coloca outro conjunto de parênteses para invocar a função exatamente no local onde você a definiu".

Este livro discute os seguintes tipos de padrões:

- Padrões de projeto (design patterns).
- Padrões de programação (coding patterns).
- Antipadrões (antipatterns).

Padrões de projeto são aqueles definidos inicialmente pelo livro *Gang of Four* ("o quarteto", nomeado assim por causa de seus quatro autores), originalmente publicado em um distante 1994 sob a alcunha de *Design Patterns: Elements of Reusable Object-Oriented Software* (Padrões de Projeto: Elementos Reutilizáveis de Software Orientado a Objeto). Exemplos de padrões de projeto são singleton, fábrica (factory), decorador (decorator), observador (observer) e assim por diante. A questão dos padrões de projeto em relação ao JavaScript é que, apesar de serem independentes de linguagem, os padrões de projeto foram na sua maior parte estudados do ponto de vista de linguagens fortemente tipadas, como C++ e Java. Por vezes não faz sentido algum aplicá-los literalmente em uma linguagem fracamente tipada como o JavaScript. Algumas vezes esses padrões buscam contornar problemas relacionados à natureza fortemente tipada das linguagens e da herança baseada em classes. Talvez existam alternativas mais simples no JavaScript. Este livro discute implementações JavaScript de vários padrões de projeto no capítulo 7.

Os *padrões de programação* são muito mais interessantes; eles são padrões específicos ao JavaScript, além de serem boas práticas relacionadas às funcionalidades exclusivas da linguagem, como os vários usos de funções. Padrões de programação JavaScript são o principal tópico deste livro.

Você pode trombar ocasionalmente com um *antipadrão* no livro. Antipadrões têm certo tom negativo ou até mesmo insultante em seu nome, mas este não é necessariamente o caso. Um antipadrão não é o mesmo que um bug ou um erro de codificação, é apenas uma abordagem comum que causa mais problemas do que soluções. Os antipadrões estão marcados claramente com comentários no código.

JavaScript: conceitos

Vamos revisar rapidamente alguns conceitos importantes que fornecem o contexto para os próximos capítulos.

Orientado a objeto

JavaScript é uma linguagem orientada a objetos, o que costuma surpreender desenvolvedores que previamente observaram a linguagem e dispensaram-na. Tudo o que você vê em um trecho de código JavaScript tem boa chance de ser um objeto. Apenas cinco tipos primitivos não são objetos: numérico, string, booleano, `null` e `undefined`, e os primeiros três têm representações correspondentes de objetos na forma de encapsuladores primitivos (discutidos no próximo capítulo). Valores primitivos numéricos, booleanos e de string são facilmente convertidos em objetos, seja pelo próprio programador ou, às vezes, nos bastidores pelo interpretador JavaScript.

Funções também são objetos. Elas podem ter propriedades e métodos.

A coisa mais simples que você pode fazer em uma linguagem é definir uma variável. Bem, no JavaScript, ao definir uma variável, você já está lidando com objetos. Primeiro, a variável automaticamente torna-se uma *propriedade* de um objeto interno conhecido como um Objeto de Ativação (ou uma propriedade do objeto global, caso seja uma variável global). Segundo, essa variável também é, na verdade, algo como um objeto, porque ela tem suas propriedades particulares (chamadas *atributos*) que determinam se a variável pode ser modificada, apagada ou enumerada em um loop `for-in`. Esses atributos não são expostos explicitamente na ECMAScript 3, mas a edição 5 oferece métodos descritores especiais para manipulá-los.

Afinal, o que são esses objetos? Eles devem ser um tanto especiais, já que podem fazer tantas coisas. Na verdade eles são extremamente simples. Um *objeto* é apenas uma coleção de propriedades nomeadas, uma lista de pares nome-valor (quase idêntico a um array associativo em outras linguagens). Algumas das propriedades podem ser funções (objetos-função), que, no caso, chamamos de *métodos*.

Outra coisa sobre os objetos que você cria é que você pode modificá-los a qualquer momento (apesar de a ECMAScript 5 introduzir APIs que previnem mutações). Você pode pegar um objeto e adicionar, remover e atualizar seus membros. Se estiver preocupado sobre privacidade e acesso, também veremos padrões para isso.

E uma última coisa para se ter em mente é que existem dois tipos principais de objetos:

Nativo

Descrito pela norma ECMAScript.

De hospedeiro (host)

Definido pelo ambiente hospedeiro (por exemplo, o ambiente do navegador web).

Os objetos nativos também podem ser categorizados como embutidos (por exemplo, `Array`, `Date`) ou *definidos pelo usuário* (`var o = {};`).

Objetos de hospedeiro são, por exemplo, o objeto `window` e todos os objetos DOM. Se quiser saber se você está utilizando objetos de hospedeiro, tente executar seu código em um ambiente diferente do ambiente de navegador. Se funcionar corretamente, provavelmente você está usando apenas objetos nativos.

Sem classes

Você verá esta afirmação repetida em várias ocasiões ao longo do livro: não há classes no JavaScript. Isso é um conceito novo para os programadores experientes de outras linguagens e leva mais do que algumas repetições, e mais do que um pouco de esforço, para "desaprender" classes e aceitar que o JavaScript lida apenas com objetos.

Não ter classes torna os seus programas mais curtos – você não precisa ter uma classe para criar um objeto. Considere essa criação de objeto no estilo Java:

```
// criação de objeto no estilo Java
HelloOO hello_oo = new HelloOO();
```

Repetir três vezes a mesma coisa parece um exagero quando se quer criar objetos simples. E normalmente desejamos manter nossos objetos simples.

No JavaScript, você cria um objeto em branco quando precisa de um e, então, começa a adicionar membros interessantes a ele. Você compõe objetos adicionando tipos primitivos, funções e outros objetos a eles como sendo suas propriedades. Um objeto "em branco" não é totalmente vazio; ele já vem com algumas propriedades embutidas, mas não possui propriedades particulares. Falaremos mais sobre isso no próximo capítulo.

Uma das regras gerais do livro *Gang of Four* diz assim: "Prefira composição de objetos a herança de classes". Isso significa que, se você puder criar objetos a partir de elementos disponíveis que estão dando sopa, isso é uma abordagem muito melhor do que criar longas cadeias de herança e classificações pai-filho. No JavaScript, é fácil seguir esse conselho – simplesmente porque não há classes, e composição de objetos é o que você vai acabar fazendo de qualquer maneira.

Protótipos

O JavaScript possui herança, apesar de isso ser apenas uma das formas de reutilizar código (e teremos um capítulo inteiro sobre reutilização de código). Herança pode

ser realizada de várias formas, normalmente fazendo uso de protótipos. Um *protótipo* é um objeto (o que não é uma surpresa) e toda função que você cria recebe automaticamente uma propriedade `prototype` que aponta para um novo objeto em branco. Esse objeto é quase idêntico a um objeto criado a partir de um objeto literal ou pelo construtor `Object()`, exceto que sua propriedade `constructor` aponta para a função que você criou, e não para o objeto embutido `Object()`. Você pode adicionar membros a esse objeto em branco e, mais tarde, ter outros objetos herdando desse objeto e utilizando as propriedades dele como se fossem criadas por você.

Discutiremos herança em detalhes, mas por ora tenhamos em mente que o protótipo é um objeto (não uma classe ou algo especial) e que toda função tem uma propriedade `prototype`.

Ambiente

Programas JavaScript precisam de um ambiente para serem executados. O habitat natural de um programa JavaScript é o navegador web, mas esse não é o único ambiente disponível. Os padrões mostrados neste livro são na sua maioria relacionados ao núcleo (core) do JavaScript (ECMAScript), então eles são independentes do ambiente. As exceções são:

- o capítulo 8, que lida especificamente com padrões de navegador;
- alguns outros exemplos que ilustram aplicações práticas de um padrão.

Os ambientes podem fornecer *objetos de hospedeiro* próprios, que não são definidos na norma ECMAScript e que podem ter comportamento não especificado ou indefinido.

ECMAScript 5

O núcleo (core) da linguagem de programação JavaScript (que exclui o DOM, o BOM e objetos de hospedeiro extras) é baseado na norma ECMAScript, ou ES para abreviar. A versão 3 da norma foi aceita oficialmente em 1999 e é a versão atualmente implementada nos navegadores. A versão 4 foi abandonada e a versão 5 foi aprovada em dezembro de 2009, 10 anos após a versão prévia.

A versão 5 inclui na linguagem alguns objetos embutidos, propriedades e métodos novos, mas sua inclusão mais importante foi o chamado *modo estrito*, que, na verdade, remove algumas funcionalidades da linguagem, tornando os programas mais simples e menos propensos a erros. Por exemplo, o uso da instrução `with` tem sido questionado ao longo dos anos. Agora, no modo estrito do ES5 ela gera um erro, apesar de não haver problemas em utilizá-la no modo não estrito. O modo estrito é ativado por uma string comum, que as implementações mais antigas da linguagem simplesmente ignoram. Isso significa que o uso do modo estrito é compatível com

as versões anteriores, já que ele não gera erros em navegadores mais antigos que não o reconheçam.

Uma vez por escopo (seja escopo de função, escopo global ou no início de uma string passada com `eval()`), você pode usar a seguinte string:

```
function my() {
    "use strict";
    // o resto da função...
}
```

Isso significa que o código na função é executado dentro do subconjunto estrito da linguagem. No caso de navegadores antigos, isso é apenas uma string não atribuída a uma variável, então ela não é usada, e ainda assim não é um erro.

O plano para a linguagem é que, no futuro, o modo estrito seja o único modo permitido. Nesse sentido, a ES5 é uma versão de transição – os desenvolvedores são encorajados, mas não forçados, a escrever código que funcione no modo estrito.

Este livro não explora padrões relacionados às inclusões específicas da ES5, porque quando o estávamos escrevendo não havia navegadores que a implementassem. Mas os exemplos neste livro promovem uma transição ao novo padrão:

- garantindo que as amostras de código oferecidas não irão gerar erros no modo estrito;
- evitando e indicando construções obsoletas, como `arguments.callee`;
- invocando padrões da ES3 que tenham equivalentes na ES5 embutidos, como `Object.create()`.

JSLint

JavaScript é uma linguagem interpretada sem verificações estáticas em tempo de compilação. Assim, é possível gerar um programa defeituoso com apenas um erro de digitação sem se aperceber do fato. É aqui que o JSLint ajuda.

O JSLint (*http://jslint.com*) é uma ferramenta de qualidade de código JavaScript, criada por Douglas Crockford, que inspeciona seu código e avisa sobre problemas em potencial. É altamente recomendável que você execute seu código por meio do JSLint. A ferramenta vai "ferir seus sentimentos", como o próprio criador alerta, mas apenas no início. Você pode aprender rapidamente com seus erros e adotar hábitos essenciais de um programador JavaScript profissional. Não ter erros JSLint em seu código também ajuda a ter mais *confiança* no código, sabendo que, na pressa, você não cometeu uma simples omissão ou erro de sintaxe.

No próximo capítulo, você verá que o JSLint é bastante mencionado. Todo o código no livro passa com sucesso na verificação do JSLint (usando as configurações padrão correntes no momento em que escrevemos o código), exceto por algumas poucas ocasiões claramente demarcadas como antipadrões.

Em suas configurações padrão, o JSLint espera que seu código seja compatível com o modo estrito.

Objeto console

O objeto `console` é utilizado ao longo do livro. Esse objeto não faz parte da linguagem, e sim do ambiente, e está presente na maioria dos navegadores atuais. No Firefox, por exemplo, ele vem com a extensão Firebug. O console do Firebug possui uma interface de usuário que lhe permite digitar e testar rapidamente trechos de código JavaScript, além de brincar com a página carregada atualmente (Figura 1.1). Ele também é altamente recomendado como ferramenta de aprendizado e exploração. Funcionalidades semelhantes estão disponíveis nos navegadores WebKit (Safari e Chrome) como parte do Web Inspector, e no IE a partir da versão 8 como parte das Developer Tools (Ferramentas do Desenvolvedor).

A maioria dos exemplos de código no livro usa o objeto `console`, em vez de usar `alert()` ou ter de atualizar a página atual, porque isso é uma maneira simples e não intrusiva de imprimir saídas.

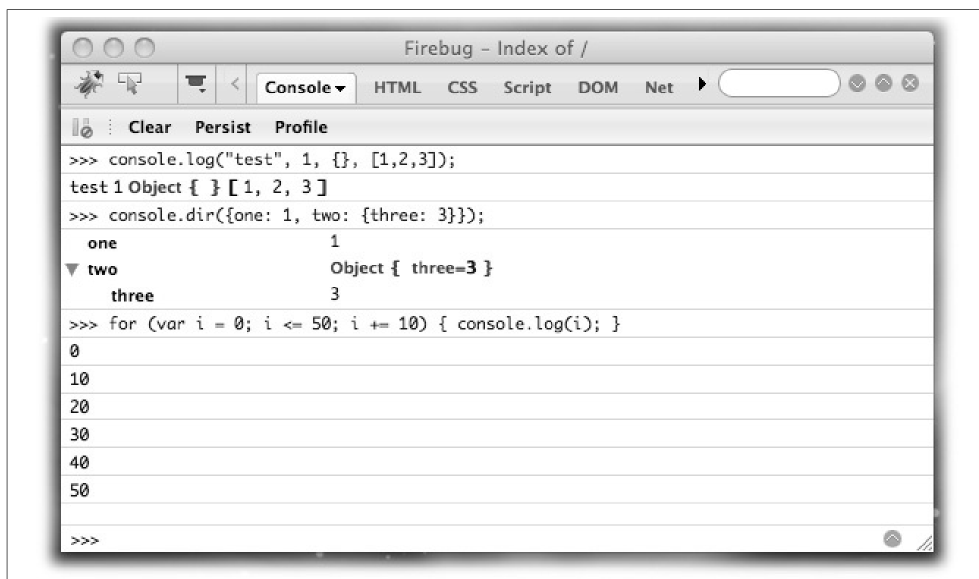

Figura 1.1 – Usando o console do Firebug.

Normalmente usamos o método `log()`, que imprime todos os parâmetros passados a ele, e algumas vezes usamos o método `dir()`, que enumera o objeto passado a ele e imprime todas as propriedades. Aqui está um exemplo de sua utilização:

```
console.log("test", 1, {}, [1,2,3]);
console.dir({one: 1, two: {three: 3}});
```

Quando estiver digitando no console, não há a necessidade de usar `console.log()`; você pode simplesmente omiti-lo. Para evitar confusão, alguns trechos de código não o utilizam e assumem que você esteja testando o código no console.

```
window.name === window['name'];   // true
```

Isso é como se usássemos o seguinte:

```
console.log(window.name === window['name']);
```

e ele tivesse imprimido `true` no console.