

Introdução

O exame 70-480 é um exame detalhado de suas habilidades com o uso de HTML5 e CSS3. Este livro irá guiá-lo através dos objetivos necessários que você deve saber para passar neste exame. Espera-se que você tenha pelo menos 2 anos de experiência trabalhando Tecnologias. Este livro é estruturado de tal forma que fornece uma referência às principais informações Necessário para cada objectivo. Este livro não ensina todos os conceitos, mas fornece detalhes do que você deve saber para cada objetivo coberto no exame.

Este livro cobre todos os objetivos do exame, mas não cobre todas as perguntas do exame. Somente a equipe de exame da Microsoft tem acesso às perguntas do exame e à Microsoft regularmente acrescenta novas questões ao exame, tornando impossível cobrir questões específicas. Você deve considerar este livro como um complemento para a sua experiência real do mundo materiais de estudo. Se você encontrar um tópico neste livro que você não se sinta completamente confortável com os links que você encontrará no texto para encontrar mais informações e tomar o tempo para pesquisar e estudar o tema. Grandes informações estão disponíveis no MSDN, TechNet e em blogs e fóruns.

Implementar e manipular estruturas de documentos e objetos

Os desenvolvedores da Web precisam entender a complexidade das construções envolvidas construindo aplicações interativas e dinâmicas com HTML e JavaScript. A introdução do HTML5 trouxe um novo padrão para definir a estrutura de suas páginas bem como mudanças em como você interage com eles via script.

Este capítulo demonstra como criar documentos HTML5 com a nova marcação semântica HTML5. Você vai explorar o processo de criação do código necessário para manipular e interagir com HTML5 marcação e aplicação e estilos para elementos HTML5.

Objetivo 1.1: Criar a estrutura do documento

Escrever uma página HTML pode ser muito fácil ou muito assustador, dependendo de seus objetivos. Qualquer página HTML processa em um navegador, mesmo se ele contém apenas texto sem formatação. Mas esse tipo de web Aplicativo pode não ser eficaz na entrega da mensagem pretendida ou no fornecimento de interatividade para manter os usuários voltando ao site para mais. Isto é onde a marcação HTML vem ser acessível.

O HTML permite que você aplique uma estrutura organizada e fluida em páginas da web. Emparelhado com uma ferramenta poderosa como JavaScript, o HTML5 permite que você forneça conteúdo altamente interativo que desperta

o interesse dos seus utilizadores. Este objetivo centra-se nos elementos semânticos disponíveis para si em HTML5, que, juntamente com o JavaScript, permitem que você crie a experiência do usuário final.

O objetivo da estrutura de um documento é informar ao navegador como o conteúdo deve ser exibida. Sem qualquer estrutura declarativa em sua página, o navegador não detectará Estrutura, de modo que irá estabelecer o seu conteúdo de acordo com as regras implementadas por seu motor renderização.

Ao usar a marcação HTML5 apresentada neste objetivo, você está dizendo ao navegador para ter sua semântica em conta ao exibir a página. Em frente, novos lançamentos de navegadores irão incorporar mais e mais padrões HTML5 em seus motores de renderização.

O exame testará sua habilidade de usar marcação semântica HTML5 para criar páginas sua capacidade de otimizar páginas da Web para uso em leitores de tela. O exame também efeito que a marcação semântica HTML5 terá na otimização de mecanismo de busca.

Este objetivo abrange:

- Usar marcação semântica HTML5
- Criar um contêiner de layout em HTML
- Otimizar para os mecanismos de pesquisa
- Otimizar para leitores de tela

Usando a marcação semântica HTML5

A Tabela 1-1 lista os elementos semânticos HTML5 conforme definidos pela especificação. Esses elementos constituem o núcleo do HTML5. Como tal, a compreensão da definição e uso apropriado de cada Elemento é crítico para a conclusão bem sucedida do exame. Nas seções a seguir, você usará cada um desses elementos de marcação semântica para criar uma estrutura de documento completa.

TABELA 1-1 Marcação semântica HTML5

Elemento HTML5 Descrição

<Article> Define áreas auto-contidas em uma página.

<Aside> Define áreas de conteúdo menores fora do fluxo de uma página da Web.

<Figcaption> Define a legenda de um elemento de figura.

<Figure> Define o conteúdo que contém uma figura, como uma imagem, gráfico ou imagem.

<Footer> Define a parte inferior de uma seção ou página.

<header> Define o topo de uma seção ou página.

<Hgroup> Define um grupo de títulos (elementos H1-H6).

<Mark> Define o texto que deve ser destacado.

<Nav> Define a navegação para outras páginas do site.

<Progress> Define o progresso da tarefa.

<Section> Define o conteúdo distinto de um documento.

NOTA USE SOMENTE O QUE VOCÊ PRECISA

Ao projetar uma página da web, você não precisa necessariamente usar cada elemento disponível. Em vez disso, use apenas os elementos que você precisa para fazer seu trabalho.

Compreendendo a estrutura principal de uma página HTML5

Embora um navegador possa renderizar qualquer arquivo de texto simples, para fornecer qualquer estrutura para o documento, a página deve conter os elementos básicos que você está prestes a aprender. Embora este livro assuma que você tem uma compreensão básica de como as páginas são estruturadas, os seguintes código HTML demonstra o modelo básico de uma página HTML5:

```
<! DOCTYPE html>
```

```
<Html>
```

```
  <Head>
```

```
    <Meta charset = "utf-8" />
```

```
    <Title> </ title>
```

```
  </ Head>
```

```
  <Body>
```

```
    <! - o conteúdo da página vai para aqui ->
```

```
  </ Body>
```

```
</ Html>
```

Esta seção elabora seja uma estrutura básica à medida que a marcação semântica é introduzida no objetivo. Todo o conteúdo a ser introduzido na página irá dentro do elemento <body>.

Porque essa seção de página exibe conteúdo no navegador. Tal como é agora, este código nada mais do que uma página em branco. Para ver algum conteúdo enquanto você trabalha com o seguinte código, use o código na Listagem 1-1.

Codigo pagina.html

Usando os elementos <header> e <footer>

A maioria dos documentos da página web contém conteúdo comum na parte superior e inferior de todas as páginas. Embora usar os elementos <header> e <footer> não forneça automaticamente essa funcionalidade, os elementos fornecem a capacidade de definir o conteúdo no cabeçalho e rodapé de do site.

Normalmente, um cabeçalho da página contém conteúdo como um logotipo ou banner da empresa. Em alguns casos, ele também pode conter um menu de navegação. (Consulte a próxima "Usando o elemento <nav>" para isso.) Inicie a página de exemplo adicionando o elemento <header> à sua página:

O elemento <header> não se limita apenas ao início da sua página, pois fornece uma maneira de declarar o cabeçalho para qualquer área da página da web. Você pode usar o elemento <header> como um cabeçalho para um elemento <section> ou para um elemento <article>. O elemento <header> é destinado a conter um elemento H1-H6 conforme necessário; No entanto, você pode preencher um cabeçalho com qualquer marcação que se adapte às suas necessidades para criar o melhor cabeçalho para essa área específica do site.

Usando o elemento <nav>

Usando o elemento <nav> em um documento HTML5, os usuários podem navegar nos principais elementos do documento da Web ou da aplicação web como um todo. Estes podem ser representados como uma lista de links na parte superior da página para navegar no site atual.

Ele também pode listar seus sites favoritos ao longo do lado da página, como em um blog onde você lista outros blogs favoritos que você segue.

Normalmente, a lista de links na parte superior, comumente conhecida como o menu principal da Aplicativo web, está contido no cabeçalho (mas não tem que ser). Uma lista de URLs favoritos mais provável ser colocado em um <aside>, a partir do conteúdo principal, mas prontamente acessível.

O outro elemento principal comumente usado sob o elemento <header> é o rightfully denominado <hgroup>.

Usando o elemento Hgroup

O elemento `<hgroup>` é um método semântico que organiza cabeçalhos e subtarefas. Esse elemento normalmente contém os elementos padrão e familiares `<h1>` para `<h6>`. O elemento `<hgroup>` agrupa cabeçalhos relacionados em sequência. Você pode adicionar um novo elemento `<hgroup>` à sua página da Web para atender a essa finalidade, da seguinte forma:

```
<body>
...
<article>
<header>
<hgroup>
<h1>Our first new Article</h1>
</hgroup>
</header>
<p>Provide some useful information in the article</p>
</article>
</body>
```

Este exemplo de código efetivamente informa o processador que o artigo tem um título principal (`<h1>`). Você pode continuar, adicionando elementos `<h2>` a `<h6>` se necessário. Quantos desses elementos (`<h1>` a `<h6>`) você usa no elemento `<hgroup>` obviamente depende da estrutura do documento que você deseja apresentar. Agora que a página está começando a tomar forma, você pode aprender sobre os dois principais conteúdos elements: `<article>` e `<section>`.

Usando o elemento <article>

Um elemento `<article>` representa uma composição completa e completa ou entrada. Exemplos de um elemento `<article>` poderia ser um artigo de revista ou um post de blog, onde o conteúdo pode ser redistribuído de forma independente e não perder o seu significado. Cada artigo está inteiramente próprio. Você pode ter um artigo com subarticles; No entanto, cada sub-artigo deve ser uma extensão e relacionadas com o artigo de raiz.

Agora que você sabe sobre este novo elemento `<article>`, você pode voltar ao seu exemplo de documento e organizá-lo com artigos:

Você expandiu o documento para incluir três artigos. Claramente, você não quer colocar todo o capítulo no documento HTML da amostra, mas você adicionará o suficiente para poder demonstrar a função de cada elemento semântico. Cada artigo adicionado ao documento neste exemplo representa uma parte independente do documento que pode ser totalmente contido. Normalmente, o primeiro elemento dentro do elemento `<article>` é um elemento de cabeçalho ou grupo de cabeçalho. Estreitamente relacionado ao elemento `<article>` é o elemento `<section>`, que você explora a seguir.

Usando o elemento `<section>`

O elemento `<section>` subdivide as páginas em seções. Você poderia continuar a quebrar a página de exemplo com elementos adicionais `<article>`;

No entanto, a finalidade do `<article>` não é dividir uma página em detalhes mais detalhados. Este é o local onde o elemento `<section>` torna-se útil. Cada elemento `<article>` contém zero ou mais elementos `<section>` para denotar as diferentes seções de conteúdo dentro do elemento `<article>`. Como um elemento `<article>` o primeiro elemento dentro de um elemento `<section>` é normalmente um cabeçalho ou um grupo de cabeçalho.

```
<body>
...
<article>
  <header>
    <hgroup>
      <h1>Our first new Article</h1>
    </hgroup>
  </header>
  <section>
    <h1>Section 1</h1>
    <p>Some details about section 1</p>
    <aside>Did you know that 7/10 is 70%</aside>
  </section>
  <section>
    <h1>Section 2</h1>
  </section>
```

```
</article>

<article>

<header>

<hgroup>

<h1>Second huge article</h1>

</hgroup>

</header>

<p>Provide some useful information in the article</p>

</article>

<article>

<header>

<hgroup>

<h1>Third huge article</h1>

</hgroup>

</header>

<p>Provide some useful information in the third article</p>

</article>

</body>
```

Quando você visualiza esta página no navegador agora, poderá notar algo bastante interessante. Os elementos `<h1>` dentro dos elementos `<hgroup>` são renderizados de forma diferente da anterior `<H1>` no mesmo artigo.

Considerando que você não aplicou nenhum estilo a esta página, você pode esperar que todos os elementos `<h1>` renderizem no mesmo estilo (por exemplo, o mesmo tamanho de fonte) como o elemento `<h1>` no elemento `<article>`. No entanto, quando você rodar a página no navegador, você pode ver que este não é o caso. Isso se deve ao analisador de documentos no navegador, que funciona através do documento para determinar a hierarquia implícita dos títulos, também chamado de esboço do documento. Você vê mais isso quando você aprender sobre os leitores de tela mais adiante nesta lição. Por enquanto, concentre-se no elemento `<aside>`.

Usando o elemento `<aside>`

O elemento <aside> define qualquer conteúdo que não caia dentro do fluxo principal do conteúdo da página atual - por exemplo, uma barra lateral, uma nota, um alerta ou um anúncio.

O elemento <aside> não se coloca automaticamente em nenhum lado particular da página da web. Ele meramente serve como uma forma de definir semanticamente uma seção de texto ou gráficos como um aparte. Mais tarde você verá como posicionar um lado usando estilos. Por enquanto, adicione o seguinte elemento <aside>.

Para sua página para uso posterior:

```
<body>
...
<article>
<header>
<hgroup>
<h1>Our first new Article</h1>
</hgroup>
</header>
<section>
<h1>Section 1</h1>
<p>Some details about section 1</p>
<aside>Did you know that 7/10 is 70%</aside>
</section>
<section>
<h1>Section 2</h1>
</section>
</article>
...
</body>
```

Como você pode ver na saída do navegador, o elemento <aside> não é tratado como especial em qualquer comparado com os outros elementos usados para estruturar sua página. No entanto, um pouco mais tarde, você pode ver o quanto mais facilmente você pode estilizar o conteúdo usando marcação semântica, como o elemento <aside>.

Usando os elementos <figcaption> e <figura>

Os elementos <figcaption> e <figure>, novos em HTML5, fornecem os elementos semânticos necessários para adicionar gráficos e figuras a páginas da web. Esses gráficos e figuras normalmente fornecem uma representação visual da informação no conteúdo textual e referenciada pelo texto. Você ve Imagens em tutoriais ou livros de texto nos quais o autor dirige o leitor para uma figura específica. Como exemplo, eu adicionei alguns HTML para o final do exemplo anterior:

```
<body>

...

<article>

...

<figure>



<figcaption>Fig 1: A really juicy orange.</figcaption>

</figure>

</article>

</body>
```

Você precisa substituir a imagem de exemplo por uma que você já tenha para obter a página para renderizar corretamente. No entanto, a essência do que você conseguir com os elementos <figcaption> e <figura> deve ser clara.

Usando o elemento <progress>

O elemento <progress> representa o progresso de um objetivo ou tarefa. Os dois tipos de tarefas de progresso são determinados e indeterminados.

Use uma tarefa de progresso determinado quando você sabe antecipadamente a quantidade de trabalho a ser concluído, em outras palavras, você sabe os valores inicial e final.

Cenários de exemplo para este caso inclui o download de um arquivo para o qual você sabe o tamanho exato ou o progresso de um esforço de angariação de fundos. Em ambas as situações, você sabe o status exato da tarefa em qualquer tempo específico, e você também sabe qual é o objetivo final - ou o número de bytes para o download do arquivo ou o número de dólares para o fundraiser. Nestes casos determinados, pode especificar HTML5 marcação como esta:

```
<p>Our goal is to have 1000 users:</p>

<span>0</span>
```

```
<progress value="50" max="1000"></progress>
```

```
<span>1000</span>
```

Conforme mostrado no código anterior, o elemento `<progress>` tem dois atributos que você precisa saber: `value` e `máx`. O atributo `value` permite especificar o valor atual ou a posição do `<Progress>` em um ponto específico no tempo. O atributo `max` informa ao navegador o que o valor máximo possível para o elemento. O navegador usa esses dois valores para determinar o quanto o elemento deve ser colorido. Normalmente, o atributo `valor` atualiza dinamicamente usando JavaScript. Na Figura 1-10, você pode ver como a exibição do elemento `<progress>` muda quando o atributo `value` é atualizado para 750.

Você usa tarefas indeterminadas quando não sabe quanto tempo uma tarefa levará para concluir, mas ainda quer mostrar aos usuários que algum trabalho está ocorrendo e que eles devem esperar. Você ainda usa o elemento `<progress>`, mas remove o atributo `value`. Quando você não especificar o valor do atributo, o navegador pode inferir que o elemento `<progress>` representa uma tarefa. Isso pode ser útil para dados recebidos de um serviço onde você não tem controle sobre ou tem o conhecimento da rapidez com que o pedido será concluído ou o quão grande será o resultado do pedido.

A marcação HTML5 a seguir demonstra uma tarefa indeterminada:

```
<p>Data download is in progress, please wait patiently:</p>
```

```
<progress max="5"></progress>
```

FIGURA 1-11 Exibindo o progresso indeterminado usando pontos móveis para demonstrar que o trabalho está ocorrendo.

Na Figura 1-11, os pontos azuis substituem a barra de progresso do exemplo determinado anterior. Esta alteração visual para o indicador de progresso ocorreu simplesmente removendo o atributo `value` do elemento `<progress>`. Na realidade, os pontos são animados, como você veria se você executar o código em um navegador.

Elemento <mark>

Com o elemento `<mark>`, você pode facilmente destacar informações importantes ou quer enfatizar. Tem essencialmente a mesma função como um highlighter. Envolvendo texto em um `<Mark>` e fornecer um atributo `background-color` ao seu elemento de estilo, você pode obter o efeito de destaque desejado. O código HTML a seguir demonstra o elemento `<mark>`:

```
<p>Some very <mark style="background-color:red;">important</mark>  
information provided here!</p>
```

Usando o elemento DIV

Os novos elementos semânticos HTML5 não (com exceção do elemento <progress>), não necessariamente fornecem qualquer padrão ou comportamento alterado. Em vez disso, eles fornecem uma definição para suas páginas da web. Isso, por sua vez, oferece uma maneira mais confiável de estruturar suas páginas e estilo-las consistentemente. O objetivo destes elementos é substituir o método mais antigo de estruturar páginas - antes do HTML5 - usando <div> elementos e nomes de acordo com sua função. No entanto, observe que o elemento <div> ainda faz parte do HTML5 e ainda desempenha um papel importante. Use os novos elementos semânticos conforme apropriado, mas lembre-se que o elemento <div> ainda é bastante útil para estilo de conteúdo.

Esta seção explorou os novos elementos semânticos em HTML5. Na próxima seção, você aprende como criar e trabalhar com contêineres de layout.

Criando um contêiner de layout em HTML

Você pode criar uma página da web de várias maneiras. Um objetivo importante aqui é pedir-lhe para pensar seriamente no layout para que sua apresentação de página seja amigável.

Se os usuários não conseguirem encontrar o que eles estão procurando, porque toda a página é denominada como um único <p> elemento dentro do elemento <body>, eles provavelmente não voltarão. Nesta seção, você olha para um par de opções de layout disponíveis em HTML. O Capítulo 4, "Usar CSS em aplicativos", explica como usar folhas de estilo em cascata (CSS) para implementar seus layouts.

Os dois métodos mais comuns de criação de um layout em HTML envolvem o uso dos elementos <div> e <Table>. Em ambos os casos, mais do que provavelmente você ainda vai usar CSS para ajudar com o posicionamento e dimensionamento. Capítulo 4 entra em mais detalhes sobre CSS, esta seção examina especificamente o layout, usando apenas HTML.

Ainda assim, os elementos do container, os familiares <div> são freqüentemente usados para dividir a página em várias seções para criar o layout. Por exemplo, você pode ver esse tipo de HTML usado para obter o layout:

```
<div id="idPageHeader"> </div>
```

```
<div id="leftSide"></div>
```

```
<div id="rightSide"></div>
```

```
<div id="footer"></div>
```

O mecanismo de renderização exibe cada <div> de acordo com suas regras. Para posicionar as divisões dinamicamente exigiria CSS.

O principal problema com o uso de <div> elementos para estruturar o documento é a sua incapacidade de transmitir significado semântico padrão para cada seção. Você vai revisar esses exemplos mais tarde, quando você explora a criação de layouts em CSS.

O elemento <div> permite uma capacidade mais dinâmica no layout da página. Para um layout mais estático declarado diretamente na página HTML, o elemento <table> é mais apropriado. O HTML a seguir define uma tabela que fornece um formato de blog comum, com uma seção de cabeçalho, uma barra lateral esquerda, uma área de conteúdo, uma barra lateral direita e uma área de rodapé:

```
<Table>
  <Tr>
    <Td colspan = "3" id = "Cabeçalho"> </ td>
  </ Tr>
  <Tr>
    <Td rowspan = "3" id = "LeftBar"> </ td>
    <Td rowspan = "3" id = "MainContent"> </ td>
    <Td id = "RightSideTop"> </ td>
  </ Tr>
  <Tr>
    <Td id = "RightSideMiddle"> </ td>
  </ Tr>
  <Tr>
    <Td id = "RightSideBottom"> </ td>
  </ Tr>
  <Tr>
    <Td colspan = "3" id = "Footer"> </ td>
  </ Tr>
</ Table>
```

O elemento <table> é muito flexível. Elementos adicionais como <thead> e <tfoot> fornecem uma abordagem mais semântica para rotular as células da tabela. A preocupação com o uso da abordagem de elementos <table> é a natureza estática da estrutura. Para alterar a estrutura geral de um site que usa tabelas para layout, você precisa ir para cada página e fazer as alterações. Vale a pena notar que alguns métodos que tornam essas mudanças mais fácil

evoluíram ao longo dos anos em resposta à dor de cabeça de manutenção envolvidos.

Otimização para motores de busca

Quando um site é necessário para criar uma presença on-line, você tem que garantir que ele pode ser encontrado entre os milhões de sites que já existem. **Search Engine Optimization (SEO)** é uma técnica usada para tornar os elementos do site facilmente descobertos e devidamente indexados pelos motores de busca, de modo que quando os usuários pesquisam conteúdo relacionado ao seu site, eles encontram suas páginas. Os motores de busca como Bing e Google constantemente vasculhar a Internet para o conteúdo. Quando eles encontram páginas web, eles passam pelo conteúdo HTML e índice, como metadados de página e imagem. Eles usam os dados indexados para permitir que os usuários pesquisem essencialmente qualquer coisa na Internet e recebam resultados relevantes. Claramente, então, existe uma relação entre o conteúdo de seus sites e a facilidade com que os usuários podem encontrar seus sites usando uma pesquisa

No passado (definido como pré-HTML5), os web designers usavam <div> elementos para segmentar a página. Esses elementos não fornecem muito contexto quanto ao que eles pretendem conter. **Mas com a marcação semântica disponível em HTML5, você pode usar mais elementos descritivos para as seções de página.** Como você viu no exemplo de layout de página de blog, os elementos HTML sozinhos deixam clara a intenção de cada segmento da página. À medida que os motores de busca exploram páginas da Web, detectam a marcação e sabem o que tirar dela para indexar adequadamente a página. **Os elementos <article> e <section> são os principais utilizados pelo algoritmo SEO. Estes elementos são conhecidos por conter o corpo principal da página.** Que uma página tem mais de um elemento <article> e / ou <section> é aceitável, todos eles são indexados dentro de cada elemento <article>, o motor procura então elementos como <hgroup> ou <h1> para obter o tópico principal do elemento <article> para a indexação relevante. No entanto, isso não significa que, se um site não tiver elementos <article> ou <section>, ele não será indexado e será pesquisável. Isso fala apenas da qualidade da indexação que os mecanismos de pesquisa podem realizar para tornar seu site mais pesquisável pelos usuários finais. SEO é uma ótima técnica para entender quando projetar sites. Criando um site apenas para deixá-lo no escuro e difícil de encontrar não serve muito propósito. Ser encontrado é muito importante e, quando o site é encontrado, você não quer limitar seu público. A acessibilidade também é muito importante. Em seguida, veja como o HTML5 afeta o uso de leitores de tela.

Com os elementos semânticos HTML discutidos no objetivo anterior em mente, você deve observar algumas coisas adicionais com relação aos motores de busca lógica usar para descobrir o que está em sites. No todo, o assunto de SEO está distante fora do espaço deste livro e exame; Livros inteiros são escritos sobre o assunto. No entanto, discutir como HTML5 impactos SEO e design do site é relevante.

Otimização para leitores de tela

Os leitores de tela contam com o esboço do documento para analisar a estrutura e apresentar as informações ao usuário. Os programas de leitura de tela podem ler o texto na página e convertê-lo em áudio por meio de um algoritmo de texto para fala. Isso é útil para usuários que podem ter dificuldade em visualizar a página da Web. Como discutido anteriormente, a maneira como o documento é delineado em HTML5 mudou. Aqui está um pouco mais de detalhes. Antes do HTML5, uma página foi descrita usando apenas os elementos de cabeçalho (<h1> a <h6>). A posição relativa de cada elemento de cabeçalho para o elemento de cabeçalho anterior dentro da página criou a hierarquia. Os leitores de tela podem usar essas informações para apresentar um índice aos usuários. No entanto, o HTML5 introduziu elementos semânticos para criar novas seções. Isso significa que os elementos <section>, <article>, <nav> e <aside> definem novas seções. A introdução dos elementos semânticos muda a forma como o contorno do documento é criado. Por exemplo, se o HTML a seguir fosse ser a hierarquia do documento de amostra, você poderia possivelmente colocá-lo como este:

```
<h1>Fruits and Vegetables</h1>
```

```
<h2>Fruit</h2>
```

```
<h3>Round Fruit</h3>
```

```
<h3>Long Fruit</h3>
```

```
<h2>Vegetables</h2>
```

```
<h3>Green</h3>
```

```
<h3>Colorful</h3>
```

O contorno mostra os estilos padrão dos diferentes elementos de cabeçalho conforme o esperado. Os elementos de título criam seções e sub-seções implícitas dentro do documento. Isso ainda é válido em HTML5. No entanto, você não deve deixar a seção de página para seção implícita como apresentado pelos elementos de cabeçalho; Em vez disso, você deve explicitamente definir as seções usando a semântica apropriada. Também é recomendado que os elementos <h1> sejam usados exclusivamente em um documento HTML5. Para produzir a mesma hierarquia desta forma, você precisaria alterar seu HTML para ser como o seguinte:

```
<section>
```

```
  <h1>Fruits and Vegetables</h1>
```

```
  <section>
```

```
    <h1>Fruit</h1>
```

```
  <section>
```

```
        <h1>Round Fruit</h1>
    </section>
    <section>
        <h1>Long Fruit</h1>
    </section>
</section>
<section>
    <h1>Vegetables</h1>
    <section>
        <h1>Green</h1>
    </section>
    <section>
        <h1>Colorful</h1>
    </section>
</section>
</section>
```

Este HTML produz a mesma saída como mostrado anteriormente na Figura 1-13. A diferença é que agora cada elemento <section> cria uma nova seção de página em vez de confiar nos elementos de cabeçalho para criar as seções. Os leitores de tela podem analisar os elementos semânticos para criar o esboço do documento e, eventualmente, podem proporcionar uma experiência de usuário muito mais rica por causa de como o HTML5 permite que os projetistas de páginas da Web definam as páginas

Objetivo 1.2: Escrever código que interage com ui

Controles de IU neste objetivo, você verifica como interagir com páginas da web no navegador usando código. Os navegadores da Web incluem um ambiente poderoso no qual você pode controlar o comportamento das páginas da Web.

E alguns novos elementos HTML5 proporcionam uma melhor interatividade para os usuários finais.

Você também verifica como modificar o modelo de objeto de documento dinamicamente, usando JavaScript.

Você examina como implementar vídeo e áudio em páginas da Web e como controlá-los programaticamente.

Finalmente, você verifica como renderizar gráficos dinamicamente ou permitir que os usuários desenhem seus próprios gráficos.

Adicionando ou modificando elementos HTML

A capacidade de modificar um documento HTML em tempo de execução é muito poderoso. Até agora, você já viu como criar suas páginas da Web, colocá-las elegantemente e torná-las para os usuários. Em muitos casos, você deve modificar o layout de suas páginas da Web em tempo de execução, dependendo do que seus usuários fazem. Isto é onde você pode tirar proveito do poder do JavaScript. O JavaScript fornece o kit de ferramentas que você precisa para escrever código que interage com elementos da página Web depois que eles já são renderizados no navegador. Antes de começar a modificar a página, você precisa saber como acessar ou fazer referência aos elementos para que você possa manipulá-los.

Modelo de objeto de documento

O DOM (Document Object Model) é uma representação da estrutura da página HTML que você pode interagir programaticamente. Como demonstrado anteriormente, uma página HTML é uma hierarquia. O navegador produz um esquema baseado na hierarquia HTML apresentada e exibe isso no navegador para o usuário. Nos bastidores, desconhecido para o usuário, o navegador constrói um DOM. A interface de programação de aplicativos (API) do DOM é exposta como objetos com propriedades e métodos, permitindo que você escreva código JavaScript para interagir com os elementos HTML renderizados para a página.

Essa noção é muito poderosa. Você pode adicionar novos elementos à página que nem sequer existiam em sua página HTML original. **Você pode modificar elementos para alterar seu comportamento, layout, aparência e conteúdo.** Teoricamente, embora isso raramente seja uma prática recomendada, você pode renderizar uma página HTML em branco para o navegador, criar a página inteira usando JavaScript e produzir exatamente os mesmos resultados. Ter esse poder sobre suas páginas da web é muito emocionante, mesmo depois que eles são renderizados. Você começa selecionando itens no DOM para obter uma referência a eles.

Selecionando itens no DOM

Para manipular o DOM, você precisa saber como acessá-lo e obter referências aos elementos que deseja manipular. Na próxima seção você verá

como alterar o DOM, mas primeiro você precisa para obter elementos do DOM para que você possa trabalhar com eles.

OBSERVAÇÃO O MODELO DE OBJETO DE DOCUMENTO COMO UMA ÁRVORE DE FAMÍLIA

O DOM é essencialmente uma coleção de nós dispostos em uma árvore. Todos os nós são relacionados um para o outro. Eles são uma grande família feliz de crianças, irmãos, pais, avós, Netos, e assim por diante. Esta essência de uma árvore de família representa a hierarquia do DOM e é importante compreender como você manipula o DOM através de código.

Você tem algumas opções quando se trata de usar o JavaScript para acessar o DOM. Você pode acessar elementos do DOM através de um objeto global fornecido pelo navegador, chamado documento, ou através dos próprios elementos depois de obter uma referência a um. A Tabela 1-2 descreve os métodos nativos principais usados para selecionar elementos no DOM.

Tabela 1 – 2

Método	Descrição de uso
getElementById	Obtém um elemento individual na página por seu valor de atributo de id exclusivo
getElementsByClassName	Obtém todos os elementos que têm a classe CSS especificada aplicada a eles
getElementsByTagName	Obtém todos os elementos da página que têm o nome de tag especificado ou nome do elemento
querySelector	Obtém o primeiro elemento filho encontrado que corresponde aos critérios de seletor CSS fornecidos
querySelectorAll	Obtém todos os elementos filho que correspondem aos critérios de seletor CSS fornecidos

Para a maior parte, os métodos são simples de usar. Nesta seção, você começa com uma estrutura de documento HTML simples que você usará em muitos outros exemplos neste livro para destacar vários conceitos. Crie uma

página da Web com a marcação HTML na Listagem 1-2 para prosseguir com os exemplos a seguir.

```
<body>
  <div id="outerDiv">
    <p class='mainPara'>Main Paragraph</p>
    <ul>
      <li>First List Item</li>
      <li>Second List Item</li>
      <li>Third List Item</li>
      <li>Fourth List Item</li>
    </ul>
    <div id="innerDiv">
      <p class='subPara' id='P1'>Paragraph 1</p>
      <p class='subPara' id='P2'>Paragraph 2</p>
      <p class='subPara' id='P3'>Paragraph 3</p>
      <p class='subPara' id='P4'>Paragraph 4</p>
    </div>
    <table>
      <tr>
        <td>Row 1</td>
      </tr>
      <tr>
        <td>Row 2</td>
      </tr>
      <tr>
        <td>Row 3</td>
      </tr>
      <tr>
        <td>Row 4</td>
      </tr>
    </table>
```

```

        <td>Row 5</td>
    </tr>
</table>
<input type="text"/><input type="submit" value="Submit"/>
</div>
</body>

```

Esta página de exemplo é muito simples, mas serve para demonstrar várias maneiras de acessar elementos através de código. Para demonstrar essa funcionalidade, você precisa de um ponto de entrada. Adicione o seguinte bloco de script à seção head da página da Web:

```

<script>
window.onload = function () {
...
}
</script>

```

Isso deve parecer familiar, mas se não, você vai rever os conceitos mais tarde. Por agora, este código essencialmente diz o tempo de execução para executar o código **após a janela terminar o carregamento**. Você pode usar seu código para experimentar como os vários métodos listados na Tabela 1-2 nesta função, começando com getElementById. O método getElementById **retorna o elemento na página que corresponde ao ID específico valor que você passa para ele**. Retorna **null** se nenhum elemento na página tem o ID especificado. Cada elemento na página deve ter um ID exclusivo. Por exemplo, se você quiser fazer referência ao elemento <div> com o ID outerDiv, use o seguinte código:

```

var element = document.getElementById("outerDiv");
alert(element.innerHTML);

```

O método de alerta JavaScript, que exibe uma caixa de mensagem, é usado aqui para mostrar se você realmente acessou o DOM com êxito. O alerta não é tudo o que é útil no mundo real, mas é para fins de desenvolvimento.

Quando você executar a página, observe a caixa de mensagem do navegador com todo o conteúdo innerHTML do <div> que você selecionou fora do DOM com seu código (consulte a Figura 1-14).

FIGURA 1-14 Um alerta JavaScript demonstrando o acesso bem-sucedido ao DOM

Agora que você conseguiu uma referência ao seu <div>, você pode fazer qualquer coisa que você quer dinamicamente-exatamente como você poderia ter definido ou aplicado tais mudanças para ele estatisticamente. O método `getElementById` é ótimo quando você conhece o ID de um elemento específico na página com a qual você deseja trabalhar, mas em outros casos você pode querer fazer algo com todos os elementos de um tipo específico - por exemplo, todos os elementos de parágrafo. Nesse caso, o método `getElementsByTagName` é mais apropriado. Você pode usar o código a seguir para obter uma referência a todos os elementos <p>:

```
<script>
window.onload = function () {
    var paragraphs = document.getElementsByTagName("p");
    alert(paragraphs.length);
}
</script>
```

Nesse código, o objeto retornado do método `getElementsByTagName` é um pouco diferente; É um tipo especial que atua como um wrapper para todos os elementos que correspondem ao seu parâmetro, chamado `NodeList`. Este objeto não é especialmente útil por si só. Na verdade, ele realmente não fornece nada útil além de um comprimento, que permite que você saiba quantos itens ele contém e a capacidade de acessar cada item individual. No exemplo anterior, o alerta JavaScript exibe quantos itens foram retornados na lista (`paragraphs.length`). Você pode ver que o método retornou todos os cinco elementos <p> na página, como mostrado na Figura 1-15.

FIGURA 1-15 Uma mensagem mostrando o número de elementos <p>

Da mesma forma que você poderia usar o método `getElementsByTagName` para obter todos os elementos do mesmo tipo, você pode usar o método `getElementsByClassName` para obter elementos da mesma classe CSS. Isso é útil quando você tem muitos elementos com o mesmo estilo, mas talvez queira modificá-los em tempo de execução. Esse método também retorna um `NodeList`. O snippet a seguir demonstra este uso:

```
<script>
window.onload = function () {
    var paragraphs = document.getElementsByClassName("subPara");
    alert("<p> elements with class subPara: " + paragraphs.length);
}
</script>
```

A Figura 1-16 mostra a saída desse script.

FIGURA 1-16 Uma mensagem mostrando o número de <p> elementos com o nome de classe especificado subPara.

Este exemplo adiciona um pouco mais de texto para a caixa de mensagem para que ele se pareça diferente do exemplo anterior, mas a idéia é o mesmo.

Todos os elementos <p> com a classe subPara atribuída a eles foram retornados em um NodeList. Você pode ver que a chamada retornou quatro elementos HTML. Ao selecionar elementos no DOM por nome de classe, o NodeList contém todos os elementos cuja classe corresponde à classe especificada, e não apenas elementos do mesmo tipo. Se, por exemplo, você atribuiu a classe subPara a um dos seus elementos <div> e, em seguida, executou a função novamente, o NodeList retornado conteria os quatro elementos <p> eo elemento <div> porque todos têm a mesma classe. Isso é importante quando você pretende iterar sobre os elementos e fazer algo para eles. Na Figura 1-17, o mesmo código JavaScript é executado, mas com um atributo de classe subPara adicionado a um elemento <div>.

FIGURA 1-17 O mesmo script é executado com um <div> atribuído ao nome da classe subPara

A caixa de mensagem agora está realmente incorreta, porque o NodeList contém um único elemento <Div> e os quatro elementos <p>. Tenha este comportamento no método **GetElementsByClassName**.

Todos os métodos que você olhou até agora para encontrar elementos no DOM fornecem para uma finalidade específica. Se você quiser um **único elemento por seu ID exclusivo, use o método getElementById;**

Se você quiser encontrar **um elemento ou todos os elementos de uma classe CSS específica, use o método getElementsByClassName.**

Agora olhe alguns exemplos que usam os métodos muito mais flexíveis de **querySelector** e **querySelectorAll**. Os métodos **querySelector** e **querySelectorAll** permitem que você alcance a maior parte do que já fez com os outros métodos. Ambos os **métodos tomam um parâmetro na forma de um Seletor CSS**. O método **querySelector** **retorna o primeiro elemento encontrado que coincide com os critérios de seleção passados a ele**, enquanto o método **querySelectorAll** **retorna todos os elementos que correspondem aos critérios de seleção passados**. Os elementos ainda são retornados na forma de um objeto NodeList. Ambos os métodos existem não só no próprio documento, mas também em cada elemento. Portanto, quando você tem uma referência a um elemento, você pode usar esses métodos para pesquisar seus filhos sem ter que percorrer todo o documento. Você pode ver alguns exemplos mais simples nesta seção.

Para encontrar todos os elementos <p> em uma página, você pode usar esta sintaxe:

```
document.querySelectorAll("p");
```

Para encontrar um elemento por seu ID exclusivo, você pode usar esta sintaxe:

```
document.querySelector("#outerDiv");
```

Essas duas linhas em seu arquivo HTML e experimente-as. Você vai explorar muito mais avançada e interessante no Capítulo 4. Por enquanto, você pode usar o que você viu sobre como encontrar elementos no DOM para aplicar esse conhecimento para adicionar ou modificar o DOM através do código:

dica de exame

JQuery é provavelmente a biblioteca mais popular disponível até à data para simplificar e estender as capacidades principais do JavaScript. Embora jQuery não seja uma tecnologia da Microsoft, é essencialmente um padrão da indústria e totalmente suportado pela Microsoft. Como tal, os desenvolvedores web hoje são geralmente entendido como ter uma compreensão de usar jQuery intercambiáveis com núcleo JavaScript.

O exame esperará que você possa usar jQuery efetivamente no lugar dos métodos de seletor de objeto de documento.

Alterando o DOM

Ter acesso ao DOM por meio de JavaScript pode ser usado para fornecer experiência de usuário rica ao criar páginas da web dinâmicas. Até agora, tudo o que você fez é obter referências aos elementos, o que não é particularmente útil por si só. A finalidade de recuperar elementos do DOM é poder fazer algo com eles. Nesta seção, você verá como manipular o DOM usando código JavaScript para **adicionar e remover** itens. Depois de ter uma referência a um elemento do contêiner, você pode adicionar elementos filho dinamicamente.

Você pode remover elementos dele ou simplesmente ocultar elementos. Quando você remove um elemento do DOM, ele é ido. **Então, se você quiser tornar algo invisível para o usuário, ser capaz de usá-lo novamente mais tarde, você pode simplesmente escondê-lo usando o CSS apropriado em vez de removê-lo.** Aqui está um exemplo:

```
var element = document.getElementById("innerDiv");  
alert(element.innerHTML);  
document.removeChild(element);  
var afterRemove = document.getElementById("innerDiv");  
alert(afterRemove);
```

O primeiro alerta mostra corretamente a propriedade innerHTML do innerDiv, mas o código nunca atinge o segundo alerta. Em vez disso, o método

getElementById lança um erro porque o ID do elemento especificado não existe mais no documento. Esteja ciente de vários métodos quando se trata de adicionar elementos e removê-los do DOM.

O primeiro método a ser observado é `document.createElement`. Utilize este método do objecto de documento para criar um novo elemento HTML. O método recebe um `único parâmetro` - o nome do elemento que você deseja criar. O código a seguir cria um novo elemento `<article>` para usar em sua página:

```
var element = document.createElement("article");  
element.innerText = "My new <article> element";
```

Este novo elemento `<article>` não está visível para ninguém neste momento; Ele simplesmente existe no DOM para uso em sua página. Porque você não tem muita necessidade de criar elementos, mas então não usá-los, vamos olhar para os métodos disponíveis para obter o seu novo elemento `<article>` em sua página. O primeiro desses métodos é `appendChild`. Utilize este método para adicionar um novo elemento HTML à coleção de elementos subordinados pertencentes ao contêiner de chamada. O nó é adicionado ao final da lista de filhos que o nó pai já contém. O método `appendChild` existe no objeto de documento, bem como em outros elementos de recipiente HTML. Ele retorna uma referência para o nó recém-adicionado. Este exemplo anexa um novo elemento `<article>` ao `outerDiv`:

```
var outerDiv = document.getElementById("outerDiv");  
var element = document.createElement("article");  
element.innerText = "My new <article> element";  
outerDiv.appendChild(element);
```

Como a maioria dos outros métodos explicados nesta seção, o método `appendChild` retorna uma referência ao novo elemento anexado aos elementos filho. Esta é uma boa maneira de garantir que você sempre tenha uma referência a um elemento para uso futuro, especialmente ao excluir elementos. Também permite simplificar ou reestruturar o código. O seguinte código obtém o mesmo resultado:

```
var element = document.getElementById("outerDiv").appendChild(document.  
createElement("article"));  
element.innerText = "My new <article> element";
```

A Figura 1-18 mostra a saída desse código.

FIGURA 1-18 Um novo elemento `<article>` anexado ao final da página

Você pode ver que o elemento `<article>` foi colocado no final da sua página. O `AppendChild` sempre adiciona o novo elemento ao final da lista de nó filho do elemento pai. Para inserir o novo elemento `<article>` em algum lugar mais preciso, o método `insertBefore` poderia ser mais adequado. Este método leva

dois parâmetros: o novo elemento propriamente dito eo nó antes do qual você deseja anexar o novo elemento. Por exemplo, para inserir o novo artigo antes do elemento innerDiv, você poderia escrever o seguinte código:

```
var element = document.getElementById("outerDiv").insertBefore(  
document.createElement("article"),  
document.getElementById("innerDiv"));  
element.innerText = "My new <article> element";
```

Este exemplo usa o método `getElementById` para obter uma referência ao nó antes do qual você queira inserir o elemento `<article>` no DOM. Você pode usar outras ferramentas para tornar esse código mais simples em alguns casos, dependendo da estrutura do documento. Cada elemento ou nó tem as propriedades listadas na Tabela 1-3 para ajudar a obter referências aos nós mais comuns ao trabalhar com o DOM.

Propriedade	Descrição
childNodes	Uma coleção de todos os nós filhos do elemento pai.
firstChild	Uma referência ao primeiro nó filho na lista de nós filhos do nó pai.
lastChild	Uma referência ao último nó filho na lista dos nós filhos do pai nó.
hasChildNodes	Uma propriedade útil que retorna true se o elemento pai tiver qualquer nó filho em tudo.Uma boa prática é verificar esta propriedade antes de acessar outras propriedades, como <code>FirstChild</code> ou <code>lastChild</code> .

Para obter um exemplo destas propriedades, pode alterar o código anterior para elemento `<article>` como o primeiro elemento no elemento innerDiv:

```
var inner = document.getElementById("innerDiv");  
  
var element =  
inner.insertBefore(document.createElement("article"),inner.firstChild);  
  
element.innerText = "My new <article> element";
```

Este código produz a saída mostrada na Figura 1-19.

FIGURA 1-19 O novo elemento `<article>` inserido como o primeiro filho de um elemento `<div>`

Seu elemento `<article>` agora está posicionado como o primeiro elemento filho do innerDiv elemento. Experimente as outras propriedades para se familiarizar com o modo como se comportam. Cada elemento que pode ter

elementos filhos suporta toda essa funcionalidade; No entanto, se você tentar inserir elementos em um nó que não oferece suporte a nós filho - como um , por exemplo - o interpretador lança um erro em tempo de execução.

Assim como você pode adicionar novos elementos ao DOM através do código, você também pode remover elementos do DOM usando código. Nesta seção você olha para os métodos disponíveis para fazer exatamente isso, chamado `removeChild` e `removeNode`. O método `removeChild` remove um nó filho do contêiner de chamada. Esse método existe no objeto de documento, bem como outros elementos de recipiente HTML.

O `removeChild` retorna uma referência ao nó removido. Isto é especialmente útil se você planeja retornar esse nó para o DOM - talvez em resposta a alguma outra interação do usuário com a página. Lembre-se, no entanto, que se você não manter a referência retornada para o nó removido, você não tem nenhuma maneira de adicionar o elemento de volta sem totalmente recriá-lo. O exemplo a seguir remove o primeiro <p> elemento do seu elemento `innerDiv`:

```
var innerDiv = document.getElementById("innerDiv");  
var p = innerDiv.removeChild(document.getElementById("P1"));
```

Este código fornece a saída na Figura 1-20. Você pode ver que o primeiro <p> elemento foi removido. Como você capturou o elemento removido na variável `p`, você poderia usá-lo mais tarde se quisesse colocar o elemento <p> em outro lugar.

FIGURA 1-20 Remoção do primeiro elemento <p> pelo método `removeChild`

Outro método útil para remover nós ou elementos é `removeNode`, que leva um Parâmetro booleano. Definir o parâmetro como `true` diz ao método para fazer uma remoção profunda, o que significa que todas os filhos também são removidos. O código a seguir demonstra isso:

```
var innerDiv = document.getElementById("innerDiv");  
innerDiv.removeNode(true);
```

A Figura 1-21 mostra que quando esse código é executado no navegador, o elemento `innerDiv` foi removido.

FIGURA 1-21 Usando o método `removeNode` para remover o nó <div>

Agora, suponha que você deseja alterar o conteúdo da página de forma mais dramática - talvez até reescrevendo todo o conteúdo HTML. Isso é completamente possível com as técnicas que você viu até agora, mas você não tentou alguns métodos ainda: `replaceNode` e `replaceChild`.

Esses dois métodos operam da mesma maneira que `removeNode` e `removeChild` em termos de parâmetros que eles tomam e quais elementos eles afetam. A diferença, no entanto, é que você pode substituir o elemento de destino por um elemento completamente novo.

O código a seguir converte todos os seus parágrafos internos para elementos de âncora e adiciona quebras de linha, porque você não obtém esses automaticamente como você faz a partir do elemento <p>:

```
var innerDiv = document.getElementById("innerDiv");
var newDiv = document.createElement("div");
for (var i = 0; i < innerDiv.childNodes.length; i++) {
    var anchor = newDiv.appendChild(document.createElement("a"));
    anchor.setAttribute("href", "http://www.bing.ca");
    anchor.text = innerDiv.childNodes[i].textContent;
    newDiv.appendChild(document.createElement("br"));
}
innerDiv.replaceNode(newDiv);
```

Este código produz a saída do navegador como mostrado na Figura 1-22.

FIGURA 1-22 Convertendo todos os elementos <p> para elementos <a>

Todos os seus parágrafos de texto simples agora são exibidos como hiperlinks. Seu elemento innerDiv original desaparecido e não está mais no DOM. Sua única referência a ele está dentro do código JavaScript. Você precisaria manter essa referência se pretender coloca-lo novamente no DOM.

Além disso, como o código não atribuiu o novo elemento <div> a um único id, a única maneira para obter uma referência a ele no DOM é através de sua referência de código existente. Por esta razão, **uma prática recomendada é sempre dar seus novos elementos um id único.** Se as variáveis JavaScript ficarem fora do escopo antes de inseri-las no documento, você perde completamente as referências aos seus elementos.

Nesta seção, você viu como acessar elementos HTML usando JavaScript para manipular O DOM no navegador. Agora você pode recuperar referências aos elementos ou nós que compõem o documento HTML, bem como modificar, adicionar e remover elementos no documento HTML.

Em seguida, você verá como implementar controles de mídia em suas páginas.

Implementando controles de mídia

A incorporação de elementos multimídia em páginas da Web não é um conceito novo. Esta capacidade tem sido em torno de um longo tempo e apresentou desafios em várias situações. Um dos principais desafios tem sido a dependência de um objeto de terceiros integrado com o navegador para meios de comunicação. **Nesta seção, você observa dois novos elementos adicionados à especificação HTML5 que trabalham com multimídia nativamente no**

navegador da Web e com o JavaScript. Você também examina os elementos <video> e <audio>.

Usando o elemento <video>

Incorporação de vídeo em uma página web tornou-se muito popular, e muitos sites agora incluem um elemento de vídeo em seu design. HTML5 tem incluído vídeo em suas páginas muito mais fácil do que era anteriormente. Aqui, você aprende sobre o novo elemento <video> fornecido pelo HTML5 e veja os atributos e eventos disponíveis que você pode usar para controlar o vídeo de forma declarativa, através de HTML estático ou dinamicamente usando JavaScript.

Incorporar um vídeo na página é tão simples como adicionar a seguinte marcação:

```
<video src="samplevideo.mp4" autoplay> </video>
```

Esse é o mínimo. Entretanto, você sabe que o mínimo desencapado é raramente útil para um Web site profissional projetado. Você precisa trabalhar com mais propriedades e eventos. Você também precisa considerar o suporte do navegador para vários formatos de vídeo. Primeiro, você precisa examinar os atributos-chave disponíveis para uso no elemento <video>, conforme listado na Tabela 1-4.

Atributo	Descrição
Src	Este atributo especifica o vídeo a ser reproduzido. Pode ser um recurso local dentro do seu próprio site ou algo exposto através de um URL público na Internet.
Autoplay	Esse atributo diz ao navegador para começar a reproduzir o vídeo assim que ele é carregado. Se este atributo é omitido, o vídeo é reproduzido apenas quando informado através de JavaScript.
Controls	Este atributo diz ao navegador para incluir seus controles de vídeo embutidos, como play e pausa. Se isso for omitido, o usuário não tem nenhuma maneira visível para reproduzir o conteúdo. Você poderia usar a reprodução automática ou fornecer algum outro mecanismo através de JavaScript para reproduzir o vídeo.

Height/width	Estes atributos controlam a quantidade de espaço que o vídeo ocupará na página, a omissão dessas causas faz com que o vídeo seja exibido em seu tamanho nativo.
Loop	Esse atributo indica ao navegador para reproduzir continuamente o vídeo depois que ele for concluído. E se este atributo é omitido, o vídeo pára depois que ele é reproduzido completamente.
poster	Este atributo especifica uma imagem a ser exibida no local alocado ao vídeo até que o usuário comece a reproduzir o vídeo. Use isso quando não estiver usando a reprodução automática. É muito útil para fornecer uma imagem profissional ou arte para representar o vídeo. Se for omitido, o cartaz aparece no primeiro quadro do vídeo.

TABELA 1-4 Atributos disponíveis no elemento <video>

Com todas essas novas informações sobre os atributos disponíveis, você pode fornecer um pouco mais detalhes no elemento <video> para controlar como você gostaria que se comportasse:

```
<video src="samplevideo.mp4" controls poster="picture.jpg" height="400"
width="600">
</video>
```

O elemento <video> anterior especifica que ele deve exibir inicialmente uma imagem de cartaz, define os parâmetros de altura e largura e indica que os controles padrão devem estar disponíveis.

A ausência de um atributo loop significa que quando o vídeo estiver terminado, ele não deve repetir automaticamente. E a ausência de um atributo de reprodução automática informa o navegador que você não deseja que o vídeo comece a ser reproduzido automaticamente;

Em vez disso, deve esperar até que o usuário invoque a reprodução. Operação com os controles ou até que você invoque a operação de reprodução com JavaScript. Quando você incluir os controles padrão, o usuário obtém um conjunto básico. A Figura 1-23 mostra o aspecto dos controles padrão no Internet Explorer.

FIGURA 1-23 Os controles de mídia padrão do Internet Explorer

Da esquerda para a direita, os controles padrão fornecem um botão de reprodução que muda para um botão de pausa enquanto o vídeo está sendo

reproduzido. Um temporizador mostra a posição atual do vídeo e quanto tempo que permanece no vídeo. Uma barra deslizante permite que os usuários naveguem até um ponto específico no vídeo. O botão de controle de áudio exibe uma barra deslizante de volume quando pressionado e, finalmente, na extrema direita, é um controle que permite aos usuários exibir o vídeo em tamanho de tela cheia.

Até agora, tão bom para os usuários do Internet Explorer. Mas você também precisa garantir que seu vídeo será reproduzido com êxito em outros navegadores. O problema é que nem todos os navegadores suportam todos os formatos de vídeo. Tenha isto em mente ao implementar seus elementos <video>;

O que cada navegador suporta pode (e vai) mudar também. Você precisa garantir que você forneça opções para o navegador para que ele possa escolher qual o formato de vídeo para reproduzir. Se você não tiver todos os formatos de vídeo compatíveis apropriados e sua página for visitada por um visitante com um navegador que não pode reproduzir o formato de vídeo que você tem, você também precisa fornecer uma alternativa ou pelo menos as informações que o navegador do usuário não suporta este vídeo. O código a seguir demonstra isso:

```
<video controls height="400" width="600" poster="picture.jpg">
```

```
<source src="samplevideo.ogv" type="video/ogg"/>
```

```
<source src="samplevideo.mp4" type="audio/mp4"/>
```

```
<object>
```

```
<p>Video is not supported by this browser.</p>
```

```
</object>
```

```
</video>
```

Este exemplo removeu o atributo src do elemento <video> e acrescentou <Source> em vez disso. O elemento <video> suporta vários elementos <source>, portanto você pode incluir um para cada tipo de vídeo. Um navegador passa pelos elementos <source> de cima para baixo e reproduz o primeiro que suporta.

Observe que o exemplo também tem um elemento <object> para cobrir a possibilidade de que o navegador do cliente não tem suporte para o elemento <video>. Nesses casos, você pode ter uma versão em Flash do vídeo para reproduzir;

Mas se nenhuma outra versão do vídeo estiver disponível para ser reproduzida, você pode apenas exibir uma mensagem de que o vídeo não é suportado, como mostrado no snippet de código. Os navegadores que não suportam o elemento <video> ignoram o elemento completamente, mas mostram o objeto <object> elemento que eles entendem. Isso permite que os

navegadores mais antigos "recuem" nos métodos anteriores de exibição de vídeo, garantindo que você possa alcançar o maior número de usuários possível.

Finalmente, o elemento <p> é um último recurso para fornecer pelo menos algumas informações aos usuários que um vídeo é suposto estar rodando aqui, mas que seu navegador não o suporta.

dica de exame

Se o navegador suportar o elemento de vídeo HTML5, ele não mostrará o retorno. Caso, certifique-se de que você tem o elemento <source> válido especificado para esse navegador. Se você não fizer isso, o contêiner de vídeo mostra um erro no lugar da barra de controle, dizendo que um link inválido ou arquivo é especificado.

As vezes, ter mais controle sobre as coisas é bom, ou talvez você simplesmente não gosta do olhar e sentir os controles padrão. É aqui que entra o JavaScript. Você pode criar sua própria barra de controle e substituir seus próprios botões de controle para permitir que os usuários controlem o vídeo.

O exemplo a seguir adiciona alguns elementos de imagem personalizados à página e acciona algum JavaScript para controlar o vídeo:

```
<head>

<style>
img:hover {
  cursor: pointer;
}
</style>

<script>
var video;

window.onload = function () {
  video = document.getElementById("sampleVideo");
}

function play() {
  video.play();
}

function pause() {
  video.pause();
}
```

```

function back() {
video.currentTime -= 10;
}
</script>
</head>
<body>
<table>
<tr>
<td>
<video height="400" width="600" id="sampleVideo">
<source src="samplevideo.mp4" type="audio/mp4"/>
</video>
</td>
<td>
<br/>
<br/>
<br/>
</td>
</tr>
</table>
</body>

```

Este HTML produz os controles de mídia mostrados na Figura 1-24.

FIGURA 1-24 Uma barra de controle de mídia personalizada

Como você pode ver, o código criou uma pequena barra de controle personalizada e posicionou-a direito do quadro de vídeo. O elemento <video> oferece muitos métodos. A Tabela 1-5 descreve as mais comuns.

TABELA 1-5 Métodos e propriedades no objeto <video>

Você aprendeu tudo sobre como exibir vídeo em suas páginas da web. Agora vire sua atenção para reproduzir sons usando o elemento <audio>.

Metodo/Propriedade	Descrição
Play()	Reproduz o vídeo a partir da sua posição atual.
Pause()	Pausa o vídeo em sua posição atual.

Volume()	Permite que o usuário controle o volume do vídeo.
currentTime	Representa a posição atual do vídeo. Aumente ou diminua esse valor para Avançar ou retroceder no vídeo.

Usando o elemento <audio>

O elemento <audio> é essencialmente idêntico ao elemento <video>. Ele tem todos os mesmos atributos e os mesmos métodos. **A única diferença real é como ele é exibido no navegador.**

Como nenhum vídeo está disponível para exibição, o **elemento <audio> não ocupa espaço na tela**. No entanto, você pode mostrar os controles padrão - ou você pode escolher novamente não mostrar os controles padrão e criar seu próprio mecanismo para controlar o áudio, Elementos de interface ou por trás das cenas em JavaScript. Aqui está um exemplo do que uma <audio> declaração se parece em sua página da web:

```
<audio controls>
<source src="sample.mp3" type="audio/mp3"/>
<source src="sample.ogg" type="audio/ogg"/>
<p>Your browser does not support HTML5 audio.</p>
</audio>
```

Esse HTML fornece a saída no Internet Explorer mostrada na Figura 1-25.

FIGURA 1-25 Os controles de áudio padrão no Internet Explorer

A Figura 1-25 mostra a saída do elemento <audio> quando você opta por usar os controles. Da esquerda para a direita, você obtém um botão de pausa / reprodução, o contador, uma barra de progresso, o tempo total no áudio e uma barra deslizante de volume. Como nenhum outro espaço de tela é necessário como nas amostras de vídeo, **o elemento <audio> não possui propriedades de altura ou largura disponíveis**. Se você não gosta da barra de controle de áudio incorporada, você pode optar por não incluí-la em sua declaração e, em vez disso, criar uma barra de controle personalizada que atenda às suas necessidades.

Os elementos <audio> e <video> são muito semelhantes. O ponto-chave a respeito desses elementos é que eles fornecem uma maneira padronizada de representar mídia em páginas HTML para simplificar a leitura do código HTML e saber exatamente o que a página deve estar fazendo.

Agora que você sabe como usar áudio e vídeo em suas páginas da web, você pode dar atenção ao uso de gráficos.

Implementação de gráficos com HTML5 <canvas> e SVG

HTML5 fornece um novo mecanismo para trabalhar com gráficos em suas páginas da web. A especificação HTML5 introduz o elemento da página da Web <canvas>, que fornece uma tela em branco na qual você pode desenhar dinamicamente. **Você pode desenhar linhas, texto e imagens na tela e manipulá-las com JavaScript.**

Adicionar uma tela à sua página é tão simples como declarar um no HTML. **O elemento <canvas> é similar ao elemento <div>.** No entanto, é um contêiner para **elementos gráficos** em oposição a elementos baseados em texto. Aqui está a marcação para um elemento <canvas>:

```
<canvas id="drawingSurface" width="600" height="400">
```

```
  Your browser does not support HTML5.
```

```
</canvas>
```

O HTML é muito simples. **Basta definir um <canvas> e especificar um tamanho.** Além disso, se o navegador do usuário **não suportar o elemento <canvas>**, **você pode colocar texto de retorno dentro do elemento <canvas> para ser exibido em seu lugar.** Quando você executar este HTML no navegador, você deve notar absolutamente nada! Isso ocorre porque - assim como com um elemento <div> ou qualquer outro container - o elemento <canvas> não tem visibilidade padrão; em outras palavras, ele é visível, mas é branco sem bordas e, portanto, é invisível em uma página HTML em branco. O próximo exemplo adiciona um estilo simples ao elemento <canvas> para que você possa ver suas bordas:

```
<style>
```

```
  canvas {  
    border: 1px solid black;  
  }
```

```
</style>
```

Agora você pode ver sua tela, que deve olhar Figura 1-26.

FIGURA 1-26 Um elemento <canvas> em branco

Uma tela em branco ainda não é terrivelmente excitante. Mas agora que você tem uma tela básica em funcionamento, você pode trabalhar com todos os vários métodos para criar gráficos na tela. Para fazer isso, você **deve criar um evento onload para sua janela** (como você tem em exemplos anteriores) **para encapsular seu código e fazer com que os gráficos sejam renderizados quando a página é carregada.** Para desenhar na tela, você precisa entender o sistema de coordenadas que a tela usa. A tela fornece um sistema de coordenadas fixo (x, y) no qual o canto superior esquerdo da tela é (0,0). Neste caso, o canto inferior esquerdo da tela é (0,400), o canto superior direito é (600,0) eo canto

inferior direito é (600,400). Você deve ser bastante usado para este tipo de sistema porque ele corresponde ao sistema de coordenadas da janela do navegador, com (0,0) no canto superior esquerdo. No entanto, a posição da tela na janela do navegador é irrelevante para o rawing

Métodos que você usa para desenhar na tela. As coordenadas para desenhar na tela são sempre baseadas nas coordenadas dentro da tela propriamente dita, onde o pixel superior esquerdo é (0,0). Como com qualquer elemento HTML, para trabalhar com ele através de código que você precisa para obter uma referência a ele em seu JavaScript. Comece escrevendo o seguinte código em sua página:

```
window.onload = function () {  
var drawingSurface = document.getElementById("drawingSurface");  
var ctx = drawingSurface.getContext("2d");  
}
```

No código anterior, você obtém uma referência ao seu elemento canvas seguido por uma referência a um contexto "2d". O contexto é um objeto que fornece os métodos de API que você usa para desenhar na tela. Agora, o <canvas> suporta apenas um contexto 2d, mas você pode esperar para ver um contexto 3d no futuro. Tendo adquirido uma referência ao contexto, agora você pode começar a olhar para os vários métodos para desenhar em sua tela.

Desenhando linhas

No nível mais básico, você pode desenhar linhas na tela com o objeto de contexto 2d que você está fazendo referência. O objeto de contexto fornece os seguintes métodos para desenhar linhas, conforme listado na Tabela 1-6.

TABELA 1-6 Métodos para desenhar linhas

Metodo	Descrição
beginPath	Reinicia / inicia um novo caminho de desenho
moveTo	Move o contexto para o ponto definido no método beginPath
lineTo	Define o ponto final de destino para a linha
stroke	Traça a linha, o que torna a linha visível

Com essa informação, na sua forma mais simples, você pode desenhar uma linha em toda a sua tela como esta:

```
ctx.beginPath();  
ctx.moveTo(10, 10);
```

```
ctx.lineTo(225, 350);  
ctx.stroke();
```

Este código produz a linha na tela mostrada na Figura 1-27.

FIGURA 1-27 Uma linha desenhada na tela

No ponto onde a linha termina, você pode continuar extraindo mais linhas adicionando mais métodos `lineTo`, como no exemplo a seguir:

```
ctx.beginPath();  
ctx.moveTo(10, 10);  
ctx.lineTo(225, 350);  
ctx.lineTo(300, 10);  
ctx.lineTo(400, 350);  
ctx.stroke();
```

Execute esse código e observe a saída. Você obtém um gráfico que se assemelha à Figura 1-28. Você pode usar linhas retas desta maneira ao traçar pontos conectados em um gráfico de linha, para

exemplo.

FIGURA 1-28 Uma polilinha desenhada na tela

Explorando o método de AVC em mais profundidade vale a pena o esforço. Se você estiver criando um gráfico, talvez queira **alterar a cor de suas linhas para que elas se destaquem do eixo. Você pode querer alterar a espessura.** Você faz isso alterando algumas propriedades no objeto de contexto antes de chamar o método de **stroke**:

```
ctx.lineWidth = 5;  
ctx.strokeStyle = '#0f0';
```

A propriedade **lineWidth** aceita um valor que **determina a largura da linha.** A propriedade **strokeStyle** permite alterar a cor da linha. Esta propriedade aceita todos os formatos de estilo comuns para especificar cores em HTML, incluindo valores hexadecimais ou cores nomeadas. Essas mudanças produzem uma saída nova e mais atraente, como mostrado na Figura 1-29.

FIGURA 1-29 A polilinha renderizada com uma cor diferente

Você também pode experimentar com a propriedade **lineCap**, que aceita alguns valores que controlam como o final da linha que será renderizado. Por exemplo, você pode definir a propriedade **lineCap** para rodar para dar a linha **uma tampa arredondada.** A aplicação de uma tampa afeta o comprimento da linha. A tampa é adicionada ao final da linha e seu comprimento corresponde ao que você definiu para a largura da linha. Neste exemplo, a linha tem uma largura

de 5. Com uma tampa redonda definida na linha, o comprimento total da linha seria estendido por 5. Você já viu tudo o que você precisa saber sobre como trabalhar com linhas retas. Agora volte sua atenção para curvas.

Desenhando curvas

Curvas de desenho é um pouco mais envolvido porque você tem mais parâmetros a considerar.

A Tabela 1-7 lista os métodos usados ao trabalhar com curvas na tela HTML5.

TABELA 1-7 Métodos para desenhar curvas

Metodo	Descrição
arc	Um arco padrão baseado em um ângulo inicial e final e um raio definido
quadraticCurveTo	Um arco mais complexo que permite controlar a inclinação da curva
bezierCurveTo	Outro arco complexo que você pode inclinar

Cada método de desenho pode ter estilos aplicados a ele, assim como os exemplos de linha. Você pode controlar as propriedades `lineWidth`, `strokeStyle` e `lineCap` para alterar como as curvas são exibidas. Comece criando alguns arcos básicos em sua tela. O método `arc` leva os parâmetros listados na Tabela 1-8.

TABELA 1-8 Parâmetros necessários para desenhar um arco

Parametro	Descrição
X, Y	Os dois primeiros parâmetros são as coordenadas X e Y para o centro do círculo.
radius	O terceiro parâmetro é o raio. Este é o comprimento da distância Ponto central do círculo para a curva.
startAngle, endAngle	Os quarto e quinto parâmetros especificam os ângulos inicial e final da Arco a ser desenhado. Isto é medido em radianos, não em graus.
counterclockwise	O parâmetro final especifica a direção de desenho do arco.

Adicione o seguinte código à sua página:

```
ctxt.beginPath();
```

```
ctxt.arc(150,100,75,0,2 * Math.PI, false);
```

```

ctxt.lineWidth = 25;
ctxt.strokeStyle = '#0f0';
ctxt.stroke();
ctxt.beginPath();
ctxt.arc(450, 100, 75, 1.5 * Math.PI, 2 * Math.PI, false);
ctxt.lineWidth = 25;
ctxt.strokeStyle = 'blue';
ctxt.stroke();
ctxt.beginPath();
ctxt.arc(150, 300, 75, 1 * Math.PI, 1.5 * Math.PI, false);
ctxt.lineWidth = 25;
ctxt.strokeStyle = '#0ff';
ctxt.stroke();
ctxt.beginPath();
ctxt.arc(450, 300, 75, .5 * Math.PI, 1 * Math.PI, false);
ctxt.lineWidth = 25;
ctxt.strokeStyle = '#f00';
ctxt.stroke();

```

Este exemplo de código desenha quatro arcos: um círculo completo seguido por três círculos de quarto, cada um com um estilo diferente. Observe que algumas fórmulas matemáticas são especificadas nos parâmetros para a matemática arc. Este é necessário para obter o valor em radianos porque os parâmetros para startAngle e endAngle são especificados em radianos, e não em graus. O código produz o desenho mostrado na figura 1-30.

FIGURA 1-30

Desenho de arcos na tela em cores diferentes O exemplo anterior demonstrou um **arco simples**. Agora olhe para o **próximo método de arco, o quadraticArc**. A curva de um arco quadrático é uniformemente distribuída de uma extremidade para a outra em termos de sua distância do ponto central. O método **quadraticCurveTo** permite especificar alguns parâmetros adicionais para alterar a "inclinação" da curva - em outras palavras, para alterar a distância do ponto central ao longo da curva. Desenho de uma curva quadrática é um pouco como desenhar uma linha reta, mas depois beliscar no meio e puxá-lo para fora para criar uma curva onde os pontos inicial e final da linha ficar fixo. Quanto mais longe você puxar o ponto central, mais íngreme a curva torna-se. Aqui está um exemplo:

```

ctxt.beginPath();
ctxt.moveTo(10,380);
ctxt.quadraticCurveTo(300,-250,580,380);
ctxt.lineWidth = 25;
ctxt.strokeStyle = '#f00';
ctxt.stroke();

```

Primeiro você precisa usar o método `moveTo` para informar o contexto onde deseja que sua curva seja iniciada. Em seguida, você passar os quatro parâmetros descritos na Tabela 1-9 para o método `quadraticCurveTo`.

TABELA 1-9 Parâmetros necessários para o método `quadraticCurveTo`

Parametro	Descrição
controlX, controlY	Esses parâmetros definem o ponto de controle, em relação à parte superior da tela, que é usado para "esticar" a curva para longe da linha formada pelo pontos iniciais e finais.
endX, endY	Este é o ponto onde a curva deve terminar.

O exemplo de código produz a imagem na Figura 1-31.

FIGURA 1-31 Uma curva quadrática em uma tela

À medida que o ponto de controle se afasta da linha formada pelos pontos inicial e final, você obtém uma curva mais acentuada. Este exemplo usou um número negativo para indicar que o ponto de controle deve estar acima da parte superior de sua tela para esticar a curva para onde você deseja. A curva final a observar é a curva de Bezier. Uma curva de Bezier é semelhante à curva quadrática, exceto que tem dois pontos de controle em vez de apenas um. Ter dois pontos permite que a curva de Bezier crie curvas mais complexas. Em ambos os exemplos que você viu até agora, a curva foi criada em torno do contexto de um único ponto. A curva Bezier muda isso. É mais fácil de ver em um exemplo, e então eu vou explicar os parâmetros. Crie o seguinte código:

```

ctxt.beginPath();
ctxt.moveTo(125, 20);
ctxt.bezierCurveTo(0, 200, 300, 300, 50, 400);
ctxt.lineWidth = 5;
ctxt.strokeStyle = '#f00';
ctxt.stroke();

```

O método `bezierCurveTo` segue uma chamada de método `moveTo` da mesma maneira que o método `quadraticCurveTo` fez. Você precisa passar três conjuntos de coordenadas `BezierCurveTo`, conforme listado na Tabela 1-10.

TABELA 1-10 Parâmetros necessários para o método `bezierCurveTo` O exemplo de código produz a saída mostrada na Figura 1-32. Você pode ver que esta curva é distorcida por causa dos dois pontos de controle.

Parametro	Descrição
controlX, controlY	Os dois primeiros parâmetros especificam o primeiro ponto de controle que é usado para esticar Fora da curva.
Control2X, control2Y	Os segundos dois parâmetros especificam o segundo ponto de controle usado para Esticar a curva.
endX, endY	Os dois parâmetros finais especificam o ponto final para a curva.

FIGURA 1-32 Uma curva de Bézier desenhada em uma tela na próxima seção, você aprenderá sobre o uso de métodos de caminho para combinar tudo o que você viu até agora.

Usando métodos de caminho

Ao usar o objeto de contexto para desenhar, você sempre precisa de um **ponto inicial e um ponto final**. O ponto final para um traço também pode se tornar o ponto de partida para o traçado seguinte. Você faz isso chamando o método **`beginPath`** no objeto de contexto e, em seguida, desenhando todas as suas linhas antes de chamar o método **`closePath`** (que termina a linha) ou o método **`beginPath`** (que inicia uma nova linha) novamente. Lembre-se de que o primeiro exemplo de arco chamado `beginPathmethod` antes de desenhar cada arco. Se o código não tivesse feito isso, a linha teria continuado através da tela de um arco para o próximo. Mas chamando o método **`beginPath`** novamente, você redefine o ponto de partida do caminho. Portanto, essencialmente, você pode juntar todas as chamadas para os vários métodos de desenho para criar um traço complexo. O traço - não importa quão simples ou complexo - é chamado de caminho. Execute o seguinte código e ver que tipo de imagem você acabar com:

```
ctxt.beginPath();
```

```
ctxt.arc(300, 200, 75, 1.75 * Math.PI, 1.25 * Math.PI, false);
```

```

ctx.lineTo(150, 125);
ctx.quadraticCurveTo(300, 0, 450, 125);
ctx.lineTo(353, 144);
ctx.strokeStyle = "blue";
ctx.lineCap = "round";
ctx.lineWidth = 10;
ctx.stroke();

```

Este código produz a saída mostrada na Figura 1-33.

FIGURA 1-33 Um caminho personalizado desenhado na tela

Embora este seja ainda um exemplo simples, você pode ser criativo e juntar imagens. Dependendo de suas habilidades matemáticas, você pode criar alguns gráficos muito complexos usando esses métodos. Em seguida, observe os outros métodos que existem para desenhar formas.

Usando o método rect

Anteriormente, você viu como desenhar círculos usando o método de arco. E como você viu na seção anterior, você pode desenhar formas personalizadas de qualquer tipo e tamanho usando o método `beginPath` e encadeando juntos uma série de métodos de desenho. Mas você nem sempre precisa fazer isso;

Algumas formas são incorporadas. Nesta seção, você olha para a funcionalidade interna para construir retângulos.

Então você olha a funcionalidade para encher suas formas desenhadas com cores e padrões. O objeto de contexto ao qual você tem uma referência tem um método chamado `rect`. O método `rect` retoma os parâmetros listados na Tabela 1-11.

Parametro	Descrição
X, Y	A coordenada x e a coordenada y definem a posição inicial do retângulo. Isto é o canto superior esquerdo do retângulo.
Width	Isso define a largura do retângulo.
height	Isso define a altura do retângulo.

Uma chamada simples para o método `rect`, como no código a seguir, desenha um retângulo:


```
ctxt.beginPath();  
ctxt.rect(300, 200, 150, 75);  
ctxt.stroke();
```

Esse código desenha um retângulo como mostrado na Figura 1-34.

FIGURA 1-34 Um retângulo desenhado na tela usando o método `rect`

Observe que, embora os parâmetros passados para o canto superior esquerdo foram (300, 200), que é o centro de sua tela, o retângulo está fora do centro. Para centrar seu retângulo, você precisaria fazer um pouco de matemática para calcular o centro com base no tamanho de sua tela, bem como o tamanho do seu retângulo desejado. O código a seguir deve centralizar seu retângulo:

```
ctxt.beginPath();  
  
var x, y;  
  
x = 150;  
  
y = 75;  
  
ctxt.rect(300—(x/2), 200—(y/2), x, y);  
  
ctxt.stroke();
```

Agora que você pode desenhar formas e retângulos, você pode ver como preencher suas formas com cores ou padrões.

Usando o método de preenchimento

Nesta seção você examina como você pode preencher suas formas. Você desenhou vários tipos de formas, mas até agora eles estão vazios. Aqui você verá como preenchê-los com cores, gradientes e padrões. Preencher uma forma com uma cor é tão simples como definir a propriedade `fillStyle` para uma cor e chamar o método de preenchimento. Inserindo o código a seguir antes de chamar o método de traçado preenche sua forma com uma cor azul:

```
ctxt.fillStyle = "blue";  
  
ctxt.fill();
```

Com relação ao método `rect`, você obtém um método de preenchimento especial especificamente para `rect` chamado `fillRect`. Com este método, você pode criar e preencher o retângulo em uma chamada:

```
ctxt.fillStyle = "blue";  
  
ctxt.fillRect(300—(x / 2), 200—(y / 2), x, y);
```

Usar o método `fillRect` reduz a quantidade de código necessário. É tão simples preencher formas não retangulares, como o gráfico de Caminho complexo que você criou anteriormente, com uma única chamada para o método de preenchimento:

```
ctxt.beginPath();
ctxt.arc(300, 200, 75, 1.75 * Math.PI, 1.25 * Math.PI, false);
ctxt.lineTo(150, 125);
ctxt.quadraticCurveTo(300, 0, 450, 125);
ctxt.lineTo(353, 144);
ctxt.strokeStyle = "blue";
ctxt.lineCap = "round";
ctxt.lineWidth = 10;
ctxt.fillStyle = "Green";
ctxt.fill();
ctxt.stroke();
```

Você pode ver na Figura 1-35 que a lógica de coloração nesta forma complexa é completamente tratada pelo navegador.

FIGURA 1-35 Usando o método de preenchimento para colorir em um objeto complexo

Isso é tudo que leva para preencher uma forma com uma cor sólida. Encher formas com um gradiente requer alguns passos extras. A criação de um gradiente envolve a utilização de um novo objeto `CanvasGradient`. Você primeiro chama o método `createLinearGradient` disponível no objeto de contexto para obter um objeto `CanvasGradient`. Nesse objeto `CanvasGradient`, você define as paradas de cores que deseja misturar para criar o efeito de gradiente. Em seguida, atribua o objeto `CanvasGradient` à propriedade `fillStyle` do contexto. O código a seguir cria e preenche um retângulo com um gradiente linear:

```
var ctxt = drawingSurface.getContext("2d");
ctxt.lineWidth = 3;
ctxt.rect(150, 150, 200, 125);
var gradient = ctxt.createLinearGradient(150, 150, 200, 125);
gradient.addColorStop(0, "Black");
gradient.addColorStop(0.5, "Gray");
gradient.addColorStop(1, "White");
ctxt.fillStyle = gradient;
```

```
ctxt.fill();  
  
ctxt.stroke();
```

Esse código cria o objeto CanvasGradient passando os pontos inicial e final de uma linha de gradiente. Em seguida, adicione três paradas de cor. O método addColorStop leva dois parâmetros. O primeiro é um valor de 0 a 1, onde 0 é o ponto inicial da linha de gradiente e 1 é o ponto final. O segundo parâmetro é a cor para começar a preencher com essa parada. Este exemplo tem três paradas, de modo que o gradiente transita por três cores. A saída de gradiente é exibida na Figura 1-36.

FIGURA 1-36 O elemento <canvas> colorido com um gradiente linear

Você também pode criar um gradiente radial usando o método createRadialGradient. Este método leva seis parâmetros, que especificam o ponto central eo raio de dois círculos e as transições de cor através das paradas ao longo do cone formado pelos dois círculos. O código a seguir produz um gradiente radial no qual o cone está apontado para o observador:

```
var ctxt = drawingSurface.getContext("2d");  
  
ctxt.lineWidth = 3;  
  
ctxt.rect(150, 150, 250, 175);  
  
var gradient = ctxt.createRadialGradient(200, 200, 5, 250, 250, 100);  
  
gradient.addColorStop(0, "Red");  
  
gradient.addColorStop(.5, "Orange");  
  
gradient.addColorStop(1, "Blue");  
  
ctxt.fillStyle = gradient;  
  
ctxt.fill();  
  
ctxt.stroke();
```

A Figura 1-37 mostra a saída desse gradiente:

FIGURA 1-37 Um gradiente radial colorido em uma tela

A última opção de preenchimento a ser observada envolve o uso de um padrão de preenchimento. Você precisa de uma imagem externa, que é aplicado como um padrão em toda a forma. Por exemplo, você pode usar uma textura que você criou como um plano de fundo para a sua tela usando o seguinte código:

```
var ctxt = drawingSurface.getContext("2d");  
  
ctxt.lineWidth = 3;  
  
ctxt.rect(150, 150, 200, 125);
```

```

var img = new Image();
img.src = "texture.png";
img.onload = function () {
var pat = ctx.createPattern(img, "repeat");
ctx.fillStyle = pat;
ctx.fill();
ctx.stroke();
}

```

A Figura 1-38 mostra a saída desse código.

FIGURA 1-38 A tela preenchida com um padrão desenhado nele

O código anterior chama o método `createPattern` e passa-o uma referência para um objeto `Image` e um padrão de repetição. O padrão de repetição pode ser `no-repeat`, `repeat-x` ou `repeat-y`, mas o padrão é repetir se você não especificar nada. É necessário atribuir um manipulador de eventos ao evento `onload` do objeto `Image` para garantir que você desene o padrão somente após as imagens. Caso contrário, o código pode ser executado antes que a imagem seja processada e o padrão não será exibido.

Esta seção abordou muito sobre como trabalhar com formas e preenchê-las. Todos os seus gráficos foram criados usando código para desenhar formas manualmente. Em seguida, você vê como desenhar gráficos existentes de arquivos externos em sua tela, e então você olha para o desenho de texto.

Desenho de imagens

Desenho de imagens em uma tela é tão simples como os outros métodos de desenho que você já viu. Para desenhar uma imagem em uma tela, use o método `drawImage` do objeto de contexto. Este método usa um objeto `Image` e algumas coordenadas (x, y) para definir onde a imagem deve ser desenhada. Assim como com o retângulo, o canto superior esquerdo da imagem é desenhado no especificado (x, y). O tamanho padrão da imagem é o tamanho real da imagem, mas como você verá à direita depois, você também pode redimensionar a imagem como você desenhá-la. Para simplesmente desenhar a imagem, crie o seguinte código:

```

var drawingSurface = document.getElementById("drawingSurface");
var ctx = drawingSurface.getContext("2d");
var img = new Image();
img.src = "orange.jpg";

```

```
img.onload = function () {  
  ctx.drawImage(img, 0, 0);  
  ctx.stroke();  
}
```

Este código produz um elemento <canvas> com a imagem desenhada sobre ele, como mostrado em Figura 1-39.

FIGURA 1-39 Uma imagem desenhada em uma tela

Se pretender redimensionar a imagem, pode substituir a chamada de método `drawImage`

Seguinte linha:

```
ctx.drawImage(img, 0,0,img.width * .5, img.height * .5);
```

Isso reduz o tamanho da imagem em 50%.

Agora veja como você pode desenhar texto em sua tela.

Desenho de texto

O desenho de texto na tela envolve a adição de algumas ferramentas adicionais ao peito a partir do objeto de contexto, usando o método `strokeText` e a propriedade `font`. Você vê como aplicar cor ao seu texto e, finalmente, como gerenciar seu alinhamento.

Na sua forma mais simples, o desenho de texto requer apenas o seguinte código:

```
ctx.strokeText("1. Text with default font", 100, 100);
```

Lembre-se de que você precisa se certificar de que a janela tenha terminado de carregar e que você precisa para obter um objeto de contexto. Aqui está o código completo para este exemplo:

```
window.onload = function () {  
  var drawingSurface = document.getElementById("drawingSurface");  
  var ctx = drawingSurface.getContext("2d");  
  ctx.strokeText("1. Text with default font", 100, 100);  
}
```

É isso aí. A chamada `strokeText` desenha o texto especificado nas coordenadas especificadas na tela. Os parâmetros especificam o texto a desenhar e as coordenadas (x, y) especificam onde o desenho deve começar. O método `strokeText` desenha o estilo de fonte padrão. Você pode facilmente alterar a propriedade de fonte do objeto de contexto para melhorar a aparência

do seu texto. Por exemplo, executar o seguinte código altera o tamanho da fonte para 24 ea família de fontes para Arial:

```
ctx.font = "24px arial";
```

```
ctx.strokeText("2. Text with altered font", 100, 125);
```

Para colorear seu texto, você pode adicionar este código:

```
ctx.font = "24px arial";
```

```
ctx.strokeStyle = "Red";
```

```
ctx.strokeText("3. Text with altered colored font", 100, 160);
```

Quando você executar o código anterior, observe que seu texto é delimitado. Este é o padrão comportamento quando você aumenta o tamanho da fonte; Ele é desenhado como descrito. Para desenhar texto de cor sólida, adicione o seguinte código, que define a propriedade `fillStyle` e chama o método `fillText` em vez dos métodos `strokeStyle` e `StrokeText`:

```
ctx.font = "24px arial";
```

```
ctx.fillStyle = "Red";
```

```
ctx.fillText("4. Text with altered colored font", 100, 185);
```

Você também pode definir o alinhamento do texto dentro da tela. Por exemplo, para garantir que o texto esteja centralizado, adicione este código:

```
ctx.font = "24px arial";
```

```
ctx.textAlign = "center";
```

```
ctx.fillStyle = "Red";
```

```
ctx.fillText("5. Text with altered colored font Centered.", drawingSurface.width / 2, drawingSurface.height / 2);
```

Ao definir a propriedade `textAlign` para o centro de valor, você está dizendo ao contexto para considerar a coordenada (x, y) especificada como o ponto central da sequência em vez do ponto inicial da sequência de caracteres. Assim, você divide a largura ea altura da tela por dois para obter o ponto central da tela, e você obterá uma sequência centralizada horizontal e verticalmente.

A Figura 1-40 mostra a progressão do seu texto:

FIGURA 1-40 Progresso do texto com a mudança de estilos

A tela é um utilitário forte para apresentar gráficos dinamicamente no navegador. No entanto, não é a única ferramenta gráfica disponível. Na próxima seção, você pode usar o Scalable VectorGraphics.

Gráficos vetoriais escaláveis (SVG)

Scalable Vector Graphics (SVG) é uma linguagem baseada em XML para criar gráficos bidimensionais. É implementado usando tags definidas pelo namespace XML SVG e incorporadas em documentos HTML5 dentro de elementos <svg> de abertura e fechamento.

Os objetos SVG não perdem qualquer qualidade à medida que os usuários aumentam ou diminuem o zoom. Você pode acessar objetos SVG através do DOM e, semelhante a elementos HTML, os elementos SVG suportam atributos, estilos e manipuladores de eventos. O elemento <svg> fornece um contêiner no qual renderizar gráficos;

SVG processa em linha com o layout da página. Aqui está um exemplo de um gráfico SVG com manipuladores de eventos:

```
<!DOCTYPE html>

<html>

<head>

<title>Test Web Page</title>

<script language="javascript">

function Red(evt) {
var circle = evt.target;
circle.setAttribute("style", "fill: red");
}

function Green(evt) {
var circle = evt.target;
circle.setAttribute("style", "fill: green");
}

</script>

</head>

<body>

<svg>

<circle id="Circle" cx="50" cy="50" r="50" fill="green" onmouseover="Red(evt)"
onmouseout="Green(evt)"/>

</svg>

</body>

</html>
```

Este código produz a saída mostrada na Figura 1-41. Os manipuladores de eventos JavaScript ativam o círculo vermelho quando o mouse paira sobre ele e volta para verde quando o mouse é movido para fora do círculo.

FIGURA 1-41 Um círculo desenhado usando SVG

Todas as funcionalidades de desenho de formas e desenho de linha que você viu na discussão do elemento <canvas> também existem para SVG, embora a sintaxe seja diferente, é claro. O código a seguir produz um gráfico um pouco mais elaborado.

```
<svg>
<rect id="lightStandard" x="100" y="100" width="60" height="200" fill="black"/>
<circle id="redLight" cx="129" cy="145" r="25" fill="red"/>
<circle id="amberLight" cx="129" cy="205" r="25" fill="yellow"/>
<circle id="greenLight" cx="129" cy="265" r="25" fill="green"/>
</svg>
```

A imagem na Figura 1-42 mostra a saída deste código:

FIGURA 1-42 Múltiplas formas desenhadas usando SVG

Neste exemplo, o elemento <rect> foi usado para criar o retângulo de fundo, e uma série de elementos <circle> foram usados para criar as luzes. Cada forma SVG requer os mesmos tipos de parâmetros que sua contrapartida de tela, e as mesmas regras se aplicam. O elemento <rect> precisa de uma coordenada (x, y) para estabelecer onde deve ser desenhada, juntamente com uma largura e altura para estabelecer o tamanho. O mesmo é verdadeiro para cada círculo, exceto que você especifica o raio para o tamanho. O atributo fill define a cor a ser usada para preencher a forma.

O SVG também suporta as mesmas funções básicas de desenho de formas que o contexto da tela. O seguinte segmento de código mostra o

Uso da polilinha, polígono, linha e elipse e produz a saída mostrada na Figura 1-43:

```
<svg>
<polygon points="10,15 30,35 10,85 100,85, 70,35,100,15" fill="purple"/>
<polyline points="10,150 30,170 50,132 62,196 78,165 96,170"
style="stroke:orange; fill:none; stroke-width:5;"/>
<line x1="150" y1="100" x2="150" y2="150" style="stroke:blue;stroke-
width:3"/>
<ellipse cx="250" cy="150" rx="30" ry="55" fill="green"/>
<text x="10" y="10" style="stroke: black;stroke-width:1;">
```


Examples of SVG Shapes and Text</text>

</svg>

FIGURA 1-43 Texto, uma linha, um polígono, uma elipse e uma polilinha desenhada em cores diferentes

Dica de exame

Em alguns casos, usar gráficos SVG é mais simples do que usar o elemento <canvas>. Como os exemplos mostraram, você pode criar imagens SVG declarativamente diretamente dentro do próprio HTML. No entanto, à medida que aumenta o número de objetos em uma renderização SVG, o desempenho pode se tornar uma preocupação. Nos casos em que o desempenho é uma consideração, usar o elemento <canvas> é uma abordagem preferível.

O SVG também suporta renderizar gráficos existentes na forma de arquivos de imagem externos, como mostrado aqui:

```
<svg id="mySVG">
```

```
<image href="orange.jpg" width="250" height="100"/>
```

```
</svg>
```

Experimento de pensamento

Criando um jogo Neste experimento de pensamento, aplique o que aprendeu sobre esse objetivo. Você pode encontrar respostas para essas perguntas na seção "Respostas" no final deste capítulo.

Você foi contratado para criar um jogo. Para a primeira fase do jogo, você deve fazer uma bola mover do lado esquerdo da tela para o lado direito. Como você pode conseguir isso com o elemento HTML5 <canvas>? Como sobre SVG? Você demonstra isso fase para as partes interessadas, e eles adoram. Para a próxima etapa, quando a bola é clicada, ele deve dividir em duas bolas. Como o número de bolas aumenta devido a ser clicado, o que considerações? Usaria o SVG ou o <canvas>

Elemento ser a melhor solução para este jogo?

Resumo do objetivo

■ JavaScript é uma poderosa ferramenta que permite aos desenvolvedores manipular o DOM

Programaticamente no navegador.

■ O HTML5 suporta controles rich media para incorporar vídeo usando o recurso <video>

Elemento e áudio usando o elemento <audio>.

- O elemento <video> suporta vários formatos de mídia usando o elemento <source>.
- Os elementos HTML5 <canvas> e <svg> suportam uma API rica para criar E gráficos complexos no navegador.
- Os motores gráficos <canvas> e <svg> podem desenhar texto, linhas, formas, fontes, preenchimentos, E gradientes.
- O elemento <canvas> é desenhado através de JavaScript, obtendo uma referência ao contexto.
- O elemento <svg> processa gráficos usando uma sintaxe declarativa.

Objetivo 1.3: Aplicar estilo aos elementos HTML por meio de programação

A seção cobre aplicar estilos aos elementos HTML na página dinamicamente, usando JavaScript. Quando você recupera referências de elemento usando métodos como getElementById, você pode manipular esses elementos, incluindo seus estilos.

Este objetivo abrange como:

- Alterar a localização de um elemento
- Aplicar uma transformação
- Mostrar e ocultar elementos

Alterar a localização de um elemento

Usando os métodos para recuperar um elemento do DOM em JavaScript, você pode aplicar estilos dinamicamente através de código que pode alterar a posição do elemento na página. Como os elementos são definidos na página pode afetar como os elementos se comportam quando são reposicionados.

Algumas opções determinam como os elementos HTML são posicionados em uma página da Web. **Por padrão, todos os elementos HTML fluem**

estaticamente da esquerda para a direita na mesma ordem em que são declarados na página HTML. No entanto, CSS fornece um mecanismo para especificar algumas opções avançadas na posição do elemento.

Você pode posicionar elementos usando posicionamento absoluto (**absolute**) ou posicionamento relativo (**relative**).

Com o posicionamento absoluto, o elemento é colocado no local exato especificado, em relação às bordas do seu contêiner. No entanto, com posicionamento relativo, o elemento é posicionado em relação às coordenadas do irmão esquerdo imediato. Você pode aplicar quatro propriedades individualmente ou em combinação para controlar a posição de um elemento: **Top**, **Left**, **Right** e **Bottom**. Cada propriedade toma um parâmetro de distância que especifica a distância relativa do objeto a partir de uma referência

Ponto baseado no atributo de posicionamento especificado. Ao usar posicionamento absoluto ou relativo, as configurações de borda ou margem padrão são ignoradas porque o objeto está posicionado onde os atributos de posicionamento direcionam o elemento a ser.

O código na Listagem 1 a 3 demonstra isso.

```
<!DOCTYPE html>
```

```
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
```

```
<meta charset="utf-8"/>
```

```
<title></title>
```

```
<style>
```

```
html, body {
```

```
height: 100%;
```

```
width: 100%;
```

```
}
```

```
img {
```

```
height: 150px;
```

```
width: 225px;
```

```
}
```

```
</style>
```

```
<script>
```

```
window.onload = function () {
```

```
var top = document.getElementById("topText");
```

```

        var left = document.getElementById("leftText");
        var pos = document.getElementById("positioning");
        document.getElementById("btnPosition").onclick = function
        () {
            var img = document.getElementById("orange2");
            img.style.position = pos.value;
            img.style.left = left.value + "px";
            img.style.top = top.value + "px";
        }
    }
</script>
</head>
<body>
<table style="width: 100%; height: 100%; border: 1px solid black;">
    <tr>
        <td style="vertical-align: top; width: 80%">
            
            
        </td>
        <td style="vertical-align: top;">Left:
            <input type="text" id="leftText"/><br/>
            Top:
            <input type="text" id="topText"/><br/>
            Position:
            <select id="positioning">
                <option>relative</option>
                <option>absolute</option>
            </select><br/>
            <input type="button" id="btnPosition" value="Update"/>
        </td>
    </tr>

```

```
</table>

</body>

</html>
```

Quando o código é processado no navegador, a posição padrão está em vigor, como mostrado em

Figura 1-44.

Todos os atributos de posicionamento que foram discutidos estão disponíveis declarativamente no atributo de estilo do elemento HTML, mas também podem ser acessados programaticamente e manipulados via JavaScript. A página da Web é aprimorada para fornecer alguma funcionalidade do usuário final para controlar o posicionamento das duas imagens. O código na Listagem 1-3 fornece uma página HTML com as duas imagens e alguns controles de entrada para controlar o posicionamento da segunda imagem. Você pode inserir as posições superior e esquerda, bem como se a posição em relação à primeira imagem laranja ou para posicionar como absoluto para o elemento da tabela pai.

Quando o topo e a esquerda estiverem configurados para 50px e o posicionamento for relativo, você verá o resultado mostrado em:

Figura 1.45

Manter os valores iguais, mas mudar o posicionamento para absoluto altera o posicionamento dos elementos, como mostrado na Figura 1-46.

Você pode empregar ainda outro mecanismo para alterar a aparência de um elemento HTML, transform, que você examina em seguida.

Aplicando uma transformação

Aplicar transformações é uma maneira de alterar um objeto na página da Web. As transformações permitem alterar a aparência de um elemento. Você pode fazer um elemento maior ou menor, rodá-lo, e assim por diante. Alguns métodos de transformação estão disponíveis. Para adicionar uma transformação a um elemento, você declará-lo no CSS para adicionar a propriedade no elemento transform da seguinte maneira:

```
.rota {transform: rotate(90deg);}
```

Este código aplica o método rotate a um objeto quando adiciona a classe .rota CSS à coleção de estilos do objeto. Como mencionado, vários métodos de transformação estão disponíveis, e você vai examinar cada um por sua vez. Use o seguinte código para todos os exemplos nesta seção:

```
<!DOCTYPE html>
```

```
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
<meta charset="utf-8"/>
<title></title>
<style>
#orange1 {
height: 150px;
width: 225px;
}
.trans {
transform: scale(1) ;
}
</style>
<script>
window.onload = function () {
document.getElementById("orange1").onclick = function () {
this.classList.add("trans");
}
}
</script>
</head>
<body>

</body>
</html>
```

Este código cria um único objeto de imagem ao qual você aplicará as transformações; No entanto, as transformações podem funcionar com êxito com qualquer elemento HTML. A imagem também é atribuída a um manipulador de eventos para o evento click. Isso é suficiente para fins de demonstração. Você pode usar qualquer evento suportado para acionar uma transformação. Nos exemplos a seguir, você precisará para substituir a classe .trans CSS no código anterior com os métodos de transformação apropriados para demonstrá-los. Você será solicitado a substituir o código quando necessário.

Usando o método de rotação

O método de rotação de transformação permite que você gire um objeto por um número em graus. O método aceita um único parâmetro que especifica o número de graus. No código anterior utilizado para a transformação de escala, substitua o método que segue:

```
transform: rotate(90deg);
```

Agora, execute a página web no navegador. Clique na imagem para ver o efeito da transformação.

Neste caso, a imagem é girada 90 graus no sentido horário (veja a Figura 1-47). Se preferir girar a imagem no sentido anti-horário, você pode especificar um número negativo de graus.

FIGURA 1-47 O efeito da transformação de rotação em uma imagem.

A transformação também suporta os métodos rotateX e rotateY, que aceitam um único parâmetro em graus para especificar um ângulo em torno do eixo x ou do eixo y no qual girar. Você pode, por exemplo, usar esses métodos para inverter um elemento verticalmente ou horizontalmente especificando 180 deg como o parâmetro. Neste caso, o elemento roda 180 graus ao longo do eixo especificado - o que resulta essencialmente na imagem sendo invertida ou espelhada ao longo desse eixo.

Usando o método translate

O método translate permite mover um elemento HTML alterando sua posição relativa X e Y na página implementando o método translate especificando o método de translação na propriedade de transformação. Na lista de exemplo, substitua o método de transformação com o seguinte:

```
Transform: translate (50px, 0px);
```

O método translate move o elemento HTML para o qual é aplicado por 50 pixels na direção X e 0 pixels na direção Y em relação a onde ela agora reside (consulte a Figura 1-48).

Novamente, os métodos translateX e translateY estão disponíveis se o efeito desejado for mover o objeto em torno do eixo x ou do eixo y.

FIGURA 1-48 O efeito do método de translação aplicado a uma imagem.

Usando o método skew

Você pode inclinar um elemento HTML usando o método skew da propriedade transform. Inclinando o objeto para que ele não é paralelo ao eixo vertical ou horizontal. No código de exemplo, substitua a propriedade transformar com a seguinte linha de código.

```
transform: skew(10deg, 10deg);
```

A Figura 1-49 mostra a transformação de efeito: inclinação (10deg, 10deg);

FIGURA 1-49 O efeito do método skew em uma imagem

Usando o método da escala

O método de escala permite redimensionar elementos por uma proporção especificada. O método scale leva um parâmetro: um valor decimal que representa a porcentagem a ser escalonada. Especificar um valor maior que 1 torna o objeto maior; Especificando um valor menor que 1 mas maior que 0 torna o objeto menor. Especificar um valor de -1 inverte o objeto sobre seu eixo horizontal. No código de exemplo, substitua a propriedade transformar com o seguinte:

```
transform: scale(1.5);
```

Esta escala transform aumenta o tamanho do elemento em 50 por cento, essencialmente multiplicando os valores de altura e largura existentes por 1,5. O objeto escala para fora de seu centro absoluto de modo que expanda em todas as direções; Não se estende apenas para baixo e para a direita.

Figura 1-50 mostra o resultado de uma transformação de escala:

FIGURA 1-50 O efeito da transformação de escala em uma imagem.

Combinando transformações

As transformações individualmente dão grande flexibilidade ao que você pode realizar alterando a aparência de elementos HTML, mas o estilo de transformação não se limita a especificar um único método de transformação. Você pode combinar os métodos para aplicar vários efeitos ao elemento.

No código de exemplo, altere a propriedade transformar para o seguinte código:

```
transform: translate (50px, 0px) scale (1,5) skew (10deg, 10deg);
```

Neste código, três efeitos são aplicados. A ordem importa. Os efeitos são aplicados na ordem que eles são especificados na propriedade de transformação. Nesse caso, a propriedade translate é aplicada primeiro e, em seguida, o objeto translate é dimensionado. Finalmente, o objeto resultante é

enviesado. O efeito no elemento HTML é que ele é movido 50 pixels ao longo do eixo x, dimensionado em 50 por cento e, em seguida, inclinado 10 graus.

Mostrando e ocultando elementos

Você pode **mostrar e ocultar elementos declarativamente na marcação HTML** ou **programaticamente modificando as propriedades CSS do objeto através de JavaScript**. Você pode criar as propriedades CSS que mostram ou ocultar um elemento diretamente na propriedade de estilo de um objeto ou em um estilo CSS e ele é adicionado à coleção de estilos do elemento. Os exemplos desta seção usam o código da Listagem 1-2,

Atualizado como segue:

```
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
...
<script>
window.onload = function () {
document.getElementById("btnHideAnElement").onclick = function () {
if (document.getElementById("innerDiv").style.display == 'inline') {
document.getElementById("innerDiv").style.display = 'none';
}
else {
document.getElementById("innerDiv").style.display = 'inline';
}
}
}
}
</script>
...
<button type="button" id="btnHideAnElement" >Show/Hide Element</button>
</form>
</body>
</html>
```

Este código modifica o bloco de script e adiciona um novo botão para a parte inferior da página. O botão é ligado a um evento onclick após o carregamento da janela terminar. Nesse evento, você modificar programaticamente a visibilidade dos elementos HTML. O elemento `innerDiv` padrão é oculto quando a página é carregada.

Quando o botão é clicado, o código avalia o estado da propriedade CSS de exibição para determinar se o elemento agora está visível ou oculto. Dependendo do resultado, a propriedade é alternada.

(Display inline/none)

Propriedade de exibição aceita dois valores possíveis. Um valor de `inline` indica ao navegador para mostrar o item, enquanto um valor de `none` significa que o navegador deve ocultar o item.

A segunda propriedade disponível para controlar a visibilidade do elemento é chamada `visibility`. Essa propriedade aceita quatro valores possíveis, como descrito na Tabela 1-12.

TABELA 1-12 Valores disponíveis para a propriedade de visibilidade.

Valor	Efeito
Visible	Define a propriedade como visível para mostrar o elemento
Hidden	Oculto o elemento
Colapse	Colapsa o elemento quando aplicável, como em uma linha da tabela
Inherit	Herdar o valor da propriedade de visibilidade do pai

Alguns destes valores têm comportamentos interessantes. Quando você usa a propriedade CSS de exibição e defini-lo para o valor de `none`, o elemento HTML está oculto. Mas ocultar o elemento dessa maneira também o remove do layout. Todos os elementos circundantes se realinham como se o elemento não estivesse lá. Quando o `display` é definido como `inline`, o elemento é mostrado novamente e todos os elementos envolventes se movem para fora do caminho, de volta para onde eles estavam originalmente.

A propriedade CSS de `visibility` comporta-se de forma ligeiramente diferente. Definir a propriedade de visibilidade para `hidden` escondem um elemento, mas os elementos circundantes do elemento oculto agem como se ainda está lá. O espaço ocupado pelo elemento é mantido intacto, mas o conteúdo do elemento está oculto. Quando a propriedade é definida de volta para visível, o elemento reaparece exatamente onde estava, sem afetar quaisquer elementos envolventes.

O valor de `colapso`, por outro lado, age mais como a propriedade de exibição. Se você especificar colapso em algo como uma linha de tabela, as linhas da tabela acima e abaixo recolher e assumir o espaço que ocupava a linha

recolhida. Quando você definir a propriedade de visibilidade volta a visível, os elementos circundantes mover para fora do caminho para mostrar o elemento. Isso é útil para situações em que você deseja ter conteúdo que pode ser recolhido ou exibido um item de cada vez para preservar espaço, como em um FAQ, onde a resposta a uma pergunta é exibida quando um usuário clica na pergunta e, em seguida, recolhida quando O usuário clica em uma pergunta diferente.

Dica de exame

Se você precisa preservar o layout da página ao alterar a visibilidade, use a propriedade de visibilidade com o valor hidden. Se você não precisa preservar o layout, você pode definir a propriedade de exibição como none ou definir visibilidade para collapse.

Experimento de pensamento

Criando uma pesquisa dinâmica

Neste experimento mental, aplique o que aprendeu sobre esse objetivo. Você pode encontrar respostas para essas perguntas na seção "Respostas" no final deste capítulo. Você foi encarregado de construir uma página da web que requer que um usuário responda a uma uma série de perguntas. No entanto, as perguntas são dinâmicas, com base na resposta do usuário da pergunta anterior.

O usuário deve ver apenas questões relevantes. Tome o seguinte fluxo de perguntas como as regras para esta página:

And the following HTML5 page:

```
<html>
```

```
<head>
```

```
<meta charset="utf-8"/>
```

```
<title></title>
```

```
</head>
```

```
<script>
```

```
...
```

```
</script>
```

```
<body>
```

```
<header>
```

Um questionário dinâmico.

```
</header>
```

```
<section>
<article>
<hgroup>
<h1>Questionnaire</h1>
<h2>Answer the questions in order as they
appear.</h2>
</hgroup>
<div id="Question1">
1. ....
</div>
<div id="Question2">
2. ....
</div>
<div id="Question3">
3. ....
</div>
<div id="Question4">
4. ....
</div>
<div id="Question5">
5. ....
</div>
<div id="Question6">
6. ....
</div>
<div id="Question7">
7. ....
</div>
<div id="Question8">
8. ....
```

```
</div>
<div id="Question9">
9. ....
</div>
<div id="Question10">
10. ....
</div>
</article>
</section>
</body>
</html>
```

Crie todo o JavaScript necessário para mostrar e ocultar os elementos necessários, dependendo da resposta a cada pergunta. Suponha que a resposta a cada pergunta é uma seleção de botão de opção com apenas as opções Sim / Não.

Resumo do objetivo

- Você pode usar CSS para definir efeitos de transformação.
- Você pode aplicar transformações via JavaScript para manipular o DOM com efeitos tais como girar, inclinar, dimensionar e traduzir.
- A propriedade de visibilidade fornece opções para controlar a visibilidade de um elemento dentro da página.

Revisão objetiva

Responda às seguintes perguntas para testar seu conhecimento da informação neste objetivo.

Você pode encontrar as respostas a essas perguntas e explicações de por que cada escolha de resposta é

Correta ou incorreta na seção "Respostas" no final deste capítulo.

1. O posicionamento absoluto posiciona um objeto em relação ao quê?

A. O canto superior esquerdo da janela do navegador.

B. O canto superior esquerdo do elemento pai.

C. Centrado dentro da janela.

D. Centrado dentro de seu elemento pai.

2. Que transformação permite que você altere o tamanho de um elemento?
- A. rodar
 - B. inclinação
 - C. traduzir
 - D. Escala**
3. Qual sintaxe preserva o layout da página ao ocultar um elemento no DOM?
- A. display = 'hidden'
 - B. display = 'inline'
 - C. visibility = 'none'
 - D. visibility = 'hidden'**

Objetivo 1.4: implementar APIs HTML5

As APIs de JavaScript forneceram algumas novas e poderosas funcionalidades, como a capacidade de armazenar mais dados localmente e disponibilizar esses dados para a página da Web por meio da Web Storage API.

A API do AppCache permite que você tome aplicativos da Web offline. A Geolocation API fornece métodos para trabalhar com posicionamento global dentro do aplicativo.

Este objetivo abrange como:

- Usar a API de armazenamento
- Use a API AppCache
- Use a API de geolocalização

Usando a API de armazenamento

O Web Storage é uma nova API para armazenar dados de páginas da Web localmente. O Web Storage, descrito nesta seção, substitui o conceito de cookies.

OBSERVAÇÃO SUPORTE DO NAVEGADOR

É claro que você deve considerar o suporte de navegador do seu público-alvo para HTML5 e Armazenamento da Web antes de você optar por usá-lo exclusivamente.

Existem duas formas de Armazenamento Web: armazenamento local (Local storage) e de sessão (session storage). O armazenamento local é persistente; Os dados armazenados no armazenamento local estão disponíveis

para a página mesmo se o usuário fechar o navegador completamente e, em seguida, reabri-lo para seu site. O armazenamento de sessão está disponível somente para a duração da sessão atual, portanto, se o usuário fechar o navegador, o armazenamento da sessão será limpo automaticamente e não estará mais disponível. A API de armazenamento da Web está disponível como um objeto global. Para acessar o armazenamento local, use o objeto `localStorage`; Para acessar o armazenamento da sessão, use o `sessionStorage` objeto.

Dica de exame

Os objetos `localStorage` e `sessionStorage` fornecem exatamente a mesma API. Todos os exemplos mostrados nesta seção funcionam exatamente o mesmo com qualquer objeto. A única diferença é a vida útil do armazenamento. Lembre-se de que `sessionStorage` é desmarcado quando a sessão é fechada, enquanto `localStorage` ainda está acessível depois que uma sessão é fechada e uma nova sessão é aberta.

A Tabela 1-13 lista os métodos da API e seu uso. Armazenamento da Web é implementado como nome e valor e armazenados como seqüências de caracteres. Todos os dados que você pode colocar em um formato de seqüência de caracteres podem ser armazenados no armazenamento da Web. Isso não é tão limitante quanto parece. Você verá alguns exemplos de armazenamento de Objetos.

TABELA 1-13 Métodos disponíveis em objetos de armazenamento

Metodo	Descrição
<code>setItem</code>	Adiciona um par chave / valor ao armazenamento. Se nenhum item com a chave especificada existir, o item é Adicionado; Se essa chave existir, seu valor será atualizado.
<code>getItem</code>	Recupera dados do armazenamento com base em um valor de chave ou índice especificado
<code>Clear</code>	Limpa todo o armazenamento que foi salvo. Use esse método para limpar o armazenamento quando necessário.
<code>Key</code>	Recupera a chave em um índice especificado. Você pode usar a chave resultante para passar como um Parâmetro para um dos outros métodos que aceita uma chave.
<code>removeItem</code>	Remove o par chave / valor especificado do armazenamento

Além dos métodos descritos na Tabela 1-13, os objetos de armazenamento expõem um **propriedade comprimento que retorna o número de pares chave / valor no armazenamento**. Use o código de exemplo na Listagem 1 a 4 para explorar a Web Storage API.

LISTA 1-4 Explorando a API de Armazenamento da Web

```
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta charset="utf-8"/>
<title></title>
<style>
section {
margin-top: 15px;
}
</style>
<script>
window.onload = function () {
document.getElementById("btnAdd").onclick = function () {
}
document.getElementById("btnRemove").onclick = function () {
}
document.getElementById("btnClear").onclick = function () {
}
function LoadFromStorage() {
}
}
</script>
</head>
<body>
<section>
Key:
<input type="text" id="toStorageKey"/>
Value:
```



```

<input type="text" id="toStorageValue"/><br/>
</section>
<section>
<button type="button" id="btnAdd">Add To Storage</button>
<button type="button" id="btnRemove">Remove from Storage</button>
<button type="button" id="btnClear">Clear Storage</button>
</section>
<div id="storage">
<p>Current Storage Contents</p>
</div>
</body>
</html>

```

O código na Listagem 1-4 cria caixas de texto para aceitar uma chave e um valor, respectivamente. Botões permitem adicionar itens ao armazenamento, remover um item ou limpar completamente o armazenamento. Cada capacidade é implementada por sua vez. Para exibir o conteúdo do armazenamento, a página contém um div que mostra o conteúdo do armazenamento anexado a ele.

O método `loadFromStorage` é chamado para cada operação para **atualizar a página** com os dados disponíveis no armazenamento. Todos os exemplos a seguir usam o **armazenamento local**, mas, novamente, eles funcionariam da mesma maneira com o armazenamento da sessão.

Se você quiser testar esses exemplos usando o armazenamento de sessão, basta substituir a referência `localStorage` por uma referência a `sessionStorage`. Você primeiro precisa implementar o método `LoadFromStorage` para que quando a página seja carregada, você possa ver todos os itens que já foram colocados em armazenamento. Digite o seguinte código na função `LoadFromStorage` no bloco de script:

```

window.onload = function () {
    LoadFromStorage();
    document.getElementById("btnAdd").onclick = function () {
        ...
    }
    function LoadFromStorage() {
        var storageDiv = document.getElementById("storage");
        var tbl = document.createElement("table");
        tbl.id = "storageTable";
    }
}

```

```

if (localStorage.length > 0) {
  for (var i = 0; i < localStorage.length; i++) {
    var row = document.createElement("tr");
    var key = document.createElement("td");
    var val = document.createElement("td");
    key.innerText = localStorage.key(i);
    val.innerText = localStorage.getItem(key.innerText);
    row.appendChild(key);
    row.appendChild(val);
    tbl.appendChild(row);
  }
}
else {
  var row = document.createElement("tr");
  var col = document.createElement("td");
  col.innerText = "No data in local storage.";
  row.appendChild(col);
  tbl.appendChild(row);
}

if (document.getElementById("storageTable")) {
  document.getElementById("storageTable").replaceNode(tbl);
}
else {
  storageDiv.appendChild(tbl);
}
}

```

Observe que este código adicionou uma chamada para o método **LoadFromStorage** para o evento `window.onload`, de modo que `localStorage` é verificado após o carregamento da página. O método `LoadFromStorage` toma quaisquer elementos disponíveis no armazenamento local e os exibe em uma tabela HTML. Este código tira vantagem da propriedade **length** para determinar se quaisquer valores de armazenamento local precisam ser exibidos. Caso contrário, a página exibirá uma mensagem sobre nenhum dado no

armazenamento local. Adicione o seguinte código ao botão onclick eventos para começar a manipular localStorage:

```
document.getElementById("btnAdd").onclick = function () {  
    localStorage.setItem(document.getElementById("toStorageKey").value,  
        document.getElementById("toStorageValue").value);  
    LoadFromStorage();  
}  
  
document.getElementById("btnRemove").onclick = function () {  
    localStorage.removeItem(document.getElementById("toStorageKey").value);  
    LoadFromStorage();  
}  
  
document.getElementById("btnClear").onclick = function () {  
    localStorage.clear();  
    LoadFromStorage();  
}
```

O código anterior implementa o evento onclick de cada botão. Um usuário agora pode adicionar itens para o armazenamento local e ver o que está armazenado. O usuário pode continuar adicionando ao armazenamento local em nesta aplicação até que o armazenamento esteja cheio. A disponibilidade de armazenamento local é limitada e esta disponível não é consistente entre navegadores. Até a presente redação, a documentação o Microsoft Internet Explorer 10 suporta até cerca de 10 MB de armazenamento. Contudo, isso poderia alterar e pode não ser o mesmo em outros navegadores; Alguns agora suportam apenas 5 MB de armazenamento.

Tenha isso em mente ao projetar aplicativos da web que usufruam do Web Storage API.

Execute o exemplo anterior e adicione os seguintes itens ao localStorage:
("Red","FF0000"), ("Green","00FF00"), ("Blue","0000FF").

A Figura 1-51 mostra a saída na tela depois de adicionar esses itens ao armazenamento local.

FIGURA 1-51 Armazenando itens e recuperando-os do armazenamento na web

Agora, se você fechar o navegador e reabrir sua página, os itens ainda estarão disponíveis em armazenamento local. Tente substituir todos os usos de

localStorage com sessionStorage. Desta vez, observe que fechar o navegador automaticamente limpa quaisquer dados no armazenamento.

O benefício de usar a API de armazenamento da Web em vez de cookies é que os dados residem Localmente e permanece local. Os dados não são enviados de um lado para outro do servidor, como é o caso dos cookies. Os dados armazenados no armazenamento da web são organizados pelo domínio raiz. O espaço a alocação está disponível em uma base de domínio por raiz. Por exemplo, domínios como localhost ou Microsoft.com cada obter seu próprio espaço de armazenamento seguro da correia fotorreceptora.

Conforme definido pela API, o armazenamento na Web permite o armazenamento somente de pares chave / valor onde a chave e o componente valor são armazenados como uma seqüência de caracteres. Se você precisar armazenar objetos armazenamento na web, você pode usar algumas técnicas. Por exemplo, adicione o seguinte código à direita antes da primeira chamada para LoadFromStorage no evento onload:

```
var customer = new Object();  
customer.firstName = "Rick";  
customer.lastName= "Delorme";  
customer.shirtSize = "XL";  
localStorage.setItem("cart1", JSON.stringify(customer));  
LoadFromStorage();
```

Este código cria um objeto personalizado para representar um cliente que está navegando no site e o tamanho da camisa do cliente. Esta informação deve ser mantida e utilizada localmente, por isso não precisa ser postado no servidor.

O armazenamento local é uma ótima solução para isso. Entretanto, para armazenar objeto em dados locais, você precisa de um método para converter o objeto personalizado em uma seqüência que corresponda o modelo de armazenamento local. Isto é onde a Notificação de Objeto JavaScript (JSON) pode ser acessível. Você pode serializar o objeto em uma seqüência de caracteres JSON, dar-lhe uma chave e, em seguida, armazená-lo na web.

Quando você executa este aplicativo agora, ele mostra o objeto de cliente representado como um JSON na Figura 1-52.

FIGURA 1-52 Conteúdo de armazenamento da Web

A disponibilidade de armazenamento na Web local pode melhorar tanto a experiência e desempenho de seus aplicativos da Web, salvando viagens de ida e volta ao servidor para recuperar ou armazenar dados temporários. Você deve considerar o armazenamento da web local como temporário. Mesmo quando você está usando localStorage em oposição a sessionStorage, você deve pensar no armazenamento como temporário e projetar seus aplicativos para que eles possam recuar em valores padrão e comportamento se o usuário limpa o

armazenamento da web. O armazenamento na Web fornece uma maneira de disponibilizar dados localmente e mesmo persistir em sessões de navegador. Essas técnicas funcionam com um site conectado ao vivo.

Se você quiser tornar um aplicativo disponível off-line, de uma maneira desconectada, você pode usar o API AppCache, que é abordada a seguir.

Usando a API AppCache

A capacidade de continuar a trabalhar com aplicações Web quando desligado de uma fonte de Internet tornou-se particularmente importante no mundo móvel de hoje. Esta seção fala sobre como criar um aplicativo que funciona quando desconectado usando o cache de aplicativo API, também chamada comumente de AppCache API.

A API do AppCache torna o conteúdo e as páginas disponíveis mesmo quando um aplicativo da Web está no modo offline. O AppCache armazena arquivos no cache do aplicativo no navegador. Assim como com Armazenamento na Web, a quantidade de dados que o navegador pode armazenar localmente é limitada para uso off-line. Dois componentes que compõem a AppCache API: o arquivo de manifesto e uma API JavaScript para apoiá-lo.

Usando o manifesto AppCache

Especificar que uma página deve estar disponível para uso offline é tão fácil quanto adicionar um atributo a um elemento HTML na página. Aqui está um exemplo:

```
<html manifest="webApp.appcache">  
</html>
```

O atributo manifesto no elemento html informa o navegador que esta página da Web precisa estar disponível offline. O valor do atributo manifesto aponta para um arquivo de manifesto. O nome do arquivo é uma convenção mais do que um requisito. Você pode nomear o arquivo com qualquer coisa, mas a extensão do arquivo geralmente é .appcache.

Dica de exame

Se você realmente deseja alterar a extensão do arquivo, você precisa configurar o servidor web para que sua extensão de arquivo escolhida seja retornada com um tipo MIME de texto / cache-manifesto.

O arquivo de manifesto do cache do aplicativo deve listar todos os arquivos e recursos necessários para ser armazenados para uso off-line. Quando o navegador analisa o atributo manifest do elemento html, ele baixa o

manifesto e o armazena localmente. Ele também garante que ele baixe todos os arquivos listados no manifesto para que eles estejam disponíveis off-line. O arquivo de manifesto contém três seções:

CACHE, NETWORK e FALLBACK. Cada seção pode aparecer apenas uma vez, várias vezes no arquivo, ou não. Cada um atende a um propósito específico de como o aplicativo cache funciona ao lidar com os recursos em cenários específicos. Um arquivo de manifesto típico tem esta aparência:

CACHE MANIFEST

My Web Application Cache Manifest

v.1.0.0.25

#

#Cache Section. All Cached items.

CACHE

/pages/page1.html

/pages/page2.html

#Required Network resources

NETWORK:

login.html

#Fallback items.

FALLBACK:

login.html fallback-login.html

A primeira linha de um arquivo de manifesto deve ser CACHE MANIFEST. O arquivo do manifesto, como código, pode ter linhas de comentário adicionadas a ele para explicações adicionais, como indicado pelo símbolo #. A seção CACHE lista todos os recursos que devem ser armazenados em cache offline. Isso deve incluir todos os arquivos CSS, arquivos JPG, arquivos de vídeo e áudio e qualquer outro recurso para funcionar corretamente.

Se você omitir um item do arquivo de manifesto, ele não será armazenado em cache, o que pode resultar em comportamento inesperado quando o aplicativo é executado off-line.

A seção NETWORK declara todos os recursos que devem estar disponíveis na Internet. Esses itens não podem ser armazenados em cache. Qualquer coisa que a página requer da Internet, como ser incorporado elementos de terceiros, devem ser listados aqui. Se esse recurso não estiver listado aqui, o navegador não sabe para verificar na Internet para ele quando no modo offline. Quando o navegador está no modo offline, ele não tenta ir para a Internet para qualquer coisa, a menos que esteja listado no NETWORK.

A seção FALLBACK permite que você forneça instruções de retorno para o navegador no caso de um item não estar disponível no cache e o navegador estiver no modo off-line. No arquivo de exemplo, se login.html não estiver disponível no cache, render fallback-login.html. Você pode usar atalhos na seção FALLBACK para fornecer redirecionamentos mais gerais, como o seguinte:

```
/resources /resource.jpg
```

Isso informa ao navegador que se o navegador estiver off-line e não puder acessar recursos, deve substituir quaisquer referências a itens na pasta de recursos resource.jpg. Observe que resource.jpg é armazenado em cache porque ele foi especificado na seção FALLBACK. Você não precisa especificar também resource.jpg na seção CACHE.

Usando a API AppCache

Como com o Armazenamento da Web, o cache do aplicativo está disponível em JavaScript como um objeto global. O código a seguir obtém uma referência para o objeto global AppCache:

```
Var appCache = window.applicationCache;
```

Quando você estiver usando o cache do aplicativo para tornar as páginas disponíveis off-line, uma das coisas úteis que você pode fazer quando a página é carregada é verificar seu status. Você consegue isso avaliando a propriedade status do objeto AppCache. A propriedade status pode ser uma dos valores listados na Tabela 1-14.

Status	Descrição
Uncached	O aplicativo da Web não está associado a um manifesto de aplicativo.
Idle	A atividade de cache está ociosa e a cópia mais atualizada do cache está sendo usada.
Checking	O manifesto do aplicativo está sendo verificado para atualizações.
Downloading	Os recursos no manifesto do aplicativo estão sendo baixados.
UpdateReady	Os recursos listados no manifesto foram baixados com êxito.
Obsolete	O manifesto não pode mais ser baixado, portanto, o cache do aplicativo está sendo eliminado.

Depois de saber o status do cache, dois métodos no objeto AppCache podem ser úteis.

A Tabela 1-15 lista essas informações.

TABELA 1-15 Métodos disponíveis com o objeto applicationCache

Metodo	Descrição
swapCache	Indica que o cache deve ser substituído por uma versão mais recente.
Update	Diz ao navegador para atualizar o cache se uma atualização estiver disponível.

Quando o método de atualização é chamado, uma atualização para o cache é preparada. Quando isso estiver pronto para fazer o download, o status do cache do aplicativo é alterado para o UpdateReady. Quando isso é chamado para o método swapCache indica ao aplicativo para alternar para o cache mais recente.

Dica de exame

A chamada para o método de atualização é assíncrona. Portanto, você deve usar Onupdateready para determinar quando a atualização concluiu o processo de download.

Além das propriedades e métodos, o objeto AppCache pode levantar uma série de eventos que você pode manipular. O cache do aplicativo normalmente opera em segundo plano e você não precisará desses eventos. No entanto, em alguns casos, lidar com alguns dos eventos e forçar uma atualização pode ser útil. A Tabela 1-16 lista os eventos disponíveis.

TABELA 1-16 Eventos disponíveis a partir do objeto applicationCache

Evento	Descrição
Onchecking	O navegador está verificando se há uma atualização para o manifesto do aplicativo ou aplicativo está sendo armazenado em cache pela primeira vez.
Onnoupdate	O manifesto do aplicativo não tem nenhuma atualização disponível.
Ondownloading	O navegador está baixando o que foi solicitado a fazer pelo arquivo de manifesto.
Onprogress	Os arquivos estão sendo baixados para o cache off-line. Este evento é acionado relatório de progresso.
Oncached	O download do cache foi concluído.
Onupdateready	Os recursos listados no manifesto foram recentemente redownloaded e o método SwapCache pode ser chamado.

Onobsolete	Um arquivo de manifesto não está mais disponível.
onerror	Ocorreu um erro. Isso pode resultar de muitas coisas. O registro apropriado é necessário para obter as informações e resolver.

A maioria desses eventos pode não ser usada com frequência, se for o caso. O cenário mais comum é lidar com o método `onupdateready` e, em seguida, fazer uma chamada para o método `swapCache`, como neste exemplo:

```

window.onload = function () {
    var appCache = window.applicationCache;

    appCache.ondcached = function (e) { alert("cache successfully
downloaded."); };

    appCache.onupdateready = function (e) { appCache.swapCache(); };
}

```

Usar o cache do aplicativo é mais sobre a configuração do que sobre a codificação. No entanto, é importante saber que a API está disponível para cenários avançados em que você precisa de mais controle sobre o processo, ou quando você precisa receber informações oportunas sobre o processo, como por manipulação do evento `onprogress`.

Usando a API de geolocalização

Os serviços de localização tornaram-se uma grande parte da vida das pessoas. Do encaminhamento e navegação para apenas encontrar pontos de interesse próximos ou verificar em seu favorito social, comunidade locais, mais e mais pessoas estão usando algum tipo de serviços de localização. Localização dependem do Sistema de Posicionamento Global (GPS), endereços IP e outros dispositivos característicos. Você pode tirar proveito da geolocalização em aplicativos da Web, alavancando navegadores que suportam a Geolocation API.

Você pode obter uma referência à API de geolocalização a partir da propriedade `window.navigator`, como seguinte:

```
Var geoLocator = window.navigator.geolocation;
```

Esse código salva uma referência à API de geolocalização em uma variável para fornecer acesso à API durante o uso futuro. Uma boa prática é garantir que o navegador do cliente suporta a API de geolocalização, certificando-se de que a referência está realmente presente.

A API de geolocalização suporta três métodos principais que você usa para interagir com ele:

GetCurrentPosition, watchPosition e clearWatch.

Usando o método getCurrentPosition

Aqui está um exemplo de como usar o método getCurrentPosition:

```
GetCurrentPosition (positionCallback, [positionErrorCallback],  
[positionOptions])
```

Você usa getCurrentPosition para obter exatamente o que seu nome indica - a posição atual do usuário ou do dispositivo no qual o aplicativo está sendo executado. Este método requer um parâmetro e dois parâmetros opcionais. O primeiro parâmetro é um método de retorno de chamada que a API chamadas após a posição atual ser determinada. O segundo parâmetro é opcional, mas também é uma função chamada callback quando ocorre um erro. O método de retorno de chamada que você especificar aqui deve lidar com quaisquer erros que possam ocorrer ao tentar obter a posição atual. A última opção é um objeto especial chamado PositionOptions, que permite definir algumas opções especiais que controlam como o método getCurrentPosition se comporta. A Tabela 1-17 lista os valores possíveis.

TABELA 1-17 Propriedades disponíveis no objeto PositionOptions.

Propriedade	Descrição
enableHighAccuracy	Isso faz com que o método seja mais intensivo em recursos se definido como true. O padrão é false. Se true, o método getCurrentPosition tenta obter o mais próximo como pode para a localização real
timeout	Isso especifica um período de tempo limite para quanto tempo o método getCurrentPosition pode levar para concluir. Esse número é medido em milissegundos e padrões Para zero. Um valor de zero representa infinito.
MaximumAge	Se isso for definido, a API está sendo informada para usar um resultado armazenado em cache se disponível, Fazer uma nova chamada para obter a posição atual. O padrão é zero, então uma nova chamada é sempre feita. Se maximumAge estiver definido como um valor eo cache Não é mais antiga do que a idade permitida, a cópia em cache é usada.

	Esse valor é medido em milissegundos.
--	---------------------------------------

A Listagem 1-5 mostra o método `getCurrentPosition` em uso, com todos os parâmetros especificados.

LISTAGEM 1-5 Usando o método `getCurrentPosition`

```
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta charset="utf-8"/>
<title></title>
<script>
window.onload = function () {
var geoLocator = window.navigator.geolocation;
var posOptions = {enableHighAccuracy: true,timeout: 45000};
geoLocator.getCurrentPosition(successPosition, errorPosition,
posOptions);
}
function successPosition(pos) {
alert(pos);
}
function errorPosition(err) {
alert(err);
}
</script>
</head>
<body>
<div id="geoResults">
<p>Current Location is:</p>
</div>
</body>
</html>
```

Quando o código é executado no navegador, algumas coisas interessantes podem acontecer. Primeiro, o navegador por segurança começa; Os usuários são perguntados se eles querem permitir que este aplicativo determine localização. No Internet Explorer, a mensagem se parece com a imagem na Figura 1-53.

FIGURA 1-53 O aviso de segurança apresentado pelo Internet Explorer ao acessar a API de geolocalização

Se o usuário optar por permitir que a aplicação prossiga, tudo é ótimo. Caso contrário, o método gera uma exceção. Para fins de demonstração do código, selecione permitir para que a página possa prosseguir. Pode demorar alguns segundos, mas a chamada retorna e mostra a caixa de mensagem que um objeto de posição existe como passado para o método de retorno de chamada de sucesso.

Ambos os métodos de retorno de chamada de êxito e erro recebem um parâmetro da Geolocalização API. O método de sucesso recebe um objeto de posição, enquanto o método de erro recebe um objeto de erro. O objeto de posição expõe duas propriedades: `coords` e `timestamp`. O `Timestamp` propriedade indica a hora em que os `coords` foram recebidos. A propriedade `coords` é um objeto de coordenadas que contém a latitude, longitude, altitude, título e velocidade da posição atual do dispositivo e / ou relativamente à última posição adquirida. O `positionError` contém duas propriedades: uma para o código e outra para a mensagem. Você pode usar estes objetos na Listagem 1-5 adicionando os seguintes fragmentos:

```
<script>

function successPosition(pos) {
    var sp = document.createElement("p");
    sp.innerHTML = "Latitude: " + pos.coords.latitude +
    " Longitude: " + pos.coords.longitude;
    document.getElementById("geoResults").appendChild(sp);
}

function errorPosition(err) {
    var sp = document.createElement("p");
    sp.innerHTML = "error: " + err.message; + " code: " + err.code;
    document.getElementById("geoResults").appendChild(sp);
}

</script>
```

A Figura 1-54 mostra a saída de executar este código com êxito.

FIGURA 1-54 Exibindo o local atual recuperado pela Geolocation API

Usando o método watchPosition

O segundo método disponível no objeto de geolocalização é o método `watchPosition`, que fornece um mecanismo embutido que pesquisa continuamente para a posição atual. Aqui está um exemplo de utilização do método:

```
GeoLocator.watchPosition (successCallBack, errorCallback,  
positionOptions)
```

O método `watchPosition` tem o mesmo conjunto de parâmetros que o método `getCurrentPosition` mas retorna um objeto `watchPosition`:

```
Var watcher = geoLocator.watchPosition ...
```

Depois de executar este código, a variável `watcher` contém uma referência para o `watchPosition` instância sendo invocada, o que pode ser útil mais tarde. O método chama o retorno de chamada de sucesso

Cada vez que a Geolocation API detecta um novo local. A votação continua para sempre a menos que você pare. Este é o lugar onde o objeto `watcher` vem a calhar; Você pode cancelar a pesquisa por Chamando o método `clearWatch`. Você pode chamar esse método no sucesso ou no erro

Callback - por exemplo, para cancelar a pesquisa quando tiver capturado informações de posição suficientes

Ou quando você deseja interromper a pesquisa por um período de tempo:

```
GeoLocator.clearWatch (watcher);
```

A Listagem 1-6 mostra o código completo da solução para o exemplo `watchPosition`.

LISTA 1-6 Uso da API de geolocalização para monitorar a posição

```
var watcher;  
var geoLocator;  
window.onload = function () {  
    geoLocator = window.navigator.geolocation;  
    var posOptions = {enableHighAccuracy: true, timeout: 45000};  
    watcher = geoLocator.watchPosition(successPosition, errorPosition,  
posOptions);  
}  
function successPosition(pos) {
```

```

var sp = document.createElement("p");
sp.innerText = "Latitude: " + pos.coords.latitude + " Longitude: "
+ pos.coords.longitude;
document.getElementById("geoResults").appendChild(sp);
geoLocator.clearWatch(watcher);
}

function errorPosition(err) {
var sp = document.createElement("p");
sp.innerText = "error: " + err.message; + " code: " + err.code;
document.getElementById("geoResults").appendChild(sp);
}

```

A Figura 1-55 mostra a saída desse código em um dispositivo móvel.

FIGURA 1-55 Várias posições sendo gravadas pelo método watchPosition

Experimento de pensamento

Combinando as APIs JavaScript

Neste experimento mental, aplique o que aprendeu sobre esse objetivo. Você pode

Encontre respostas para essas perguntas na seção "Respostas" no final deste capítulo.

Considere um aplicativo como um utilitário em execução ou andando que mede a distância

viajei. Uma característica deste aplicativo é reproduzir uma rota completa para

Se assim o desejarem. Como você usaria a Geolocation API and Web Storage

API em combinação para salvar os pontos de dados à medida que os usuários percorrem a rota para que a aplicação

Pode reproduzi-los mais tarde? (Suponha que você está interagindo com o software de mapas

Para desenhar as linhas.)

Resumo do objetivo

■ ■ A nova API de Armazenamento Web permite armazenar dados localmente no computador cliente.

■ ■ O **Web Storage** suporta **localStorage** e **sessionStorage**.

■ ■ Os dados no Armazenamento da Web são armazenados como pares de chaves e valores.

■ ■ A API AppCache fornece uma maneira de tornar as páginas da web disponíveis quando os usuários estão offline.

■ ■ O manifesto do AppCache define o que está disponível off-line.

■ ■ A Geolocation API fornece uma maneira de integrar serviços de localização em uma página da Web.

■ ■ A Geolocation API fornece dois métodos: `getPosition` e `watchPosition`.

Revisão objetiva

Responda às seguintes perguntas para testar seu conhecimento da informação neste objetivo.

Você pode encontrar as respostas a essas perguntas e explicações de por que cada escolha de resposta é

Correta ou incorreta na seção "Respostas" no final deste capítulo.

1. Ao usar a API de armazenamento Web, onde você deve armazenar dados para garantir quando o usuário fecha o navegador?

A. localStorage

B. cookieStorage

C. sessionStorage

D. Um elemento de entrada oculto

2. O que você precisa fazer para designar uma página como disponível off-line?

A. Especifique em JavaScript como `document.offLine = true`.

B. Especifique o atributo manifesto no elemento de formulário.

C. Especifique o atributo manifesto no elemento HTML.

D. Diga aos usuários para alternar para o modo off-line usando seu navegador. Nenhum código é necessário.

3. Quais das seguintes não são seções válidas do manifesto do AppCache?

A. Manifesto do cache

B. Manifesto da sessão

C. Manifesto de rede

D. Manifestação de reposição

4. Qual evento é disparado pelo objeto AppCache quando o download do cache está concluído?

A. oncached

B. onupdateready

C. emdownloading

D. onchecking

5. Ao usar a Geolocation API, como você configura a capacidade de usar dados em cache?

A. Defina a propriedade enableCache como true no objeto PositionOptions.

B. Defina a propriedade maximumAge como um valor diferente de zero no objeto PositionOptions.

C. Defina a propriedade timeout do objeto PositionOptions.

D. O uso do cache está sempre ativado para economizar largura de banda, portanto nenhuma configuração é necessária.

Objetivo 1.5: Estabelecer o escopo dos objetos e Variáveis

Um componente-chave de qualquer linguagem de programação é como ele usa variáveis e o JavaScript não é exceção. Para usar as variáveis efetivamente, você deve entender seu escopo e vida útil.

Declarar variáveis e instanciar objetos consome recursos. O recurso do sistema primário usado por variáveis é a memória. Quanto mais memória um aplicativo usa, maior o potencial de que o uso de outros recursos também aumentará - como a energia do uso de memória adicional.

Este objetivo abrange como:

- Estabelecer o tempo de vida das variáveis eo escopo da variável
- Evite usar o namespace global

- Aproveite a palavra-chave **this**

Estabelecimento do tempo de vida das variáveis e do escopo variável

As variáveis começam sua vida com uma declaração. O restante da vida dentro da aplicação depende de onde as variáveis são declaradas.

Para declarar uma variável em JavaScript, use a palavra-chave **var**.

```
var myVariable;
```

Você pode declarar muitas variáveis simultaneamente. Por exemplo, o código a seguir declara três variáveis:

```
var x, y, z;
```

Você também pode inicializar suas variáveis em linha com a declaração, dando-lhes valores padrão:

```
Var x = 0, y = 0, z = 0;
```

Até que uma variável seja inicializada, ela não é realmente "viva" - tem um valor de indefinido. Depois de uma variável estar disponível para uso, é considerado "no escopo". A duração durante a qual a variável permanece no escopo depende de onde a variável é declarada. Uma variável que tem escopo global está disponível em toda a página da Web. Uma variável com escopo local está disponível somente onde foi especificado, como mostra a Listagem 1-7.

LISTA 1-7 Um exemplo para demonstrar o escopo de variável

```
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta charset="utf-8"/>
<style>
div {
width: 100px;
height: 100px;
border: 1px solid black;
}
</style>
<script>
var globalVar = "global";
```

```

window.onload = function () {
var localVar = "local";
document.getElementById("Div1").onclick = function () {
var insideDiv1Click;
//Do some logic here...
};
document.getElementById("Div2").onclick = function () {
};
document.getElementById("Div3").onclick = function () {
};
function AFunction() {
var x;
}
function BFunctionWithParam(p) {
}
}
</script>
</head>
<body>
<div id="Div1"></div>
<div id="Div2"></div>
<div id="Div3"></div>
</body>
</html>

```

Na Listagem 1-7, observe que o bloco <script> cria uma seção de script disponível para a página inteira. A primeira linha na seção de script é a variável **globalVar**, que é considerada global para toda a página. Qualquer JavaScript em qualquer lugar nesta página poderia acessar essa variável. Em próximo nível, o código implementa um manipulador de eventos window.onload. Dentro deste manipulador de eventos, a primeira linha declara uma variável chamada **localVar**, que é local para o manipulador de eventos onload. Dentro do manipulador de eventos onload, o código tem acesso à variável **globalVar**. Agora as coisas começam a ficar interessantes.

O manipulador de eventos onload acessa o DOM para conectar alguns outros manipuladores de eventos. Dentro desses manipuladores de eventos, o código tem acesso às variáveis `localVar` e `globalVar`. A variável `localVar` é local para o evento onload, mas porque os outros manipuladores de eventos são declarados no manipulador de eventos onload, eles também têm acesso a variáveis locais declaradas no manipulador de eventos onload. Atualize o manipulador do onclick Div1 para este código:

```
document.getElementById("Div1").onclick = function () {  
    var insideDiv1Click = "insideDiv1";  
    alert(globalVar);  
    alert(localVar);  
    alert(insideDiv1Click);  
};
```

Quando esse código é executado e um usuário clica no elemento Div1, todos os três alertas exibem com êxito

O valor de cada variável, o que significa que eles estão todos no escopo.

Agora, atualize o manipulador onclick Div2 com este código, o mesmo que foi colocado em Div1:

```
document.getElementById("Div2").onclick = function () {  
    alert(globalVar);  
    alert(localVar);  
    alert(insideDiv1Click);  
};
```

Quando você executar esse código e clique Div2, a variável `globalVar` está no escopo, o `localVar` está no escopo, mas a variável `insideDiv1Click` não está no escopo. Essa variável só vive dentro do manipulador onclick Div1, por isso está no escopo apenas enquanto essa função é executada. Este exemplo gera uma exceção indefinida quando ele tenta acessar a variável `insideDiv1Click`.

Como exemplo final, atualize o código para Div3 e outras funções da seguinte maneira:

```
document.getElementById("Div3").onclick = function () {  
    var insideDiv3 = "Div3";  
    AFunction();  
    BFunctionWithParam(insideDiv3);
```

```

};

function AFunction() {

var x;

alert(insideDiv3);

}

function BFunctionWithParam(p) {

alert(p);

alert(localVar);

}

```

Neste código, a variável `insideDiv3` é local para o manipulador de eventos `onclick` para `Div3`. O manipulador de eventos `onclick` tem acesso às variáveis `globalVar` e `localVar` como os outros manipuladores de eventos fizeram. O manipulador de eventos para `Div3` também chama o `AFunction` e `BFunctionWithParam` métodos. O método `AFunction` tenta acessar o `insideDiv3` variável. Infelizmente, essa variável vive apenas dentro do escopo do manipulador `onclick Div3`.

As funções chamadas a partir da função de evento de clique `Div3` não herdam ou não têm acesso as variáveis locais do método `Div3`. Para acessar as variáveis locais declaradas no evento `Div3` o manipulador de outra função, você **precisa passá-los como parâmetros para essas funções**.

Você também pode ver uma ilustração de passar uma variável local como um parâmetro para outra função no código. Após a chamada para o método `AFunction`, o manipulador de eventos `BFunctionWithParam` método. Esta função espera um único parâmetro chamado `p`. O manipulador de evento `onclick` passa o valor da variável `insideDiv3` para o método. Agora, a variável `p` é uma variável local para o método `BFunctionWithParam`, para que ele possa mostrar o valor da variável `insideDiv3`. Esta é a única maneira de tornar uma variável local de uma função acessível a outra função-passando um parâmetro.

Em seguida, o método `BFunctionWithParam` tenta acessar a variável `localVar`. Ele assume que teria acesso, mas não pelo mesmo motivo que o método `AFunction` não tem acesso à variável `insideDiv3`. A variável `localVar` está acessível apenas para o código `Onload` no qual ele foi declarado. Para funções fora desse escopo ter acesso a ele, você precisa passá-lo como um parâmetro. Mais uma coisa a considerar com relação ao tempo de vida e escopo de variáveis é hierarquia.

Se você planeja usar os valores das variáveis `globalVar` ou `localVar` nos manipuladores de eventos `onclick` para os vários elementos `div`, você não deve declarar quaisquer variáveis no nível local com o mesmo nome. Variáveis com escopo local substituem variáveis de escopo de nível mais alto do mesmo nome. Note que eles não os sobrescrevem, eles os substituem - o que significa que

you cannot access the substituted values. The code below demonstrates this point:

```
window.onload
...
var scaleX = 0.0;
...
document.getElementById("Div4").onclick = function () {
var scaleX = -3;
alert(scaleX);
}
function scaleDiv() {
//code to scale the Div by a factor of scaleX
}
```

In this code, if your intention is to use the variable `scaleX` declared globally in the `ScaleDiv` function, the results should be unexpected. This occurs because the event handler `onclick` also declares a variable called `scaleX`. The value in the alert box inside the `onClick` function is `-3`, not `0.0`, and when the `scaleDiv` function accesses the `scaleX` variable, the value is `0.0`. Problems of scope, like these, highlight why you should always use meaningful names for variables. Meaningful names can help avoid accidental variable names.

Evitando usar o namespace global

The global namespace is where all native JavaScript libraries live. In Internet Explorer, **the `window` object references the global namespace**. Everything that this object exposes is global. The global namespace has much more functionality than this book can cover; However, you have seen that some examples of objects in the global namespace used in this chapter, including the **Web Storage API** and the **Geolocation API**. The global namespace includes other nested namespaces, such as `Math`, `WebSocket` and `JSON`.

The global namespace is available for all code within a session of an application. With the increase in the number of third-party libraries in use, and as applications become more complex and require the use of these libraries, the potential for naming conflicts increases. Names of classes in a namespace should be exclusive. If several developers

definirem um namespace com o mesmo nome, o tempo de execução do JavaScript não pode identificar qual namespace eles pretendiam usar. É por isso que manter seus objetos fora do namespace global é importante.

Uma estratégia para evitar colisões de nomes é criar seus próprios namespaces para as suas bibliotecas JavaScript. Um padrão a considerar criar nomes de namespace exclusivo é o nome de domínio em sentido inverso, como com.microsoft. Como os nomes de domínio são exclusivos, esse padrão ajuda a reduzir a possibilidade de nomear colisões. O código a seguir demonstra essa estratégia para criar um namespace para uma biblioteca desenvolvida para uma livraria:

```
var com = {};  
com.Bookstore = {};  
com.Bookstore.Book = {  
  title: 'my book',  
  genre: 'fiction'  
};  
com.Bookstore.Author = {  
  firstName: 'R',  
  lastName: 'D'  
}
```

Ao criar os objetos dessa maneira, você pode estar razoavelmente certo de que se outro desenvolvedor cria uma biblioteca útil para gerenciar livros que você deseja incluir em seu site, você não terá que preocupar-se com uma colisão de nomeação entre seus objetos do livro e do autor e aqueles fornecidos pela outra biblioteca. Ao desenvolver bibliotecas JavaScript reutilizáveis, nunca implemente seus objetos no namespace global.

Aproveitando palavra-chave this

A palavra-chave `this` é um termo especial que permite aos desenvolvedores de JavaScript referenciar os objetos diretamente. O snippet de código a seguir demonstra o contexto da palavra-chave `this`:

```
<script>  
  
//Here, "this" references the global namespace  
  
this.navigator.geolocation  
  
window.onload = function () {  
  
  //Here, "this" references the window object  
  
  this...
```

```
document.getElementById("aDiv").onclick = function()
{
  //Here, "this" references the DIV element
  this...
}
}
</script>
```

Neste fragmento de código, o primeiro desta referência está no namespace global - portanto, fornece uma referência direta ao namespace global. À medida que você se move para baixo através do código, o contexto da palavra-chave desta alteração. No evento onload para a janela, este refere-se à objeto window (sim, que está no namespace global, mas continue lendo). Dentro da função onclick, a palavra-chave this refere-se ao elemento div retornado do método getElementById. Esta palavra-chave sempre se refere ao objeto que contém o código em execução no momento. Esse é o seu contexto.

Experimento de pensamento

Criando uma biblioteca de objetos personalizados

Neste experimento mental, aplique o que aprendeu sobre esse objetivo. Você pode encontrar respostas para essas perguntas na seção "Respostas" no final deste capítulo.

Você precisa criar vários sites que alavancem o mesmo banco de dados back-end. Você usará AJAX para fazer solicitações de servidor para dados. Você deseja criar uma biblioteca que pode ser usada entre os vários sites. Você está preocupado com nomes de objetos porque você sabe que muitas outras bibliotecas estão integradas nesta solução global.

Como você pode projetar sua biblioteca para que não entre em conflito com outras bibliotecas?

Resumo do objetivo

- As variáveis são indefinidas até serem inicializadas.
- As variáveis são escopadas e acessíveis, dependendo de onde elas são declaradas. Se eles estão dentro de uma função, por exemplo, eles são locais para a função.
- A passagem de parâmetros é a única forma de disponibilizar uma variável local em outra função.
- O namespace global não deve ser usado porque é compartilhado por todos.

■ Você deve aplicar um namespace a objetos personalizados para evitar conflitos no Namespace.

■ This, palavra-chave fornece acesso direto ao objeto que gerou o evento.

Revisão objetiva

Responda às seguintes perguntas para testar seu conhecimento da informação neste objetivo.

Você pode encontrar as respostas a essas perguntas e explicações de por que cada escolha de resposta se é Correta ou incorreta na seção "Respostas" no final deste capítulo.

1. Em JavaScript, como você determina o escopo de uma variável?

A. O escopo de uma variável é global dentro do contexto da página.

B. O escopo de uma variável depende de onde o script é declarado.

C. O escopo de uma variável varia de acordo com o tipo que ele representa.

2. Por que é importante evitar a criação de objetos JavaScript personalizados no Namespace

A. O namespace global é reservado para o navegador.

B. O espaço de nomes global está disponível para todos os aplicativos na sessão e usá-lo pode resultar em um conflito de nomeação.

C. O namespace global cria um risco de segurança para os sistemas dos usuários.

3. Qual palavra-chave JavaScript em um manipulador de eventos pode ser facilmente usada para fazer referência ao objeto que levantou o evento?

A. A palavra-chave it fornece uma referência ao objeto.

B. A propriedade document.current fornece uma referência ao objeto.

C. A palavra-chave this fornece uma referência ao objeto.

D. Não está disponível outra forma senão usar uma consulta de seletor para recuperar o objeto do DOM.

Objetivo 1.6: Criar e implementar objetos e métodos

O JavaScript é uma linguagem de programação orientada a objetos, o que significa que para desenvolver aplicações em JavaScript de forma eficaz, você deve entender como trabalhar com objetos.

Essencialmente, existem dois tipos de objetos em JavaScript:

- Objetos JavaScript **nativos**, fornecidos com o JavaScript em si
- Objetos **personalizados**, que os desenvolvedores criam para representar suas próprias construções de dados e comportamentos.

Em alguns casos, não é necessário criar um objeto totalmente novo. Você pode basear objetos em outros objetos (se forem um subtipo desse objeto) usando herança de objeto, em que um objeto herda todos os atributos e comportamentos de outro objeto, mas também pode implementar aspectos adicionais que lhe são exclusivos.

Os objetos encapsulam a funcionalidade e informações de estado que são relevantes para eles. **A funcionalidade é fornecida na forma de métodos**, enquanto **as informações de estado são forma de propriedades**.

Este objetivo examina o trabalho com objetos em JavaScript

Este objetivo abrange como:

- Implementar objetos nativos.
- Criar objetos personalizados e propriedades personalizadas para projetos nativos usando protótipos e funções.
- Implementar herança
- Implementar métodos nativos e criar métodos personalizados

Implementando objetos nativos

Os objetos nativos estão disponíveis para os desenvolvedores diretamente através do JavaScript. O JavaScript fornece um grande número de objetos que fornecem funcionalidade para facilitar a vida dos desenvolvedores. Apesar de abranger cada objeto nativo em JavaScript está fora do escopo para este livro e este exame, você deve ser capaz de criar e trabalhar com objetos JavaScript nativos.

Alguns objetos nativos estão disponíveis estaticamente, o que significa que você não precisa criar um exemplo deles. Outros exigem que você crie uma instância. Você pode encontrar ambos os tipos entre objetos no namespace global. Um exemplo de um objeto **estático é Math**, que está disponível no namespace global e fornece uma grande quantidade de funcionalidade sem que você tenha que criar uma instância:

```
Var squareValue = Math.sqrt (144);
```

Outros objetos, como Array mostrado no código a seguir, exigem que você crie uma instância para trabalhar com eles:

```
Var listofPrimeNumbers = new Array (1, 2, 3, 5, 7, 11, 13, 17, 19, 23);
```

Esse código introduz a palavra-chave **new**, que você usa para instanciar um objeto. Isso em tempo de execução para alocar um novo objeto do tipo especificado. Neste caso, um novo objeto Array está sendo requerido. A lista após o tipo de objeto Array é chamada de **construtor do objeto**. Essa informação pode ser passada para o objeto como parâmetros para construir o estado inicial do objeto. Alguns objetos têm muitos construtores para escolher, com diferentes conjuntos de parâmetros. A adição de vários construtores é chamada de construtor sobrecarregado.

O JavaScript também fornece objetos wrapper. Por exemplo, estes tipos finais de envolvimento. Tipos nativos são definidos como **integer, string, char**, e assim por diante. Quando uma variável é declarada como esta.

```
var txt = "my long string";
```

```
var num = 5;
```

Você pode acessar o método na variável como este:

```
var index = txt.indexOf("long",0);
```

```
var exp = num.toExponential(5);
```

Os tipos subjacentes para string e integer não possuem nativamente métodos ou funcionalidade;

No entanto, o tempo de execução JavaScript cria um objeto wrapper para eles dinamicamente para que alguns métodos úteis estejam disponíveis. Por exemplo, você pode criar a seguinte sequência de caracteres e número

Variáveis com a palavra-chave new, mas isso não é muito comum.

```
var txt = new String("my long string");
```

```
var num = new Number(5);
```

A sintaxe revisada até agora se aplica a objetos nativos e objetos personalizados.

Objetos são criados pelo desenvolvedor, enquanto objetos nativos são fornecidos pelo JavaScript principal.

Criando objetos personalizados

A criação de objetos personalizados é prática padrão quando se trabalha com informações em aplicativos personalizados. Como o JavaScript é uma linguagem orientada a objetos, você deve aplicar práticas ao desenvolver aplicativos JavaScript. Em quase todos os casos, isso envolve criar objetos personalizados para encapsular a funcionalidade dentro de entidades lógicas.

Por exemplo, o código a seguir cria um objeto de livro. Trata-se de um objeto dinâmico, que ele foi criado inline com uma declaração de variável.

```
var book = {  
  ISBN: "55555555",  
  Length: 560,  
  genre: "programming",  
  covering: "soft",  
  author: "John Doe",  
  currentPage: 5  
}
```

O objeto criado representa um livro. Fornece uma maneira de encapsular em um único objeto as propriedades que se aplicam a um livro - neste caso, uma entidade de livro. O código especifica cinco propriedades. Ao usar a variável de livro, você pode acessar todas as propriedades como outra propriedade; Se desejar, você poderia sair para a tela, colocando os valores no DOM.

As propriedades de um objeto representam seu estado, enquanto que os métodos de um objeto fornecem seu comportamento. Neste ponto, o objeto de livro tem somente propriedades. Para dar ao objeto do livro algum comportamento, você pode adicionar o seguinte código:

```
var book = {  
  ISBN: "55555555",  
  Length: 560,  
  genre: "programming",  
  covering: "soft",  
  author: "John Doe",  
  currentPage: 5,  
  title: "My Big Book of Wonderful Things",  
  flipTo: function flipToAPage(pNum) {  
    this.currentPage = pNum;  
  },  
  turnPageForward: function turnForward() {  
    this.flipTo(this.currentPage++);  
  },  
  turnPageBackward: function turnBackward() {
```

```
this.flipTo(this.currentPage--);  
}  
}
```

No objeto de livro, três métodos foram adicionados: `turnPageForward`, `turnPageBackward`, E `flipTo`. Cada método fornece alguma funcionalidade para o objeto do livro, permitindo que um leitor se mova através das páginas. As partes interessantes deste código são as próprias declarações de função.

Por exemplo, quando você olha para o código para a função `flipTo`, você pode pensar que a função é chamada `FlipToAPage` porque isso é o que foi declarado. No entanto, este não é o caso. Os métodos são chamados usando a propriedade de `alias` que atribuiu a função. Ao usar o código, o runtime sabe que é um método, não uma propriedade, e espera que o método para ser chamado com parênteses:

```
// Esta linha lança uma exceção porque o objecto não suporta este método
```

```
book.FlipToAPage(15);
```

```
// Esta linha funciona porque este é o nome do método.
```

```
book.flipTo(15);
```

Criar objetos inline como o objeto de livro está no exemplo de código anterior é útil somente quando ele é usado na página onde ele é definido, e talvez apenas algumas vezes. No entanto, se você planeja usar um objeto com frequência, considere criar um `protótipo` para que você possa construir um sempre que você precisar dele. Um protótipo fornece uma definição do objeto para que você possa construir o objeto usando a palavra-chave `new`. Quando um objeto pode ser construído, com a palavra-chave `new`, o construtor pode tomar parâmetros para inicializar o estado do objeto, e o próprio objeto pode internamente tomar medidas extras conforme necessário para se inicializar. O código a seguir cria um protótipo para o objeto do livro:

```
function Book() {  
  this.ISBN = "55555555";  
  this.Length = 560;  
  this.genre= "programming";  
  this.covering = "soft";  
  this.author = "John Doe";  
  this.currentPage = 5,  
  this.flipTo = function FlipToAPage(pNum) {  
    this.currentPage = pNum;
```

```

},
this.turnPageForward = function turnForward() {
this.flipTo(this.currentPage++);
},
this.turnPageBackward = function turnBackward() {
this.flipTo(this.currentPage--);
}
}
var books = new Array(new Book(), new Book(), new Book());
books[0].Length

```

Dica de exame

O JavaScript consiste em objetos. Tudo em JavaScript é um objeto. Cada objeto é baseado em um **protótipo**. Sempre que você cria uma nova instância de um objeto, essa instância é baseada no protótipo do objeto.

No código anterior, o objeto Book é construído para que você possa criar um conjunto de propriedades padrão. Em seguida, o código cria um Array contendo uma lista de livros. Você pode acessar cada elemento da matriz para inicializar cada objeto Book conforme for necessário.

Acessar cada elemento Book para fornecer valores de inicialização não é muito eficiente. Seria mais conveniente se o objeto de livro suportasse mais de um construtor. Dessa forma, você poderia criar um livro em branco ou criar um com propriedades específicas. Aqui é onde a prototipagem vem a calhar. O código a seguir cria um protótipo contendo dois construtores que suportam as necessidades de qualquer usuário do objeto Book:

```

function Book()
{
//just creates an empty book.
}

function Book(title, length, author) {
this.title = title;
this.Length = length;
this.author = author;
}

```

```

Book.prototype = {
  ISBN: "",
  Length: -1,
  genre: "",
  covering: "",
  author: "",
  currentPage: 0,
  title: "",
  flipTo: function FlipToAPage(pNum) {
    this.currentPage = pNum;
  },
  turnPageForward: function turnForward() {
    this.flipTo(this.currentPage++);
  },
  turnPageBackward: function turnBackward() {
    this.flipTo(this.currentPage--);
  }
};

var books = new Array(new Book(), new Book("First
Edition",350,"Random"));

```

Com este novo código, você pode criar um objeto de livro vazio usando o construtor ou você pode criar um objeto Book usando parâmetros específicos para inicializar os campos.

Os objetos podem conter outros objetos conforme necessário. Neste exemplo, a propriedade Author poderia ser facilmente incorporado num novo protótipo, tornando-o mais extensível e encapsulando informações relacionadas a um autor. Adicione o seguinte código ao protótipo Book:

```

Book.prototype = {
  ISBN: "",
  Length: -1,
  genre: "",
  covering: "",
  author: new Author(),

```

```

currentPage: 0,
title: "",
...
}
function Author(){
}
function Author(firstName, lastName, gender)
{
this.firstName = firstName;
this.lastName = lastName;
this.gender = gender;
}
Author.prototype = {
firstName:"",
lastName:"",
gender:"",
BookCount: 0
}
var books = new Array(new Book(),
new Book("First Edition",350, new Author("Random","Author","M"))
);

```

Agora, o autor do livro é um objeto personalizado em vez de apenas uma sequência de caracteres. Isso fornece mais extensibilidade na concepção. Se você decidir mais tarde que você precisa adicionar informações sobre o autor, você pode simplesmente adicionar a propriedade para o protótipo Author.

Dica de exame

Você pode adicionar propriedades a um protótipo dinamicamente em vez de usar o método anterior. O seguinte código obtém o mesmo resultado. Usar esse código é apenas uma questão de preferência.

```
Book.prototype.ISBN = "";
Book.prototype.Length = 350;
Book.prototype.genre = "";
Book.prototype.covering = "";
Book.prototype.author = new Author();
Book.prototype.currentPage = 0;
Book.prototype.title = "";
```

Implementando herança

Na programação orientada a objetos, a herança é um conceito fundamental. No padrão Programação orientado a objetos, as classes são criadas em uma hierarquia relacional, de modo que os atributos e funcionalidade de uma entidade pode ser reutilizada dentro de outra entidade sem ter que recriar o código. No jargão orientado a objetos, se uma entidade satisfaz a pergunta de relação "é um tipo de", é um candidato para a herança. Por exemplo, uma organização é composta por funcionários,

Uma entidade **empregado** tem certos atributos (propriedades) e comportamentos (métodos). **Gestor**, **executivos** e **funcionários** são todos os tipos de empregados. Um funcionário é um empregado. Assim, em um Projeto objeto orientado, um objeto do funcionário herdaria de um empregado. Este tipo de herança é bastante fácil de construir em linguagens orientadas a objetos de plano direito. No entanto, o JavaScript é um porque não usa classes. Como você viu nas seções anteriores, tudo é um objeto; Um objeto personalizado é composto de propriedades onde algumas propriedades são tipos nativos e algumas propriedades são atribuídas a funções para implementar métodos. Esta seção examina herança de objeto como ele funciona em JavaScript.

Com base no código utilizado na seção anterior, esta seção explica a herança de objectos. No exemplo de código anterior, você criou um objeto chamado Book. Mas muitos tipos de livros existem. Para estender a definição de Livro, você deve separar as diferenças de funcionalidade entre eles, por exemplo, livros pop-up e outros livros. Os livros pop-up têm alguma funcionalidade extra, como exibir o pop-up na página atual e talvez tocar um som. Em outras palavras, enquanto um livro pop-up "é-um" tipo de livro, ele também tem essa funcionalidade extra que não se aplica a todos os livros. Neste caso, seria útil herdar do Livro para que todos os atributos básicos e comportamentos de um livro estão disponíveis sem que você tenha que recriá-los. Então você poderia adicionar a funcionalidade específica para livros pop-up. Você pode estender o objeto Book de algumas maneiras. (**Estender é outra maneira de pensar sobre a herança** - um objeto é estendido para outro objeto.) Aqui está a primeira maneira de se estender um objeto:


```

var popupBook = Object.create(Book.prototype,{ hasSound: {value:true},
showPopUp:{ value: function showPop() {
//do logic to show a popup
}
}
});

```

Object.create é um método disponível a partir da classe Object no namespace global. o

O método create leva dois parâmetros: o objeto que você deseja criar e uma lista de propriedades

Descritores.

O primeiro parâmetro espera receber o protótipo do objeto para criar ou null. E se

Null é especificado, o objeto usa somente as funções ou propriedades especificadas no segundo

parâmetro. Se um protótipo de objeto for especificado, como no caso Book.prototype, o objeto

É criado com todas as propriedades e funções declaradas nesse protótipo de objeto. Isto é

Outra razão para projetar código de uma maneira adequada orientada a objetos é importante - para que você possa

Aproveitar esse tipo de funcionalidade para manter o código mais legível e sustentável.

O segundo parâmetro permite adicionar propriedades ou comportamentos ao objeto que está sendo criada. Essencialmente, você define essa informação adicional do protótipo em linha com o objeto criação. Este exemplo adiciona a propriedade hasSound, que tem um valor padrão especificado como falso. Você também pode especificar informações adicionais aqui, como se a propriedade é Somente leitura e se é enumerable. A criação de objetos dessa forma é semelhante ao exemplo embutido No início da seção anterior sobre objetos personalizados. Novamente, essa abordagem não é Muito modular ou reutilizável. Para cada instância de um livro pop-up, você precisa declarar o Propriedade adicional e método. Então, novamente, para objetos que você pode querer reutilizar muitas vezes,

Estender o protótipo Livro é melhor.

Estender o protótipo do Livro é muito parecido com a criação de um novo protótipo. Você precisa Apenas uma linha de código para dizer ao JavaScript para herdar a funcionalidade e os atributos de outro objeto. Você faz isso inicializando o protótipo para o objeto pai:

```

function PopUpBook() {
Book.call(this);
}
PopUpBook.prototype = Book.prototype;
PopUpBook.prototype.hasSound = false;
PopUpBook.prototype.showPopUp = function ShowPop() {};

```

Desta forma, o PopUpBook agora estende a implementação do objeto Book e adiciona sua funcionalidade para reutilização. A função PopUpBook faz uma chamada de método para Book.call(..). este é uma chamada para o construtor da classe super (a classe sendo herdada de). Se a classe super tem um construtor que leva parâmetros, este método permitirá você passar o parâmetro de valores para os construtores de super-classe para inicialização de objeto.

Experimento de pensamento

Criar sinergia entre objetos personalizados e objetos nativos

Neste experimento mental, aplique o que aprendeu sobre esse objetivo. Você pode encontrar respostas para essas perguntas na seção "Respostas" no final deste capítulo.

Neste objetivo você viu o uso de objetos nativos e objetos personalizados. Como você uniria esses dois mundos? Em JavaScript, herdando de objetos não é totalmente suportado. No entanto, e se você quisesse adicionar funcionalidade para objetos nativos estendendo-os? Como você iria fazer isso?

Resumo do objetivo

- ■ Tudo em JavaScript é um objeto - funções par.
- ■ O JavaScript suporta objetos nativos e objetos personalizados.
- ■ Os objetos são criados com a palavra-chave new.
- ■ Acessar métodos e propriedades em objetos com a notação ponto: object.method ou Object.property.
- ■ Você pode criar objetos personalizados dinamicamente ou usando protótipos.
- ■ Os protótipos fornecem a reutilização da definição de objeto, enquanto os objetos dinâmicos requerem atributos e métodos definidos para cada uso.
- ■ A herança é conseguida em JavaScript através da extensão de protótipos.

Revisão objetiva

Responda às seguintes perguntas para testar seu conhecimento da informação neste objetivo. Você pode encontrar as respostas a essas perguntas e explicações de por que cada escolha de resposta é Correta ou incorreta na seção "Respostas" no final deste capítulo.

1. No JavaScript, qual dos seguintes itens não é um objeto nativo?
 - A. Uma função
 - B. Array
 - C. Inteiro

D. Pessoa

2. Qual dos snippets a seguir mostra a maneira correta de criar um objeto de livro personalizado?

A. `var book = "Título: 'Meu livro sobre coisas'" + "Autor: 'Jane Doe'" + "Páginas: 400";`

B. `var book = {Título: "Meu livro sobre as coisas", Autor: "Jane Doe", Páginas: 400};`

C. `var book = (Title = "Meu livro sobre as coisas", Author = "Jane Doe" = Páginas: 400);`

D. `var book = new {Título: "Meu livro sobre as coisas", Autor: "Jane Doe", Páginas: 400};`

3. Herança é realizada em JavaScript através do uso de que construir?

A. herda a palavra-chave

B. implementa palavra-chave

C. esta palavra-chave

D. Protótipos

Capítulo 2

Implementando fluxo do programa

Ser capaz de manipular o Document Object Model (DOM), criar animações e usar as várias interfaces de programação de aplicativos (APIs) fornecidos pela biblioteca JavaScript é uma grande habilidade para ter. Para aproveitar ao máximo o poder da experiência do usuário, no entanto, você precisa fornecer aos usuários certas funções do site somente sob certas condições, um conceito conhecido como **fluxo de programa**. Sem o fluxo do programa, os programas JavaScript processariam de cima para baixo na ordem em que o código foi escrito. Isso é útil em alguns casos, mas na maioria das situações em que é necessária uma experiência dinâmica do usuário, a lógica precisa ser processada condicionalmente. **O fluxo do programa pode ser condicional, iterativo ou comportamental:**

■ ■ O fluxo condicional do programa é baseado na avaliação do estado para tomar uma decisão sobre qual código deve ser executado.

■ O fluxo iterativo é a capacidade de processar listas ou coleções de informações sistematicamente e consistentemente.

■ ■ O fluxo comportamental pode ser definido como um evento ou retorno de chamada no qual a lógica específica deve ser aplicada com base no envolvimento do usuário com a aplicação web ou na conclusão de outra tarefa.

O fluxo pode - e quase sempre irá - incluir uma combinação de todos os três.

Outro tipo especial de fluxo de programa envolve o tratamento de exceções. Estruturas de manipulação de exceção fornecem a capacidade de executar lógica específica no caso de um erro no programa.

Objectivos deste capítulo:

- ■ Objetivo 2.1: Implementar o fluxo do programa
- ■ Objetivo 2.2: Levantar e lidar com um evento
- ■ Objetivo 2.3: Implementar o tratamento de exceções
- ■ Objetivo 2.4: Implementar um callback
- ■ Objetivo 2.5: Criar um processo de trabalho na Web (Web Work)

Objetivo 2.1: Implementar o fluxo do programa

Para o exame, você precisa entender o fluxo de programa condicional e iterativo. O fluxo de programa condicional permite que uma aplicação examine o estado de um objeto ou variável para decidir qual caminho de código deve ser processado. Os comandos que você usa para aplicar o conceito de fluxo condicional são a instrução `if ... else`, a instrução `switch` eo operador ternário.

Este objetivo abrange como:

■ ■ Avaliar expressões, incluindo usar instruções `switch`, declarações `if / then` e Operadores.

■ ■ Trabalhar com matrizes

■ Implementar tipos especiais de matrizes

■ ■ Usar métodos avançados de matriz

■ Implementar fluxo de controle iterativo

Avaliando expressões

Para usar uma instrução de **fluxo condicional**, você **deve avaliar alguns dados contra alguma condição**, que você faz usando operadores condicionais. Você pode usar operadores lógicos para combinar operadores condicionais. A combinação de operadores é útil quando mais de uma condição deve ser atendida - ou pelo menos uma condição de um conjunto de condições deve ser atendida - antes de processar uma lógica específica. A Tabela 2-1 descreve os operadores disponíveis.

Operador	Tipo	Descrição
> Maior que	Condicional	Avalia se o valor à esquerda é maior que o valor à direita
< menor que	Condicional	Avalia se o valor esquerda à direita é menor que o valor à esquerda
>= <= (maior/menor igual)	Condicional	Avalia o mesmo como > ou < mas com a lógica adicional que os valores podem ser igual
!= (diferente)	Condicional	Avalia se os valores não são iguais
== (igual valor)	Condicional	Avalia se os valores são iguais independentemente do tipo de dados subjacente
=== (igual tipo/valor)	Condicional	Avalia se os valores são iguais tanto no valor quanto no tipo de dados subjacente
&& (e)	Lógico	O operador lógico AND, no qual as expressões em ambos os lados Avaliar como verdadeiro
(ou)	Lógico	O operador lógico OR, no qual pelo menos uma expressão de cada lado deve avaliar como verdadeiro

Use esses operadores para avaliar dados e tomar decisões. Por exemplo, se um site exige que seus usuários tenham uma idade mínima para se inscrever para uma conta, a lógica na página de inscrição pode incluir algo como isto:

```
if(users age >= minimum age)
{
  //allow sign-up.
}
```

Usando declarações if

Use a **instrução if** para avaliar o estado para controlar a direção na qual o código será executado. A instrução if pode estar sozinha, como mostrado no snippet anterior, ou ser combinada com else para formar construções mais complexas:

```

if(exp1, exp2, exp3...expn){
//true logic
}else {
//false logic
}

```

Esta instrução if começa em uma nova linha com a palavra-chave if. Em parênteses seguindo a declaração é uma série de uma ou mais expressões, separadas por operadores lógicos quando mais de uma expressão é fornecida. O bloco de código imediatamente após a expressão condicional de instrução if é executado somente quando a expressão é avaliada como verdadeira. Quando a expressão é avaliada como falsa, o bloco imediatamente após a palavra-chave else é executado.

A palavra-chave else é opcional. Uma instrução if pode existir como uma declaração autônoma quando a lógica não está disponível para executar quando a expressão é avaliada como false.

Dica de exame

Dois operadores condicionais estão disponíveis para verificar a igualdade: == (operador de igualdade) e === (Operador de identidade). Verificar a igualdade com o operador == irá ignorar o subjacente tipo de dados, enquanto o operador de identidade === considerará o tipo de dados. Olhe para o seguinte

exemplo:

```
var n = 2000, s = '2000';
```

```
alert(n == s); true
```

```
alert(n === s); false
```

A primeira expressão, que utiliza o operador de igualdade, é avaliada como verdadeira porque a cadeia é moldada para um número para efeitos da avaliação. A segunda expressão, que usa o operador de identidade, é avaliada como falsa porque a string '2000' não é igual ao inteiro 2000.

As instruções condicionais, como a instrução if, podem ser aninhadas, como neste exemplo:

```

var userAge = 10, gender = 'M';
var minimumAge = 11;
if (userAge > minimumAge) {
  if (gender == 'M') {
    //do logic for above age male
  }
}

```

```

else {

//do logic for above age female.

}

} else if (gender == 'M') {

//do logic for underage male

} else {

//do logic for underage female.

}

```

Neste exemplo, a lógica verifica se um usuário é mais antigo do que uma idade especificada. Se o usuário estiver sobre a idade especificada, a lógica no ramo verdadeiro será executada. Nesse ponto, outra declaração if avalia o sexo do usuário. Se a idade do usuário for menor do que o mínimo exigido, o ramo falso é processado. Aqui uma instrução else if executa processamento condicional adicional no ramo falso com base no sexo. Novamente, o código processa um ramo específico dependendo

Sobre se o usuário é masculino ou feminino.

Você não está limitado em quão profundamente você pode aninhar as declarações, mas aninhando-os muito profundamente pode tornar o código bastante confuso e difícil de ler.

O exemplo a seguir examina a cor de plano de fundo de um elemento e processa um comportamento específico com base na cor:

```

var canvas = document.getElementById("canvas1");

if (canvas.style.backgroundColor == 'green') {

alert('proceed');

} else if (canvas.style.backgroundColor == 'yellow') {

alert('slow down/safely stop');

} else if (canvas.style.backgroundColor == 'red') {

alert('stop');

}

```

Esse código recupera uma referência a um elemento de página chamado canvas1 e, em seguida, avalia a cor de plano de fundo desse elemento para determinar uma ação apropriada a ser executada. Em alguns lugares, por lei ou preferência, o amarelo às vezes significa "avançar mais rápido". Nesse caso, o código poderia ser adaptado para usar o operador lógico OR:

```

var canvas = document.getElementById("canvas1");

```

```

if      (canvas.style.backgroundColor == 'green' ||
canvas.style.backgroundColor == 'yellow')
{
    alert('proceed');
} else if (canvas.style.backgroundColor == 'red') {
    alert('stop');
}

```

Este código fornece a instrução "continuar" quando a cor é verde OU amarelo.

Dica de exame

Ao usar o operador OR lógico em uma instrução if, o mecanismo JavaScript sabe que pode prosseguir se alguma das instruções for verdadeira. Como tal, avalia as expressões da esquerda para a direita até encontrar uma que é verdadeira. Assim que o fizer, ele não avaliará quaisquer outras expressões, mas imediatamente saltará para o bloco de código verdadeiro. No exemplo anterior, se o fundo for verde, a verificação para saber se o fundo é amarelo nunca seria avaliada.

Estruturar código desta forma é sintaticamente correto. No entanto, as instruções longas podem provar difícil de ler e ainda mais difícil de manter. Se suas instruções if estão se tornando bastante longas - por exemplo, se o exemplo de código anterior tivesse que testar 15 cores diferentes - uma instrução switch poderia ser mais apropriada.

Usando instruções switch

A instrução switch fornece uma construção na qual você pode testar uma lista de valores para igualdade (como com o operador ==). O exemplo a seguir demonstra uma instrução switch:

```

switch (canvas.style.backgroundColor) {
    case 'yellow':
        alert('slow down');
        break;
    case 'green':
        alert('proceed');
}

```



```

        break;
    case 'red':
        alert('stop');
        break;
    default:
        alert('unknown condition');
        break;
}

```

A instrução switch consiste em várias partes. O primeiro é a própria **palavra-chave switch**, seguido por parênteses em torno de uma expressão para avaliar. Este exemplo específico avalia a cor de fundo do elemento canvas.

Seguindo a linha de comutação é uma série de declarações de caso encerrado em chaves. A instrução case fornece os valores a avaliar contra. Este exemplo fornece três casos para avaliar: um para cada uma das possíveis cores de fundo vermelho, verde e amarelo.

Cada instrução case contém uma palavra-chave **break** necessária. Esta palavra-chave **indica o final dessa instrução de caso particular**. Somente o primeiro caso que avaliar true em uma instrução switch será processado. A omissão da palavra-chave break causará um comportamento inesperado. A última parte da instrução switch é a palavra-chave **default opcional**, que serve como um failsafe. Se nenhuma das declarações de caso é avaliada como verdadeira, a instrução padrão fornece uma maneira de lidar com a situação.

Talvez você não queira tomar nenhuma ação quando nenhuma das declarações de caso for avaliada como verdadeira, caso em que você pode omitir a instrução padrão. No entanto, ele permite que você lide com o cenário em que uma das condições deveria ter sido atingida, mas não foi, possivelmente devido a dados ruins sendo passado para um método ou um caso válido ser perdido incluindo um padrão para contabilizar ambos os cenários. É uma boa prática.

Você não pode forçar o fluxo lógico em uma instrução switch para mover de um caso para o seguinte omitido a palavra-chave break; Em outras palavras, **apenas um bloco condicional é processado dentro de uma instrução switch**. Isso significa que logicamente, você não pode usar o operador lógico **AND**. No entanto, você pode aproveitar o operador lógico **OR**. **O código a seguir demonstra um caso no qual você deseja que o mesmo código seja executado para as condições de plano de fundo verde e amarelo:**

```

switch (canvas.style.backgroundColor) {
    case 'yellow':
    case 'green':

```

```

        alert('proceed');
        break;
    case 'red':
        alert('stop');
        break;
    default:
        alert('unknown condition');
        break;
}

```

Neste código, várias instruções de maiúsculas e minúsculas são empilhadas uma sobre a outra. Se qualquer uma das instruções de caso empilhado é avaliada como verdadeira, o bloco de código que segue essa instrução de caso é processado, implicando assim um OR lógico. Você não precisa usar explicitamente o operador OR lógico (||) para levantar lógica OU semântica.

IMPORTANTE UMA DECLARAÇÃO DE COMUTADOR VÁLIDA

Os valores utilizados na declaração do caso para efeitos da avaliação devem ser expressos como uma **constante**. Por exemplo, a ativação de um valor inteiro para determinar se é divisível por outro número não funcionará porque o caso **exigiria uma expressão** em vez de **um valor constante**. Por exemplo, caso $x / 10$: seria uma instrução caso inválido. No entanto, a instrução switch mesmo pode aceitar uma expressão para avaliar contra todos os casos dentro do bloco de comutação.

Usando operadores ternários

O operador ternário é essencialmente um mecanismo abreviado para uma instrução if. A sintaxe da operação ternária é

<expression> ? <true part>: <false part>

Quando a expressão é avaliada como verdadeira, a parte verdadeira é executada; Caso contrário, a parte falsa é executada.

Este código demonstra usando o operador ternário para verificar a cor de plano de fundo da tela:

```

canvas.style.backgroundColor == 'green' ? document.write('proceed') : document.write('stop');

```

Trabalhando com Arrays

Arrays são objetos JavaScript e são criados exatamente como qualquer outro objeto JavaScript, com a palavra-chave `new`.

```
var anArray = new Array();  
var anArray = new Array(5);  
var anArray = new Array('soccer', 'basketball', ..., 'badminton');
```

Este exemplo de código mostra um objeto Array sendo instanciado e demonstrando os três construtores disponíveis. A primeira linha cria um objeto Array vazio sem um tamanho padrão. A segunda linha cria um objeto Array com um tamanho padrão. Cada valor na matriz é indefinido porque nada é atribuído a ele ainda. O último exemplo cria uma matriz inicializada com dados. Além dos construtores de objetos, você pode criar uma matriz da seguinte maneira:

```
var anArray = ['soccer', 'basketball', ..., 'badminton'];
```

Sob o capuz, o JavaScript converte a variável em um array no tipo de objeto Array. Depois de criar uma matriz, você pode acessar seus elementos usando colchetes seguindo o nome da variável, como mostrado neste exemplo:

```
var anArray = new Array(5);  
anArray[1] = 'soccer';  
alert(anArray[1]);
```

Você acessa elementos dentro de uma matriz por sua posição indexada. Este exemplo acessa o elemento na posição de índice 1 e atribui um valor a ele. Arrays em JavaScript são iniciados em zero, o que significa que o primeiro elemento na matriz está no índice zero, não no índice um. O último elemento está no índice `Array.length - 1` - no exemplo anterior, $5 - 1 = 4$. Assim, os índices de elementos de matriz são 0, 1, 2, 3 e 4.

Dica de exame

Dimensionar arrays é muito dinâmico. No exemplo anterior, mesmo que a matriz é declarada inicialmente para ter um comprimento de 5, se você tentar acessar o elemento 10, a matriz redimensiona automaticamente para acomodar o comprimento solicitado. O exemplo a seguir demonstra esse conceito:

```
var anArray = new Array(5);  
alert(anArray.length);  
anArray[9] = 'soccer';  
alert(anArray.length);
```

Um array multi-dimensional pode conter outros arrays. O código a seguir demonstra isso:

```
var multiArray = new Array(3);
multiArray[0] = new Array(3);
multiArray[1] = new Array(3);
multiArray[2] = new Array(3);
```

Este exemplo cria um array bidimensional 3 × 3. Não é necessário que cada matriz seja do mesmo tamanho; Este exemplo foi apenas codificado dessa forma. Acessar os elementos de um array bidimensional é muito semelhante ao acesso a um array unidimensional, mas você usa dois índices:

Este exemplo atribui um valor ao primeiro índice da primeira dimensão e ao segundo índice da segunda dimensão, conforme ilustrado na Figura 2-1.

[0][0]	[0][1]	[0][2]
[1][0]	[1][1]	[1][2]
[2][0]	[2][1]	[2][2]

FIGURA 2-1 Layout de um array bidimensional

Como os arrays são objetos, eles expõem uma série de métodos poderosos para facilitar o trabalho com eles. As seções a seguir explicam cada método e propriedade disponíveis.

Usando a propriedade length

A propriedade length fornece informações sobre a duração da matriz - ou seja, quantos elementos a matriz alocou no momento em que a propriedade é avaliada. Esta propriedade é útil para situações em que você precisa iterar em uma matriz ou para mostrar aos usuários quantos itens estão na matriz em um ponto específico no tempo, como em uma fila. O exemplo a seguir mostra como acessar a propriedade length:

```
var anArray = new Array(5);
alert(anArray.length);
```

Muitas funções permitem que você manipule o conteúdo da matriz rapidamente e facilmente.

Dica de exame

Alguns métodos de matriz afetam o objeto Array diretamente, enquanto outros métodos retornam um novo objeto Array. Para o exame, você deve entender quando cada caso é aplicável.

Usando o método concat

O método **concat combina dois ou mais arrays em um array:**

```
var sports = new Array( 'football', 'cricket', 'rugby', 'tennis', 'badminton');  
var moreSports = new Array('soccer', 'basketball', 'hockey');  
var combinedSports = sports.concat(moreSports);
```

A matriz retornada pelo método concat é armazenada na variável combinedSports contém todos os elementos de ambos os arrays em sequência. O conteúdo da matriz moreSports aparece após os elementos da matriz de sports neste exemplo.

Usando os métodos indexOf e lastIndexOf

O método indexOf **fornece uma maneira de encontrar o índice de um elemento conhecido.** O exemplo de código a seguir demonstra isso:

```
var sports = new Array('soccer', 'basketball', 'hockey', 'football', 'cricket',  
'rugby', 'tennis', 'badminton');  
var index = sports.indexOf('football', 0);
```

Este exemplo chama o método indexOf para determinar o índice do elemento soccer. O método indexOf **aceita dois parâmetros: o que pesquisa e o índice no qual começar a procurar.** Este exemplo procura toda a matriz, então a pesquisa **começa no índice 0.** O resultado dessa chamada para o método indexOf é 3, porque o elemento é encontrado na quarta posição. **Se o elemento a ser procurado não for encontrado, o método retorna um valor de -1.**

O método indexOf usa o **operador de identidade** para verificar a igualdade, o que significa que se uma matriz contém seqüências de caracteres como '1', '2' e '3' e você está procurando o inteiro 3, o resultado é -1 porque A operação de igualdade retorna false para todos os elementos na matriz. **O método indexOf pesquisa em ordem ascendente de índice.** Para pesquisar em ordem decrescente - ou seja, para procurar **do final da matriz para o início** - use o **método lastIndexOf**, que aceita os mesmos parâmetros.

Usando o método de associação (join)

O método `join` concatena todos os elementos de uma matriz em uma única sequência de caracteres separados por um separador de sequência especificado. Por exemplo, para converter uma matriz de seqüências de caracteres em uma lista separada por vírgulas, você pode usar o seguinte código:

```
var sports = new Array('soccer', 'basketball', 'hockey', 'football', 'cricket',  
'rugby','tennis', 'badminton');  
var joined = sports.join(',');
```

O método `join` aceita uma seqüência de caracteres como um parâmetro, que é a seqüência usada como um delimitador para separar os valores na matriz. O resultado é uma seqüência de todos os elementos separados pela seqüência de caracteres passada para o método de associação.

Usando o método reverso (reverse)

O método `reverse` **inverte a seqüência de todos os elementos na matriz**. Este exemplo inverte a matriz de esportes:

```
var sports = new Array('soccer', 'basketball', 'hockey', 'football', 'cricket',  
'rugby','tennis', 'badminton');  
sports.reverse();
```

O método inverte todos os itens para que 'futebol' se torne o último item na matriz e 'badminton' se torna o primeiro item.

Usando o método sort

O método `sort` **classifica os itens na matriz em ordem crescente**. Na matriz de esportes, o tipo seria alfabético, como mostrado no exemplo a seguir:

```
var sports = new Array('soccer', 'basketball', 'hockey', 'football', 'cricket',  
'rugby', 'tennis', 'badminton');  
alert(sports.indexOf('soccer'));  
sports.sort();  
alert(sports.indexOf('soccer'));
```

O resultado é que o conjunto de esportes agora está classificado. As caixas de alerta mostram o índice do elemento 'soccer' antes e depois da

classificação, demonstrando que o elemento se moveu da posição 0 para a posição 6 na matriz.

Usando o método de slice

O método slice **remove um ou mais itens em uma matriz e os move para uma nova matriz**. Considere a seguinte matriz com a lista de esportes:

```
var sports = new Array('soccer', 'basketball', 'hockey', 'football', 'cricket',  
'rugby',  
'tennis', 'badminton');  
var someSports = sports.slice(1, 2);
```

O método slice leva dois parâmetros: os índices onde a operação de fatia deve começar e terminar. O índice final não está incluído na fatia. Todos os elementos copiados são retornados. Neste exemplo, porque 'basquete' está no índice 1 e o índice final é especificado no índice 2, a matriz resultante someSports contém apenas um elemento: 'basquete'.

Usando o método splice

O método de splice **fornece uma maneira de substituir itens em uma matriz com novos itens**. O código a seguir demonstra isso:

```
var sports = new Array('soccer', 'basketball', 'hockey', 'football', 'cricket',  
'rugby', 'tennis', 'badminton');  
var splicedItems = sports.splice(1, 3, 'golf', 'curling', 'darts');
```

O método splice retorna uma matriz contendo os itens que são emendado fora da matriz de origem. O primeiro parâmetro é o índice na matriz onde a operação de junção deve iniciar. O segundo parâmetro é o número de itens a substituir, a partir do índice especificado no primeiro parâmetro. O último parâmetro opcional lista os itens que devem substituir os itens que estão sendo processados. A lista não precisa ter o mesmo comprimento que os itens que estão sendo processados. De fato, se o último parâmetro for omitido, os itens emendados simplesmente são removidos da matriz e não substituídos. Neste exemplo, três itens são substituídos, começando pelo índice 1. Assim, 'basquete', 'hóquei' e 'futebol' são substituídos por 'golf', 'curling' e 'dardos'.

Implementando tipos especiais de matrizes

O JavaScript não fornece nativamente um objeto personalizado para representar coleções ou matrizes especiais. Em vez disso, os métodos são fornecidos no objeto Array que permite que você implemente vários tipos de coleções especializadas, como uma **fila** ou uma **pilha**.

Uma fila é essencialmente um tipo de coleção first-in-first-out. Sempre que itens são adicionados à lista, eles devem ir no final da linha. Em contraste, é a pilha, um tipo de coleção de last-in-first-out em que o último item colocado na pilha é o primeiro item que você pode tirar da pilha. Os métodos de matriz que facilitam esse tipo de comportamento são **pop**, **push**, **shift** e **unshift**.

Usando os métodos pop e push

Os métodos **pop** e **push** fornecem funcionalidade de **pilha**. O método **push** adiciona os itens especificados ao final da matriz. O método **pop** remove o último item da matriz. O código a seguir demonstra o método **push**:

```
var sports = new Array();  
sports.push('soccer', 'basketball', 'hockey');  
sports.push('football');
```

Este código cria um objeto Array e, em seguida, **push** (insere) três itens para a matriz. Os itens são adicionados à pilha na mesma ordem em que aparecem na lista de parâmetros. Em seguida, o código empurra um item adicional para a pilha. O método **pop** remove e retorna o último item na matriz:

```
var nextSport = sports.pop();
```

Quando este código é executado, a variável **nextSport** contém o valor 'futebol' porque esse foi o último valor adicionado à matriz.

NOTA UTILIZANDO PUSH E POP EM QUALQUER ARRAY

Você pode usar os métodos **pop** e **push** em qualquer contexto para adicionar e remover itens da extremidade de uma matriz. O conceito de pilha é útil, mas não é um mecanismo de confinamento que limita o uso desses métodos para apenas arrays de pilha.

Usando os métodos shift e unshift

Os métodos shift e unshift funcionam da maneira exatamente oposta à do pop e push

métodos. O método shift remove e retorna o primeiro elemento da matriz, enquanto o método unshift insere novos elementos no início da matriz. O código a seguir usa os métodos shift e unshift:

```
var sports = new Array();  
sports.unshift('soccer', 'basketball', 'hockey');  
sports.unshift('football');  
var nextSport = sports.shift();
```

O resultado líquido deste código é exatamente o mesmo que para o pop e código de envio, exceto as operações ocorrem na frente da matriz em vez da extremidade. Em outras palavras, este exemplo ainda ilustra a funcionalidade de pilha de "último em, primeiro a sair".

Tomados em conjunto, os dois conceitos que você acabou de ver (pop / push e shift / unshift) podem ser

Combinado para criar o conceito de uma fila first-in-first-out. O código a seguir demonstra o uso de uma fila em que a frente da linha é o início da matriz eo final da linha é o final da matriz:

```
var sports = new Array();  
sports.push('soccer');  
sports.push('basketball');  
sports.push('hockey');  
var get1 = sports.shift();  
sports.push('golf');  
var get2 = sports.shift();
```

Esse código primeiro empurra alguns itens para a matriz. Isso significa que cada item é adicionado ao final da matriz. Quando um item é necessário a partir da matriz, o método shift obtém o primeiro item fora do início da matriz - o item no índice 0. Você pode facilmente implementar o mecanismo oposto usando os métodos unshift e pop, que iria atingir o mesmo Resultados, mas entrar e recuperar itens das extremidades opostas da matriz a partir deste exemplo.

Usando métodos avançados de matriz

Esta seção examina alguns dos métodos de matriz mais avançados. Esses métodos envolvem o uso de um retorno de chamada, que você examinará com mais detalhes no Objetivo 2.4, "Implementar um callback". Se os callbacks forem um novo conceito para você, você deve estudar esse objetivo antes de concluir esta seção. O objeto Array expõe métodos que permitem processar a lógica personalizada em cada único elemento na matriz. As seções a seguir demonstram cada método.

Usando o método every

O método every permite processar lógica específica para cada elemento de matriz para determinar se algum deles atende a alguma condição. Olhe para o seguinte código:

```
var evenNumbers = new Array(0, 2, 4, 6, 8, 9, 10, 12);
var allEven = evenNumbers.every(evenNumberCheck, this);
if (allEven) {
  ...
} else {
  ...
}
function evenNumberCheck(value, index, array) {
  return (value % 2) == 0;
}
```

Neste código, suponha que a matriz evenNumber é criada com uma lista do que você esperava ser todos os números pares. Para validar isso, você pode usar todos os métodos.

Cada método tem dois parâmetros:

- O nome da função que deve ser processada para cada elemento
- Uma referência opcional ao objeto de matriz.

A função evenNumberCheck chamada para cada item na matriz retorna true ou false para cada item, dependendo se ele atende aos critérios desejados. Neste exemplo, o valor é testado para garantir que ele é um número par. Se for, a função retorna true; Caso contrário, retorna false. Assim que cada método obtém o primeiro resultado falso para qualquer item na matriz, ele sai e retorna false. Caso contrário, se todos os elementos na matriz retornarem true, o método

every retornará **true**. No exemplo de código anterior, uma instrução **if** foi adicionada para avaliar o valor de retorno de cada método e tomar uma ação apropriada. Neste exemplo, a função **evenNumber-Check** retorna **false** no sexto item na matriz, porque 9 é um número ímpar, de modo que o teste para mesmo falha.

Usando o método **some**

O método **some** funciona muito bem como todo o método. **A diferença é que algumas verificações somente se algum item na matriz atende aos critérios.** Nesse caso, o método **some** retorna **true** se a função chamada retorna **true** para qualquer elemento único. Se todos os elementos na matriz retornam **false**, o método **some** retorna **false**. Por esta definição, você pode usar alguns para conseguir exatamente o oposto de cada método quando o método **some** retorna **false**. O código a seguir é atualizado a partir do exemplo anterior para que ele usa o algum método:

```
var evenNumbers = new Array(0, 2, 4, 6, 8, 9, 10, 12);
var allEven = evenNumbers.some(evenNumberCheck, evenNumbers);
if (allEven) {
    ...
} else {
    ...
}
function evenNumberCheck(value, index, array) {
    return (value % 2) == 0;
}
```

Com o código atualizado para usar o método **some**, o resultado de retorno não é verdadeiro, porque alguns dos valores na matriz são números pares. Se esse resultado retornasse **false**, você saberia que todos os elementos da matriz eram números ímpares.

Usando o método **forEach**

O método **forEach** permite que um aplicativo processe alguma lógica em relação a cada item na matriz. Esse método é executado para cada item único e não produz um valor de retorno. **ForEach** tem a mesma assinatura que os outros

métodos que você viu até agora nesta seção. O código a seguir demonstra o método `forEach`:

```
var sportsArray = ['soccer', 'basketball', 'hockey', 'football', 'cricket', 'rugby'];

sportsArray.forEach(offerSport);

function offerSport(value, index, array) {
  var sportsList = document.getElementById("sportsList");
  var bullet = document.createElement("li");
  bullet.innerText = value;
  sportsList.appendChild(bullet);
}
```

Neste exemplo, o código assume que um elemento de lista na página HTML está pronto para ser preenchido com a lista de esportes, cada formatado como um nó filho. Cada elemento na lista é passado para a função e adicionado como um elemento ``. Os elementos de matriz não são classificados nesse caso. Você pode encadear os métodos juntos para garantir que os elementos sejam, por exemplo, em ordem alfabética:

```
sportsArray.sort().forEach(offerSport);
```

Como com todos os métodos avançados mostrados até agora, os elementos são passados para a função em ordem ascendente de índice. Então você pode chamar o método `sort` e encadeá-lo junto com o método `forEach` para garantir que os elementos sejam exibidos para o usuário em ordem.

Usando o método filter

O método `filter` fornece uma maneira de remover itens para uma matriz com base em algum processamento feito na função de retorno de chamada. O método `filter` retorna uma nova matriz contendo os elementos que são incluídos com base em um valor de retorno de `true` ou `false` da função `callback`. No exemplo de número par, você pode usar o método de filtro para scrub a matriz e garantir que o programa continua a usar apenas uma matriz que contém números pares, como demonstrado aqui:

```
var evenNumbers = new Array(0, 2, 4, 6, 8, 9, 10, 12);

var allEven = evenNumbers.filter(evenNumberCheck, evenNumbers);

// trabalho com os números pares ....

function evenNumberCheck(value, index, array) {
```

```
return (value % 2) == 0;

}
```

Neste exemplo, o método `evenNumberCheck` é o mesmo que o utilizado anteriormente. No entanto, em vez de usar todos ou qualquer método para determinar a qualidade dos dados no que diz respeito a conter apenas números pares, o método de filtro simplifica a remoção dos números ímpares. Você pode usar qualquer lógica na função de retorno de chamada para processar o elemento e determinar se ele deve ser incluído na matriz retornada, como correspondência de padrões ou uma pesquisa de banco de dados.

Usando o método map

O método de `map` permite substituir valores na matriz. Cada elemento na matriz é passado para uma função de retorno de chamada. O valor de retorno da função de chamada substitui o valor para a posição na matriz que foi passada. O exemplo a seguir demonstra ter cada número em uma matriz arredondado apropriadamente:

```
var money = [12.8, 15.9, 21.7, 35.2];

var roundedMoney = money.map(roundOff, money);

...

function roundOff(value, position, array) {
return Math.round(value);
}
```

Este exemplo fornece o quadrado de uma série de números:

```
var numbers = [1, 2, 3, 4, 5, 6, 7, 8];

var squares = numbers.map(squareNumber, numbers);

...

function squareNumber(value, position, array) {
return value * value;
}
```

Usando os métodos reduce e reduceRight

Os métodos `reduce` e `reduceRight` são recursiva. Cada resultado da função de retorno de chamada é passado de volta para o método de retorno de chamada como o valor de retorno anterior juntamente com o elemento atual a ser passado dentro. Isso fornece alguns cenários interessantes. O método `reduce` processa os elementos da matriz em ordem crescente, enquanto que `reduceRight` processa os elementos da matriz em ordem decrescente. O exemplo a seguir demonstra usando o método `reduce` para calcular um fatorial:

```
var numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];  
var factorials = numbers.reduce(factorial);  
function factorial(previous, current) {  
  return previous * current;  
}
```

Nesta função, o fatorial para 10 é calculado. No mundo da matemática, é denotado como 10!

Dica de exame

Algumas funções avançadas permitem alterar a matriz de origem, enquanto outras não. Este é um aspecto importante para manter claro.

Implementando fluxo de controle iterativo

Você já viu como usar as instruções para controlar o fluxo do programa. Outro conceito que você pode usar para controlar o fluxo de programas JavaScript é o **controle de fluxo iterativo**, que permite que você faça um **loop sobre um bloco de código muitas vezes**. Você já viu algumas operações iterativas quando você revisou os métodos avançados no objeto de matriz que usam callbacks. Lá, o controle de fluxo iterativo foi construído nos vários métodos de matriz. Nesta seção, você examinará as instruções de controle iterativas nativas, incluindo os **loops for e while**.

Usando o loop for

O loop for é útil nos casos em que um bloco de código **deve ser executado com base em um número determinístico** de itens. Em alguns casos, você pode querer iterar sobre uma lista de itens em uma matriz ou lista; Em outros casos, você pode querer executar um bloco de código um número específico de vezes para executar algum tipo de animação ou para criar um número específico de objetos.

A sintaxe do loop for é a seguinte:

```
for(<counter>;<expression>;<counter increment>)  
{  
  <code to run>  
}
```

O loop for necessita de três elementos: um contador, uma expressão e um incremento.

■ A variável contador contém o número atual de vezes que o loop foi executado. É necessário inicializar o contador apropriadamente, como para 1 ou 0.

■ O elemento **expression** avalia o contador contra algum outro valor. Seu objetivo é definir um limite para o número de vezes que o loop for executa. Por exemplo, se você quisesse que o loop fosse executado 10 vezes, poderia inicializar o contador para 0 e a expressão seria `counter < 10`. Fazer isso garantiria que o loop seria executado somente enquanto a expressão retornasse true, ou seja, enquanto a variável de contador é menor que 10. Assim que o contador for igual a 10, o processamento de loop será interrompido e o processamento de código continuará após o loop for.

■ Com o incremento do contador, o loop for deve ser informado como ajustar a variável do contador após cada iteração do loop. O incremento pode ser positivo ou negativo dependendo de como o loop está configurado. Você pode definir o incremento para que o loop conta sequencialmente, ou usar operadores matemáticos para incrementar por um valor diferente.

O código ou corpo do loop é um bloco de código rodeado por chaves. Esta seção de código é executada para cada iteração de loop. Os exemplos de código a seguir demonstram várias maneiras de usar um loop for.

Primeiro, aqui está um simples para loop que executa um bloco de código 10 vezes:

```
for (var i = 0; i < 10; i++) {  
    document.write(i);  
}
```

Observe que porque o contador está começando em 0, a expressão deve ser menor que 10. Se o contador for iniciar em 1, a expressão seria `<= 10`. Isso é importante para manter um olho em. O incremento do contador usa a abreviação de adição `++` para aumentar o contador por um em cada iteração. O seguinte código vai na ordem inversa:

```
for (var i = 10; i > 0; i--) {  
    document.write(i);  
}
```

Além de usar os operadores de incremento ou decremento, você pode multiplicar ou dividir o valor do contador. O código a seguir imprime um conjunto de números que aumentam em um fator de 2 até 100:

```
for(var i= 1; i<100;i*=2){  
    document.write(i);  
    document.write("<br />");  
}
```

```
}
```

O pedaço expressão do loop for não precisa ser um valor codificado, como o que foi mostrado até agora. Em vez disso, você pode derivar a expressão a partir do comprimento de um objeto ou outra variável, como neste exemplo:

```
var alphabet = 'abcdefghijklmnopqrstuvwxyz';  
for (var i = 0; i < alphabet.length; i++) {  
    document.write(alphabet[i]);  
    document.write("<br />");  
}
```

Uma vez que uma cadeia é apenas uma matriz de caracteres, este código pode iterar sobre a cadeia de caracteres e imprimir cada carácter para a tela. O comprimento da sequência determina quantas vezes o loop é executado.

Usando o loop for...in

O loop for ... in é um método para iterar sobre as propriedades de um objeto. Tome o exemplo a seguir:

```
var person = { firstName: "Jane", lastName: "Doe", birthDate: "Jan 5,  
1925", gender:  
    "female" };  
for (var prop in person) {  
    document.write(prop);  
}
```

Isso para loop imprime o nome de cada propriedade no objeto pessoa personalizada. Se você deseja que o loop para imprimir os valores de propriedade em vez disso, cada propriedade precisa ser acessada através do indexador de propriedades do objeto, como neste exemplo:

```
var person = { firstName: "Jane", lastName: "Doe", birthDate: "Jan 5,  
1925", gender:  
    "female" };  
for (var prop in person) {  
    document.write(person[prop]);  
}
```

Usando o loop while

O loop while permite que você execute um bloco de código até que alguma condição seja avaliada como falsa. A construção do loop while é a seguinte:

```
while(<expression>){  
  <code block>  
}
```

A expressão é algo que avalia como um booleano. Enquanto a expressão for verdadeira, o loop while continua a ser executado. O código que é executado está contido dentro do bloco de código dentro das chaves. A condição deve ser verdadeira para que o loop while seja executado. Como o loop while não usa um incrementador como o loop for, o código dentro do loop while deve ser capaz de definir a expressão como false, conforme apropriado; Caso contrário, o loop será um loop infinito. Você pode realmente querer usar um loop infinito, mas você deve garantir que o processamento do loop não bloqueia o thread principal do aplicativo. O código a seguir demonstra um loop de tempo:

```
var i = 0;  
while (i < 10) {  
  //do some work here.  
  i++;  
}
```

Neste código, o loop while é executado até que a variável i seja igual a 10. A expressão pode conter qualquer coisa, desde que avalie como um Boolean.

O exemplo anterior é razoavelmente determinista, pois você sabe que o loop será executado 10 vezes. No entanto, em algumas situações, um bloco de código deve ser executado até que algo mais muda que poderia estar fora do controle do loop. Suponha que um aplicativo está movendo o tráfego através de uma interseção. Este tipo de aplicação pode mover o tráfego enquanto o sinal de trânsito estiver verde. O código a seguir demonstra isso:

```
var canvas = document.getElementById("canvas1");  
while (canvas.styles.backgroundColor == 'green') {  
  //move traffic  
}
```

Esse loop while nunca terminará até que o fundo da tela não esteja mais verde. O loop depende da lógica em outro lugar na aplicação para alterar a cor de fundo da tela, como um timer que controla quanto tempo o sinal de trânsito permanece verde ou vermelho. Outra forma do loop while é o loop do ... while.

Usando o loop do...While

A principal diferença entre o loop while e o loop do ... while é que fazer ... enquanto sempre é executado pelo menos na primeira vez. Em contraste, o loop while primeiro avalia a expressão para determinar se ele deve ser executado em tudo e, em seguida, continua a ser executado, desde que a expressão seja avaliada como true. O loop do ... while sempre é executado uma vez porque nesta forma de loop, a lógica de expressão está na parte inferior. O loop do ... while faz isso com a seguinte estrutura:

```
do{  
  <code block>  
}while(<expression>)
```

Esse código processa o bloco de código e, em seguida, enquanto uma expressão é verdadeira, continua a processar esse bloco de código. O segmento de código a seguir demonstra isso:

```
var canvas = document.getElementById("canvas1");  
  
do {  
  //stop traffic  
}while(canvas.styles.backgroundColor == 'red')
```

Neste segmento de código, a lógica para parar o tráfego é executada uma vez. Em seguida, avalia a expressão que verifica se o fundo da tela é vermelho. O ciclo continua a ser executado, desde que esta expressão seja avaliada como verdadeira.

Curto-circuito nos loops

Dois mecanismos permitem que você curto-circuito um loop. A palavra-chave **break** deixa completamente o loop atual, enquanto a palavra-chave **continue** quebra do bloco de código e continua para a próxima iteração do loop.

Dica de exame

A palavra-chave **break** é interrompida apenas no loop em execução no momento. Se o loop contendo a quebra está aninhado dentro de outro loop, o loop externo continua a iterar como controlado por sua própria expressão.

Experimento de pensamento

Identificando sutilezas na sintaxe deste experimento de pensamento, aplique o que aprendeu sobre esse objetivo. Você pode encontrar respostas para essas perguntas na seção "Respostas" no final deste capítulo. Várias construções executam a mesma função, mas com uma sintaxe diferente. Por exemplo, a diferença real entre uma instrução switch e as instruções aninhadas if ... else é mínima. Além disso, os loops while e for ambos avaliam uma condição para saber se o loop deve prosseguir.

1. Quando um loop for melhor que um loop while?
2. Como a legibilidade do código é afetada?

Resumo do objetivo

- ■ O for e for ... in iterate para um comprimento de valores conhecidos.
- ■ Os toques while e do ... while são executados até que uma condição Booleana seja definida como false.
- ■ Arrays fornecem um mecanismo no qual criar listas de coisas.

Revisão objetiva

1. Quais das seguintes palavras-chave fornecem um fluxo de controle iterativo?
 - A. if statement
 - B. switch statement
 - C. for
 - D. break
2. Qual dos seguintes métodos de matriz combina duas matrizes?
 - A. join
 - B. combine
 - C. split
 - D. concat
3. Que sintaxe de controle iterativo pode garantir que o loop seja processado pelo menos uma vez?
 - A. for...in loop
 - B. while loop
 - C. do...while loop
 - D. for loop
4. Qual palavra-chave é usada para sair de um loop?
 - A. continue
 - B. break
 - C. stop
 - D. next

Objetivo 2.2: Levantar e lidar com um evento

O navegador fornece comportamento dinâmico através de eventos. As ações processadas pelo usuário do navegador podem desencadear uma oportunidade para o seu código reagir e criar uma experiência para o usuário. Esta oportunidade é apresentada sob a forma de um evento. Os elementos DOM fornecem nativamente eventos que podem ser manipulados e você pode implementar eventos personalizados em objetos personalizados. Os eventos normalmente seguem uma convenção de nomenclatura. Ao examinar quais eventos estão disponíveis em um determinado objeto, é possível identificar esses eventos como propriedades que começam com o prefixo `on`. Por exemplo, alguns eventos comuns são `onkeypress` ou `onblur`. Para que os eventos funcionem, você precisa "conectá-los" atribuindo um manipulador de eventos. O manipulador de eventos é uma função JavaScript chamada quando uma ação aciona o evento. Os eventos estão disparados o tempo todo no navegador;

No entanto, é ou não um manipulador é atribuído que determina se ou não você pode executar seu próprio código quando o evento é acionado.

Usando eventos

A razão pela qual uma API fornece eventos é para que os desenvolvedores possam injetar seu próprio processamento em meio a toda a ação que ocorre em um programa. O JavaScript permite que você faça exatamente isso em todo o DOM. Esta seção discute a capacidade de conectar-se a esses eventos. A ideia de ligar um evento é dizer ao navegador que quando ocorre um determinado evento, ele deve chamar uma função especificada. A função atribuída a um evento é dito ser um ouvinte de evento que escuta esse evento. A necessidade, então, é atribuir uma função a um evento para ouvir quando esse evento ocorre.

Você pode conectar um evento de três maneiras:

- Declare-o diretamente na marcação HTML.
- Atribua a função à propriedade de evento do objeto através de JavaScript.
- Use os métodos de inclusão e remoção mais recentes no objeto para associar eventos manipuladores

Ao atribuir manipuladores de eventos através de JavaScript, você tem duas opções: fornecer uma função nomeada e atribuir uma função anônima. A diferença entre estes dois será examinada.

Para começar, você precisa entender o conceito de um único objeto comum a todos os manipuladores de eventos DOM, e esse é o objeto de evento em si.

Objetos de evento

Em geral, o objeto de evento é um objeto comum disponível em manipuladores de eventos que fornece metadados sobre o evento. Por exemplo, se eventos de teclado estão a ser tratados, você pode querer saber qual tecla foi pressionada. Se os eventos do mouse estiverem sendo manipulados, você pode querer saber qual botão do mouse foi pressionado. O objeto de evento contém todas essas propriedades.

O objeto de evento é acessado dentro de uma função de manipulador de eventos, usando o objeto de janela como mostrado:

```
var evnt = window.event;
```

O objeto de evento é uma propriedade do objeto window. Neste exemplo, uma referência ao objeto de evento é atribuída a uma variável, mas também pode ser usada diretamente. No contexto do manipulador de eventos atual, o objeto de evento contém as informações pertinentes - ou seja, as respectivas propriedades são definidas. Por exemplo, em um evento keydown, os detalhes do estado do teclado estão disponíveis, mas os botões do mouse não são porque não são relevantes para um evento keydown.

NOTA ACESSANDO O CONTEXTO DO EVENTO

No Internet Explorer, o evento de janela é o método necessário para acessar o objeto de evento. No entanto, em alguns navegadores, o objeto de evento é passado para a função de evento como um parâmetro.

Manipulação de eventos declarativos

Manipular eventos declarativamente na marcação HTML é possível configurando uma linha de manipuladores de eventos dentro dos elementos HTML. Isso efetivamente não é diferente de atribuir um valor a qualquer outra

propriedade ou atributo do elemento HTML. Olhe para o seguinte exemplo HTML:

```
<html>
<head>
<script>
function onloadHandler() {
alert("hello event.");
}
</script>
</head>
<body onload="onloadHandler();">
...
</body>
</html>
```

Nesta marcação HTML, o atributo **onload** do elemento body é atribuído JavaScript para ser executado. O evento **onload é acionado quando o documento em si é totalmente carregado no navegador**. Quando o documento é **carregado**, o evento **onload é acionado**, **que chama a função onloadHandler** e, por sua vez, mostra a caixa de alerta. Todos os eventos que serão analisados através deste objectivo podem ser configurados desta forma directamente na marcação HTML. Em seguida, você verá como configurar eventos programaticamente atribuindo a função para a propriedade de evento em JavaScript.

Manipulação de eventos de atribuição

Atribuir a função de evento à propriedade de evento através do JavaScript é outra maneira de configurar manipuladores de eventos. Este método tem sido utilizado em torno de um longo tempo e ainda é amplamente utilizado. Para o exemplo anterior do evento onload, as seguintes alterações são necessárias para refletir a atribuição de um manipulador de eventos por meio de JavaScript:

```
<html>
<head>
<script>
```

```

window.onload = onloadHandler();
function onloadHandler() {
    alert("hello event.");
}
</script>
</head>
<body >
...
</body

```

Neste código, o elemento HTML para o corpo é limpo e o evento `onload` é atribuído em JavaScript. O objeto de janela não é o mesmo que o elemento de corpo, mas demonstra o conceito de atribuir código que precisa ser executado assim que a página é carregada. Observe que a atribuição do `onloadHandler` está no bloco de script, mas não dentro de qualquer função. Para que isso tenha êxito, o objeto de janela deve existir. Uma vez que o objeto `window` é um objeto global ele existirá. No entanto, para acessar elementos da página, a página deve ser carregada ou o script deve ser executado após o processador processar o HTML. Por exemplo, se a página tiver uma tela e a funcionalidade para permitir que os usuários desenhem com um mouse, os manipuladores de eventos para as atividades do mouse teriam que ser atribuídos na parte inferior da página ou no evento `onload` da janela. O evento `onload` é acionado quando toda a página é carregada, portanto é possível obter uma referência aos elementos da página e conectar os manipuladores de eventos.

Uma maneira mais comum de fazer isso é atribuir uma função anônima à janela

O evento `Onload` liga todos os eventos necessários. O conceito de uma função anônima é discutido em breve. Ele é usado em todo o livro como mostrado aqui:

```

window.onload = function () {
    //do event setup in here.
}

```

Usando os métodos addEventListener e removeEventListener

`addEventListener` e `removeEventListener` são os dois métodos preferidos para conectar uma função a um evento e, em seguida, para removê-lo mais tarde conforme necessário. O método `addEventListener` aceita dois parâmetros necessários e um parâmetro opcional:

```
addEventListener(<event name>,<event function>,<optional cascade rule>)
```

O nome do evento é o nome do evento a ser tratado. O nome do evento será como você viu nos exemplos anteriores, exceto sem o prefixo `on`. Por exemplo, o nome do evento `onload` é apenas carga. A função de evento é aquela que deve ser executada quando o evento ocorre, o ouvinte. A regra de cascata opcional fornece alguma flexibilidade na forma como os eventos se movem através de elementos DOM aninhados. Isso será examinado mais detalhadamente mais adiante na discussão sobre o borbulhamento de eventos. O `removeEventListener` leva exatamente os mesmos parâmetros. O que isso implica é que mais de um evento ouvinte pode ser adicionado para o mesmo evento e, em seguida, removido. Pensando no contexto de um programa complicado, como um jogo, talvez seja necessário ativar e desativar eventos para o mesmo evento. Considere o seguinte exemplo:

```
<script>
window.addEventListener("load", onloadHandler, false);
window.addEventListener("load", onloadHandler2, false);
window.addEventListener("load", onloadHandler3, false);
function onloadHandler() {
    alert("hello event 1.");
}
function onloadHandler2() {
    alert("hello event 2.");
}
function onloadHandler3() {
    alert("hello event 3.");
}
```


</script>

Cada evento é acionado na ordem em que ele foi adicionado quando o carregamento da janela é terminado. Para remover o evento `onloadHandler2`, tudo o que é necessário é uma chamada para o `removeEventListener`:

```
window.removeEventListener("load", onloadHandler2, false);
```

Ao manipular eventos DOM, os eventos personalizados criados não são uma substituição para a funcionalidade interna fornecida pelo elemento DOM. O tratamento do evento permite que você faça alguma lógica ou manipulação personalizada, mas quando o tratamento de eventos for concluído, o processamento retorna à API JavaScript, que processa sua própria implementação para o evento. Se isso não for desejável, você pode interromper o processamento do evento.

Usando funções anônimas

Nos exemplos até agora, os manipuladores de eventos foram atribuídos por meio de funções nomeadas. A vantagem de usar funções nomeadas é que você pode remover mais tarde os listeners de eventos conforme necessário. Não é possível identificar funções anônimas depois que elas são atribuídas como ouvintes de eventos para manipulá-las. No exemplo na seção anterior, três ouvintes de evento foram adicionados ao mesmo evento e, em seguida, um evento foi removido. Isso era possível somente porque o nome da função de ouvinte de evento era conhecido.

Como esperado, uma função anônima não tem nome. É completamente anônimo e não pode ser chamado de outros segmentos de código. Observe o seguinte exemplo:

```
window.onload = function () {  
  }  
}
```

Este exemplo é usado em todo o livro para garantir que a página está totalmente carregada antes de acessar elementos no DOM; Caso contrário, os elementos não estariam disponíveis. O evento `onload` para o objeto `window` está sendo atribuído a uma função anônima. Esta função não tem um nome e não

pode ser chamado por qualquer outro código. A implementação interna do objeto de janela executa esta função ao aumentar o evento onload. No JavaScript, as funções são objetos que podem ser atribuídos a variáveis. É assim que funciona o ouvinte de eventos de função anônima. **Atribui um objeto de função à propriedade onload do objeto window, que por sua vez manipula o evento quando a janela está completamente carregada.** Você pode usar funções anônimas na maioria dos casos em que uma função é esperada como um parâmetro também. Leve o seguinte exemplo de código:

```
window.addEventListener("load",  
function () {  
    document.getElementById("outer").addEventListener("click",  
outerDivClick, false);},  
false);
```

Neste exemplo, o método **addEventListener** é usado. No entanto, em vez de passar o nome da função para chamar quando o evento é acionado, uma função anônima é passada. **O único problema potencial com essa abordagem é a capacidade de posteriormente remover o ouvinte de evento com o método removeEventListener.** Que o código a seguir funcionaria pode parecer lógico:

```
window.removeEventListener("load",  
function () {  
    document.getElementById("outer").addEventListener("click",  
outerDivClick, false); },  
false);
```

Mas este não é o caso. **Como os ouvintes de eventos que o addEventListener adiciona são armazenados por suas assinaturas, esse método removeEventHandler não pode conhecer a assinatura da função anônima anterior.** Mesmo passar a mesma implementação anônima não funciona porque esta não é a mesma função anônima; É um novo e, portanto, não corresponde à assinatura do adicionado.

Cancelar um evento

A capacidade de **cancelar o processamento de eventos pode ser útil quando você deseja substituir completamente a implementação da**

funcionalidade nativa de um elemento DOM. Um exemplo perfeito é se foi necessário para substituir a funcionalidade inerente de um elemento de âncora. Um ouvinte de evento seria configurado para o evento de clique. Em seguida, no evento click, através do objeto de evento, o ReturnValue propriedade é definida como false ou a própria função pode retornar false. Isso informa o tempo de execução para interromper qualquer processamento posterior do evento. O código a seguir demonstra isso:

```
window.onload = function () {  
  var a = document.getElementById("aLink");  
  a.onclick = OverrideAnchorClick;  
}  
function OverrideAnchorClick() {  
  //do custom logic for the anchor  
  window.event.returnValue = false;  
  //or  
  //return false;  
}
```

Neste caso, quando a âncora é clicada, o manipulador de eventos personalizado é executado mas nenhuma lógica adicional é processada. Portanto, a navegação normalmente fornecida pelo elemento <a> é impedida de execução. Outro aspecto a ser considerado é a ordem em que os eventos são executados quando você está trabalhando com um elemento DOM aninhado. Neste caso, o conceito que é tratado é borbulhar evento.

Declaração e tratamento de eventos borbulhados

O bubbling de eventos é o conceito que se aplica quando o documento HTML tem elementos aninhados.

Considere o seguinte exemplo HTML:

```
<style>  
#outer {  
  width: 200px;  
  height: 200px;  
  background-color: red;
```

```

}
#middle {
width: 50%;
height: 50%;
position: relative;
top: 25%;
left: 25%;
background-color: green;
}
#inner {
width: 50%;
height: 50%;
position: relative;
top: 25%;
left: 25%;
background-color: blue;
}
</style>
<script>
window.onload = function () {
    document.getElementById("outer").addEventListener("click",
outerDivClick, false);
    document.getElementById("middle").addEventListener("click",
middleDivClick, false);
    document.getElementById("inner").addEventListener("click",
innerDivClick, false);
    document.getElementById("clearButton").addEventListener("click",
clearList);
}
function outerDivClick() {
    appendText("outer Div Clicked");
}
function middleDivClick() {
    appendText("middle Div Clicked");
}

```

```

}
function innerDivClick() {
  appendText("inner Div Clicked");
}
function appendText(s) {
  var li = document.createElement("li");
  li.innerText = s;
  document.getElementById("eventOrder").appendChild(li);
}
function clearList() {
  var ol = document.createElement("ol");
  ol.id = "eventOrder";
  document.getElementById("bod").replaceChild(ol,document.
getElementById("eventOrder"));
}
</script>
<body id="bod">
<div id="outer">
<div id="middle" >
<div id="inner">
</div>
</div>
</div>
<ol id="eventOrder"> </ol>
<button type="button" id="clearButton">Clear</button>
</body>

```

Quando esse HTML é renderizado no navegador, o resultado é três elementos div, como mostrado na Figura 2-2. Neste código são três div elementos empilhados em cima uns dos outros. O estilo é aplicado para fornecer uma distinção visual entre as caixas. Quando uma caixa div é clicada, o evento clique dispara. O código de ouvinte de evento no manipulador atribuído emite o nome do div clicado em uma lista ordenada para que a ordem na qual os eventos são clicados é identificada.

FIGURA 2-2 Três elementos <div> aninhados para exibir o efeito de borbulhar eventos

O último parâmetro do método `addEventListener` aceita um parâmetro booleano opcional. Este parâmetro permite que você especifique o efeito cascata ou borbulhante do evento, ou seja, em que ordem ocorre o processamento do evento. O evento de clique para cada div tem um ouvinte de evento atribuído. No exemplo anterior, os três elementos div são aninhados. Um usuário que clica no interior ou no meio div também clica no div pai porque os elementos div compartilham o mesmo espaço físico na tela. Quando a caixa azul dentro é clicada, a seguinte saída é exibida:

1. inner Div Clicked
2. middle Div Clicked
3. outer Div Clicked

Um evento de clique acionou todos os três eventos para acionar. Este conceito é chamado borbulhar eventos. Clicando diretamente no div verde médio produz a seguinte saída:

1. middle Div Clicked
2. outer Div Clicked

Finalmente, ao clicar na div externa vermelha, você produz essa saída:

1. outer Div Clicked

O evento tem borbulhado até o topo. Se você preferir que os eventos sejam Ordem inversa - ou seja, para os ter em cascata para baixo - o último parâmetro especificado pelo método `addEventListener` é especificado como verdadeiro. Com esta alteração feita, como se segue,

```
document.getElementById("outer").addEventListener("click",    outerDivClick,
true);
document.getElementById("middle").addEventListener("click",  middleDivClick,
true);
document.getElementById("inner").addEventListener("click",    innerDivClick,
true);
```

A saída da tela agora é a seguinte:

1. outer Div Clicked
2. middle Div Clicked
3. inner Div Clicked

A ordem do processamento de eventos foi invertida para ser cascata em vez de borbulhar.

O efeito de cascata ou borbulhante dos eventos é conveniente quando você quer. No entanto, o design da página da Web pode envolver elementos aninhados, mas **o evento de clique de cada elemento deve ser executado somente se o elemento for clicado diretamente**. Nesse caso, você pode usar uma propriedade do objeto de evento chamado **cancelBubble**. Se esta propriedade estiver definida como true, o evento borbulhando ou em cascata pára com o ouvinte de eventos que o define. Isso só para o comportamento borbulhando ou em cascata. O código para cancelar o borbulhamento do evento é adicionado ao ouvinte de evento do elemento div interno:

```
function innerDivClick() {  
  appendText("inner Div Clicked");  
  window.event.cancelBubble = true;  
}
```

Agora, quando o div interno é clicado, a saída é a seguinte:

1. inner Div Clicked

O bubbling do evento até o div médio e div externo foi impedido.

Gerenciando eventos DOM

O DOM fornece um grande número de eventos internos. Os eventos mais comuns usados em uma base mais diária são abordados nesta seção. O DOM fornece esses eventos por meio da API JavaScript. As funções podem ser especificadas como ouvintes de evento e o comportamento personalizado pode ser implementado em páginas da web com base no evento ocorrido. Esses eventos se aplicam à maioria dos elementos DOM.

Alterar eventos

Um evento de alteração ocorre quando o valor associado a um elemento é alterado. Isso ocorre mais comumente em elementos de entrada, como entradas baseadas em texto e outros, como o elemento de intervalo. Um exemplo do evento de alteração em ação é mostrado aqui:

```
<script>
window.onload = function () {
    document.getElementById("aRange").addEventListener("change",
rangeChangeEvent);
}
function rangeChangeEvent() {
    document.getElementById("rangeValue").innerText = this.value;
}
</script>
...
<body>
<input id="aRange" type="range" max="200" min="0" value="0"/>
<div id="rangeValue"></div>
</body>
```

Neste exemplo, como o controle deslizante de intervalo muda com o mouse arrastando-lo de um lado para o outro, o div exibe o valor da barra deslizante.

Dica de exame

Este exemplo usa a palavra-chave `this`. Nesse contexto, a palavra-chave `this` fornece uma referência direta ao elemento que criou o evento. Desta forma, isto fornece acesso directo ao elemento em vez de obter uma referência através de um dos métodos de procura de documentos.

Com o controle de entrada de texto, o mesmo tipo de código pode ser processado:

```
...
document.getElementById("aText").addEventListener("change",
rangeChangeEvent);
...
<body>
```



```
<input id="aRange" type="range" max="200" min="0" value="0"/>
<input id="aText" type="text"/>
<div id="rangeValue"></div>
</body>
```

Agora, quando o valor de texto da caixa de texto muda, o div mostra o valor. O evento de alteração de caixa de texto é gerado quando o cursor sai da caixa de texto, não como cada caractere é digitado.

Eventos de foco

Eventos de foco ocorrem quando um elemento recebe ou perde o foco. A Tabela 2-2 lista os eventos disponíveis relacionados ao foco.

TABELA 2-2 Os eventos de foco DOM

Evento	Descrição
Focus	Criado quando o elemento recebe o foco
Blur	Criado quando o elemento perde o foco
Focusin	Criado antes de um elemento receber o foco
<u>focusout</u>	Criado antes de um elemento perder o foco

O número de eventos de foco fornece uma flexibilidade muito boa em como o foco de qualquer elemento DOM particular é tratado com respeito ao tempo. O evento de desfocagem é comumente usado para validar campos de formulário. Você pode usar o método **focus ()** para definir o foco para qualquer elemento que faz com que a hierarquia de eventos de foco ocorra. O código a seguir mostra como usar o evento de desfocagem:

```

<script>
window.onload = function () {
document.getElementById("firstNameText").focus();
document.getElementById("firstNameText").addEventListener("blur",
function () {
if (this.value.length < 5) {
document.getElementById("ruleViolation").innerText =
'First Name is required to be 5 letters.';
document.getElementById("ruleViolation").style.color = 'red';
this.focus();
}
});
}
</script>

```

Eventos de teclado

Eventos de teclado ocorrem quando as teclas são pressionadas no teclado. Os eventos de teclado na tabela 2-3 está disponível para ser capturada.

TABELA 2-3 Eventos de teclado disponíveis

Evento	Descrição
Keydown	Levantado quando uma chave é empurrada para baixo
Keyup	Criado quando uma chave é liberada
KeyPress	Criado quando uma tecla está completamente pressionada

O exemplo a seguir escuta o evento keydown na caixa de texto e mostra a tecla pressionada:

```

document.getElementById("firstNameText").addEventListener("keydown",
function () {
document.getElementById("outputText").innerText = window.event.keyCode;
});

```

Código como esse pode ser usado para filtrar caracteres inválidos de serem inseridos em uma caixa de texto. Com eventos de teclado, propriedades extras estão disponíveis no objeto de evento para ajudar. Por exemplo, talvez seja necessário saber se a tecla Shift ou a tecla Control também foram pressionadas. A Tabela 2-4 lista as propriedades do objeto de evento para eventos de teclado.

TABELA 2-4 Propriedades do objeto de evento para eventos de teclado

Propriedade	Descrição
Altkey	Um valor booleano para indicar se a tecla Alt foi pressionada
KeyCode	O código numérico da tecla que foi pressionada
Ctrlkey	Um valor booleano se a tecla Control foi pressionada
shiftkey	Um valor booleano se a tecla Shift foi pressionada

Dica de exame

Em alguns casos, dependendo da chave, somente o evento `keydown` é acionado. As teclas de seta são um exemplo: `keydown` mas não `keyup` ou `KeyPress`.

Você pode usar propriedades como `ctrlKey` com o evento `keyCode` para dar aos usuários algo semelhante à funcionalidade `hotkey` para navegar automaticamente o foco para campos específicos:

```
document.onkeydown = function () {  
    if (window.event.ctrlKey && String.fromCharCode(window.event.keyCode) ==  
        'F')  
        document.getElementById("firstNameText").focus();  
    if (window.event.ctrlKey && String.fromCharCode(window.event.keyCode) ==  
        'L')  
        document.getElementById("lastNameText").focus();  
    return false;  
}
```

Eventos de mouse

O DOM fornece exposição extensiva à atividade do mouse através dos eventos do mouse. A Tabela 2-5 descreve os eventos de mouse disponíveis.

Evento	Descrição
click	Criado quando o mouse executa um clique
dblClick	Criado quando o mouse executa um duplo clique
Mousedown	Criado quando o botão do mouse é pressionado
Mouseup	Criado quando o botão do mouse é liberado
Mouseenter ou mouseover	Criado quando o cursor do mouse entra no espaço de um elemento HTML
Mouseleave	Criado quando o cursor do mouse deixa o espaço de um elemento HTML
mousemove	Criado quando o cursor do mouse se move sobre um elemento HTML

Os eventos de mouse fornecem informações adicionais sobre o objeto de evento.

A Tabela 2-6 lista as propriedades aplicáveis do objeto de evento.

Propriedade	Descrição
clientX	A posição x ou horizontal do cursor do mouse em relação aos limites de viewport
clientY	A posição y ou vertical do cursor do mouse em relação aos limites da viewport

offsetX	A posição x ou horizontal do cursor do mouse em relação ao elemento de destino
offsetY	A posição y ou vertical do cursor do mouse em relação ao elemento alvo
screenX	A posição x ou horizontal do cursor do mouse em relação ao canto superior esquerdo da tela
screenY	A posição y ou vertical do cursor do mouse em relação ao canto superior esquerdo da tela

O código a seguir demonstra a captura de cada conjunto de coordenadas:

```

window.onload = function () {
    document.getElementById("yellowBox").addEventListener("click",
yellowBoxClick);
}
function yellowBoxClick() {
    document.write("Client X: " + window.event.clientX + " ClientY: "
+ window.event.clientY);
    document.write("<BR />");
    document.write("offsetX: " + window.event.offsetX + " offsetY: "
+ window.event.offsetY);
    document.write("<BR />");
    document.write("screen X: " + window.event.screenX + " screenY: "
+ window.event.screenY);
}

```

Este código assume um div chamado yellowBox que aumenta seu evento de clique quando o mouse clica nele. Você pode facilmente mudar o evento para mousedown ou mouseup para alcançar o mesmo resultado. Os eventos mouseenter e mouseleave indicam quando a posição do cursor do mouse entrou ou saiu da área coberta por um elemento particular, respectivamente. O código a seguir demonstra aplicar uma transformação para o elemento div no mouseenter e removê-lo no mouseleave:

```

<style>
.scale {
transform:scale(1.5);
}
</style>
<script>
window.onload = function () {
document.getElementById("yellowBox").addEventListener("mouseenter",
yellowBoxEnter);
document.getElementById("yellowBox").addEventListener("mouseleave",
yellowBoxLeave);
}
function yellowBoxEnter() {
this.classList.add("scale");
}
function yellowBoxLeave() {
this.classList.remove("scale");
}
</script>
<body>
<div id="yellowBox" style="width: 50%;height:50%;margin: 0 auto;
background-color:yellow;"></div>
</body>

```

Quando o mouse se move sobre o div amarelo-cheio, o div escalas acima. Quando o mouse é movido para fora do div, ele retorna para o tamanho original.

Funcionalidade arrastar e largar DRAG-AND-DROP

A funcionalidade arrastar e soltar permite que os usuários peguem um elemento com o mouse e o coloquem em outro local. A Tabela 2-7 lista os eventos relacionados à funcionalidade de arrastar e soltar.

TABELA 2-7 Eventos disponíveis para arrastar e soltar

Evento	Descrição
--------	-----------

Drag	Criado continuamente enquanto o elemento está sendo arrastado
Dragend	Levantado sobre o elemento sendo arrastado quando o mouse é liberado para terminar a queda da operação
Dragenter	Criado em um elemento de destino quando um elemento arrastado é arrastado para seu espaço
Dragleave	Criado em um elemento de destino quando um elemento arrastado deixa seu espaço
Dragover	Criado continuamente no elemento alvo enquanto o elemento arrastado está sendo Arrastado por ela
Dragstart	Criado no elemento sendo arrastado quando a operação de arrastar está começando
drop	Criado no elemento de destino quando o elemento arrastado é liberado

dragstart. O evento de arrastar continua a disparar enquanto o elemento está sendo arrastado. À medida que o elemento é arrastado sobre outros elementos, cada um dos eventos de outros elementos, draguer, dragover e dragleave dispara. Quando o elemento termina sendo arrastado, seu evento de arrastar é acionado eo evento de queda de um elemento de destino é acionado. Você pode usar todos esses eventos em combinação para fornecer feedback visual aos usuários que a operação de arrastar está ocorrendo eo que pode ser um local de descarte potencialmente válido.

O HTML a seguir demonstra essa funcionalidade:

```
<head>
```

```
<style>
```

```
.dropped {
width: 50%;
height: 50%;
position: relative;
top: 25%;
left: 25%;
background-color:black;
}

.over {
transform: scale(1.1);
}

.bucket {
width: 100px;
height: 100px;
margin: 10px 10px 10px 10px;
position:absolute;
}

.chip {
width:20px;
height:20px;
position:absolute;
}

div:first-of-type {
background-color: red;
}

div:nth-of-type(2) {
background-color: green;
left:25%;
}

div:nth-of-type(3) {
background-color: blue;
left:50%;
}

#chip {
```



```

background-color: black;
width:50px;
height:50px;
}
.begin {
position:absolute;
left: 150px;
top: 150px;
}
</style>
</head>
<body>
<div id="bucket1" class="bucket"></div>
<div id="bucket2" class="bucket"></div>
<div id="bucket3" class="bucket"></div>
<div id="chip" draggable="true" class="chip"></div>
</body>

```

O conceito é que três baldes são definidos usando elementos div, e um chip é definido. O usuário pode arrastar o chip para qualquer um dos três baldes. Para que o chip possa ser arrastado, ele deve ser arrastável.

Para iniciar o evento de arrastar, o dragstart deve ser tratado:

```

var chip = document.getElementById("chip");
chip.addEventListener("dragstart", function ()
{ window.event.dataTransfer.setData("Text", this.id); });

```

Neste manipulador, o objeto dataTransfer método setData é usado para armazenar o que exatamente está sendo transferido. Nesse caso, o ID do objeto de origem é especificado.

Em seguida, os listeners de evento do elemento desejado devem ser configurados. O código a seguir mostra isso:

```

var b1 = document.getElementById("bucket1");
b1.addEventListener("dragenter", function () {
b1.classList.add("over");
window.event.returnValue = false;
});

```

```
b1.addEventListener("dragleave", function () {  
  b1.classList.remove("over");  
});  
b1.addEventListener("dragover", function () {  
  window.event.returnValue = false;  
});  
b1.addEventListener("drop", function () {  
  window.event.returnValue = false;  
  var data = event.dataTransfer.getData("Text");  
  var d = document.getElementById(data);  
  d.classList.remove("begin");  
  d.classList.add("dropped");  
  this.appendChild(d);  
});
```

Neste código, o ouvinte de evento `dragenter` é estabelecido para que o usuário obtenha uma sugestão visual com uma transformação que o elemento pode ser descartado. No mesmo token, o ouvinte de evento de arrastar é configurado para remover o efeito. O evento `dragover` é definido para ser ignorado cancelando-o. Isso ocorre apenas porque os elementos `div` não podem ser arrastados e descartados por padrão. A última peça é o manipulador de eventos `drop`. Com este manipulador de eventos, a queda é recebida. O método `getData` do objeto `dataTransfer` é chamado para recuperar o que está sendo descartado. O ID do elemento de origem obtém uma referência ao elemento e coloca-o dentro do alvo. O mesmo código pode ser repetido para os outros dois baldes, e então o chip pode ser arrastado para cada balde.

NOTA SUPORTE DE DRAG-AND-DROP

Para os elementos que não suportam a funcionalidade arrastar e largar por predefinição, o mecanismo de evento predefinido tem de ser cancelado. É por isso que `event.returnValue` é definido como `false`.

Criando eventos personalizados

Os eventos DOM oferecem uma grande variedade de funcionalidades. Em alguns casos, convém criar um evento personalizado para usar de forma mais genérica. Para criar um personalizado evento, você usar o objeto CustomEvent.

Para usar eventos personalizados, primeiro você precisa criar um usando o window.CustomEvent

```
myEvent = new CustomEvent(  
  "anAction",  
  {  
    detail: { description: "a description of the event",  
      timeofevent: new Date(),  
      eventcode: 2 },  
    bubbles: true,  
    cancelable: true  
  }  
);
```

O construtor de objeto CustomEvent aceita dois parâmetros:

- **O primeiro parâmetro é o nome do evento.** Isso é tudo o que faz sentido para o que o evento deve representar. Neste exemplo, o evento é chamado deAction.

- **O segundo parâmetro é um objeto dinâmico que contém uma propriedade de detalhe que pode ter propriedades atribuídas a ele contendo informações que devem ser passadas para o manipulador de eventos.** Além disso, o parâmetro fornece a capacidade de especificar se o evento deve bolha e se o evento pode ser cancelado.

O próximo passo é atribuir o evento a um elemento na página usando o Método addEventListener:

```
document.addEventListener("anAction", customEventHandler);
```

Finalmente, o evento é gerado usando o método dispatchEvent:

```
document.dispatchEvent(myEvent);
```

Uma função chamada customEventHandler deve existir para que tudo isso funcione:

```
function customEventHandler() {  
  alert(window.event.detail.description);  
}
```

Dica de exame

Até à data desta escrita, Internet Explorer não suporta esta funcionalidade. No entanto, eventos personalizados funcionam corretamente em outros navegadores. Esteja ciente de como os eventos personalizados funcionam para o exame, no entanto, porque eles fazem parte das habilidades oficiais que estão sendo medidos.

Experimento de pensamento

Criação de uma página da Web completa para o evento

Neste experimento mental, aplique o que aprendeu sobre esse objetivo.

Você pode

Encontre respostas para essas perguntas na seção "Respostas" no final deste capítulo.

Considere um aplicativo em que os campos se tornam automaticamente preenchidos com base na

Outros campos. A auto-população é vista algumas vezes em formulários que exigem que os usuários preencham

Seu endereço. Quando o código postal é inserido, a cidade, país / região, e assim por diante

São preenchidos com base nessa informação. Como você aplicaria o evento apropriado

Para um formulário que contém caixas de texto, caixas de seleção, botões de opção, barras deslizantes e

On, de modo que como o formulário é preenchido, outros campos são preenchidos automaticamente? Possivelmente

Quando um campo é preenchido automaticamente, seu valor dispara outros campos a serem preenchidos.

Como você pode implementar a solução, considerando o momento em que os eventos são

Disparado

Resumo do objetivo

■ Os eventos fornecem uma forma de interagir com os usuários quando executam ações página da web.

- ■ Eventos em cascata ou bolhas em toda a hierarquia de DOM.
- ■ Os eventos de foco ocorrem quando um objeto obtém ou perde o foco.
- ■ Os eventos de teclado ocorrem quando as teclas do teclado são pressionadas em um objeto focado.
- ■ Eventos de mouse ocorrem quando o mouse clica em um objeto ou o ponteiro é movido sobre ou fora de um objeto.
- ■ A funcionalidade arrastar e soltar fornece uma maneira de mover elementos de um contêiner para outro.

Revisão objetiva

1. Qual das opções a seguir não é uma maneira suportada para adicionar um manipulador de eventos a um DOM elemento?

A. Declarando dentro do elemento HTML, atribuindo o atributo event a uma função JavaScript.

B. Configurando o atributo em CSS para uma função JavaScript válida

C. Dinamicamente através do JavaScript atribuindo uma função JavaScript ao objeto propriedade de evento

D. Dinamicamente através do JavaScript através dos métodos de atribuição / remoção de ouvintes de eventos

2. Qual dos seguintes não é um atributo de uma função anônima?

A. Funções anônimas não podem ser chamadas por qualquer outro código.

B. Funções anônimas não têm um nome claramente definido.

C. As funções anônimas podem ser passadas como parâmetros.

D. Funções anônimas não podem ser atribuídas a um elemento DOM declarativamente.

3. Que linha de código cancelaria com êxito um evento?

A. `window.event.returnValue = false;`

B. `return false;`

C. `window.event.Return ();`

D. window.Stop ();

4. Que evento ocorre quando um elemento DOM recebe o cursor?

A. focus

B. change

C. keydown

D. mouseleave

5. Qual opção fornece a sequência correta de eventos em uma operação de arrastar e soltar?

A. dragstart, drag, dragenter, drop

B. dragstart, drag, dragenter, dragstop

C. drag, dragstart, drop, dragenter

D. drag, dragstart, dragenter, dragstop

Objetivo 2.3: Implementar o tratamento de exceções

Que um programa pode lidar com erros e condições desconhecidas é fundamental em qualquer desenvolvimento de software. O JavaScript não é uma exceção e fornece construções estruturadas de manipulação de erros para lidar com essas situações. Manipulação de erro estruturado em JavaScript é **conseguido com o try ... catch ... finally** construtor. Uma boa programação defensiva também inclui a **verificação de valores nulos**, quando apropriado, para evitar erros. Além de manipular erros, o código pode aumentar erros personalizados conforme necessário para enviar informações de erro para um programa em execução a partir de objetos personalizados ou bibliotecas.

Este objetivo abrange o uso do try ... catch ... finally construtor, avaliando para a condição nulo e aumento de erros personalizados.

Este objetivo abrange como:

- Implementar os blocos try-catch-finally, incluindo a definição e resposta ao erro de códigos e lançando exceções

- Verificar valores nulos

Implementando try ... catch ... finally constructs

O try ... catch ... finally constrói lida com exceções que ocorrem durante o processamento do programa. Ele permite que você veja que tipo de erro ocorreu e fazer o que é apropriado com ele. Se try ... catch ... finally não é implementado no programa, os erros seriam tratados como não tratados e poderiam causar falha no navegador ou, no mínimo, exibir muitas caixas de mensagem irritante para os usuários, como este mostrado na Figura 2-3 que é causado por este código:

```
window.dosomeunsupportedmethod();
```

FIGURA 2-3 Caixa de diálogo de erro de exceção não tratada

Com erros como este, os usuários provavelmente pararão de chegar ao site ou usar o aplicativo.

Para evitar tais problemas, você precisa envolver o código em um bloco try:

```
try{  
    window.dosomeunsupportedmethod();  
} catch (e) {  
    alert("Browser does not support the desired functionality.");  
}
```

Usando o bloco try ... catch, você pode lidar com a condição de erro graciosamente. Os usuários vêem um alerta padrão ea página da Web continua a ser executada como de costume. Este código de exemplo resulta na mensagem mostrada na Figura 2-4.

FIGURA 2-4 Uma caixa de mensagem limpa para mostrar erros

O bloco try ... catch é dividido em duas partes. A primeira parte, a parte try, diz: "Tente fazer este trabalho." Se algo der errado ao tentar fazer o trabalho, o bloco catch recebe um objeto de exceção com informações sobre o erro. Qualquer código dentro da parte try do bloco try ... catch está protegido contra o encontro de um erro não tratado.

O bloco catch é onde o erro pode ser tratado conforme apropriado para o aplicativo. O bloco catch recebe um parâmetro que é um objeto de exceção. A Tabela 2-8 mostra as propriedades para o objeto de exceção.

TABELA 2-8 Propriedades disponíveis no objeto de exceção

Propriedade	Descrição
Message	Uma descrição textual do erro que ocorreu
Number	Um código de erro numérico
name	O nome do objeto de exceção

Você pode usar as informações fornecidas no objeto de exceção para decidir o que fazer em termos do fluxo geral do programa. Por exemplo, se o programa precisar de acesso a um recurso que ele não pode ter e uma exceção é lançada, o programa pode voltar a um processo diferente para alcançar a funcionalidade desejada ou simplesmente dizer ao usuário que algo precisa ser alterado. Exemplo, se os cookies ou outra API HTML5 forem necessários para que o site funcione. Outras maneiras de verificar esse tipo de coisa são demonstradas em breve.

Outra parte do bloco try ... catch é o bloco finally. Este bloco é adicionado diretamente após o bloco catch. O significado do bloco finally é que o código dentro dele é executado o tempo todo. Isso não quer dizer que o código no bloco finally não pode ter seus próprios erros resultando em exceções, mas se o código no bloco try tem ou não um erro, o código no bloco finally ainda é executado. Considere o seguinte código:

```
function WorkthroughArray() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.fillStyle = "blue";
    context.arc(50, 50, 25, 0, 360);
    context.fill();
    context.strokeStyle = "red";
    context.stroke();
}
```

Esta função contém um erro ortográfico intencional, contígo, que resulta em uma exceção. Nada depois da linha que faz gera a exceção será executada.

No entanto, colocando um try ... catch ... finally bloco em torno deste código fornece mais controle sobre o fluxo:

```
try{
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.fillStyle = "blue";
  context.arc(50, 50, 25, 0, 360);
  context.fill();
  context.strokeStyle = "red";
  context.stroke();
}
catch (e) {
  console.log(e.message);
}
finally {
  //do any final logic before exiting the method
}
```

Agora, com a manipulação de erro estruturada no lugar, quando a linha com o erro de digitação é atingido, o processamento **salta para o bloco catch**. Neste bloco, o erro poderia ser registrado para diagnósticos futuros. Após a conclusão do **bloco catch**, o **bloco finally** é executado. Se qualquer limpeza ou reinicialização variável precisa ser feito, pode ser feito aqui mesmo que uma exceção ocorre. O bloco finally também é executado. Se o erro de digitação é corrigido para que nenhuma exceção ocorra no bloco try, o catch não ocorre por causa de nada para capturar, mas o bloco finally ainda é executado. **O bloco finally sempre é executado como a última parte de um try ... catch ... finally block**. O escopo de variável se aplica a cada bloco dentro do bloco try ... catch. Se uma variável for declarada dentro da porção try, ela não será acessível a partir da captura do final. Se você quiser ter acesso nesses blocos, as variáveis precisam ser declaradas fora do bloco try. Veja este exemplo:

```
var canvas;
var context;
try {
  document.getElementById("myCanvas");
```

```

context = canvas.getContext("2d");
contxt.arc(50, 50, 25, 0, 360);
context.fillStyle = "blue";
context.fill();
context.strokeStyle = "red";
context.stroke();
}
catch (e) {
context.strokeText(e.message, 50, 50);
console.log(e.message);
}
finally {
//do any final logic before exiting the method
}

```

A declaração para a referência à tela e ao contexto da tela é movida para fora do bloco try para que ela possa ser acessada no bloco catch. O bloco catch agora pode escrever o erro para a tela.

NOTA UTILIZANDO FERRAMENTAS DE DEBUGGING

Uma chamada para console.log foi adicionada ao bloco catch. Esta é uma ótima maneira de adicionar informações que podem ser exibidas no depurador do cliente. Por exemplo, no Internet Explorer, você pode acessar as ferramentas do depurador pressionando F12.

Exceções borbulham até a pilha de chamadas, uma pilha especial no ambiente de processamento que representa as funções atualmente sendo processadas em ordem seqüencial. Leve o seguinte exemplo de código:

```

window.onload = function () {
try {
WorkWithCanvas();
} catch (e) {
console.log(e.message);
}
}
function WorkWithCanvas() {

```

```

var canvas = document.getElementById("myCanvas");
var context = canvas.getContext("2d");
context.arc(50, 50, 25, 0, 360);
context.fillStyle = "blue";
context.fill();
context.strokeStyle = "red";
context.stroke();
}

```

Como o método `WorkWithCanvas` não tem tratamento de exceção, a exceção borbulha até o método chamador, o próximo método na pilha. Isso continua através da pilha até que um manipulador de exceção seja atendida ou o navegador recebe a exceção e trata-a como uma exceção não tratada. É claro que, neste caso, as variáveis no método `WorkWithCanvas` não podem ser acessadas, então, se alguma coisa precisava ser feita em um bloco `finally`, o bloco `try ... catch ... finally` deve ser movido para o método `WorkWithCanvas` ou o `WorkWithCanvas` Método pode lidar com o erro e `throw-lo` para processamento adicional. O conceito de levantar um erro também é conhecido como lançando uma exceção. Objetos personalizados e bibliotecas lançam exceções conforme necessário para os consumidores das bibliotecas. Os objetos ou bibliotecas esperam que você atenda a determinadas condições e se essas condições não forem cumpridas, eles podem lançar uma exceção para o consumidor para lidar com eles. Para continuar com o exemplo, a exceção é tratada no método `WorkWithCanvas` e, em seguida, `thrown`. Uma exceção é lançada usando a palavra-chave `throw`:

```

function WorkWithCanvas() {
  try {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.arc(50, 50, 25, 0, 360);
    context.fillStyle = "blue";
    context.fill();
    context.strokeStyle = "red";
    context.stroke();
  } catch (e) {

```

```
//handle the exception as appropriate
throw e;
} finally {
}
}
```

Neste exemplo, a exceção pode ser tratada no bloco catch conforme necessário e, em seguida, jogado de volta a pilha de chamadas para ser tratado novamente em outro nível.

Mais comumente ao trabalhar com bibliotecas personalizadas, você pode criar exceções personalizadas para fornecer aos usuários informações específicas sobre a situação ocorrida:

```
var ball = {
  x: -1,
  y: -1,
  draw: function DrawBall(c) {
    if (this.x < 0)
      throw new Error(25, "Invalid X coordinate");
  }
}

window.onload = function () {
  try {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    ball.draw(context);
  } catch (e) {
    alert(e.message);
  }
}
```

Neste código, um objeto personalizado para representar uma bola é criado. Tem um método de espera que um contexto de tela se desenhe. No entanto, se as coordenadas para a bola não são inicializadas, o objeto bola lança um erro personalizado. O código de chamada tem um bloco try ... catch para que ele possa lidar com quaisquer erros inesperados. Neste exemplo, o consumidor do objeto bola receberia uma mensagem significativa de que a coordenada x

precisa ser definida como algo válido. Um novo objeto, `Error`, é usado aqui para criar a exceção. O construtor de objeto `Error` leva dois parâmetros, nesta ordem: o número de erro seguido por uma descrição de erro. Estas informações devem ser tão específicas quanto possível para fornecer o máximo de detalhes possível para o código de chamada.

Verificação de valores nulos

Uma maneira de evitar muitos erros é verificar se há valores nulos antes de usar algo. Um valor nulo em um programa JavaScript é o que uma variável é igual a antes de ser inicializada. O JavaScript sabe sobre a existência da variável, mas ainda não tem um valor.

Um lugar comum para garantir que as variáveis têm valores está em funções que aceitam parâmetros. Considere a seguinte função:

```
window.onload = function () {  
  try {  
    var a, b, c;  
    a = 5;  
    b = 10;  
    var result = multiplyNumbers(a, b, c);  
    alert(result);  
  } catch (e) {  
    alert(e.message);  
  }  
}  
  
function multiplyNumbers(first, second, third) {  
  if (first == null || second == null || third == null)  
  {  
    throw new Error(5, "Forgot to initialize a number.");  
  }  
  return first * second * third;  
}
```

Neste código, o desenvolvedor esqueceu de inicializar a variável `c`, resultando em um valor nulo. No método `multiplyNumbers`, os parâmetros são

avaliados para um valor nulo e, se encontrado, um erro é lançado. Se esse método não verificar valores nulos e assumir que cada desenvolvedor chamado nunca faria um erro, os resultados seria inesperado para o consumidor do método. Neste caso, o resultado seria NaN (não um número), um tipo de JavaScript especial. Isto é devido à tentativa de executar uma operação matemática com um valor nulo.

Experimento de pensamento

Micromanaging exceções Neste experimento de pensamento, aplicar o que você aprendeu sobre este objetivo. Você pode encontrar respostas para essas perguntas na seção "Respostas" no final deste capítulo. A razão por trás manipulação de exceção é, obviamente, para lidar com exceções. Em alguns casos, as condições de erro podem ser previstas e, se a situação ocorrer, ela é tratada e resolvida. Neste caso, você pode não querer simplesmente quebrar um bloco de código inteiro em um grande bloco try ... catch e ter a aplicação deixar de prosseguir. Como você iria incorporar o uso de tentar ... catch blocos em uma rotina JavaScript mais longo com vários pontos de potencial erro? Alguns erros podem ser corrigidos, e alguns podem não ser estar. Os erros corrigíveis devem permitir que o script continue com êxito.

Resumo do objetivo

- O tratamento estruturado de erros é fornecido pela linguagem JavaScript na forma de try ... catch ... finalmente bloco.

- O bloco try ... catch ... finally fornece uma maneira de tentar alguma lógica, pegar um erro e lidar com isso adequadamente e, finalmente, fazer algumas limpar.

- O bloco finally sempre executa se uma exceção é lançada ou não.

- Verificar a existência de um valor nulo antes de acessar qualquer objeto para garantir que eles sejam inicializados é uma boa prática.

Revisão objetiva

1. Qual instrução descreve corretamente o tratamento correto de erros usando try ... catch ... finally blocks?

A. Manuseio adequado de erros permite que o processamento de código continue e forneça feedback do usuário.

B. O correto tratamento de erros permite aos usuários corrigir problemas com a página da Web.

C. O tratamento adequado de erros permite depurar o aplicativo em tempo de execução.

D. O tratamento adequado de erros permite que você suprima todos os bugs em seus scripts.

2. Qual das seguintes não é uma propriedade do objeto de exceção?

A. message

B. description

C. number

D. name

3. Por que a verificação de nulo é uma boa prática?

A. A verificação de null impede o uso de um objeto antes que ele seja inicializado.

B. A verificação de null evita erros resultantes de NaN.

C. A verificação de null evita a necessidade de lançar erros personalizados.

Objectivo 2.4: Implementar um call-back

Callbacks são um padrão de design para implementar quando você estiver trabalhando com vários segmentos ou apenas precisando ter algo trabalhando **assincronamente**. O conceito do retorno de chamada é bastante simples e é usado em todo este livro muito fortemente. **A idéia de um retorno de chamada é chamar uma função para executar, mas é feito para chamar de volta uma função especificada com geralmente algum tipo de resultado ou status da operação.** A seção "Usando métodos avançados de matriz" neste capítulo demonstra algumas das funções disponíveis no objeto de matriz que tomam um retorno de chamada como um parâmetro. O padrão geral é mostrado aqui:

<script>

```
window.onload = function () {  
  WillCallBackWhenDone(MyCallBack, 3, 3);  
}  
  
function WillCallBackWhenDone(f, a, b) {  
  var r = a * b;  
  f(r);  
}  
  
function MyCallBack(result) {  
  alert(result);  
}  
  
</script>
```

Neste exemplo de código, duas funções são declaradas: **WillCallBackWhenDone** e **MyCallBack**. Um parâmetro para a função **WillCallBackWhenDone** é uma função seguida por duas outras variáveis, que neste caso são números que serão multiplicados. O produto da multiplicação é passado para a função de retorno de chamada. Este caso é um pouco sobre o topo para o uso de callbacks, mas demonstra o padrão envolvido. **Sempre que uma função é chamada que espera uma função como um parâmetro, isso é o que está fazendo.** Saber quais parâmetros a função de retorno de chamada receberá é importante para que eles possam ser especificados na lista de parâmetros.

Outro uso comum para callbacks é com eventos. Sempre que um evento DOM é disparado, ele está usando um padrão de retorno de chamada. Uma função é fornecida como um parâmetro ou propriedade para indicar que quando ocorrem coisas específicas, **como um mouseover, para chamar de volta para a função especificada para executar alguma lógica personalizada relacionada à ação do usuário final.**

Muitas APIs que o JavaScript e o navegador expõem como parte da especificação HTML5 envolvem o uso de callbacks. Neste objetivo, a API do WebSocket é examinada. Além disso, o uso de jQuery é introduzido como se aplica a fazer JavaScript assíncrono e XML (**AJAX**) chamadas. A capacidade de ligar um evento e implementar um retorno de chamada usando funções anônimas também é discutida. Finalmente, este objetivo abrange o uso deste ponteiro.

Este objetivo abrange como:

- Implementar comunicação bidirecional com a WebSocket API
- Crie páginas dinâmicas com jQuery e AJAX
- Conecte um evento com jQuery
- Implementar um callback com uma função anônima
- Utilize o ponteiro this

Implementação de comunicação bidirecional com o API do WebSocket

A WebSocket API fornece suporte de comunicação bidirecional para seus aplicativos da Web. O WebSocket simplificou bastante a forma como os dados podem ser enviados e recebidos. Métodos tradicionais, como longa sondagem, têm existido por um longo tempo e são amplamente utilizados em toda a web hoje. No entanto, as técnicas tradicionais usam os mais pesados mecanismos HTTP, o que torna a aplicação inerentemente menos eficiente. O uso da API do WebSocket permite a conexão diretamente a um servidor em um soquete. Esta é uma conexão de peso muito mais leve e é totalmente bidirecional; Tanto os dados binários como os dados baseados em texto podem ser enviados e recebidos.

OBSERVAÇÃO

ACEITAÇÃO DE CONEXÕES DE SOCKET

A implementação completa de uma API do WebSocket requer que um servidor web tenha uma Implementação do lado do servidor que pode aceitar conexões de soquete. Tecnologias como o Node.js funcionam bem para essa finalidade. A implementação de tais tecnologias está além do escopo deste livro, e essas amostras de código assumem que tal implementação existe.

O uso da API do WebSocket é ideal para aplicativos em tempo real, como aplicativos de mensagens / bate-papo, jogos baseados em servidor e cenários mais avançados, como WebRTC (Web Comunicação em tempo real) videoconferência. Os dados transmitidos através do WebSockets podem ser baseados em texto ou binários. O código na Listagem 2-1 demonstra a API do WebSocket.

LISTA 2-1 Implementação da API do WebSocket

```
<script type="text/javascript">
window.onload = function () {
var wsConnection;
var chatBox = document.getElementById("chatWindow");
var disconnectButton = document.getElementById("Disconnect");
var connectButton = document.getElementById("Connect");
var sendButton = document.getElementById("Send");
var msgToSend = document.getElementById("msgSendText");
disconnectButton.onclick = function () {
wsConnection.close();
}
connectButton.onclick = function () {
//Or the use of wss for secure WebSockets. IE:
wss://studygroup.70480.com
//Opens the WebSocket
wsConnection= new WebSocket('ws://studygroup.70480.com', ['soap',
'xmpp']);
}
sendButton.onclick = function () {
//check the state of the connection
if (wsConnection.readyState == WebSocket.OPEN) {
//send message to server.
wsConnection.send(msgToSend.value);
}
else
return;
//show message in chat window.
NewLine();
chatBox.value = chatBox.value + "You: " + msgToSend.value;
//clear message text box
msgToSend.value = "";
}
// event handler for when the WebSocket connection is established
```

```

wsConnection.onopen = function () {
  chatBox.textContent = chatBox.textContent +
  "System: Connection has been established";
}
//event handler for when the WebSocket encounters an error
wsConnection.onerror = function (err) {
  //write an error to the screen
  NewLine();
  chatBox.value = chatBox.value + "System: Error Occurred. ";
}
wsConnection.onclose = function () {
  //write the connection has been closed.
  NewLine();
  chatBox.value = chatBox.value + "System: Connection has been closed.";
}
wsConnection.onmessage = function (msg) {
  //write message
  NewLine();
  chatBox.value = chatBox.value + "Them: " + msg.data;
}
//helper functions.
function NewLine()
{
  chatBox.textContent = chatBox.textContent + '\r\n';
}
}
</script>
<body>
<div align="center">
<div>
70-480 Study Group Chat Forum
</div>
<div>
<textarea id="chatWindow" style="height: 500px; width: 300px">

```

```

</textarea>
</div>
<div>
<input type="text" id="msgSendText" style="width: 300px"/>
</div>
<div>
<button id="Disconnect">Disconnect</button>
<button id="Connect">Connect</button>
<button id="Send">Send</button>
</div>
</div>
</body>

```

O objeto principal com o qual você trabalhará é o objeto WebSocket, que se conecta ao soquete quando seu construtor é invocado. Na Listagem 2.1, uma variável é declarada mas não instanciada até que um usuário chame o botão de conexão. Quando o usuário clica no botão, o WebSocket é instanciado com as informações de conexão apropriadas:

```
wsConnection= new WebSocket('ws://studygroup.70480.com', ['soap',
'xmpp']);
```

O construtor WebSocket aceita dois parâmetros:

- O URL do socket do lado do servidor para se conectar, que é sempre prefixado com ws ou wss para conexões WebSocket seguras

- Uma lista opcional de subprotocolos.

Quando o construtor WebSocket é chamado, a API do WebSocket estabelece uma conexão com o servidor. Uma das duas coisas pode acontecer nesta fase. O WebSocket se conectará com êxito ao servidor ou a conexão falhará, resultando em um erro. Ambos os casos devem ser manipulados de forma que o feedback adequado seja fornecido ao usuário da aplicação. A API do WebSocket fornece um evento para cada um, chamado onopen e onerror, como mostrado anteriormente na Listagem 2-1:

```

// event handler for when the WebSocket connection is established
wsConnection.onopen = function () {
  chatBox.textContent = chatBox.textContent +
  "System: Connection has been established";

```

```

}
//event handler for when the WebSocket encounters an error
wsConnection.onerror = function (err) {
//write an error to the screen
NewLine();
chatBox.value = chatBox.value + "System: Error Occurred.";
}

```

Neste exemplo, ambos os manipuladores de eventos estão fornecendo feedback na janela de bate-papo para permitir que os usuários fiquem informados de uma conexão bem-sucedida ou a ocorrência de um erro. O evento de erro pode acontecer a qualquer momento, não apenas ao estabelecer a conexão inicial.

Quando uma conexão bem-sucedida é estabelecida, você pode enviar e receber mensagens sobre o soquete. Para enviar mensagens, a API do WebSocket fornece a função `Send`. Para receber mensagens, a API do WebSocket fornece o manipulador de eventos `onmessage`. Esses dois métodos mostram as funções e eventos que lidam com a comunicação bidirecional:

```

wsConnection.onmessage = function (msg) {
//write message
NewLine();
chatBox.value = chatBox.value + "Them: " + msg.data;
}

sendButton.onclick = function () {
//check the state of the connection
if (wsConnection.readyState == WebSocket.OPEN) {
//send message to server.
wsConnection.send(msgToSend.value);
}
else
return;
//show message in chat window.
NewLine();
chatBox.value = chatBox.value + "You: " + msgToSend.value;
//clear message text box

```

```
msgToSend.value = "";  
}
```

O primeiro método é um manipulador de eventos para o botão enviar fornecido no HTML. Usuários clique neste botão para enviar mensagens para outros usuários do aplicativo de bate-papo. A API do WebSocket fornece um mecanismo para verificar o status atual da conexão. Para evitar um erro, a propriedade `readyState` é avaliada para garantir que ele está agora aberto. `ReadyState` fornece quatro valores possíveis, conforme descrito na Tabela 2-9.

TABELA 2-9 Valores possíveis do WebSocket `readyState`

| Valor | Descrição |
|------------|--|
| Open | A conexão está aberta |
| Connecting | A conexão está em processo de conexão e ainda não está pronta para uso. Isto é o valor padrão. |
| Closing | A conexão está em processo de fechamento. |
| Closed | A conexão esta fechada. |

Depois de confirmar que a conexão está no estado apropriado para enviar uma mensagem, o método de `send` é chamado com o texto que o usuário inseriu no aplicativo de bate-papo. Além disso, para que cada usuário possa ver que sua mensagem é realmente parte do bate-papo, a mensagem é adicionada à janela de bate-papo. Quando outros usuários do aplicativo de bate-papo enviam mensagens para o sistema, o servidor chama o manipulador de eventos especificado em `onmessage`. O evento `onmessage` recebe um parâmetro de mensagem.

Com a propriedade de dados que contém a mensagem. Esta mensagem é extraída e exibida na janela de bate-papo para que os usuários vejam. Quando terminar com uma sessão de bate-papo, um usuário deve ser capaz de sair e limpar. Isso é realizado chamando o método `close` do objeto `WebSocket`. O método `close` pode ser chamado sem parâmetros. Também permite o uso de dois parâmetros opcionais. Um código numérico e uma razão podem ser fornecidos, mas não é obrigatório. Neste exemplo, a conexão é fechada sem

parâmetros. Quando uma conexão é fechada, o manipulador de eventos `onclose` é gerado:

```
disconnectButton.onclick = function () {  
    wsConnection.close();  
}  
wsConnection.onclose = function () {  
    //write the connection has been closed.  
    NewLine();  
    chatBox.value = chatBox.value + "System: Connection has been closed.";  
}
```

Quando o usuário clica no botão fechar, o método `close` é chamado. Em seguida, a chamada subsequente para o manipulador de eventos `onclose` é implementada para que uma mensagem pode ser fornecida ao usuário que a conexão foi realmente fechada.

Tornar as páginas dinâmicas com jQuery e AJAX

Até agora, ao longo do livro, você já viu algumas ótimas maneiras de tornar dinâmicas as páginas da web usando o JavaScript. JavaScript é o idioma que o navegador da web compreende. Em alguns casos, o uso de JavaScript simples ou a biblioteca JavaScript padrão disponível no navegador pode ser complicado. Aqui é onde o jQuery pode ser útil. **jQuery é uma biblioteca de JavaScript que se especializa em trabalhar com o DOM para tornar as páginas web dinâmicas.**

Na seção anterior, você explorou como usar a API do WebSocket para canal de comunicação bidirecional com o servidor. Nesta seção, você verá **jQuery e AJAX para fazer solicitações de servidor para recuperar conteúdo atualizado para suas páginas.** No tradicional desenvolvimento da web, quando o conteúdo precisa ser atualizado em uma página, um pedido é feito para o servidor para a página em si onde o código do lado do servidor pode ser executado para obter o novo conteúdo, talvez de um banco de dados, e rederizá-lo a página com informações atualizadas para uma melhor experiência do usuário

É uma cintilação como a página inteira precisa ser atualizada. O uso de AJAX resolveu isso permitindo que você faça solicitações do lado do servidor via JavaScript sem ter que atualizar a página inteira. Você pode implementar AJAX sem jQuery. Entretanto, por causa da popularidade e facilidade de uso que jQuery fornece, usando jQuery para implementar este tipo de funcionalidade é muito mais desejável.

Ao solicitar dados de um servidor com JavaScript via jQuery e AJAX, você pode recuperar os dados nos bastidores e, em seguida, usar as várias técnicas de manipulação DOM que você aprendeu para atualizar áreas específicas da página que precisam ser atualizadas. Isso evita a necessidade de enviar uma solicitação para toda a página de volta para o servidor e criar uma agradável experiência ao usuário.

Para este exemplo, você criará um site fictício para pesquisar frutas. A página consiste de uma caixa para inserir um adjetivo sobre a fruta e retornar qualquer fruta que correspondam aos resultados. É configurada como mostrado na Listagem 2-2.

LISTA 2-2 A página do Fruit Finder

```
<html>
<head>
<script src="jquery-2.0.3.min.js" type="text/javascript"></script>
<script type="text/javascript">
window.onload = function () {
$('#searchButton').click(function () {
var searchPath;
$('#searchResults').empty();
switch ($('#searchFruit').val()) {
case 'long':
searchPath = "Fruit/Long.xml";
break;
case 'round':
searchPath = "Fruit/Round.xml";
break;
case 'orange':
searchPath = "Fruit/Orange.xml";
```



```

break;
default:
InvalidSearchTerm();
}
$.ajax({
url: searchPath,
cache: false,
dataType: "xml",
success: function (data) {
$(data).find("fruit").each(
function () {
$('#searchResults').append($(this).text());
$('#searchResults').append("<BR />");
})
}
});
});
}
function InvalidSearchTerm() {
$('#searchResults').empty();
$('#searchResults').append('Invalid Search Term. Please try again.');
```

```

</script>
```

```

</head>
```

```

<body>
```

```

<div>
```

```

Enter search term: <input type="text" id="searchFruit"/>
```

```

<input type="button" id="searchButton" value="Search"/>
```

```

</div>
```

```

<div>
```

```

<h1>Results:</h1>
```

```

</div>
```

```

<div id="searchResults">
```

```

</div>
```

```
</body>
```

```
</html>
```

Nesta listagem, os usuários são apresentados com uma interface de usuário muito simples na qual eles podem inserir um termo de pesquisa e recuperar um conjunto de resultados com base nesse termo de pesquisa. Nesse caso, os usuários podem inserir um dos termos de pesquisa suportados e recuperar os dados do servidor. A solicitação de dados é feita usando AJAX e, como tal, a página inteira não precisa atualizar, apenas a área que exibe os resultados. A parte da página onde os dados são necessários é a única parte da página que é afetada pelos novos dados sendo recebidos.

A primeira coisa que este código faz é configurar um ouvinte de evento para o evento de clique de botão de pesquisa. Toda a magia ocorre nesta função. O termo de pesquisa é avaliado para garantir que ele corresponde a um dos termos de pesquisa suportados. Se não, o usuário é apresentado com uma mensagem indicando isso. Se o fizer, o código prossegue para efectuar uma chamada AJAX para o servidor para o conjunto de dados correcto que corresponde ao termo de procura. Nesse caso, é um arquivo XML codificado. No entanto, a fonte de dados é irrelevante contanto que o XML retornado corresponda ao esquema que a página da Web espera para que ele possa ser analisado e exibido.

Uma chamada AJAX tem algumas notas importantes que você pode definir. Observe uma chamada AJAX da Listagem 2-2:

```
$.ajax({  
  url: searchPath,  
  cache: false,  
  dataType: "xml",  
  success: function (data) {  
    $(data).find("fruit").each(  
      function () {  
        $('#searchResults').append($(this).text());  
        $('#searchResults').append("<BR/>");  
      }  
    )  
  }  
});
```

O primeiro parâmetro que está sendo definido é o `url` que a chamada AJAX está solicitando. Por razões de segurança, para evitar scripts entre sites, esse URL deve estar dentro do mesmo domínio da página da web.

O próximo parâmetro, `cache`, é opcional e indica se a chamada pode usar uma cópia em cache. O terceiro parâmetro, `dataType`, indica o tipo de dados esperado, que pode ser XML ou JavaScript Object Notation (JSON), por exemplo.

O último parâmetro definido neste exemplo é a propriedade `success`. Este parâmetro assume uma função que os resultados das chamadas AJAX devem ser passados para a página da Web para fazer algum trabalho com ele. Neste exemplo, os resultados são analisados e adicionados ao DOM para que os usuários possam ver os resultados.

Outra propriedade que pode ser definida na chamada AJAX, como boa prática, é a propriedade de `error` para que quaisquer condições de erro podem ser tratadas com graça. Esta é a listagem atualizada com um conjunto de funções de erro:

```
$.ajax({
  url: searchPath,
  cache: false,
  dataType: "xml",
  success: function (data) {
    $(data).find("fruit").each(
      function () {
        $('#searchResults').append($(this).text());
        $('#searchResults').append("<BR />");
      }
    ),
    error: function (xhr, textStatus, errorThrown) {
      $('#searchResults').append(errorThrown);
    }
  });
```

A função de erro passa três parâmetros úteis:

■ ■ O pedido HTTP em si

■ ■ O número de erro HTTP (como 404)

■ ■ O texto de erro (como não encontrado)

Você pode imitar um erro 404, alterando uma das palavras de pesquisa para retornar um caminho inválido.

O jQuery AJAX toolkit suporta não apenas a obtenção de dados, mas também a postagem de dados no servidor. O tipo de solicitação padrão é GET. Para alterar uma chamada para uma postagem, altere o valor da propriedade type para POST:

```
$.ajax({  
  url: searchPath,  
  cache: false,  
  dataType: "xml",  
  type: "POST",  
  success: function (data) {  
    ...  
  },  
  error: function (xhr, textStatus, errorThrown) {  
    $('#searchResults').append(errorThrown);  
  }  
});
```

Talvez o usuário do site saiba de frutas adicionais que se encaixam em uma determinada categoria. A página pode ser aprimorada para permitir que os usuários digitem o nome de uma fruta e enviá-la para o arquivo XML para que as pesquisas subsequentes a incluam. Idealmente, neste caso, você usaria o método POST para uma função do lado do servidor que aceitaria esses dados e os armazenaria no arquivo XML.

Ligando um evento com jQuery

No Objetivo 2.2, "Aumentar e manipular um evento", você viu como configurar os listeners de eventos para várias ações que o DOM ou a interação

de um usuário com o DOM poderia invocar. Um dos problemas mais comuns encontrados pelos desenvolvedores web é a compatibilidade entre navegadores. Embora este tópico seja grande e este livro não tenha espaço para entrar em grande detalhe sobre a compatibilidade entre navegadores, o jQuery é um dos kits de ferramentas disponíveis para ajudar a resolver o problema. Na Listagem 2-2, você viu um exemplo de **sintaxe jQuery para conectar um evento**:

```
$('#searchButton').click(function () {  
    ...  
})
```

Neste exemplo, a sintaxe do seletor jQuery é usada para encontrar o botão de pesquisa na página por seu nome. Em seguida, o evento de clique é atribuído a uma função que é executado quando o botão é clicado. Esta sintaxe é bastante poderosa. Além de ser compatível com vários navegadores, ele inclui muita flexibilidade na maneira como os manipuladores de eventos são atribuídos a objetos. Esta sintaxe de seletor jQuery suporta todo o mesmo tipo de pesquisas que o objeto de documento expõe. Mas a parte que diferencia o jQuery do objeto de documento é que jQuery pode atribuir estilos ou eventos a tudo no conjunto de resultados em uma linha.

Suponha que o HTML a seguir compõe a marcação de uma página da Web:

```
<body>  
<table>  
<tr>  
<td id="door1">Door 1</td>  
<td id="door2">Door 2</td>  
<td id="door3">Door 3</td>  
</tr>  
</table>  
</body>
```

O script a seguir é o método mais tradicional para atribuir manipuladores de eventos aos elementos DOM:

```
<script type="text/javascript">
```

```

window.onload = function () {
  document.getElementById("door1").onclick = function () { };
  document.getElementById("door2").onclick = function () { };
  document.getElementById("door3").onclick = function () { };
}

```

Este script bastante simples tem apenas três células para adicionar um evento de clique para. Mas se a página é para ficar mais complexo e ter até 20 ou 50 portas, este código torna-se tedioso. Aqui é onde o jQuery pode simplificar as coisas. O código anterior pode ser substituído por este código:

```

$("document").ready(function () {
  $("td").click(function () { });
});

```

Observe como este código é mais fácil. Em uma linha, todos os elementos <td> recebem um evento de clique. Este código se aplica a todos os elementos <td> na página. Portanto, se alguns elementos <td> não fazem parte da página, você precisa garantir que o seletor é exclusivo para os elementos necessários. Isso pode ser feito com folhas de estilo em cascata (CSS) ou usando a hierarquia DOM, como neste exemplo:

```

$("document").ready(function () {
  $("#GameRow td").click(function () {
    alert( $(this).text());
  });
});
...
<table>
<tr id="GameRow">
<td id="door1">Door 1</td>
<td id="door2">Door 2</td>
<td id="door3">Door 3</td>
</tr>
</table>
<table>
<tr id="SomeOtherRow">

```

```
<td id="cell1">Not a Door 1</td>
<td id="cell2">Not a Door 2</td>
<td id="cell3">Not a Door 3</td>
</tr>
</table>
```

Os eventos de clique são atribuídos apenas aos elementos `<td>` que são filhos de um elemento chamado `GameRow`. JQuery fornece capacidades de seletor avançadas que permitem controle fino sobre como o DOM é manipulado.

Implementando um callback com uma função anônima

As funções de retorno de chamada são usadas em todos os lugares. O conceito de retorno de chamada é a base para o funcionamento dos eventos. É o mecanismo pelo qual as operações assíncronas retornam ao chamador. Em linguagens de programação tradicionais, um retorno de chamada é conseguido passando um ponteiro para uma função para outro processo de modo que quando esse processo termina ou está em estágios especificados do processo, a função é chamada para aconselhar o chamador de um status de algum tipo. Isso pode acontecer quando a operação é concluída e pode estar repassando dados para o chamador. Um exemplo disso seria uma chamada de serviço da Web assíncrona que retorna dados. O princípio é o mesmo em JavaScript.

Em JavaScript, as funções são consideradas objetos e são freqüentemente citadas como cidadãos de primeira classe. Isso significa que uma variável pode ser atribuída a uma função, ou uma função pode ser passada para outra função como um parâmetro. Ver funções utilizadas desta forma é uma convenção comum em JavaScript. Funções usadas desta forma são chamadas funções anônimas. Uma função é considerada anônima quando não tem nome. A declaração de função a seguir tem um nome, portanto, não seria considerado anônimo:

```
function SubmitClick() {
    //do some logic here
}
```

Aqui é declarada uma função que pode ser usada em toda a página. Esta função tem um nome: `SubmitClick`. Uma vez que esta função tem um nome, não é uma função anônima. No entanto, uma função nomeada como esta pode ser atribuída a tantos eventos botão que você deseja:

```
$("#Button1").click(SubmitClick);  
$("#Button2").click(SubmitClick);  
$("#Button3").click(SubmitClick);
```

Com uma função nomeada, a conveniência de reutilização está lá. No entanto, em alguns casos, isso é mais sobrecarga do que é necessário. Isso também pode tornar o código mais difícil de seguir em termos de ser capaz de ver facilmente o que realmente está acontecendo no manipulador de eventos de clique. Em uma situação que especifica um comportamento distinto para cada botão, as funções anônimas simplificam muito as coisas. O código a seguir demonstra usando funções anônimas em vez da função nomeada:

```
$("#Button1").click(function () { ... });  
$("#Button2").click(function () { ... });  
$("#Button3").click(function () { ... });
```

Cada botão tem sua própria função inline, onde a implementação pode ser personalizada para cada clique de botão. Nesse exemplo, o uso de função anônima é aparente porque a função não tem um nome. A sintaxe para uma função anônima é a seguinte:

```
function (n,n,...,n) { body };
```

A declaração de função anônima deve começar com a palavra-chave `function`, que deve ser seguida por parênteses fechados. Os parênteses podem incluir zero ou mais parâmetros. Os parênteses são seguidos por chaves fechadas nas quais o código bloco que compõe a implementação da função é codificado.

A **única diferença** entre uma função **anônima** e uma **função nomeada** é a parte de **nome da assinatura de função**. **Que a função anônima aceita parâmetros é um conceito importante ao lidar com callbacks.**

Ao trabalhar com uma API, a sua própria ou de terceiros, a funcionalidade é frequentemente desde que inclua o uso de callbacks. Como discutido anteriormente, **callbacks são funções que são processadas quando a transferência de controle retorna para o chamador.** Por exemplo, na seção anterior usando jQuery com AJAX, o exemplo de código a seguir foi usado:

```
$.ajax({  
  url: searchPath,  
  cache: false,  
  dataType: "xml",  
  error: function(hdr, num, txt){...}  
  success: function (data) {  
    ...  
  }  
});
```

Neste exemplo, a chamada AJAX permite especificar algumas funções para chamar novamente para circunstâncias diferentes que podem ocorrer. As propriedades de **erro** e **sucesso** permitem especificar uma função que o quadro AJAX chama depois que ele conclui com êxito a solicitação ou encontra um erro. Em cada caso, os parâmetros são especificados para receber os dados que acompanham cada callback.

As funções de retorno de chamada também podem ser usadas na forma de um parâmetro para outra função. Considere o exemplo a seguir que aceita a entrada de um usuário para avaliar se uma pontuação é uma aprovação ou uma falha:

```
$("document").ready( function () {  
  $("#Button1").click( function () {  
    DoLongTask($("#inputValue").val(),  
    function (result, data) {  
      if (result == "SUCCESS")  
        alert(data + " is a Success");  
      else  
        alert(data + " is a fail");  
    }  
  }  
});
```

```

});
});
} );
function DoLongTask(n,f)
{
if (n < 10)
f("SUCCESS", n);
else
f("FAIL", n);
}

```

Este código faz uso pesado de funções anônimas e callbacks. A primeira instância é o retorno de chamada do documento pronto. Nesse caso, você pede ao processador para chamar de volta para uma função anônima depois que ele atinja o estado pronto:

```

$("#document").ready( function () {...

```

Em seguida, você deseja manipular um evento de clique. Nesse caso, você indica ao processador que, quando recebe um clique de um botão específico, para retornar a sua função anônima:

```

$("#Button1").click( function () {...

```

Em seguida, em seu clique do botão é onde você está codificando sua própria lógica para a página. A entrada do usuário é retirada da caixa de entrada e passada para uma função que a avalia. A função não faz nada mais do que avaliar o valor e produzir o resultado. Qualquer chamador que está interessado no resultado pode fornecer uma função de retorno de chamada para obter o resultado.

```

DoLongTask($("#inputValue").val(),
function (result, data) {
if (result == "SUCCESS")
alert(data + " is a Success");
else
alert(data + " is a fail");
});

```

A chamada para DoLongTask aceita dois parâmetros: o valor a ser avaliado e um retorno de chamada (CALLBACK).

Função para passar os resultados para quando ele é feito. Uma função anônima é passada para a função DoLongTask como o retorno de chamada para executar. Nesse caso, o retorno de chamada é conhecido por fornecer dois parâmetros, portanto, a função de retorno de chamada aceita dois parâmetros: o valor original eo resultado da avaliação. O retorno de chamada fornece informações aos usuários sobre o resultado do cálculo.

As funções de retorno de chamada são muito úteis e amplamente utilizadas no desenvolvimento de JavaScript. Funções de retorno de chamada podem existir estaticamente com um nome ou ser fornecidas inline dinamicamente como anônimo.

Usando este ponteiro

Este ponteiro é um objeto especial fornecido pelo framework jQuery. Ao executar seleções contra o DOM usando jQuery, isso se refere ao objeto que ele encontra ou à coleção de objetos que ele encontra. Ele fornece um atalho para acessar o item dentro do contexto atual do filtro jQuery. Em um exemplo simples, a palavra-chave this pode ser demonstrada da seguinte maneira:

```
$("#document").ready(  
  function () {  
    $('#floorDiv').click(function () {  
      $(this).css("background-color", "red");  
    })  
  }  
);
```

Neste exemplo, o elemento floorDiv é atribuído uma função anônima para ser executado quando ele é clicado. Dentro da função, em vez de consultar o DOM para o elemento novamente para fazer algo com ele, a palavra-chave this fornece uma referência para o elemento que iniciou o evento. Neste caso, \$ (this) fornece uma referência ao elemento floorDiv, e você pode fazer o que quiser com esse elemento. Nesse caso, você está apenas alterando a propriedade de estilo

de cor de plano de fundo da div. Em cenários mais avançados, o resultado do seletor pode retornar mais de um elemento. O exemplo a seguir demonstra isso:

```
$("#document").ready(  
  function () {  
    $('#floorDiv').click(function () {  
      $("div").each(function () { $(this).css("background-color", "red");  
    });  
  })  
});
```

Neste exemplo, quando `floorDiv` é clicado, `$ ("div")` encontra todos os elementos `div` na página. Em seguida, ele chama o operador `each`, que chama a função de retorno de chamada passada para cada elemento que é retornado. Então, `$(this)` é usado para modificar a cor de fundo de `cada div`. Desta forma, o uso da palavra-chave `this` é extremamente eficiente porque fornece acesso direto rápido a cada elemento com muito pouco código.

Experimento de pensamento

Criando um aplicativo de bate-papo

Neste experimento mental, aplique o que aprendeu sobre esse objetivo. Você pode encontrar respostas para essas perguntas na seção "Respostas" no final deste capítulo. Você usaria o WebSockets ou usaria o AJAX para criar um aplicativo assíncrono de comunicação bidirecional em JavaScript? Para este experimento de pensamento, descreva como você criaria um aplicativo de bate-papo em HTML em HTML5 baseado em JavaScript.

Resumo do objetivo

- WebSockets fornece comunicação bidirecional com um servidor.
- ■ O WebSockets suporta conexões não seguras (ws) e seguras (wss) com o servidor.
- A estrutura do jQuery AJAX fornece um mecanismo para tornar a Web pedidos assíncrono.
- ■ Você pode conectar eventos usando a sintaxe do seletor jQuery.

Revisão objetiva

1. Qual das seguintes é uma instanciação WebSocket válida?

- A. `wsConnection = new WebSocket('http://studygroup.70480.com');`
- B. `wsConnection = new WebSocket('tcp://studygroup.70480.com',['soap','xmpp']);`
- C. `wsConnection = new WebSocket('wss://studygroup.70480.com',['soap','xmpp']);`**
- D. `wsConnection = new WebSocket('ftp://studygroup.70480.com',['soap','xmpp']);`

2. Qual das seguintes afirmações trata adequadamente a recepção de dados de um WebSocket?

- A. `wsConnection.onpost = function(msg){..};`
- B. `wsConneciton.onreceive = function(msg){...};`
- C. `wsConnection.onmessage = function(msg){...};`**
- D. `wsConnection.ongetdata = function(msg){...};`

3. Qual lista identifica as propriedades que precisam ser configuradas para fazer uma chamada AJAX?

- A. `cache`, `datatype`, `success`
- B. `url`, `cache`, `datatype`, `success`
- C. `url`, `datatype`, `onsuccess`**
- D. `url`, `datatype`, `oncomplete`

4. Por que a fiação de eventos com jQuery é mais fácil?

- A. Permite que você atribua o ouvinte do evento a muitos elementos em uma vez através do seletor sintaxe.**
- B. Não há nenhuma diferença de fiação de eventos com jQuery versus Método `addEventListener`.
- C. jQuery funciona de forma mais eficiente em um loop.
- D. O jQuery permite que funções nomeadas e anônimas sejam usadas como ouvintes de evento.

Objectivo 2.5: Criar um processo de Web worker

Os profissionais da Web apresentam uma maneira de desenvolver aplicativos de JavaScript com vários segmentos. **O JavaScript é um ambiente de thread único. Tudo executado em JavaScript está em fila de forma síncrona.** Isso pode não ser evidente na maioria das aplicações, porque o poder de processamento disponível nos computadores clientes geralmente excede em muito o que é exigido por uma página da Web em um computador cliente. No entanto, em aplicações web mais intensas, você viu mensagens de aviso do navegador que os scripts estão em execução e levando muito tempo para ser concluída. De fato, esses avisos dão aos usuários a opção de parar de executar scripts na página imediatamente. Esse tipo de experiência do usuário não terá usuários voltando ao site. É aqui que a API do Web Worker é útil.

Este objetivo abrange como:

- Começar com um processo de trabalho na Web
- Criar um processo de trabalho com a API do Web Worker
- Usar um web worker
- Compreender as limitações dos operadores da Web
- Configurar tempos limite e intervalos

Introdução a um processo de trabalho na Web

A API do Web Worker permite que você especifique que peças de trabalho devem ser processadas em seu próprio segmento. Fazer isso tem muitas vantagens, mas também algumas armadilhas que você precisa respeitar. Neste objetivo, você aprenderá a usar a API de Web Worker para tirar proveito da flexibilidade que isso traz para aplicações web. Você também aprende sobre as desvantagens e precauções que vêm com o uso de Web Worker. Este objetivo usa o exemplo de bola de saltar para demonstrar o uso de um Web Worker.

A Listagem 2-3 mostra o código básico para a bola saltitante. Ele será ajustado à medida que você trabalhar as seções dentro deste objetivo para conseguir mover o trabalho a um processo do trabalhador da correia fotorreceptora.

LISTING 2-3 Bola de salto

```
<html>
<head>
<script>
window.requestAnimationFrame = (function (callback) {
return window.requestAnimationFrame ||
window.webkitRequestAnimationFrame ||
window.mozRequestAnimationFrame ||
window.msRequestAnimationFrame ||
function (callback) {
window.setTimeout(callback, 1000 / 30);
};
})();
window.setTimeout(getDirection, 30000);
var x = 176, y = 176, w = 600, h = 600, r = 26;
var d,c,s;
var rColor,gColor,bColor;
var hd = "r";
var horizontal = true;
window.onload = function () {
try{
c = document.getElementById("c");
w = c.width;
h = c.height;
s = parseInt( document.getElementById("speedy").value);
getDirection();
drawBall();
document.onkeydown = function () {
switch (window.event.keyCode) {
case 40:
horizontal = false;
hd = "d";
break;
case 37:
```

```

horizontal = true;
hd = "l";
break;
case 38:
horizontal = false;
hd = "u";
break;
case 39:
horizontal = true;
hd = "r";
break;
}
}
} catch (e) {
alert(e.message);
}
}

function increaseSpeed() {
s++;
document.getElementById("speedy").value = s;
}

function decreaseSpeed() {
s--;
document.getElementById("speedy").value = s;
}

function changeDirection() {
var cx = window.event.offsetX;
var cy = window.event.offsetY;
x = cx;
y = cy;
document.getElementById("speedy").value = s;
}

function setNewPoint(d) {
try{

```



```

switch (horizontal) {
case true:
if (x < (w - r) && hd == "r")
x += s;
else if(x > r && hd == "l")
x -= s;
break;
case false:
if (y < (h - r) && hd == "d")
y += s;
else if (y > r && hd == "u")
y -= s;
break;
}
if (x >= (w - r))
hd = "l";
if (x <= r)
hd = "r";
if (y >= (h - r))
hd = "u";
if (y <= r)
hd = "d";
} catch (e) {
alert(e.message);
}
}

function getDirection() {
horizontal = !horizontal;
var d = Math.ceil(Math.random() * 2);
if (horizontal) {
if (d == 1) {
hd = "r";
} else {
hd = "l";
}
}
}

```

```

    }
    } else {
    if (d == 1) {
    hd = "u";
    } else {
    hd = "d";
    }
    }
    }

function drawBall() {
try {
var rgbFill = "rgb(0,0,0)";
var rgbStroke = "rgb(128,128,128)";
setNewPoint(d);
var ctxt = c.getContext("2d");
ctxt.clearRect(0, 0, c.width, c.height);
ctxt.beginPath();
ctxt.lineWidth = "5";
ctxt.strokeStyle = rgbStroke;
ctxt.arc(x, y, r, 0, 360);
ctxt.fillStyle = rgbFill;
ctxt.fill();
ctxt.stroke();
s = parseInt( document.getElementById("speedy").value);
requestAnimationFrame(function () {
drawBall();
});
} catch (e) {
alert(e.message);
}
}

</script>
</head>
<body>

```

```
<canvas id="c" width="1200" height="800" style="border: 2px solid black;
position: absolute; top: 50px; left: 50px;"></canvas>
<input id="intensiveWork" type="button" value="Do Work" /><span
id="workResult"></span>
<input id="speedy" type="range" min="0" max="10" value="10"
style="position:relative; visibility:hidden;" step="1"/>
</body>
</html>
```

Este código simplesmente exibe uma pequena bola pulando em torno de uma tela. Os usuários podem usar as setas para mudar a direção da bola. Os usuários esperam uma experiência suave. Agora você pode introduzir uma operação matemática intensiva para ocorrer com o clique de um botão. O botão já está no formulário e é chamado de `intensiveWork`. Adicione a seguinte função à parte inferior do bloco de script para fazer um pouco de matemática intensa:

```
function DoIntensiveWork() {  
    var result = document.getElementById("workResult");  
    result.innerText = "";  
    var work = 10000000;  
    var i = 0;  
    var a = new Array(work);  
    var sum=0;  
    for (i = 0; i < work; i++) {  
        a[i] = i * i  
        sum += i * i;  
    }  
    result.innerText = sum;  
}
```

Esta função não faz nada mais do que calcular a soma de uma série de quadrados e exibir o resultado para os usuários. A quantidade de trabalho a fazer é difícil neste exemplo de código, mas poderia ser estendido para obter as informações dos usuários também.

Em seguida, adicione o manipulador de eventos de clique ao botão:

```
<script>  
...  
    getDirection();  
    drawBall();  
    document.getElementById("intensiveWork").onclick = function () {  
DoIntensiveWork(); };  
...  
</script>
```

Agora, os usuários podem clicar em um botão e obter a soma dos quadrados para uma série de números sequenciais.

O problema com este código é que, embora o trabalho de matemática está ocorrendo, a interação bola é bloqueada completamente. A bola pára de se mover ea entrada do usuário é aparentemente ignorada até que a chamada de matemática retorna. A chamada para executar os cálculos demora muito tempo e interfere. Você pode experimentar com números menores e ver que eventualmente o número pode ser pequeno o suficiente para que o trabalho aconteça rápido o suficiente para que a bola não seja interrompida. Isso não significa que o aplicativo está fazendo trabalho simultaneamente, embora não ocorra nenhuma interrupção visivelmente.

Criando um processo de trabalho com a API do Web

Worker

A API do Web Worker é baseada na estrutura de mensagens de JavaScript. Esta estrutura subjacente permite que o seu código envie parâmetros para um trabalho e que o trabalho envie resultados de volta. Um trabalho básico da Web é estabelecido criando um arquivo separado para conter o script que será processado no segmento separado. O objeto Worker está disponível a partir do namespace global e é criado da seguinte forma:

```
var webWorker = new Worker("workercode.js");
```

Isso instancia um novo processo de trabalho e especifica qual arquivo contém o código a ser executado no thread de trabalho. O objeto Worker suporta a funcionalidade descrita na Tabela 2-10.

TABELA 2-10 Operações de objeto do trabalhador

Metodo	Descrição
postMessage	Inicia o processo de trabalho. Este método espera que um único parâmetro dados para passar para o segmento de trabalho. Se nada for necessário na thread de trabalho uma String vazia pode ser fornecida.
Terminate	Interrompe o processo de trabalho de continuar.

onMessage	<p>Especifica a função para o thread de trabalho a chamar de volta para quando concluída. Isso</p> <p>Aceita um único parâmetro na forma de EventData com uma propriedade nomeada de dados que contenham os valores.</p>
onerror	<p>Especifica uma função para chamar quando ocorre um erro no segmento de trabalho. O onerror</p> <p>Método recebe dados de evento, incluindo o seguinte:</p> <p>message: mensagem textual do erro</p> <p>filename: o filename o erro ocorreu no</p> <p>lineno: o número da linha no arquivo que criou o erro</p>

Assim que o objeto Worker é instanciado, ele está disponível para uso a qualquer momento. Tudo o que é necessário para iniciar o processo é chamar o método postMessage:

```
webWorker.postMessage("");
```

Assim que o WebWorker estiver em execução, o aplicativo principal continua como de costume. Se algo ocorrer que o processo de trabalho deve ser cancelado, uma chamada para o método de encerramento seria conseguir isso:

```
webWorker.terminate();
```

Depois que o processo de trabalho for concluído e os resultados precisarem ser processados, a função **onmessage** é chamada pelo trabalhador. Isso deve ser configurado antes de iniciar o trabalhador:

```
webWorker.onmessage = function(evt) {...}
```

Isso é tudo o que é necessário no lado da chamada ou no aplicativo da web para criar e gerenciar um processo de trabalho. Em seguida, você precisa criar o próprio código de trabalho. Para isso, você cria o arquivo workercode.js que foi usado no construtor. A primeira linha do arquivo será a propriedade **onmessage** sendo atribuída uma função para processar:

```
onmessage = function(e){...}
```

Isso indica ao runtime o ponto de entrada para o trabalho a ser executado dentro do processo de trabalho.

Em algum lugar no processo de trabalho, onde um resultado deve ser enviado de volta para o aplicativo de chamada, o método `postMessage` é chamado:

```
onmessage = function(e){  
  ...  
  self.postMessage(result);  
}
```

Isso é o que está envolvido na criação de um web worker. Na última peça, observe o usuário da palavra-chave `self`. A palavra-chave `self` é semelhante à palavra-chave `this`. O processo de trabalho é executado em seu próprio contexto, o que significa que ele tem seu próprio namespace global. A palavra-chave automática dá acesso ao namespace global dentro do processo de trabalho.

Usando web workers

Agora que você examinou o fundamento de web workers, você pode ir para trás ao exemplo da esfera de repercussão e mover as operações intensivas da matemática sobre a um processo de web worker de modo que não interfira com a atividade da esfera de salto. Para fazer isso, crie um novo arquivo JavaScript chamado `CalculateWorker.js` com o seguinte código nele:

```
onmessage = function (evt) {  
  var work = 10000000;  
  var i = 0;  
  var a = new Array(work);  
  var sum = 0;  
  for (i = 0; i < work; i++) {  
    a[i] = i * i;  
    sum += i * i;  
  }  
  self.postMessage(sum);  
}
```

Esse código começa com atribuir o manipulador `onmessage` uma função para executar quando gerado dentro do contexto de um trabalhador. No final da mensagem, ele chama `postMessage` para retornar um resultado para o chamador. Salve esse arquivo e, em seguida, altere o manipulador de eventos de clique para o botão de intensidade de trabalho no código bola rebatida da seguinte maneira:

```
document.getElementById("intensiveWork").onclick = function () {  
  var result = document.getElementById("workResult");  
  result.innerText = "";  
  var worker = new Worker("CalculateWorker.js");  
  worker.onmessage = function (evt) {  
    try {  
      result.innerText = evt.data;  
    } catch (e) {  
      alert(e.message);  
    }  
  }  
  worker.onerror = function (err) {  
    alert(err.message + err.filename + err.lineno);  
  }  
  worker.postMessage("");  
};
```

Neste código, o padrão descrito na seção anterior é implementado. Um novo objeto `Worker` é instanciado com `CalculateWorker.js` especificado. Em seguida, o `onmessage` é atribuído uma função para manipular o resultado do thread de trabalho. O `onerror` é atribuído uma função para lidar com quaisquer condições de erro. Finalmente, o `postMessage` é chamado para invocar o trabalhador.

Execute este código e clique no botão Fazer trabalho. A bola agora continua a se mover na tela e responde às teclas de seta. Para tornar o processo de trabalho mais demorado, basta aumentar o tamanho do número com o qual ele precisa trabalhar.

Para fornecer uma opção para parar o processo de trabalho, você precisa implementar o método. Adicione um botão para a página como:

```
<input id="stopWorker" type="button" value="Stop Work" />
```

E adicione o seguinte script abaixo da chamada `postMessage`:

```
document.getElementById("stopWorker").onclick = function () {  
    worker.terminate();  
}
```

Em seguida, clique no botão Fazer trabalho seguido pelo botão Parar trabalho para ver que o trabalho é encerrado e nenhum resultado é retornado.

Compreendendo as limitações do web worker

Web worker são muito convenientes. Eles podem resolver muitos problemas de processamento em aplicações web intensivas. No entanto, estar ciente das limitações impostas aos Web worker também.

Passando parâmetros

O método `postMessage` aceita um parâmetro que permite que ele passe dados para o trabalhador que ele pode precisar para operar. O parâmetro `postMessage` é uma seqüência de caracteres - ele pode ter qualquer objeto serializável, como tipos de dados nativos, objetos JSON ou XML. O parâmetro não pode ser uma função.

No exemplo de bola de saltar, a entrada para qual número trabalhar pode vir de usuários. Uma caixa de entrada pode ser adicionada ao HTML e o valor inserido pode ser passado para o trabalhador. Isso ficaria assim:

```
var value = document.getElementById("inputValue").value;  
worker.postMessage(value);
```

Em seguida, no trabalhador, o valor seria acessado como este:

```
onmessage = function (evt) {  
    var work = evt.data;  
    ...  
}
```

A função recebe um objeto de evento com uma propriedade chamada `data` que contém o que foi passado para o trabalhador.

Número de workers

Embora **nenhum limite seja imposto** sobre quantos workers podem ser processados ou criados simultaneamente, **o número de workers utilizados é algo que você precisa prestar atenção**. Criar workers é uma operação pesada. Cada worker cria threads no nível do sistema operacional e seu uso deve ser gerenciado adequadamente. Se você quer um alto volume de workers, considere criando um pool que pode ser usado em um round-robin moda para que não muitos trabalhadores são criados.

Acesso DOM

Os **workers** operam em seu próprio contexto global, o que significa que eles **não têm acesso ao DOM da página que os invocou**. **O DOM não deve ser manipulado a partir de um processo de workers**. **O contexto de workers não tem acesso ao objeto window, objeto de documento ou qualquer objeto pai**.

Subworkers

Seguindo os mesmos padrões que para um worker da página principal, um worker pode criar workers também. Todas as construções devem ser seguidas para passar dados e obter dados retornados. No entanto, saber quantos workers totais serão criados torna-se cada vez mais importante.

Configurando tempos limite e intervalos

Você pode configurar um web worker para ser executado em um intervalo especificado em segundo plano. Isso é feito usando qualquer método existente **setTimeout** ou **setInterval**. **O método setTimeout chama uma função especificada após o atraso especificado**. **O setInterval chama a função especificada repetidamente após cada intervalo de tempo especificado**. Por exemplo, o seguinte código executa o trabalhador após 3 segundos:

```
setTimeout(function(){  
  work.postMessage("");  
},3000);
```

No entanto, o seguinte código executa o trabalhador a cada 3 segundos:

```
var work = new Worker("workerFile.js");
setInterval(function(){
work.postMessage("");
},3000);
```

Experimento de pensamento

Criando uma página que executa um show de fogos de artifício neste experimento mental, aplique o que aprendeu sobre esse objetivo. Você pode encontrar respostas para essas perguntas na seção "Respostas" no final deste capítulo. Como um exercício para criar uma página da web usando tempos limite e intervalos, pense em como você criaria uma página que executaria um show de fogos de artifício. Diferentes tipos de Os fogos de artifício precisam disparar oem intervalos e atrasos diferentes. Cada um deles precisa viajar para uma altura diferente, e diferentes tipos de fogos de artifício têm diferentes efeitos de explosão e cores.

1. Um grande show de fogos de artifício seria muito intenso? Por que ou por que não?
2. Será que os trabalhadores da web ajudam a torná-lo mais fluido? Por que ou por que não?

Resumo do objetivo

- Os operadores da Web permitem que o tempo de execução do JavaScript forneça o multithreading.
- Os Web Workes podem ter sub-workers.
- O número de Workes que você pode usar é ilimitado, mas muitos Workes podem dificultar o desempenho.
- Os Workes da Web podem receber um único parâmetro contendo quaisquer dados Workes.
- Os Workes da Web não têm acesso ao DOM da página de chamada.
- Use setTimeout para atrasar antes de executar uma função de script. Use setInterval para repetir uma função de script após cada intervalo especificado.

Revisão objetiva

1. Qual das seguintes opções não é uma operação válida para o trabalhador da web?
 - A. `postMessage`
 - B. `onmessage`
 - C. `close`**
 - D. `terminate`

2. Qual método cancela um web worker?
 - A. `close`
 - B. `terminate`**
 - C. `suspend`
 - D. `sleep`

3. Onde você deve colocar o código JavaScript para ser executado no contexto de um web worker?
 - A. Entre os elementos `<head>` `</head>`
 - B. Em qualquer bloco `<script>` na página
 - C. Em seu próprio arquivo JavaScript**
 - D. Como uma função dinâmica atribuída ao `self.worker`

4. Quantos trabalhadores da Web / sub-trabalhadores podem executar simultaneamente?
 - A. Um múltiplo de quatro trabalhadores da correia fotorreceptora including subworkers, por o processador
 - B. 16 trabalhadores por padrão, mas você pode alterar esse número via `self.configuration`
 - C. Um número ilimitado de trabalhadores**
 - D. Um limite de oito trabalhadores, cada um com um máximo de oito sub-trabalhadores

5. Para que um script seja executado continuamente a cada 30 segundos, qual linha de código deve ser usada?
 - A. `wsConnection.repeatWork("workerFile.js",30000);`
 - B. `setTimeout(function(){ worker.postMessage("");}, 30000);`

C. setTimeout(worker.postMessage(""), 30000);

D. setInterval(function(){ worker.postMessage("");}, 30000);

CAPÍTULO 3

Acesso e dados seguros

A maioria das aplicações web requer dados estáticos ou dinâmicos. Os dados estáticos são gravados na marcação HTML, não alterado ou carregado por código, como JavaScript. É renderizado e exibido para os usuários sem qualquer forma para alterar os dados. Os dados dinâmicos podem mudar.

Dados dinâmicos pode atualizar um ticker em uma página da Web a partir de um feed de notícias, capturar dados do usuário para executar um operação e fornecer resultados, ou talvez até armazenar apenas informações de registro de um usuário em um banco de dados.

Ambas as abordagens dos dados têm benefícios e desvantagens. Os dados estáticos são porque ele não fornece muito uma superfície de ataque para um usuário mal-intencionado. Contudo, com um site que possui transições mais dinâmicas, com atualizações ao vivo de dados e a capacidade para os usuários inserirem informações em vários campos, uma superfície de ataque é aberta e o tornam-se menos seguros. Saber como evitar que usuários mal-intencionados causem danos à sua aplicação e possivelmente seus usuários é importante. Você pode implementar os mesmos mecanismos usados para Uso malicioso para simplificar a experiência do usuário e manter seus dados geralmente limpos. Certos elementos de dados, como números de telefone e endereços de e-mail, podem ser de diferentes formatos. Como essas informações podem ser muito importantes. Ter informações de endereço completas e garantir que todos os campos necessários são preenchidos também pode ser muito importante. O HTML5 suporta construções como expressões regulares e atributos necessários para apoiar a implementação esses tipos de regras. Ao longo dos objetivos deste capítulo, validar a entrada do usuário tanto declarativamente via HTML5 e também usando JavaScript é coberto.

Em outros cenários, os dados provenientes para o site é consumindo feeds de dados ou fornecer dados para outro destino. Atualmente, os sites têm comumente um link direto para atualizações de redes sociais.

Nesses casos, a recuperação e o envio dos dados é que os usuários não estão envolvidos com o processo. Estes processos devem ser simplificados e não interferir com a experiência do usuário do site. Nos objetivos deste capítulo, o consumo de dados fontes externas, transmissão de dados e serialização e desserialização de dados estão todos cobertos.

Objectivos deste capítulo:

- ■ Objetivo 3.1: Validar entrada do usuário usando elementos HTML5
- ■ Objetivo 3.2: Validar a entrada do usuário usando o JavaScript
- ■ Objetivo 3.3: consumir dados
- ■ Objetivo 3.4: Serializar, deserializar e transmitir dados

Objetivo 3.1: validar a entrada do usuário usando elementos HTML5

Este objetivo examina os elementos da interface do usuário disponibilizados pelo HTML5 que permitem aos usuários fornecerem entrada. A capacidade de capturar informações de usuários é um grande recurso. No entanto, você deve garantir que a privacidade e a segurança do usuário sejam protegidas da melhor forma possível. Você também deve garantir que o site não abre nenhum buraco que um invasor pode explorar para interromper os serviços do site. **Parte da proteção do site é escolher os controles de entrada de usuário corretos para o trabalho e definir os atributos apropriados nesses controles para garantir que os dados são validados.** Para o exame, você precisa conhecer esses controles de entrada e os atributos que eles usam para essa finalidade.

Este objetivo abrange como:

- ■ Escolha controles de entrada e tipos de entrada HTML 5
- Implementar atributos de conteúdo

Escolhendo controles de entrada

O HTML5 fornece uma ampla variedade de controles para tornar a captura de entrada do usuário simples e segura. Nesta seção, você explora os controles de entrada do usuário em maior detalhe e veja exemplos de seu uso. Uma simulação de um formulário de pesquisa será criada para demonstrar quando cada tipo de controle deve ser usado. A Listagem 3-1 mostra toda a marcação para a pesquisa.

LISTA 3-1 Marcação HTML5 para uma pesquisa com o cliente,

```
<form>
<div>
<hgroup>
<h1>Customer Satisfaction is #1</h1>
<h2>Please take the time to fill out the following survey</h2>
</hgroup>
</div>
<table>
<tr>
<td>Your Secret Code:
</td>
<td>
<input type="text" readonly="readonly" value="00XY998BB"/>
</td>
</tr>
<tr>
<td>Password:
</td>
<td>
<input type="password"/>
</td>
</tr>
<tr>
<td>First Name:
</td>
<td>
```

```
<input type="text" id="firstNameText" maxlength="50"/>
</td>
</tr>
<tr>
<td>Last Name:
</td>
<td>
<input type="text" id="lastNameText"/>
</td>
</tr>
<tr>
<td>
Your favorite website:
</td>
<td>
<input type="url"/>
</td>
</tr>
<tr>
<td>
Your age in years:
</td>
<td>
<input type="number"/></td>
</tr>
<tr>
<td>
What colors have you colored your hair:
</td>
<td>
<input type="checkbox" id="chkBrown" checked="checked"/>
Brown
<input type="checkbox" id="chkBlonde"/>
Blonde
```



```
<input type="checkbox" id="chkBlack"/>
Black
<input type="checkbox" id="chkRed"/>
Red
<input type="checkbox" id="chkNone"/>
None
</td>
</tr>
<tr>
<td>Rate your experience:
</td>
<td>
<input type="radio" id="chkOne" name="experience"/>
1 - Very Poor
<input type="radio" id="chkTwo" name="experience"/>
2
<input type="radio" id="chkThree" name="experience"/>
3
<input type="radio" id="chkFour" name="experience"/>
4
<input type="radio" id="chkFive" name="experience" checked="checked"/>
5 - Very Good
</td>
</tr>
<tr>
<td>How likely would you recommend the product:
</td>
<td>
<br/>
<br/>
<br/>
<br/>
<input type="range" min="1" max="25" value="20"/>
</td>
```

```

</tr>
<tr>
<td>
Other Comments:
</td>
<td>
<textarea id="otherCommentsText" rows="5" cols="20" spellcheck="true">
</textarea>
</td>
</tr>
<tr>
<td>
Email address:
</td>
<td>
<input type="email" placeholder="me@mydomain.com" required/>
</td>
</tr>
<tr>
<td>
<input type="submit"/>
<input type="reset"/>
<input type="button" value="Cancel"/>
</td>
</tr>
</table>
</form>

```

NOTA CONTROLES DE ENTRADA

A especificação HTML5 define muitos mais controles de entrada do que são explicados neste livro. Este livro concentra-se especificamente nos controles agora suportados pelo Internet Explorer, seguido de exemplos menores para demonstrar alguns dos outros controles como suportados por outros navegadores como o Google Chrome.

O elemento <input> em HTML denota controles de entrada. Este elemento contém um atributo de tipo que especifica o tipo de elemento de entrada a ser renderizado. As exceções à regra <input type = "> são os elementos <textarea> e <button>, que possuem seu próprio suporte de elemento. A Tabela 3-1 descreve os elementos de entrada suportados em HTML5 e indica se um elemento é agora suportado no Internet Explorer. Os atributos adicionais disponíveis para um elemento <input> são discutidos em seções posteriores.

TABELA 3-1 Elementos de entrada HTML5

Elemento	Descrição
Color*	Fornece um seletor de cores
Date*	Fornece um seletor de data
dateTime*	Fornece um seletor de data / hora
Month*	Permite que os usuários selecionem um mês numérico e ano
Week*	Permite que os usuários selecionem uma semana e um ano numéricos
Time*	Permite aos usuários selecionar uma hora do dia
Number*	Força a entrada a ser numérica
Range	Faixa Permite que os usuários selecionem um valor dentro de um intervalo usando uma barra deslizante
Tel*	Formatos inseridos dados como um número de telefone
url	Formatos dados inseridos como um URL formatado corretamente
Radio+	Permite que os usuários selecionem um único valor para uma lista de opções
Checkbox+	Permite que os usuários selecionem vários valores em uma lista de opções

Password+	Captura uma senha e glifos os caracteres inseridos
Button+	Permite que os usuários executem uma ação, como executar script
Reset+	Redefine todos os elementos HTML dentro de um formulário
Submit+	Envia os dados do formulário para um destino para processamento posterior

* Não suportado atualmente pelo Internet Explorer

+ Não é novo no HTML5

Usando texto e textarea tipos de entrada

Os controles de entrada `text` e `textarea` são os mais flexíveis. Usando esses controles, você permite que os usuários insiram qualquer texto que desejarem em uma caixa de texto normal. Uma caixa de texto fornece uma entrada de texto de linha única, enquanto uma área de texto permite uma entrada de dados de várias linhas. O HTML a seguir mostra a marcação para os dois tipos de controles:

```
<table>
<tr>
<td>
First Name:
</td>
<td>
<input type="text" id="firstNameText"/>
</td>
</tr>
<tr>
<td>
Last Name:
</td>
<td>
<input type="text" id="lastNameText"/>
</td>
</tr>
```

```

</tr>
...
<tr>
<td>Other Comments:
</td>
<td>
<textarea id="otherCommentsText" rows="5" cols="20"></textarea>
</td>
</tr>
</table>

```

A Figura 3-1 mostra a saída desse código.

Este código adiciona caixas de texto para capturar informações como nome, sobrenome e comentários adicionais. Para o primeiro e último nomes, a entrada é uma **caixa de texto** padrão como denotado por tipo = "texto". Isso informa o processador para exibir um campo de entrada em que os usuários podem inserir texto de forma livre. No entanto, este tipo de campo de entrada está limitado a uma única linha. A caixa de texto Outros comentários fornece uma área de texto de várias linhas para os usuários inserirem texto. Os atributos **rows** e **cols** definem o tamanho visível da área de texto. Nesse caso, os usuários podem inserir muitas linhas de texto na área de texto.

Outros atributos que ajudam a controlar quanta informação é inserida nos campos de texto é o atributo **maxlength**:

```
<input type="text" id="firstNameText" maxlength="50"/>
```

Os usuários não podem inserir mais de 50 caracteres no campo de texto com o **maxlength** definido como um valor de 50. Em alguns casos, convém assegurar que os usuários insiram apenas determinadas informações em um determinado formato.

url input type

O `<input type="url">` exibe uma caixa de texto semelhante à que o tipo de texto `<input type="text">` fornece. No entanto, o processador é instruído que o tipo de entrada é url, portanto, quando os usuários tentam enviar um formulário com esse tipo de informações nele, ele valida que o texto na caixa corresponde ao formato de um URL válido.

Dica de exame

Você pode validar dados de muitas maneiras. Ainda mais opções ficam disponíveis em HTML5, como o tipo de entrada url. Também estão disponíveis o atributo padrão eo uso de expressões regulares em JavaScript. Ambos são discutidos mais adiante neste capítulo.

O código a seguir demonstra um tipo de url adicionado à pesquisa:

```
<tr>
<td>Last Name:
</td>
<td>
<input type="text" id="lastNameText"/>
</td>
</tr>
<tr>
<td>
Your favorite website:
</td>
<td>
<input type="url"/>
</td>
</tr>
<tr>
<td>
Other Comments:
</td>
<td>
<textarea id="otherCommentsText" rows="5" cols="20"></textarea>
```

```
</td>
</tr>
...
<tr>
<td>
<input type="submit"/>
</td>
</tr>
```

Este código produz a saída mostrada na Figura 3-2 para a página HTML que compõe a pesquisa. Este código HTML também adiciona um botão de entrada, conforme discutido mais adiante na seção, "Usando o tipo de entrada de botão."

FIGURA 3-2 A caixa de entrada de URL adicionada à pesquisa

Este código demonstra o poder do tipo de entrada url em validar que o texto de um usuário inserido é realmente um formato de URL válido. Se um usuário digitou algo diferente de um URL ou um URL incompleto na caixa Seu site favorito, como contoso.com e, em seguida, clicou no botão Enviar consulta, o resultado seria semelhante à saída mostrada na Figura 3-3.

FIGURA 3-3 Demonstrando a validação do tipo de entrada url

Clique no botão para invocar a validação. A caixa url é delineada em vermelho, e uma dica de ferramenta aparece para explicar o erro de validação. Nesse caso, ele detectou que um URL válido não foi inserido. Se o usuário corrigir os dados especificando o URL como http://www.contoso.com, o erro de validação não ocorre e a entrada pode ser enviada com êxito.

Se você precisar de mais flexibilidade e quiser aceitar informações de URL parcialmente inseridas, como contoso.com, não use a caixa de entrada do url. Uma entrada de texto regular com um padrão especificado seria mais apropriado.

Usando o controle de entrada de senha

O controle de entrada de senha é o método padrão de solicitar aos usuários em formação. À medida que você digita sua senha, cada caractere é substituído por um glifo para que os espectadores não possam ver sua senha.

Dica de exame

Você não pode especificar o texto padrão em uma caixa de senha ou escrever para ele via JavaScript. Esta é uma garantia de segurança para ajudar a garantir a segurança das senhas. No entanto, os navegadores fornecem um mecanismo para armazenar uma senha se um usuário optar por ter a senha lembrada pelo navegador.

Você pode adicionar uma caixa de texto de senha à pesquisa para fornecer uma maneira de recuperar uma pesquisa se um usuário desejar completá-la mais tarde. A senha pode ser armazenada em um servidor para posterior recuperação. A marcação a seguir é adicionada ao HTML:

```
<tr>
<td>
Password:
</td>
<td>
<input type="password"/>
</td>
</tr>
```

Com este HTML adicionado, a pesquisa agora aparece como mostrado na Figura 3-4.

FIGURA 3-4 Um campo de entrada de senha adicionado ao formulário

Novamente, a caixa de texto de senha não parece diferente de qualquer outra caixa de texto. No entanto, digitando na caixa fornece uma experiência diferente, como mostrado na Figura 3-5.

FIGURA 3-5 Substituindo a entrada da senha pelo caractere de glifo

O tipo de entrada de senha captura informações de forma segura. Usuários digitando essas informações não querem que outros que estão

próximos possam ver o que eles estão digitando e, portanto, comprometer seus dados.

Usando o input type email

Você pode usar o tipo de entrada de e-mail para garantir que o formato do texto inserido na caixa de texto corresponda ao de um endereço de e-mail válido. Ser capaz de capturar um endereço de e-mail é muitas vezes importante para permitir mais acompanhamento com um usuário. Esse controle ajuda a garantir que as informações inseridas correspondam ao esperado na forma de um endereço de e-mail.

Dica de exame

A validação do tipo de entrada de e-mail confirma apenas que as informações inseridas correspondem ao formato esperado de um endereço de e-mail válido. De forma alguma verifica se o próprio endereço de e-mail é uma caixa de correio válida que pode receber mensagens.

O HTML a seguir adiciona um tipo de entrada de endereço de e-mail à pesquisa:

```
<tr>
<td>
Email address:
</td>
<td>
<input type="email"/>
</td>
</tr>
<tr>
<td>
<input type="submit"/>
</td>
</tr>
```

A Figura 3-6 mostra a saída deste HTML.

FIGURA 3-6 Saída do tipo de entrada do endereço de e-mail

Assim como com o tipo de entrada do url, se você digitar texto que não corresponde ao formato de um endereço de e-mail, você receberá uma mensagem de aviso (consulte a Figura 3-7).

FIGURA 3-7 Validação para o tipo de entrada do endereço de e-mail

Essa validação ajuda a garantir que você não digite seu endereço de e-mail errado. Claro que não impede que você digite um endereço de e-mail inválido, apenas um onde o formato não corresponda corretamente ao que seria esperado, como ter o símbolo @ e terminar com um .com ou outro sufixo de domínio.

Usando o input type checkbox

Em alguns casos, quando a captura de informações de usuários, você precisa ser capaz de capturar mais de uma escolha, uma vez que se relaciona com uma pergunta específica. Neste caso, o check box é a melhor escolha. Você pode fornecer uma série de caixas de seleção e permitir que os usuários selecionem todos os que se aplicam.

O inquérito agora irá adicionar uma pergunta onde os usuários podem selecionar vários itens, da seguinte forma:

```
<tr>
<td>Your age in years:</td>
<td><input type="number" /></td>
</tr>
<tr>
<td>
What colors have you colored your hair:
</td>
<td>
<input type="checkbox" id="chkBrown"/> Brown
<input type="checkbox" id="chkBlonde"/> Blonde
<input type="checkbox" id="chkBlack"/> Black
<input type="checkbox" id="chkRed"/> Red

```

```
<input type="checkbox" id="chkNone"/> None  
</td>  
</tr>
```

Neste exemplo HTML, os usuários visualizam uma lista de cores de cabelo que eles poderiam ter usado. Porque um usuário possivelmente poderia ter usado mais de um, ela tem a opção de escolher mais de um.

A Figura 3-8 mostra a saída deste HTML.

FIGURA 3-8 A caixa de seleção de entrada adicionada ao formulário HTML

Um atributo adicional disponível na caixa de seleção é o atributo **checked**. Esse atributo fornece uma maneira de usar como padrão uma caixa de seleção para o estado "marcado" (ou selecionado). Por padrão, as caixas de seleção não estão selecionadas. No entanto, adicionando o atributo da seguinte maneira, a caixa de seleção padrão para o estado "checked" quando a página é carregada:

```
<input type="checkbox" id="chkBrown" checked="checked"/> Brown
```

Em outros casos, quando apresentados com uma lista de itens, os usuários poderão escolher apenas um único item da lista.

Usando o input type radio

O **radio button** é semelhante **checkbox**, pois fornece uma lista de opções para os usuários selecionarem. A diferença da caixa de seleção é que os usuários podem selecionar apenas um único item da lista. Um exemplo seria pedir aos usuários que classificassem algo em uma escala de 1 a 5. Para adicionar esse tipo de pergunta à pesquisa, incorpore o seguinte HTML abaixo das caixas de seleção:

```
<tr>  
<td>  
Rate your experience:  
</td>  
<td>  
<input type="radio" id="chkOne" name="experience"/> 1 - Very Poor  
<input type="radio" id="chkTwo" name="experience"/> 2  
<input type="radio" id="chkThree" name="experience"/> 3  
<input type="radio" id="chkFour" name="experience"/> 4  
<input type="radio" id="chkFive" name="experience"/> 5 - Very Good  
</td>
```

</tr>

Observe que, como todos os elementos HTML, cada tipo de entrada de rádio precisa de um id exclusivo. No entanto, o atributo nome liga todos os botões de opção juntos. Com o mesmo nome especificado para cada tipo de rádio, o navegador sabe que eles fazem parte de um grupo e que apenas um botão de opção do grupo pode ser selecionado.

A Figura 3-9 mostra a saída dos botões de rádio adicionados ao levantamento.

Nesta saída, os botões de opção são mostrados da esquerda para a direita e permitem selecionar apenas uma opção. Quando um usuário altera a seleção para uma opção diferente, a opção selecionada anteriormente é automaticamente desmarcada.

FIGURA 3-9 Adicionando alguns tipos de entrada de rádio ao formulário

Tal como acontece com os tipos de entrada da caixa de verificação, o estado da entrada de rádio selecionada é possível. Isso é feito exatamente da mesma maneira, especificando o atributo **checked**:

```
<input type="radio" id="chkFive"
name="experience"checked="checked"/> 5 - Very Good
```

Nesse caso, a classificação de 5 - Muito bom padrão é selecionada para o grupo de botões de opção. Você pode ter vários grupos de botões de opção na mesma página especificando um nome diferente para cada grupo de botões. Outra maneira de fornecer aos usuários a capacidade de especificar um único valor dentro de um grupo de valores é com o uso do controle de intervalo.

Usando o input type range

Usar o **type range** permite que os usuários especifiquem um valor dentro de um intervalo predefinido usando uma barra deslizante. Este tipo pode ser usado nos casos em que uma escala mais larga dos valores é exigida para escolher usar os botões de rádio seria demasiado. Adicione outra pergunta de classificação ao questionário, conforme mostrado no HTML a seguir após os botões de opção:

<tr>

<td>How likely would you recommend the product:

</td>

<td>

<input type="range" min="1" max="25" value="20"/>

</td>

</tr>

Essa marcação HTML fornece aos usuários uma barra deslizante que eles podem usar para especificar um valor entre 1 e 25. O atributo `min` especifica o valor mínimo do intervalo; O atributo `max` especifica o valor máximo. O atributo `value` especifica um valor padrão. Se você omitir o atributo `value`, o intervalo padrão para o valor será o mínimo. Este HTML exibe a saída mostrada na Figura 3-10.

FIGURA 3-10 Um elemento de entrada de intervalo adicionado ao formulário HTML

Nesta saída, o controle de intervalo é exibido como uma barra deslizante. A barra predefinida para o valor de 20 como especificado na marcação. Os usuários podem pegar o ponto de extremidade preto do controle deslizante e alterar o valor inferior ou superior, arrastando-o para a esquerda ou para a direita. À medida que um usuário muda o valor, uma dica de ferramenta mostra o valor atual onde reside o controle deslizante. Neste caso, o usuário está agora no valor 17 (veja a Figura 3-11).

FIGURA 3-11 A dica de ferramenta exibe o valor atual do intervalo à medida que o usuário o altera.

Depois que os usuários inserem todas as informações necessárias, eles precisam de uma maneira de enviar ou em formação. O botão Enviar já foi visualizado.

Usando o input type button

O tipo de entrada que permite aos usuários enviar o formulário ou desmarcá-lo é o `button`. A entrada do botão não é nova para o HTML5, mas é uma peça essencial para o quebra-cabeça de captura de dados. Botões são o que dizem o site quando um usuário termina de fazer alguma coisa e que eles querem executar uma ação. O elemento `<input>` suporta três tipos de controles de botão: `submit`, `reset` e `button`.

Dica de exame

Qualquer coisa pode ser um "botão". Como a maioria dos elementos DOM tem um evento de clique ou pelo menos um evento `mousedown` e `mouseup`, o conceito de clicar pode ser capturado e ações personalizadas processadas. Isso pode inerentemente transformar qualquer parte do DOM em um "botão".

O tipo de entrada de `submit` informa o formulário HTML para postar suas informações para o servidor (ou, em alguns casos, para outro site ou página da Web). O tipo de `reset` automaticamente limpa todos os elementos do formulário para seus valores padrão. O tipo de `button` fornece um botão genérico sem funcionalidade predefinida. Ele pode ser usado para fornecer uma função personalizada, como cancelar esta página e retornar à página inicial. Todos os três tipos de botões são adicionados à parte inferior da página de pesquisa da seguinte forma:

`<tr>`

```
<td>
<input type="submit"/>
<input type="reset"/>
<input type="button"/>
</td>
</tr>
```

Isso é tudo o que é necessário para adicionar a funcionalidade à página para cada botão. Claro, o tipo = "button" requer algum JavaScript para ser conectado até realmente fazer algo. No entanto, os botões **submit** e **reset** vêm com a funcionalidade descrita embutida. O HTML fornece a saída no formulário como mostrado na Figura 3-12.

FIGURA 3-12 Botões adicionados ao formulário HTML

O texto nos botões é o texto padrão. O botão de envio vem com o texto **Submit Query**, e o botão de reinicialização vem com o texto **Reset**. Isso não pode ser alterado. No entanto, **o tipo de botão não tem nenhum texto nele** porque nenhum foi especificado e o botão não vem com qualquer comportamento predeterminado. Para especificar o texto para este botão, adicione o atributo value:

```
<input type="button" value="Cancel"/>
```

Isso produz um botão como mostrado na Figura 3-13.

FIGURA 3-13 O tipo de botão com o texto especificado

Isso é o que você obtém com o tipo de entrada do botão. No entanto, em alguns casos, é desejada mais flexibilidade no conteúdo do botão. Este é o lugar onde o elemento do botão vem a calhar.

Usando o elemento button

O elemento button fornece um botão na interface do usuário, tal como o nome implica.

No entanto, a partir de uma perspectiva gráfica, este elemento se comporta de forma muito diferente.

O elemento de botão também suporta um atributo de tipo, como os vistos anteriormente: submit, reset e button. Este exemplo se afasta da pesquisa e demonstra esses botões em uma página autônoma. O seguinte HTML é adicionado a uma página e a saída subsequente é

Mostrado na Figura 3-14:

```
<button type="button"/>
<button type="reset"/>
```

```
<button type="submit"/>
```

FIGURA 3-14 Todos os três tipos de elementos de botão

Esta saída exibe três botões, como esperado. No entanto, ele não fornecer qualquer texto sobre os botões. O elemento button fornece somente o comportamento de clique desejado, como enviar, redefinir ou fornecer um comportamento personalizado como com type = "button". Tudo o resto deve ser especificado no HTML, incluindo o rótulo ou o texto que vai no botão. Desta forma, você tem muito mais controle sobre o que é colocado no botão. Em vez de enviar consulta como com o elemento <input>, o texto pode ser definido como submeter pesquisa ou salvar dados. O HTML a seguir mostra o texto nos botões e a figura 3-15 mostra a saída:

```
<button type="button">Go Home</button>
<button type="reset" >Reset</button>
<button type="submit">Submit Survey</button>
```

FIGURA 3-15 Os elementos de botão com texto especificado

Você pode levar o elemento botão ainda mais. O conteúdo do elemento não precisa ser apenas texto simples. Você pode incorporar imagens dentro do elemento usando o elemento além do texto ou incorporar um parágrafo clicável inteiro. Você também pode aplicar folhas de estilo em cascata (CSS) ao botão para alterar sua aparência, como mostrado na Figura 3-16. O HTML é como segue:

```
<button type="button" style="border-radius: 15px;">
  <p>Something exciting lies behind this button</p>
  
</button>
```

FIGURA 3-16 Um elemento de botão personalizado

Dentro do elemento botão reside a capacidade de criar um botão altamente personalizado e obter comportamento padrão do navegador.

Além do que é fornecido pelos vários tipos de entrada, como intervalo, email e url, outros atributos estão disponíveis e comuns na maioria dos controles de entrada e fornecem flexibilidade adicional em como os campos são validados. Isso é abordado a seguir.

Implementando atributos de conteúdo

Os controles de entrada fornecem atributos de conteúdo que permitem controlar o comportamento navegando declarativamente em vez de ter que escrever código JavaScript.

Fazendo controles somente leitura

Parte da especificação para os controles de entrada HTML inclui um atributo `readonly`. Se você quiser apresentar informações aos usuários em elementos como caixas de texto, mas não quiser que eles sejam capazes de alterar esses dados, use o atributo `readonly`. Quando `readonly` é especificado, o processador não permitirá que os usuários alterem qualquer um dos dados na caixa de texto. O HTML a seguir demonstra a propriedade `readonly`:

```
<tr>
<td>
Your Secret Code:
</td>
<td>
<input type="text" readonly value="00XY998BB"/>
</td>
</tr>
```

Neste código, na parte superior do formulário de pesquisa, os usuários recebem um código secreto para corresponder à sua pesquisa. Eles não podem alterar isso porque o atributo `readonly` é especificado. Onde os campos não são somente leitura e os usuários podem digitar o que quiserem na caixa de texto, fornecendo-lhes a capacidade de verificar a ortografia é uma boa idéia.

Fornecer um verificador ortográfico

Verificar a ortografia é outro método disponível para validar a entrada do usuário. O atributo `spellcheck` ajuda a fornecer feedback aos usuários de que uma palavra inserida está incorreta. Novamente, esse atributo é aplicado ao elemento de entrada:

```
<textarea id="otherCommentsText" rows="5" cols="20"
spellcheck="true"></textarea>
```

Neste HTML, a opção de verificação ortográfica foi ativada para a área de texto Outros Comentários porque os usuários podem digitar o que quiserem e podem causar erros ortográficos.

A saída de uma caixa de texto com verificação ortográfica não é diferente até que um usuário comece a digitar e digite um erro ortográfico. A Figura 3-17

mostra o sublinhado vermelho para as palavras que são detectadas como soletrado incorretamente.

FIGURA 3-17 Uma área de texto com verificação ortográfica

Em alguns casos, a validação interna fornecida pelos controles de entrada não é suficiente, e fornecer um padrão personalizado para validar é melhor, conforme explicado na próxima seção.

Especificação de um padrão

Como você viu com os tipos de entrada de e-mail e URL, a validação interna é bastante cuidadosa para garantir que as informações inseridas são precisas e esperadas. No entanto, em alguns casos, você pode exigir uma validação mais flexível ou mais rigorosa. Suponha que você não deseja que os usuários tenham que especificar o protocolo HTTP em um tipo de url, mas você deseja permitir apenas sites .com ou .ca. Isso pode ser conseguido usando o atributo **pattern**, que permite o uso de uma expressão regular para definir o padrão que deve ser aceito.

Dica de exame

O atributo **pattern** aplica-se apenas a caixas de texto. Ele não pode ser usado para substituir a validação incorporada nos tipos de email ou url.

O código a seguir mostra o atributo de padrão usado para alcançar a validação desejada:

```
<input type="text" title="Only .com and .ca are permitted."  
pattern="^[a-zA-Z0-9\-\.\.]+\.(com|ca)$"/>
```

Muitas expressões regulares estão disponíveis para validar um URL; Este é bastante simples. Ao especificar o atributo **pattern**, você deve especificar o atributo **title** também. O atributo **title** especifica a mensagem de erro aos usuários na dica de ferramenta quando a validação falhar. Para garantir que os usuários digitem os dados no formato correto, você deve mostrar-lhes uma amostra do que os dados devem ser parecidos. Isso é conseguido com o atributo **placeholder**.

Usando o atributo placeholder

O atributo **placeholder** permite que você solicite aos usuários o que é esperado em uma determinada caixa de texto. Por exemplo, uma caixa de texto de e-mail pode exibir texto me@mydomain.com. Mais importante ainda, este texto de espaço reservado não interfere com os usuários quando eles começam a digitar suas informações na caixa de texto. O atributo **placeholder** consegue isto, como mostrado na HTML a seguir e na saída subsequente na Figura 3-18.

```
<input type="email" placeholder="me@mydomain.com" /></td>
```

FIGURA 3-18 O atributo do espaço reservado demonstrando aos usuários o que é esperado

O **placeholder** é ligeiramente mais claro na cor. Assim que um usuário coloca o cursor do mouse na caixa para digitar, o texto de espaço reservado desaparece e a digitação do usuário assume. Os campos HTML podem ser validados de muitas maneiras. Em alguns casos, não é tanto o que é colocado no campo, mas que o campo é realmente preenchido. O atributo necessário controla isso para os elementos HTML.

Fazendo controles necessários

Para garantir que um usuário preencha um campo, use o atributo **required** com o elemento **<input>**. Isso garante que os usuários serão informados de que o campo é necessário. Neste exemplo, o endereço de e-mail será feito com uma caixa de texto obrigatória:

```
<input type = "email" placeholder = "me@mydomain.com" required />
```

Com o controle **required** especificado, se os usuários tentarem enviar o formulário sem especificar um endereço de e-mail, receberão uma mensagem de erro (consulte a Figura 3-19). Agora os usuários não podem enviar até que eles especifiquem um endereço de e-mail válido.

FIGURA 3-19 A validação de campo obrigatória invocada

As capacidades dos controles de entrada podem fornecer uma estrutura de validação bastante robusta.

No entanto, mais precisa ser feito para garantir que o site é seguro e protegido.

Experimento de pensamento

Criar formulários dinâmicos

Neste experimento mental, aplique o que aprendeu sobre esse objetivo. Você pode encontrar respostas para essas perguntas na seção "Respostas" no final deste capítulo.

Com base no experimento de pensamento do Objetivo 1.3, considere o que você faz para adicionar validação personalizada a um controle com base no que um usuário inseriu no campo anterior. Ter um formulário dinamicamente criado onde você pode alterar as regras de validação como um usuário progride através do formulário pode ser bastante poderoso. Considere diferentes variações culturais para regras sobre números de telefone, códigos postais e sufixos de endereços de e-mail. Descreva como você implementaria a validação para que você pudesse fornecer validação rica em contexto para o usuário.

Resumo do objetivo

■ Os controles de entrada, como **text** e **textarea**, permitem aos usuários digitar informações em uma página da web.

■ Alguns controles de entrada fornecem validação interna, como para URLs e endereços de e-mail.

■ Botões de rádio e checkbox fornecem controles para que os usuários selecionem itens em uma lista.

■ ■ Reset e submit os botões de controle do comportamento do formulário HTML.

■ ■ Os usuários não podem modificar o conteúdo de um controle que tenha o atributo readonly atribuído.

■ ■ Você pode adicionar um verificador ortográfico a uma caixa de texto para ajudar os usuários a evitar erros ortográficos.

■ ■ O atributo pattern ajuda a definir uma expressão regular para validação personalizada de

Dados formatados.

■ ■ O atributo required garante que um campo seja preenchido antes que os usuários possam enviar o formulário.

Revisão objetiva

1. Qual controle de entrada é mais adequado para permitir que os usuários façam seleções múltiplas?
 - A. radio button
 - B. textarea
 - C. checkbox**
 - D. radio or checkbox
2. Que controle de entrada foi projetado para permitir aos usuários inserir informações seguras que impede os outros de ver o que é digitado?
 - A. text
 - B. textarea
 - C. url
 - D. password**
3. Que entradas de controle de entrada formam dados para um servidor?
 - A. button
 - B. Submit**
 - C. Reset
 - D. radio
4. Qual das seguintes declarações são formas válidas de tornar um controle de texto não editável?

- A. `<input type="text" edit="false"/>`
- B. `<input type="text" editable="false"/>`
- C. `<input type="text" readonly="yes"/>`
- D.** `<input type="text" readonly/>`

5. Como você pode garantir que todos os campos necessários são preenchidos antes de um formulário pode ser submetido?

- A. Escreva uma função JavaScript para avaliar todos os controles no formulário para o conteúdo.
- B. No servidor, avalie todos os controles de dados e retorne uma página de erro para conteúdo ausente.
- C.** Adicione o atributo **required** em cada controle para que os usuários recebam uma mensagem campo é obrigatório.
- D. Adicione uma etiqueta à página para permitir que os usuários saibam quais controles devem preencher.

Objetivo 3.2: validar a entrada do usuário usando o JavaScript

Os novos controles HTML discutidos no Objetivo 3.1 fornecem uma grande funcionalidade para validar os dados do usuário. No entanto, esta funcionalidade tem algumas limitações. Isto é onde a validação realizada em JavaScript vem a calhar. O JavaScript fornece funcionalidade adicionais que não está prontamente disponível nos controles HTML principais. Embora alguns controles ainda não estejam disponíveis em todos os navegadores, talvez seja necessário validar a entrada do usuário, como datas, números de telefone ou códigos postais alfanuméricos. Este objetivo demonstra como usar **expressões regulares** para validar o formato de entrada e como usar as funções internas de JavaScript para garantir que os dados são o tipo de dados correto. Este objetivo também adiciona uma camada de segurança, demonstrando como evitar a injeção de código malicioso.

Este objetivo abrange como:

- Avaliar expressões regulares
- Validar dados com funções incorporadas
- Impedir a injeção de código

Avaliando expressões regulares

Você viu o uso de expressões regulares no Objetivo 3.1. De fato, os controles de entrada de HTML principais suportam o atributo `pattern` que permite aplicar uma **expressão regular** para validar a entrada do usuário. Em alguns casos, no entanto, validar a entrada do usuário em JavaScript pode ser mais eficaz do que inline com atributos. Esta seção apresenta expressões regulares.

A sintaxe básica de uma expressão regular é explicada, como é, e como usar a expressão em JavaScript.

Expressões regulares possuem uma sintaxe exclusiva. Eles podem ser assustadores de usar, mas também pode ser muito poderoso. Embora uma instrução completa sobre expressões regulares esteja além do escopo deste livro, uma breve introdução é fornecida para apoiar os exemplos posteriores.

Dica de exame

Expressões regulares tendem a fazer o seu caminho para os exames. Você deve se preparar estudando-os em mais detalhes. Uma pesquisa na Internet deve render muitos recursos disponíveis gratuitamente sobre o tema. Estar familiarizado com a leitura de uma expressão para coisas como endereços de e-mail, URLs e números de telefone, entre outras coisas.

Expressões regulares são uma mistura de caracteres especiais e caracteres literais que compõem o padrão que alguém gostaria de corresponder. A Tabela 3-1 lista os caracteres especiais e o seu significado.

TABELA 3-1 Caracteres especiais de expressão regular

Símbolo	Descrição
<code>^</code>	O caractere circunflexo indica o início de uma seqüência de caracteres.
<code>\$</code>	O sinal de dólar indica o final de uma string.
<code>.</code>	O período indica para corresponder em qualquer personagem.
<code>[A-Za-z]</code>	Letras alfabéticas indicam que correspondem a qualquer caractere alfabético. Isso diferencia maiúsculas de minúsculas. Para combinar letras minúsculas, use <code>[a-z]</code> .
<code>\d</code>	Essa combinação indica para coincidir com qualquer caractere numérico.

+	O sinal de mais indica que o carácter anterior ou conjunto de caracteres tem de corresponder pelo menos uma vez .
*	O asterisco indica que o caractere anterior ou conjunto de caracteres pode ou não corresponder . Isso gera zero ou mais correspondências.
[^]	Quando incluído em um conjunto de caracteres, o caracter denota uma negação . [^ A] corresponderia a uma string que não tem um 'a' nele.
?	O ponto de interrogação indica que o carácter anterior é opcional .
\w	Esta combinação indica que corresponde a um carácter de palavra que consiste em qualquer, Incluindo um sublinhado.
\	A barra invertida é um caractere de escape . Se qualquer caractere especial deve ser incluído no conjunto de caracteres para corresponder literalmente, ele precisa ser escapado com um \. Por exemplo, para encontrar uma barra invertida em uma string, o padrão inclui \\.
\s	Esta combinação indica a correspondência em um espaço . Quando combinado com + ou *, ele pode corresponder em um ou mais espaços.

Esta lista abrange as principais funções disponíveis quando a correspondência de expressões. Construir expressões regulares requer ter a

definição desses caracteres e, essencialmente, criar uma máscara fora deles para ser usado pelo mecanismo de expressão regular para interpretar e decidir se há uma correspondência. Por exemplo, um código postal canadense é composto pelo formato A1A 1A1 - ou seja, alternando caracteres alfabéticos e caracteres numéricos com um espaço no meio. Alguns caracteres não são usados em códigos postais porque as máquinas os confundem com outros caracteres (por exemplo, Z e 2). Além disso, o espaço não é obrigatório. Quando você precisa reforçar o formato de dados da entrada do usuário, decidir como você deseja que os dados sejam capturados e quão flexível você quer que seja é importante. Em seguida, crie sua expressão regular para corresponder a isso. Agora, crie a expressão regular para um código postal. Primeiro você precisa denotar o início da sequência de caracteres, porque ele ajuda a eliminar espaço em branco desnecessário na liderança da sequência de caracteres: ^

A primeira parte da expressão é o caractere ^. O próximo caractere deve ser alfabético: `^[A-Z,a-z]`

Como os códigos postais não são sensíveis a maiúsculas e minúsculas, a expressão permite que o primeiro caractere seja maiúsculo ou minúsculo. O próximo caractere no código postal deve ser um dígito: `^[A-Z,a-z]d`

Como o código postal aceita todos os dígitos 0-9, `d` é usado para especificar qualquer dígito. No entanto, `[0-9]` também poderia ter sido usado. E agora o padrão continua, letra-número-letra número-letra-número:

`^[A-Z,a-z]d[A-Z,a-z]d[A-Z,a-z]d`

Como foi indicado anteriormente, o espaço no meio do código postal, enquanto convenção, é opcional. Este é o lugar onde decidir o quão flexível a validação de dados deve ser é necessário. A expressão como ela é não permitirá qualquer espaço no meio porque a expressão é definida para corresponder em consecutivos alternando letter-number-letter. Talvez, para fins de formatação, um espaço deve ser exigido. Neste caso, `s` exigiria que um espaço seja incluído:

`^[A-Z,a-z]d[A-Z,a-z]s[A-Z,a-z]d`

Agora, os usuários seriam obrigados a digitar o código postal com um espaço no meio dos dois conjuntos de três caracteres. Mas talvez o site não se preocupa com o espaço no meio, porque ele realmente não afeta nada. Neste caso, o `s` pode ser denotado com o `*`:

`^[A-Z,a-z]d[A-Z,a-z]s*[A-Z,a-z]d`

Agora, a expressão permite alternar letra-número-letra e um ou mais espaços podem ocorrer no meio. O espaço agora é opcional, mas um problema foi introduzido. O usuário agora pode inserir qualquer número de espaços e ainda passar a validação, como:

A1A 1A1

Isso A1A 1A1 passaria a validação porque um ou mais espaços é exigido pelo `\s*`. O resultado desejado aqui é permitir apenas um espaço ou nenhum espaço. Para isso, um novo elemento é adicionado para limitar o número de ocorrências a apenas um. Isto é conseguido especificando o comprimento máximo permitido para o conjunto de caracteres que está sendo correspondido:

`^[A-Z,a-z]\d[A-Z,a-z]\s{1}\d[A-Z,a-z]\d`

O `{1}` diz que corresponde ao caracter anterior apenas o número de vezes especificado - neste caso, uma vez. Agora a expressão está de volta à funcionalidade que não é diferente do que apenas especificando o `\s`. O que é necessário em seguida é algo para tornar o espaço único opcional, como denotado com o `?`. Para atingir este efeito, o segmento espacial é envolvido em colchetes para torná-lo um conjunto e seguido pelo `?` Para torná-lo opcional:

`^[A-Z,a-z]\d[A-Z,a-z](\s{1})?\d[A-Z,a-z]\d`

Agora você tem uma expressão regular que requer o padrão alfanumérico correto para um código postal canadense com um espaço opcional no meio.

Este exemplo simples demonstra os elementos-chave para uma expressão regular. Embora esta expressão regular possa ser colocada no atributo `pattern` do elemento `<input>`, este próximo discute como usar a estrutura de JavaScript para executar a correspondência de padrões com expressões regulares.

Avaliando expressões regulares em JavaScript

Assim como com strings e inteiros, expressões regulares são objetos em JavaScript. Como tal, eles podem ser criados e podem fornecer métodos para avaliar strings. Objetos de expressão regular são criados de forma semelhante a strings; No entanto, em vez de usar "para encapsular a expressão, use a barra / <expressão> / em vez disso. O JavaScript sabe que o texto rodeado por barras

diagonais desta forma é um objeto de expressão regular. Voltando ao exemplo do código postal, o seguinte HTML é fornecido:

```
<script type="text/javascript">
function CheckString() {
try{
var s = $('#regExString').val();
var regExpression = /^[A-Z,a-z]\d[A-Z,a-z][\s{1}]?\d[A-Z,a-z]\d/;
if (regExpression.test(s))
alert("Valid postal code.");
else
alert("Invalid postal code.");
} catch (e) {
alert(e.message);
}
}
</script>
<body>
<form>
<input type="text" id="regExString" />
<button onclick="CheckString();" >Evaluate</button>
</form>
</body>
```

Este HTML fornece uma página muito básica com uma caixa de texto e um botão. O botão não faz nada mais do que chamar uma função para validar se o texto digitado corresponde ao formato desejado para um código postal. Esta página não deve conter nada que você já não tenha visto, exceto a linha na qual o objeto de expressão regular é criado:

```
var regExpression = /^[A-Z,a-z]\d[A-Z,a-z][\s{1}]?\d[A-Z,a-z]\d/;
```

Com essa linha, um objeto de expressão regular é criado e, como resultado, métodos estão disponíveis.

A string é extraída da caixa de texto e passada para o método de teste da expressão. O método de `test` retorna um booleano para indicar se a sequência de entrada corresponde a expressão regular que foi criada.

O objeto de expressão regular também fornece um método chamado `exec`. Esse método retorna a parte da sequência de entrada que corresponde a expressão. O exemplo de código a seguir ilustra isso adicionando outro botão e função para usar o método `exec` em vez de `test`:

```
function CheckStringExec() {  
    var s = $('#regExString').val();  
    var regExpression = /^[A-Z,a-z]\d[A-Z,a-z][\s{1}]?\d[A-Z,a-z]\d/;  
    var results = regExpression.exec(s);  
    if(results != null)  
        alert("Valid postal code." + results[0]);  
    else  
        alert("Invalid postal code.");  
    ...  
    <button onclick="CheckStringExec();" >Evaluate with Exec</button>
```

Com este botão, a expressão é avaliada como era com o método de teste, exceto que a correspondência é retornada como uma matriz de sequência de caracteres. Que o resultado de retorno é uma matriz de sequência de caracteres é importante observar porque usar expressões regulares pode resultar em múltiplas partidas. Se uma correspondência não for feita, o resultado do retorno será nulo. Neste exemplo, os resultados são avaliados verificando se a matriz não é nula; Se não for, o código postal é válido e mostrado de volta para o usuário. Se a correspondência não for feita, o valor de retorno será nulo.

O objeto string também fornece métodos de expressão regular. A string poderia ser usada diretamente para avaliar a expressão. A sequência de caracteres fornece os métodos de `search` e `match`. O método de `search` retorna o índice do caractere na sequência onde ocorreu a primeira correspondência. O método `match` retorna a parte da sequência que corresponde ao padrão, bem como o método `exec`. Além desses dois métodos, muitos dos outros métodos de sequência de caracteres aceitam um objeto de expressão regular, como `indexOf`, `split` e `replace`. Isso fornece algumas funcionalidades avançadas para manipulação de strings em JavaScript.

Dica de exame

O exemplo usa uma expressão regular para validar a entrada do usuário de dados página da web. Tenha em mente que os dados podem vir de qualquer lugar, como um feed RSS ou servidor back-end que fornece JavaScript Object Notation (JSON). Neste contexto, quando um site está esperando dados especificamente formatados, você pode usar expressões regulares para validar os dados de entrada e evitar o possível travamento do site ou, pelo menos, erros sendo apresentados aos usuários.

Embora expressões regulares fornecem um grande poder na avaliação de strings para padrões e garantindo que os dados estão no formato desejado, o JavaScript também fornece funções internas para avaliar o tipo de dados recebidos.

Validação de dados com funções incorporadas

O JavaScript fornece funções internas para avaliar o tipo de dados. Algumas funções são fornecidas diretamente dentro do JavaScript; Outras são fornecidos pela biblioteca jQuery.

A função `isNaN` fornece uma maneira de avaliar se o valor passado para ele não é um número. Se o valor não for um número, a função retornará `true`; Se for um número, ele retorna `false`.

Se a forma esperada de dados que está sendo avaliada é numérica, esta função fornece uma maneira defensiva para determinar isso e manipulá-lo apropriadamente:

```
if (isNaN(value)) {  
    //handle the non number value  
}  
else {  
    //proceed with the number value
```

O oposto da função `isNaN` é a função `isFinite`. A função `isFinite` é usada da mesma maneira, mas retorna `true` se o valor for um número finito e `false` se não for. Ser capaz de validar dados é muito importante como descrito anteriormente. Igualmente importante para validar os dados explicitamente é garantir que os campos de entrada de dados evitam que os usuários injetem

scripts. Injeção de código é um tema amplamente discutido na segurança do site. A próxima seção discute a prevenção de injeção de código.

Prevenção da injeção de código

Injeção de código é uma técnica que os invasores usam para injetar código JavaScript em sua página da web. Normalmente, esses ataques usam conteúdo criado dinamicamente para Script executar para que os usuários mal-intencionados podem tentar obter algum tipo de controle sobre o site.

As intenções podem ser muitas, mas entre essas intenções pode ser enganar outros usuários do site com Informações confidenciais. Dependendo do conteúdo da página, diferentes medidas precisam ser consideradas.

Proteção contra entrada do usuário

Um aplicativo da Web que aceita a entrada do usuário abre uma superfície de ataque potencial para Usuários. O tamanho da superfície de ataque depende do que é feito com os dados inseridos. Se o Site leva dados e não faz nada com ele fora do âmbito da página atual, como enviá-lo para outro servidor ou armazená-lo em um banco de dados, os efeitos são limitados ao página e sessão do navegador. Pouco pode ser feito exceto para interromper o projeto da página.

Para este usuário em particular. No entanto, se os dados capturados incluem uma criação de conta por exemplo, um usuário mal-intencionado tem muito mais potencial para causar danos – especialmente quando essa informação é mais tarde processada para a página web dinamicamente. Isto permite inerentemente qualquer pessoa para adicionar script ao site, que pode abrir o site para o comportamento, como **phishing**. Como um desenvolvedor de páginas da Web, você precisa garantir que todas as entradas do usuário sejam depuradas dos elementos de script. Para por exemplo, não permita que **<>** texto seja inserido no formulário. Sem esses caracteres, um bloco de script não pode ser adicionado.

Usando a função eval

A função eval é usada para executar JavaScript dinamicamente. Ele toma uma seqüência de caracteres como um parâmetro e executa-lo como uma função JavaScript. Nunca use a função eval contra quaisquer dados fornecidos por uma fonte externa sobre a qual você não tem 100% de controle.

Usando iFrames

Os iFrames abrem uma nova oportunidade para os atacantes. Os motores de busca fornecem uma plethora de resultados lidando com explorações sobre o uso de iFrames. O atributo `sandbox` sempre deve ser usado para restringir quais dados podem ser colocados em um iFrame. O atributo `sandbox` tem quatro valores possíveis, conforme listado na Tabela 3-2.

TABELA 3-2 Valores de atributo do `sandbox` disponíveis

Valor	Descrição
""	Uma string vazia aplica todas as restrições. Este é o mais seguro.
allow-same-origin	O conteúdo do iFrame é tratado como sendo da mesma origem que o conteúdo do Documento HTML.
allow-top-navigation	O conteúdo do iFrame pode carregar conteúdo do documento HTML que o contém.
allow-forms	O iFrame pode enviar formulários.
allow-scripts	O iFrame pode executar o script.

Experimento de pensamento

Codificação de dados de entrada

Neste experimento mental, aplique o que aprendeu sobre esse objetivo. Você pode encontrar respostas para essas perguntas na seção "Respostas" no final deste capítulo.

A principal maneira pela qual os usuários mal-intencionados buscam vulnerabilidades nas suas páginas da Web é através do uso de injeções de código. Estes são usados para encontrar falhas no código onde os usuários mal-intencionados poderiam enganar os usuários legítimos para redirecioná-los para um site ou pior-roubar dados privados. Que estratégias adicionais você pode projetar em suas páginas da Web para ajudar a prevenir esses tipos de ataques?

Resumo do objetivo

■ ■ Expressões regulares são strings de caracteres especiais que um interpretador entende e usa para validar o formato de texto.

■ ■ Expressões regulares são objetos em JavaScript que fornecem métodos para testar dados de entrada.

■ ■ `isNaN` é uma função incorporada para determinar se um valor não é um número, enquanto `isFinite` valida se o valor é um número finito.

■ ■ Injeção de código é uma técnica que os invasores usam para injetar códigos maliciosos aplicação.

■ ■ Os iFrames e o JavaScript dinâmico são perigosos se não forem usados corretamente em uma página da Web.

Revisão objetiva

1. Qual dos seguintes caracteres de expressão regular denotar o final da sequência de caracteres?

A. \$

B. %

C. ^

D. &

2. Qual dos seguintes atributos do Sandbox permite que o iFrame carregue o conteúdo do documento HTML que o contém?

A. allow-script-execution

B. allow-same-origin

C. allow-forms

D. allow-top-navigation

E. allow-top-document

3. Qual função nunca deve ser usada para executar JavaScript?

A. execute

B. JSDynamic

C. eval

D. evaluate

Objectivo 3.3: consumir dados

Este objetivo abrange como consumir dados em um aplicativo da Web HTML5. A capacidade de consumir dados de fontes externas é mais popular do que nunca. Os mash-ups do site e a integração social são os principais catalisadores para isso.

Este objetivo abrange como:

- Consumir dados JSON e XML usando serviços da Web
- Use o objeto `XMLHttpRequest`

Consumindo dados JSON e XML usando serviços da Web

Os dois formatos de dados comumente utilizados na transmissão de dados são `JSON` e `XML`. `JSON` é dados não estruturados, enquanto `XML` é estruturado. `JSON` usa uma sintaxe especial que permite a definição de pares de valor de nome em um formato de sequência leve. O `XML`, como um parente do `HTML`, é mais estruturado que o `JSON` com tags nomeadas e tags de abertura e fechamento. As tags podem ter atributos. Os seguintes são exemplos do que um objeto pessoa pode parecer em ambos os formatos em que o objeto pessoa tem um primeiro nome, sobrenome, cor de cabelo e cor dos olhos:

■ JSON:

```
{firstName: "Rick", lastName: "Delorme", hairColor: "brown", eyeColor: "brown" }
```

■ XML (Elements):

```
<Person>
  <firstName>Rick</firstName>
  <lastName>Delorme</lastName>
  <hairColor>Brown</hairColor>
  <eyeColor>Brown</eyeColor>
</Person>
```

■ XML (attributes):

```
<Person    firstname="Rick"    lastName="Delorme"    hairColor="Brown"
eyeColor="Brown"/>
```

Ao publicar serviços de dados, como Serviços da Web ou uma API REST, você pode controlar como você publica os dados. Ao consumir recursos de terceiros, você não terá controle sobre como eles publicaram os dados.

Usando o objeto XMLHttpRequest

O JavaScript fornece suporte interno para receber dados HTML através do objeto XMLHttpRequest. O objeto faz uma chamada para um serviço da Web, REST API ou outros serviços do provedor de dados. A vantagem de fazer isso via JavaScript no lado do cliente é ser capaz de recarregar partes da página de uma fonte externa sem ter que postar a página inteira de volta para o servidor.

XMLHttpRequest faz uma solicitação HTTP e espera receber dados de volta em formato XML. As chamadas síncronas e assíncronas são suportadas. Tabela 3-3, Tabela 3-4 e Tabela 3-5 listam os eventos, métodos e propriedades disponíveis do objeto XMLHttpRequest.

TABELA 3-3 Eventos disponíveis do objeto XMLHttpRequest

Evento	Descrição
Onreadystatechange	Define um manipulador de eventos para quando o estado da solicitação foi alterado. Usado para chamadas assíncronas.
Ontimeout	Define um manipulador de eventos para quando a solicitação não pode ser concluída.

TABELA 3-4 Métodos disponíveis do objeto XMLHttpRequest

Método	Descrição
Abort	Cancela o pedido atual
getAllResponseHeaders	Fornece uma lista completa de cabeçalhos de resposta
getResponseHeader	Retorna o cabeçalho de resposta específico
Send	Faz o pedido HTTP e recebe a resposta
setRequestHeader	Adiciona um cabeçalho HTTP personalizado ao pedido
Open	Define propriedades para a solicitação, como URL, nome de usuário e senha

TABELA 3-5 Propriedades disponíveis do objeto XMLHttpRequest

Propriedade	Descrição
readyState	Obtém o estado atual do objeto
Response	Obtém a resposta retornada do servidor
responseBody	Obtém o corpo da resposta como uma matriz de bytes
responseText	Obtém o corpo da resposta como uma sequência de caracteres
responseType	Obtém o tipo de dados associado à resposta, como blob, texto, arraybuffer, ou documento
responseXML	Obtém o corpo da resposta como um objeto XML DOM
Status	Obtém o código de status HTTP da solicitação

statusText	Obtém o texto HTTP amigável que corresponde ao status
Timeout	Define o limite de tempo limite no pedido
withCredentials	Especifica se a solicitação deve incluir credenciais de usuário

Em sua forma mais simples, uma solicitação para o servidor usando o objeto XMLHttpRequest se parece com isso:

```
<script>
$("document").ready(function () {
$("#btnGetXMLData").click(function () {
var xReq = new XMLHttpRequest();
xReq.open("GET", "myXMLData.xml", false);
xReq.send(null);
$("#results").text(xReq.response);
});
});
</script>
```

Este script assume um botão no formulário HTML e um div para mostrar os resultados. Um novo XMLHttpRequest objeto é criado. O método **open** é chamado para especificar o tipo de solicitação, **URI** e se a chamada deve ser assíncrona. A Tabela 3-6 lista todos os parâmetros para o método **open**.

TABELA 3-6 Parâmetros para o método **open** XMLHttpRequest

Parâmetro	Descrição
Method	O método HTTP que está sendo usado para a solicitação: GET , POST , etc.
url	URL para fazer o pedido para.
Async	Um valor booleano para indicar se a chamada deve ser feita de forma assíncrona. Se true , um

	manipulador de eventos precisa ser definido para o <code>onreadystatechange</code> .
User name	Um nome de usuário se o destino exigir credenciais.
Password	Uma senha se o destino exigir credenciais.

Dica de exame

O método `open` não faz nenhuma solicitação de servidor. Se o nome de usuário e a senha for especificado, ele não enviará essas informações para o servidor no método `open`. Quando o método `send` é chamado, o nome de usuário e senha também não são passados para o servidor. As credenciais são passadas para o servidor apenas em resposta a uma resposta de segurança 401 do servidor.

O objeto `XMLHttpRequest` fornece alguns mecanismos para tratar erros. O erro mais comum para conta é um erro de tempo limite. Por padrão, o valor do tempo limite é zero, que é infinito. Um valor de tempo limite deve sempre ser especificado. O código é atualizado da seguinte maneira:

```
var xReq = new XMLHttpRequest();
xReq.open("GET", "myXMLData.xml", false);
xReq.timeout = 2000;
xReq.ontimeout = function () {
    $("#results").text("Request Timed out");
}
xReq.send(null);
$("#results").text(xReq.response);
```

Isso resulta em não permitir que a chamada demore mais de dois segundos. O tempo limite é expressa em milissegundos. Após o período de tempo limite, o manipulador de eventos `ontimeout` é chamado para permitir que essa condição seja tratada adequadamente na página da Web.

Uma consideração adicional para este código é se a chamada deve ser sincronizada ou de forma assíncrona. Idealmente, você deve garantir que a chamada para o serviço para obter os dados não irá interferir com os usuários e não irá bloqueá-los, a menos que, naturalmente, eles precisam esperar na resposta antes de tomar qualquer outra ação. Chamadas síncronas, como os exemplos até agora têm mostrado, bloquear a interface do usuário enquanto o pedido está sendo feito. Para evitar isso, a chamada deve ser assíncrona, como mostrado aqui:

```
var XMLHTTPReadyState_COMPLETE = 4;
var xReq = new XMLHttpRequest();
xReq.open("GET", "myXMLData.xml", true);
xReq.timeout = 2000;
xReq.ontimeout = function () {
    $("#results").text("Request Timed out");
}
xReq.onreadystatechange = function (e) {
    if (xReq.readyState == XMLHTTPReadyState_COMPLETE) {
        if (xReq.status = "200") {
            $("#results").text(xReq.response);
        } else {
            $("#results").text(xReq.statusText);
        }
    }
}
xReq.send(null);
```

O evento `onreadystatechange` recebe uma função a ser executada quando o estado do `XMLHttpRequest` objeto é alterado. Quando a solicitação é concluída, o `readyState` muda para concluir (`readyState == 4`). Neste momento, o estado de retorno HTTP pode ser avaliado para um valor de êxito como 200 e, em seguida, o processamento dos dados XML pode ocorrer.

O mesmo código que foi usado até agora para recuperar dados XML também pode ser usado para fazer uma solicitação de dados JSON. A seguinte atualização para o código mostra isso:

```
var XMLHTTPReadyState_COMPLETE = 4;
var xReq = new XMLHttpRequest();
xReq.open("GET", "myJSONData.json", true);
xReq.timeout = 2000;
xReq.ontimeout = function () {
    $("#results").text("Request Timed out");
}
xReq.onreadystatechange = function (e) {
    if (xReq.readyState == XMLHTTPReadyState_COMPLETE) {
        if (xReq.status = "200") {
            $("#results").text(xReq.response);
        } else {
            $("#results").text(xReq.statusText);
        }
    }
}
xReq.send(null);
```

A única diferença para este código é o nome da URL que está sendo passada. Nesse caso, o nó de extremidade é uma fonte de dados que retorna dados formatados em JSON em vez de XML. O JSON é exibido para a tela da mesma maneira que o XML é exibido.

Quando os dados são recebidos através do objeto `XMLHttpRequest`, os dados precisarão ser desserializado em um formato mais amigável. Você também pode querer enviar dados para o servidor em resposta às ações do usuário. O próximo objetivo examina esses conceitos.

Experimento de pensamento

Criando uma página da Web com um ticker de ações nesta experiência de pensamento, aplique o que aprendeu sobre esse objetivo. Você pode encontrar respostas para essas perguntas na seção

"Respostas" no final deste capítulo. Você é encarregado de construir uma página da web para o seu cliente que envolve um ticker de ações. Você precisa fornecer cotações de ações em tempo real para os usuários da página em um rolo na parte superior da página. Explicar como você iria construir um aplicativo da web que fará isso dinamicamente sem postar de volta a página inteira.

Resumo do objetivo

■ ■ **JSON** e **XML** são os formatos mais comuns usados para troca de dados.

■ ■ O JSON consiste em **pares nome / valor**.

■ ■ **XML** é um documento baseado em elementos **estruturados**.

■ ■ O JavaScript fornece **suporte interno** para receber dados através do objeto **XMLHttpRequest**.

Revisão objetivo

1. Qual das seguintes é uma string JSON válida?

- A. {firstName, Rick, lastname, Delorme, hairColor, brown, eyeColor, brown}
- B. {firstName: Rick; lastname: Delorme; hairColor: brown; eyeColor: brown}
- C. {firstName: "Rick"; lastname: "Delorme"; hairColor: "brown"; eyeColor: "brown"}
- D. {firstName: "Rick", lastname: "Delorme", hairColor: "brown", eyeColor: "brown"}**

2. Com o objeto XMLHttpRequest, qual das seguintes propriedades fornece a resposta em um formato legível humano?

- A. Response
- B. responseBody
- C. responseText**
- D. responseXML

3. Em que fase durante um XMLHttpRequest são credenciais de usuário enviadas para o servidor?

- A. Quando a conexão é aberta
- B. Quando o pedido é enviado
- C. Quando o estado pronto estiver completo
- D. Quando o servidor envia uma resposta de segurança solicitando as credenciais**

Objetivo 3.4: Serializar, desserializar e transmitir dados

Os dados podem ser recebidos e enviados de várias formas. No objectivo anterior, **JSON** e **XML** foram examinados especificamente. A ideia de apresentar dados JSON ou XML diretamente aos usuários não é ideal. Os usuários gostariam de receber os dados de uma forma mais utilizável ou legível e significativa. Para isso, você precisa ter os dados convertidos de uma sequência de caracteres XML ou JSON em outra coisa. **O conceito de converter os dados de uma forma para outra é chamado serialização ou desserialização.**

Com **serialização**, os dados são colocados em um formato para **transmissão**. Com a **desserialização**, os dados transmitidos são **convertidos em algo que pode ser trabalhado, como um objeto personalizado**. Além de trabalhar com dados de sequência de caracteres, os aplicativos podem trabalhar com dados binários. Um aplicativo pode capturar desenhos ou imagens em uma tela e enviar esses dados de volta para o servidor. Os dados precisam ser serializados em um fluxo binário para conseguir isso. Este objetivo analisa a **serialização, desserialização e transmissão de dados binários e de texto**. A capacidade de enviar dados via Formulário HTML e enviar dados com o objeto XMLHttpRequest também é revisada.

Este objetivo abrange como:

- Enviar dados usando **XMLHttpRequest**
- Serializar e desserializar dados JSON
- Serializar e desserializar dados binários

Enviando dados usando XMLHttpRequest

O envio de dados para o servidor é semelhante ao recebimento de dados. Os exemplos de código no objetivo anterior usaram o objeto

XMLHttpRequest para receber dados. O objeto XMLHttpRequest em si é agnóstico para enviar ou receber. Ele pode realizar ambas as tarefas com base no modo como o objeto é configurado. Para enviar dados, o método send deve ter dados passados para ele, e esses dados podem ser transmitido para o ponto de extremidade especificado no URL do método open. O código a seguir envia os dados XML para o servidor:

```
var xmlData = "<Person firstname='Rick' lastName='Delorme'
hairColor='Brown' eyeColor='Brown' /> ";
var xReq = new XMLHttpRequest();
xReq.open("POST", "/ReceiveXMLData.aspx", false);
xReq.responseType
xReq.send(xmlData);
```

Quando os dados são transmitidos para o servidor, ele precisa ser serializado em um formato que o URL pode entender. Se o ponto de extremidade está esperando XML, os dados devem ser XML; Se ele está esperando dados binários, os dados devem estar em um formato binário.

Serialização e desserialização de dados JSON

O navegador oferece suporte nativo para trabalhar com JSON e XML. O objeto JSON está disponível para converter uma seqüência de caracteres JSON para um objeto (serializar / desserializar). O código a seguir mostra como isso é realizado:

```
var person = {
    FirstName: "Rick",
    HairColor: "Brown"
};
var jsonPerson = JSON.stringify(person);
```

O objeto de pessoa foi serializado em uma seqüência de caracteres JSON que pode ser enviada para um URL de nó de extremidade para processamento. Para retornar a pessoa de volta a um objeto de pessoa

de uma sequência de caracteres JSON, o objeto pode ser desserializado usando o método de análise:

```
var req = new XMLHttpRequest();
req.open("GET", "MyJsonData.json", false);
req.send(null);
var jsonPerson = JSON.parse(req.responseText);
```

Quando este código é executado, o objecto de pessoa é reconstruído a partir da cadeia JSON.

Serialização e desserialização de dados binários

A captura de dados de imagens dinâmicas segue um padrão semelhante ao das outras técnicas revisadas. A diferença chave agora é que a propriedade `responseType` deve ser definida como `blob`. O código a seguir demonstra como recuperar um objeto de imagem binário e desserializá-lo na página da Web:

```
var xReq = new XMLHttpRequest();
xReq.open("GET", "orange.jpg", false);
xReq.responseType = 'blob';
xReq.send(null);
var blob = xReq.response;
document.getElementById("result").src=URL.createObjectURL(blob);
```

A propriedade `responseType` do objeto `XMLHttpRequest` foi definida como `blob`. Então usando a propriedade de resposta para extrair os dados binários, o BLOB é passado para o método `URL.createObjectURL`. O método `createObjectURL` dá ao elemento `img` uma URL ligando ao BLOB ea imagem é exibida no navegador. Para o inverso, os dados também podem ser enviados para o servidor assim que ele é serializado em um BLOB:

```
var xReq = new XMLHttpRequest();
```

```
xReq.open("POST", "saveImage.aspx", false);  
xReq.responseType = 'blob';  
xReq.send(data);
```

Usando o método Form.Submit

O elemento de formulário de uma página HTML é a área do formulário que contém elementos que normalmente são controles de entrada para coletar informações de usuários. O elemento de formulário contém um atributo de ação que informa o formulário onde enviar seus dados. Submetendo os dados dessa forma envia a página HTML inteira de volta ao servidor para processamento. No entanto, outro mecanismo disponível é conectar-se ao evento de envio do formulário e lidar com o envio via JavaScript. Isso é útil para enviar os dados do formulário por meio de uma **solicitação AJAX** para que os usuários não precisem deixar a página atual enquanto a solicitação está sendo processada. O elemento de formulário no seu mais simples é o seguinte:

```
<form id="signupForm" action="processSignUp.aspx">  
</form>
```

O formulário neste caso publicará seus dados na página do servidor **processSignUp** para processamento, que por sua vez deve redirecionar os usuários para uma página de confirmação de algum tipo. A outra opção para lidar com a submissão do formulário é conectar o evento em JavaScript:

```
$("document").ready(function () {  
    $("form").submit(function () {  
        ./////  
    });  
});
```

Iterar sobre todos os elementos do formulário, capturar os dados deles e construir uma sequência de consulta para uso com uma chamada AJAX seria possível dentro do evento click. O código a seguir revisa esse conceito:

```
$("form").submit(function () {  
    var fName = $("#firstName").val();
```

```

var lName = $("#lastName").val();
var qString = "Last Name=" + lName + "&First Name=" + fName;
$.ajax({
  url: 'processSignUp.aspx',
  type: "POST",
  data: qString,
  success: function (r) {
  }
});
return false;
});

```

Nesse caso, o método `jQuery.serialize` manipula a extração dos dados de todos os elementos de entrada e cria a seqüência de consulta. A vantagem de usar este método - além de salvar um monte de código - é que a seqüência de consulta também é codificada.

Dica de exame

O método `serialize` requer que todos os elementos tenham o atributo de nome especificado. O código anterior funciona com o HTML modificado como tal:

```

<form id="signupForm">
  First Name:
  <input type="text" id="firstName" name="firstName"/><br/>
  Last Name:
  <input type="text" id="lastName" name="lastName"/><br/>
  <button type="submit">Submit</button>
</form>

```

O método `serialize` atua em qualquer resultado do seletor passado para o segmento `$()` do jQuery. No entanto, o método de serialização tem algumas limitações que você deve saber sobre. **Apenas controles bem-sucedidos são serializados - ou seja, apenas controles que estão em um estado válido.** Para controles de entrada, como caixas de seleção e botões de opção, apenas os que estão em um estado selecionado são considerados. Para os botões de opção, porque o atributo `name` deve ser

o mesmo para todos eles para serem considerados em um grupo de botões de opção, você especificaria o atributo de valor para diferenciá-los na sequência de consulta:

Experimento de pensamento

Salvando um formulário neste experimento mental, aplique o que aprendeu sobre esse objetivo. Você pode encontrar respostas para essas perguntas na seção "Respostas" no final deste capítulo. No Objectivo 3.1, foi construído um inquérito ao cliente. Estendendo este conceito, como você pode usar o objeto XMLHttpRequest para postar os dados capturados no formulário para o servidor? Antes de enviar o formulário, como você pode processar a validação do servidor em tempo real? Adicione validação ao formulário para que você possa comparar um endereço de e-mail inserido em um banco de dados de endereços de e-mail para garantir que ele não tenha sido usado antes.

Resumo do objetivo

- Os navegadores fornecem suporte nativo através do objeto JSON para trabalhar com deserialização de seqüências JSON.

- O método `JSON.parse` deserializa uma string JSON em um objeto e o método `JSON.stringify` serializa um objeto em uma seqüência de caracteres JSON.

- Ao definir a propriedade de tipo de resposta `XMLHttpRequest` como o valor `'blob'`, você pode recuperar dados binários.

- Por padrão, a ação de envio de formulário envia a página inteira ao servidor (com base no atributo de ação) para processamento.

- A manipulação do evento de envio permite que você personalize como os dados do formulário são lançados no servidor.

- O método `jQuery.serialize` fornece um atalho conveniente para converter controles de entrada especificados em uma seqüência de consulta.

Revisão objetiva

1.Qual das seguintes linhas de código é a forma correta de criar um objeto a partir de um JSON String armazenada em uma variável chamada jsonString?

- A. `var o = JSON.split(jsonString);`
- B. `var o = JSON.stringify(jsonString);`
- C.** `var o = JSON.parse(jsonString);`
- D. `var 0 = JSON.join(jsonString);`

2. Qual das seguintes linhas de código permite que um XMLHttpRequest retornar dados binários?

- A. `request.responseType = 'binary';`
- B. `request.responseType = 'image/jpg';`
- C. `response.type = 'blob';`
- D.** `request.responseType = 'blob';`

3. Como você controla o que é enviado ao servidor ao enviar um formulário?

- A. Adicionar um botão de envio ao formulário.
- B.** Gerencie o evento de envio do formulário.
- C. Especifique o atributo de ação do elemento de formulário.
- D. Certifique-se de que todos os elementos no formulário têm um nome.

CAPÍTULO 4

Usar CSS3 em aplicativos

O uso de folhas de estilo em cascata (CSS) não é um novo conceito. No entanto, as capacidades funcionais do CSS3 avançaram tremendamente. Neste capítulo, você analisa os recursos do CSS3 e como eles podem ser alavancados em suas aplicações web HTML5 para fornecer aos usuários finais a experiência desejada.

NOTA VARIANTES RESULTADOS

Você pode ver resultados ligeiramente diferentes em alguns dos exemplos de código. Nem todos os recursos funcionam em todos os navegadores. O Internet Explorer 11 eo Chrome v35 + foram usados para validar os exemplos ao longo deste capítulo.

Objectivos deste capítulo:

- ■ Objectivo 4.1: Propriedades de texto HTML de estilo
- ■ Objectivo 4.2: Propriedades da caixa HTML do estilo
- ■ Objectivo 4.3: Criar um layout de conteúdo flexível
- ■ Objectivo 4.4: Criar uma interface de usuário animada e adaptável
- ■ Objectivo 4.5: Localizar elementos usando seletores CSS e jQuery
- ■ Objectivo 4.6: Estruturar um arquivo CSS usando seletores CSS

Objectivo 4.1: Estilo propriedades de texto HTML

O texto é a base para todas as aplicações web. Alguns aplicativos da Web têm mais texto do que outros. Em última análise, o espectador do site irá ler o conteúdo do site que está no texto. CSS fornece muitos recursos para personalizar a aparência do texto, a fim de fornecer o seu próprio olhar único e sentir a sua aplicação web. Ao longo deste objectivo, irá explorar as várias formas de estilo de texto.

Este objetivo abrange como:

- Aplicar estilos à aparência do texto
- ■ Aplicar estilos à fonte de texto
- ■ Aplicar estilos ao alinhamento de texto, espaçamento e recuo
- ■ Aplicar estilos à hifenização de texto
- Aplicar estilos para uma sombra de texto

Aplicar estilos à aparência do texto

A aplicação de estilos como **cor**, **negrito** ou **itálico** torna certo texto destacado em uma página da web. Qualquer uma destas técnicas de denominação pode ser aplicada em combinação ou individualmente.

Aplicando cor ao texto

A cor do texto pode ser alterada, especificando a propriedade de cor para o elemento CSS. CSS3 aceita o valor da cor em qualquer um dos três métodos a seguir.

■ ■ **Valor hexadecimal:** especifique a cor como um valor hexadecimal, por exemplo como #00FF00 para exibir o texto em verde. Os dois primeiros dígitos (base 16) são o valor para Vermelho, os dois segundos dígitos são o valor para Verde, e os dois últimos dígitos são o valor para Azul. Isto é comumente referido como o código RGB.

■ ■ **Nome da cor:** Use uma palavra para especificar o valor da cor, como verde para exibir o verde do texto.

■ ■ **Função RGB:** Especifique o valor da cor usando a função RGB, que leva três valores, representando cada um um valor de bit de espectro de cor de 0 a 255. Por exemplo,

Rgb (0,255,0) especifica o verde como a cor do texto. O código a seguir demonstra cada método em uso.

```
<style>
h1{
color:#00FF00;
}
h2 {
color: green;
}
h3{
color: rgb(0,255,0);
}
</style>
```

A saída na Figura 4-1 mostra o resultado da aplicação desses estilos a uma página.

FIGURA 4-1 Especificação da cor do texto usando CSS

Com este código, cada um dos cabeçalhos h1 a h3 tem um estilo CSS aplicado para alterar a cor para verde, cada um usando um método

diferente. Usando o método hexadecimal ou RGB dá-lhe mais controle granular sobre a cor, em seguida, usando o nome da cor como foi feito no elemento <h2>. Há apenas muito mais cores que existem, então poderia ser nomeado. Os métodos hexadecimal e RGB permitem que você acesse toda a gama de cores.

OBSERVAÇÃO STYLES PADRÃO

Na Figura 4-1, você vê que mesmo que os três itens de texto estejam definidos como verde puro, um deles é mais escuro que os outros. É importante saber que alguns elementos, como os vários elementos de cabeçalho, têm estilos padrão já aplicados a eles. Nesse caso, o estilo para o elemento padrão é um tipo mais ousado.

Aplicando negrito ao texto

CSS também fornece acesso a outras propriedades da exibição de texto via o objeto `font`. O objeto `font` fornece a capacidade de tornar o texto em **negrito** ou **itálico**. O código a seguir demonstra a alteração do texto em negrito para todos os elementos <p>:

```
p {  
    font-weight: bold;  
}
```

Os estilos acima produzem a saída mostrada na Figura 4-2.

FIGURA 4-2 Exibindo o estilo negrito aplicado ao texto

A propriedade `font-weight` CSS aceita os seguintes valores para especificar como negrito você gostaria que o texto fosse: **lighter, normal, bold, and bolder..** Além disso, os valores numéricos **100 (mais claro) a 900 (mais escuro) são suportados**. Os valores aumentam em 100, fornecendo nove valores no total para controlar o peso do texto.

Aplicar estilo itálico ao texto

Através do objeto font, você também pode tornar o texto específico em itálico. Isso é feito especificando o estilo de fonte para o texto. O código a seguir demonstra aplicar o estilo itálico a todos os elementos <p> na página da Web:

```
p{  
    font-style:italic;  
}
```

Aplicar o estilo itálico a um elemento de texto produz a saída mostrada na Figura 4-3.

FIGURA 4-3 O estilo itálico aplicado a um elemento de texto.

Aplicar estilos à fonte de texto

O objeto CSS da **font** contém outras propriedades para permitir que você controle como o texto é processado em suas páginas. Você pode alterar o **tipo de letra e controlar o tamanho do texto**. Você pode controlar o tipo de letra em algumas maneiras diferentes. O primeiro método é simplesmente confiar nos fonts que estão instalados no sistema de renderização da página web. Isso é conseguido usando a família de fontes como mostrado no código a seguir:

```
p{  
    font-family:Arial,'Times New Roman',Webdings;  
}
```

Este código CSS **processa as fontes em ordem da esquerda para a direita até encontrar um que está disponível no computador** cliente. Se o nome da fonte contiver espaços, ele deve estar contido entre aspas. **Se nenhuma das fontes especificadas estiverem disponíveis, o texto voltará à fonte padrão do navegador.** No exemplo anterior, o cliente procura primeiro a fonte Arial. Se isso não estiver instalado, ele então procura a fonte Times New Roman e assim por diante. Esta é uma abordagem simples, mas muitas pessoas preferem usar fontes que não estão disponíveis no sistema. Há muitas fontes personalizadas disponíveis na Internet para inclusão em suas aplicações web. Essas fontes são conhecidas como **WOFF (Web Open Font Format)**. Para usar essas

fontes em sua página da Web, você define uma família de fontes usando a palavra-chave especial `@ font-face`.

Dica de exame

Esteja ciente de que certos tipos de fontes funcionam em alguns navegadores, mas não em outros. É importante declarar cada tipo de fonte usando `@ font-face` para que o navegador tenha acesso ao que ele precisa.

O código a seguir define um `@ font-face` para uma página da Web e implementa-o para o `<p>` elementos da página:

```
@font-face {  
  font-family: "My Nicer Font";  
  src: url('fonts/my_woff_font.eot');  
  src: url('fonts/my_woff_font.woff');  
}  
  
p {  
  font-family: 'My Nicer Font';  
}
```

Primeiro, adicione a palavra-chave `@ font-face` à página e, em seguida, atribua um nome especificando a propriedade `font-family`. Em seguida, especifique onde a fonte pode ser carregada. Isso pode ser do servidor web local se você baixou as fontes para ele ou de uma fonte da Internet. Na declaração anterior, ambos os tipos eot e woff são especificados.

Depois de ter decidido sobre uma fonte para o seu aplicativo da web, você pode decidir redimensionar o texto para cenários diferentes como você vê o ajuste. É claro que os elementos `<h1>` a `<h6>` fornecem alguma formatação padrão, mas você não gostaria de usar esses elementos em um site para controlar o tamanho do texto. Em vez disso, você pode usar a propriedade `font-size`. A propriedade `font-size` aceita valores relativos que quando renderizados são relativos ao tamanho de texto padrão no navegador. Os valores disponíveis são: (xx-small, x-small, small, smaller, medium, larger, large, x-large, xx-large.) xx-pequeno, x-pequeno, pequeno, menor, médio, maior, grande, x-grande, xx-grande. O

código a seguir demonstra a configuração do tamanho da fonte para os elementos <p>:

```
p{  
    font-size: x-large;  
}
```

A aplicação do atributo **size** para a fonte resulta na renderização de texto como mostrado na Figura 4-4. A primeira linha de texto mostra o tamanho padrão enquanto o texto maior mostra o tamanho x-grande.

FIGURA 4-4 O tamanho do texto aumenta com o uso do atributo font-size

Aplicando estilos ao alinhamento de texto, espaçamento e recuo

CSS3 também pode ser usado para alterar alinhamento de texto, espaçamento e recuo. Isso fornece grande controle sobre o posicionamento de texto dentro de recipientes pai.

Alinhamento de texto

Para controlar o alinhamento de texto dentro de um contêiner, especifique o atributo de alinhamento de texto. O atributo **text-align** suporta os valores descritos na Tabela 4-1.

TABELA 4-1 Valores suportados para alinhamento de texto

Valor	Descrição
Right	Alinha o texto para o lado direito do container pai.
Left	Alinha o texto para o lado esquerdo do contêiner pai.
Center	Alinha o texto com o centro horizontal do container pai.
Justify	Estica o texto horizontalmente para preencher toda a largura do contêiner pai.

O exemplo de código a seguir demonstra o uso do atributo text-align e a Figura 4-5 exibe os resultados dentro dos limites de um elemento div definido.

```
p {  
    text-align: center;  
}
```

FIGURA 4-5 O atributo text-align usado para posicionar o texto no centro de um elemento div

Recuo de texto

O recuo de texto é configurado usando o atributo text-indent. O atributo text-indent aceita um valor inteiro para indicar quanto a recuar. O exemplo de código a seguir ilustra como recuar o texto a partir da borda esquerda do elemento div pai. A Figura 4-6 mostra os resultados desse código.

```
p {  
    text-indent: 50px;  
}
```

FIGURA 4-6 Texto recuado usando o atributo text-indent

Espaçamento de texto

Existem duas formas de controlar o espaçamento do texto. Isso pode ser feito especificando o espaçamento entre cada caractere no texto ou especificando o espaçamento entre cada palavra no texto. O seguinte código CSS demonstra ambos os exemplos. A Figura 4-7 mostra a saída desse código.

```
p {  
    letter-spacing: 8px; espaçamento entre as letras  
}  
  
p {  
    word-spacing: 8px; espaçamento entre as palavras  
}
```

FIGURA 4-7 Espaçamento entre letras e palavras usando CSS3

Aplicando estilos a hifenização de texto

Aplicar estilos a hifenização de texto permite que você controle como uma frase ou palavra quebra no final da linha. O atributo **híphen** pode ser especificado para controlar esse comportamento. Os valores disponíveis para o atributo hífen são definidos na Tabela 4-2.

TABELA 4-2 Valores disponíveis para o atributo hífen

Valor	- Descrição
None	- As palavras não irão quebrar com um hífen ea frase só quebrará em espaço em branco.
Auto	As palavras automáticas irão quebrar com base em um algoritmo predefinido.
Manual	- As palavras manuais irão quebrar com base nas dicas especificadas nas palavras indicando um espaço apropriado para o pausa. Isso é feito usando o ­ Notação.

O código a seguir demonstra usando o valor none, e os resultados são mostrados em Figura 4-8.

```
div {  
    hyphens: none;  
}
```

FIGURA 4-8 Os resultados de especificar **none** para o atributo hífen na Figura 4-8, o navegador não está hifenizando qualquer texto e está deixando-o transbordar além dos limites do recipiente. Para forçar hifenização, especifique **auto** para o valor do atributo hífen. A saída dessa alteração é mostrada na Figura 4-9.

FIGURA 4-9 Os resultados da especificação automática para o atributo hífen

Quando **auto** é especificado, o navegador calcula onde **hifenizar** as palavras com base em suas próprias regras para que o texto não vá fora dos limites do contêiner. A opção manual se comporta da mesma maneira, exceto que você pode especificar um conjunto de regras externas para usar e também fornecer dicas usando o ­ (Hífen suave).

Aplicando estilos para uma sombra de texto

Você encontrará informações sobre a aplicação de estilos para uma sombra de texto na próxima seção. A Microsoft usa a frase "sombra de texto suspenso", mas você notará que ela foi encurtada para "sombra de texto" neste livro.

Experimento de pensamento

Alterando estilos neste experimento de pensamento, aplique o que aprendeu sobre esse objetivo. Você pode encontrar uma resposta a esta pergunta na seção "Respostas" no final deste capítulo. CSS fornece a capacidade de fazer sites olhar grande. Em alguns casos, você pode querer alterar os estilos com base na entrada do usuário ou ações. Como você alteraria os estilos de sua página desta forma?

Resumo do objetivo

■ O CSS3 oferece a capacidade de modelar a aparência do texto das seguintes maneiras:

■ Alterar a cor com a propriedade de **color**

■ Alterando o texto em negrito com a propriedade **font-weight**

■ Alterar o texto para itálico com a propriedade **font-style**

■ Alterando o tipo de fonte com a propriedade **font-family**

■ Alterar o tamanho do texto com a propriedade **font-size**

■ O CSS3 fornece a capacidade de formatar o alinhamento do texto com a propriedade **text-align**.

■ CSS3 fornece a capacidade de alterar o recuo de texto com a propriedade **text-indent**.

■ CSS3 fornece a capacidade de **alterar o espaçamento entre as letras e o espaçamento entre as palavras** com as propriedades de espaçamento entre letras e espaçamento de palavras.

■ CSS3 permite que você controle como o texto hifeniza quando o texto precisa ser envolvido dentro dos limites de seu contêiner.

Revisão objetiva

Responda às seguintes perguntas para testar seu conhecimento da informação neste objetivo. Você pode encontrar as respostas a essas perguntas e explicações de por que cada opção de resposta está correta ou incorreta na seção "Respostas" no final deste capítulo.

1. Qual das seguintes CSS não mudaria a aparência do texto?

- A. font-style: italic;
- B. font-weight: heavy;**
- C. font: bolder 12px arial;
- D. color: green;

2. Qual dos seguintes alinha o texto com a largura total da caixa disponível?

- A. right
- B. full
- C. center
- D. justify**

3. Qual das opções a seguir é uma maneira de configurar a quantidade de espaço entre as palavras?

- A. word-margin
- B. letter-margin
- C. word-spacing**
- D. word-padding

Objectivo 4.2: Propriedades da caixa HTML de estilo

Cada elemento HTML tem propriedades de caixa. Estas são as propriedades que controlam como o elemento é espaçado na página e controlar a posição do conteúdo da caixa. Além, os efeitos gráficos podem ser aplicados à caixa de um elemento.

Este objetivo abrange como:

- Aplicar estilos para alterar atributos de aparência
- Aplicar estilos para alterar os efeitos gráficos

■■ Aplicar estilos para estabelecer e alterar a posição de um elemento

Aplicando estilos para alterar aparência dos atributos

Há uma variedade de maneiras de alterar a aparência de uma caixa como ela se aplica a um elemento HTML. Esta seção demonstra como alterar a aparência alterando os atributos relacionados a **tamanho**, **fronteira**, **esboço**, **preenchimento** e **margem**. (size, bordering, outlining, padding, and margin)

Alterando o tamanho

O tamanho de qualquer elemento é controlado por suas propriedades de **altura** e **largura** (**height** **width**). Estes podem ser qualquer objeto ou classe em CSS. **Por padrão, um objeto terá tamanho próprio para ser capaz de exibir seu conteúdo**. Assim, um elemento div com algum texto dentro dele terá uma largura e altura que é suficiente para exibir o texto. O tamanho pode ser alterado definindo um valor para a **largura e / ou altura**. A Largura e altura podem ser especificadas como uma medida **em pixels (px)** ou **centímetros (cm)** ou **pode ser especificado como uma porcentagem de seu elemento pai**. Por exemplo, o código a seguir definir a largura de uma tabela para ser 50 por cento de seu recipiente pai, que neste caso é apenas a janela.

```
table {  
    height: 50%;  
    width: 50%;  
}
```

Em outro exemplo, o tamanho de um elemento <div> pode ser definido como um valor específico, como neste caso:

```
div {  
    width: 200px;  
    height: 100px;  
}
```


Border

CSS fornece controle muito granular sobre os estilos das **bordas** de qualquer elemento HTML. O nome da propriedade CSS raiz para fazer isso é a propriedade **border**. Com o uso da propriedade de borda, você pode controlar o **estilo, espaçamento, cor e largura da borda**. A primeira coisa que você precisa definir na fronteira é o **estilo**. Isto trará essencialmente a beira à existência. Há uma variedade de estilos de borda que existe. Estes incluem, **borda sólida, borda tracejada, borda de linha pontilhada, borda de linha ranhurada** (**solid border, dashed border, dotted line border, grooved line border**) e assim por diante. A borda é definida especificando a propriedade **border-style**:

```
p {  
    border-style: solid;  
}
```

A cor da borda pode ser alterada especificando a propriedade **border-color**. Isto é demonstrado com o seguinte código:

```
p {  
    border-style: solid;  
    border-color: black;  
}
```

Todos os elementos <p> agora serão exibidos com uma borda preta sólida. Isso é demonstrado na Figura 4-10.

FIGURA 4-10 Um elemento <p> com propriedades de borda definidas

OBSERVAÇÃO CONFIGURAÇÃO DE ESTILO FRONTEIRO

Observe que **a borda precisa existir antes que ele possa ter quaisquer alterações visíveis feitas a ele**. É por isso que no exemplo acima, o estilo de borda é definido primeiro.

Espaçamento entre bordas é usado para definir a quantidade de espaço desejado entre elementos adjacentes. O seguinte CSS fornecerá espaçamento de borda de 250 px para todos os elementos de parágrafo na página:

```
p {  
    border-style: solid ;  
    border-color: black;  
    border-spacing: 250px;  
}
```

Outra propriedade da borda que pode ser controlada com CSS é a largura da borda. O Border-width é uma propriedade que é definida usando uma medida fixa, em pixels, por exemplo. O código a seguir define a largura da borda como 5 px:

```
p {  
    border-style: solid ;  
    border-color: black;  
    border-spacing: 250px;  
    border-width: 5px;  
}
```

A Figura 4-11 mostra a nova largura definida no elemento de parágrafo.

FIGURA 4-11 Uma largura mais espessa na borda do elemento de parágrafo definido pela propriedade **border-width**

Até agora, você definiu cada uma das propriedades da borda como linhas individuais na folha de estilos. A propriedade CSS de borda suporta uma técnica abreviada onde você pode especificar as propriedades de chave em uma única linha. O código a seguir demonstra esse conceito:

```
p {  
    border: 5px solid black;  
}
```

Nesse caso, a borda é definida como uma largura de 5 px com uma linha sólida ea cor preta.

Dica de exame

O elemento de borda suporta muitas variantes na sua capacidade de definir propriedades em uma única linha. Tome algum tempo para experimentar

com todas as combinações possíveis para que você será capaz de lê-los e identificá-los facilmente no exame.

Além de ser capaz de definir todas as propriedades discutidas até agora em uma única linha, o elemento `border` permite ainda mais controle granular. As propriedades discutidas também podem ser configuradas para diferentes valores específicos para cada lado da caixa. Por exemplo, o código a seguir produz a saída mostrada na Figura 4-12:

```
p {  
  border-top: 15px solid black;  
  border-left: 10px solid black;  
  border-right: 10px solid black;  
  border-bottom: 5px solid black;  
}
```

FIGURA 4-12 Propriedades de borda diferentes para cada lado da caixa

Preenchimento e margem

O preenchimento e a margem são métodos adicionais para criar espaço em torno de um elemento HTML. Em essência, existem três camadas em torno de um elemento HTML. Como você olha do interior para o exterior do elemento, há o `padding`, a `border` e a `margin`. Se você der uma olhada novamente na amostra na Figura 4-12, o texto é esmagado muito perto da borda. A largura da borda foi definida, mas o preenchimento (o espaço entre o texto e a borda) tem o valor padrão de 0 px. Para criar espaço entre o texto e a borda, você deve aumentar o preenchimento como mostrado no exemplo de código a seguir. Os resultados são mostrados na Figura 4-13.

```
p {  
  border-top: 15px solid black;  
  border-left: 10px solid black;  
  border-right: 10px solid black;  
  border-bottom: 5px solid black;  
  padding: 25px;  
}
```

FIGURA 4-13 O uso do preenchimento para criar espaço entre texto e uma borda

Neste código, o preenchimento é definido como 25 px. Com apenas um único valor especificado, presume-se que este valor é aplicado a todos os quatro lados da caixa. No entanto, o preenchimento pode ser especificado como valores diferentes para todos os quatro lados. O código acima é, na verdade, o mesmo que dizer:

```
padding: 25px 25px 25px 25px; ou
padding-top: 25px;
padding-bottom: 25px;
padding-left: 25px;
padding-right: 25px;
```

A próxima área onde você pode criar espaçamento para seus elementos HTML está na margem. A margem é o espaço entre a borda do elemento e os elementos circundantes. O navegador fornece uma margem padrão com base no elemento HTML que é usado. A Figura 4-14 **mostra a margem padrão para o elemento de parágrafo**. A Figura 4-15 mostra o efeito do aumento do tamanho da margem. A margem pode ser controlada exatamente da mesma maneira que o padding. Você pode especificar um único valor a ser aplicado igualmente a todos os quatro lados, especificar valores individuais em uma única linha ou especificar cada lado da caixa individualmente. O código a seguir demonstra o aumento da margem do elemento de parágrafo e os resultados são mostrados na Figura 4-15:

```
p {
border-top: 15px solid black;
border-left: 10px solid black;
border-right: 10px solid black;
border-bottom: 5px solid black;
padding-top: 25px;
padding-bottom: 25px;
padding-left: 25px;
padding-right: 25px;
margin: 40px;
```

```
}
```

FIGURA 4-14 O layout de dois parágrafos com suas margens padrão

FIGURA 4-15 O layout de dois parágrafos com margens ampliadas

Você pode ver na Figura 4-15 que não só a margem cria espaço entre os dois

Parágrafo, mas também criou espaço entre a parte superior da janela eo topo do primeiro parágrafo elemento.

Aplicando estilos para alterar efeitos gráficos

Há uma variedade de opções na aplicação de efeitos gráficos a uma caixa para fornecer uma exibição exclusiva para os usuários finais. Esta seção demonstra configuração de **transparência**, **opacidade**, uma **imagem de plano de fundo**, **gradientes**, **sombras e recorte**.

Aplicando transparência / opacidade

Definir a opacidade (também conhecido como transparência) de um elemento em CSS fornece a capacidade de tornar o elemento efetivamente ver através. O valor utilizado na definição da opacidade é uma razão de **0 a 1.0**. Uma configuração de **0** indica que o elemento é **totalmente transparente**, essencialmente **invisível**. Um valor de 1.0 indica que o elemento é totalmente **opaco**, **o padrão quando nenhum valor de opacidade é especificado**. O código a seguir define um nível de opacidade de 0,4 para um elemento de texto:

```
p {  
  opacity: 0.4;  
}
```

A Figura 4-16 demonstra a saída de aplicar esse efeito ao elemento de texto. A saída sem a opacidade especificada também é fornecida para uma comparação.

FIGURA 4-16 A utilização da propriedade de opacidade

Aplicando uma imagem de plano de fundo

Qualquer elemento HTML também pode conter uma **imagem de plano de fundo** (**background image**). Isso é conseguido especificando a propriedade **background-image**. A propriedade de plano de fundo em si tem muitas outras

opções para controlar seu tamanho e repetir padrão. No entanto, para simplesmente definir uma imagem de plano de fundo, você precisa apenas especificar a imagem para a propriedade `background-image`. O código a seguir demonstra o uso dessa propriedade para atribuir uma imagem de plano de fundo:

```
p {  
  background-image: url('orange.jpg');  
  color: white;  
}
```

A Figura 4-17 mostra a saída desse código. A cor do texto é alterada para branco para torná-lo mais visível sobre a imagem. A Tabela 4-3 explica mais opções disponíveis para formatar a imagem de plano de fundo.

FIGURA 4-17 Uma imagem de fundo em um elemento de texto

TABELA 4-3 Opções de configuração para a imagem de fundo

Property	Description
size	Altera as dimensões da imagem
repeat	Especifica se a imagem deve ser repetida / em mosaico através do espaço disponível da caixa
clip	Especifica se a imagem deve ser cortada em uma borda
position-x/position-y	Especifica a posição de origem da imagem dentro da caixa

Todas as propriedades são prefixadas com o token `background-`. O exemplo a seguir demonstra usando a propriedade `repeat` e a propriedade `size`. Especifica que a imagem deve ser menor e repetir. Este exemplo especifica a repetição contínua, horizontal e verticalmente. No entanto, você pode especificar `repeat-x` ou `repeat-y` para repetir apenas na direção especificada.

```
p {  
  background-image: url('orange.jpg');  
  background-size: 20px;  
  background-repeat: repeat;
```

```
width: 200px;
height: 200px;
text-align: center;
color: white;
}
```

Este código anterior produz a saída na Figura 4-18.

FIGURA 4-18 Uso das propriedades de tamanho e repetição para uma imagem de fundo.

Aplicando gradientes

Um efeito gradiente altera gradualmente a cor de um objeto de um espectro para outro. Existem dois tipos de efeitos de gradiente suportados. O primeiro é um gradiente linear onde a cor muda em uma linha através do objeto em qualquer direção. O outro gradiente é um gradiente radial onde a cor começa no centro e muda para as bordas externas. O gradiente pode ser aplicado ao plano de fundo do elemento da seguinte maneira:

```
background:linear-gradient(black,gray);
```

A função de gradiente linear leva alguns parâmetros. Os parâmetros são Tabela 4-4.

TABELA 4-4 Parâmetros para a função de gradiente linear

Parametro	Descrição
Direction	Especifique a direção do gradiente para a direita ou para a esquerda. Este parâmetro é opcional e o padrão quando em branco é um efeito de gradiente para cima / para

	baixo . Um efeito diagonal também pode ser aplicado especificando para a parte inferior direita ou para a esquerda inferior. Você também pode especificar um ângulo, como em 100deg.
Color stop...n	O segundo e os subsequentes parâmetros são a cor a começar, seguida pela <u>as</u> cores de transição conhecidas como as paradas de cor. Isso informa ao navegador qual a cor a começar e transição para o efeito de gradiente.

Aplicando um efeito de sombra

Os efeitos de sombra permitem aplicar uma sombra à caixa do elemento HTML ou ao texto. Existem duas propriedades CSS3 para controlar o efeito de sombra: **caixa-sombra** e **texto-sombra** (box-shadow and text-shadow.). O **box-shadow** controla o efeito de sombra em torno da caixa do elemento HTML. A propriedade text-shadow controla a sombra do texto.

A propriedade box-shadow suporta os parâmetros descritos na Tabela 4-5. Os dois primeiros

Parâmetros são necessários para criar o efeito de sombra. Os parâmetros de **desfocagem e propagação** são efeitos opcionais que podem ser aplicados à caixa-sombra.

TABELA 4-5 Parâmetros para a propriedade box-shadow

Parâmetro	Descrição	
H	shadow	Especifica a posição da sombra horizontal. O valor também pode

		ser um número negativo.
V	shadow	Especifica a posição da sombra vertical. O valor também pode ser um número negativo.
Blur	Especifica a distância do efeito de desfocagem. Esse parâmetro é opcional e o padrão é 0.	
Spread	Especifica o tamanho da sombra.	
Color	Especifica a cor da sombra.	
Inset	Especifica que a sombra deve estar dentro da caixa em vez de fora da caixa.	

Em sua forma mais simples, a propriedade **box-shadow** requer apenas que **h-shadow** e **v-shadow** sejam especificados. O código a seguir mostra uma sombra básica aplicada a um elemento div:

```
div{
  position: absolute;
  left: 50px;top: 50px;
  width: 100px;
  height: 100px;
  border: solid 1px black;
  box-shadow: 10px 10px;
}
```

O elemento `div` é renderizado com uma sombra, como mostrado na Figura 4-19.

FIGURA 4-19 Um efeito caixa-sombra simples

O efeito de sombra na Figura 4-19 é uma sombra de caixa sólida.

Para fornecer um efeito onde a sombra desaparece gradualmente, você precisará especificar o parâmetro de desfocagem. Ao adicionar o parâmetro de desfocagem, você pode criar o efeito mostrado na Figura 4-20. O código a seguir adiciona o parâmetro de desfocagem:

```
div{
  position: absolute;
  left: 50px;top: 50px;
  width: 100px;
  height: 100px;
  border: solid 1px black;
  box-shadow: 10px 10px 10px;
}
```

FIGURA 4-20 Um efeito caixa-sombra com a adição de um borrão

O próximo parâmetro que adiciona um efeito especial à sombra é o parâmetro `spread`. Este parâmetro especifica o tamanho da sombra. O código a seguir especifica um valor de propagação para aumentar o tamanho da sombra:

```
div{
  position: absolute;
  left: 50px;top: 50px;
  width: 100px;
  height: 100px;
  border: solid 1px black;
  box-shadow: 10px 10px 10px 20px;
}
```

Este código produz a saída na Figura 4-21.

FIGURA 4-21 Um efeito caixa-sombra com adição do parâmetro `spread` para aumentar o tamanho da sombra.

A Figura 4-21 mostra a saída do código anterior com a adição do `spread` parâmetro. O parâmetro `spread` especifica o tamanho da sombra. No contexto de objetos do mundo real e suas sombras, a propagação é como indicar o quão

próxima é uma fonte de luz de um objeto. Quanto mais próxima a fonte de luz, maior a sombra que ela produz. O parâmetro `spread` pode ser usado para obter esse tipo de efeito no elemento HTML. Na Figura 4-21, a sombra foi criada intencionalmente para ser maior que a própria caixa HTML. Isso foi feito para demonstrar um conceito adicional de efeitos de sombra. A sombra em si é uma caixa de tamanho normal. Neste caso, a sombra foi feita para ser maior do que a caixa original para que seja visível em torno de todos os quatro lados da caixa. Normalmente, para uma caixa onde o `spread` não é especificado para ser maior, a sombra é apenas visível nos dois eixos especificados pelos dois primeiros parâmetros. Isso ocorre porque a caixa de sombra é deslocada do centro do elemento HTML que está sendo sombreado. O resto da sombra ainda está atrás do elemento HTML sendo sombreado. Você pode demonstrar isso definindo a posição da sombra para ser completamente longe do elemento HTML. Isso é conseguido especificando a posição para ser um valor de número maior, em seguida, o tamanho do elemento HTML. O código a seguir demonstra isso:

```
div{  
position: absolute;  
left: 50px;top: 50px;  
width: 100px;  
height: 100px;  
border: solid 1px black;  
box-shadow: 100px 100px 10px;  
}
```

Neste código, a posição da sombra da caixa é definida como 100px ao longo do eixo horizontal e 100px ao longo do eixo vertical, que colocará a sombra no canto inferior direito do elemento `div`, como mostrado na Figura 4-22 .

FIGURA 4-22 A sombra inteira de um elemento exibido especificando a posição a ser maior do que o tamanho do elemento HTML.

Outro parâmetro para a caixa sombra é o parâmetro `inset`. Se omitido, este parâmetro cria a sombra na parte externa da caixa. Se a inserção for especificada, a sombra será criada no interior da caixa. Isto é demonstrado no seguinte código:

```
div{
position: absolute;
left: 50px;top: 50px;
width: 100px;
height: 100px;
border: solid 1px black;
box-shadow: 10px 10px inset;
}
```

Este código produz a saída mostrada na Figura 4-23. A sombra agora é exibida no interior da caixa em vez da parte externa da caixa.

FIGURA 4-23 A utilização do parâmetro inserir para colocar a sombra no interior da caixa

A propriedade de color aceita valores como um código hexadecimal, rgb() ou uma cor literal. Esta propriedade altera a cor da sombra. Uma vez que o livro é impresso em preto e branco, demonstrando o uso deste parâmetro não apresentará bem. Então, no seu próprio código, altere a cor da sombra para ver o efeito que ela tem - a cor deve mudar.

NOTA COMPREENDENDO OS PARÂMETROS H-SHADOW E V-SHADOW

Os parâmetros h-shadow e v-shadow podem aceitar valores negativos. Para colocar a sombra no lado esquerdo da caixa em vez da direita, especifique um valor negativo para a sombra h. Para colocar a sombra na parte superior da caixa em vez da inferior, especifique um valor negativo para o parâmetro v-shadow.

As sombras de texto podem ser criadas da mesma forma que as sombras de caixa. A propriedade CSS para aplicar uma sombra ao texto é chamada de sombra de texto. A propriedade text-shadow tem parâmetros semelhantes aos da propriedade box-shadow. Os parâmetros estão descritos na Tabela 4-6.

TABELA 4-6 Parâmetros de sombra de texto

Parâmetro - Descrição

H-shadow	- Especifica a posição da sombra ao longo do eixo horizontal. Esse valor aceita números.
V-shadow	- Especifica a posição da sombra ao longo do eixo vertical. Esse valor aceita números.
Blur	- Especifica a distância do efeito de desfocagem. Esse parâmetro é opcional e o padrão é 0.
Color	- Especifica a cor da sombra.

Todos os parâmetros de sombra de texto são familiares. Todos eles têm o mesmo efeito contrapartes no elemento text-shadow. O código a seguir demonstra a aplicação de um efeito sombra para texto em uma página HTML:

```
p {
  position: absolute;
  left: 250px;
  top: 250px;
  text-shadow: -10px -10px;
}
```

Neste exemplo, os números negativos são fornecidos aos parâmetros h-shadow e v-shadow para colocar a sombra no canto superior esquerdo do texto. A saída deste código é mostrada na Figura 4-24.

FIGURA 4-24 Uma sombra de texto em um elemento HTML de parágrafo

Aplicando recorte

A propriedade **clip** permite que você especifique qual parte de um elemento está visível. A propriedade de clipe leva apenas um parâmetro, **a forma para o clipe**. Atualmente, a única forma de suporte é um **retângulo**, portanto, o único valor de parâmetro que irá produzir qualquer resultado é a função rect (). Por exemplo, o seguinte código é a sintaxe válida para especificar uma propriedade de clipe:

```
img{
  position: absolute;
  clip: rect(25px, 50px, 50px, 25px);
}
```

No exemplo de código acima, a região de clipe é definida como um retângulo. Os dois primeiros parâmetros da função `rect` compilam as coordenadas para qual parte da imagem será cortada. Os parâmetros são executados no sentido horário como superior, direito, inferior e lado esquerdo do retângulo. Além disso, todas as medições são tomadas a partir da esquerda da borda superior da caixa de origem sendo cortada. Assim, no exemplo de código acima, uma região da imagem é definida como 25 px a partir do topo para formar a borda superior da região cortada, 50 px a partir da esquerda para formar a borda direita da região cortada, 50 px da Topo para formar a borda inferior da região cortada, e 25 px a partir da esquerda para formar a borda direita da região cortada. Isto essencialmente cria um retângulo a partir do ponto (25px, 25px) e com uma altura e largura de 25px. A Figura 4-25 mostra uma imagem antes e depois de ser cortada com esses valores.

FIGURA 4-25 A imagem à esquerda é a imagem completa de um arranjo floral. A imagem à direita é uma versão cortada da mesma imagem.

Na Figura 2-25, você pode ver a imagem completa à esquerda. À direita, a imagem é cortada. Somente a seção da imagem de origem como especificado na função `rect` atribuída à propriedade de clipe é visível. O seguinte é o código completo:

```
<html>
<head>
<style>
.clipper{
position: fixed;
left: 325px;
clip: rect(25px, 100px, 100px, 25px);
}
</style>
</head>
<body>


</body>
```

</html>

NOTA ENTENDENDO A PROPRIEDADE CLIP

A propriedade clip funciona apenas em elementos cuja posição é definida como **fixed** ou **absolute**.

Aplicar estilos para estabelecer e alterar a posição de um elemento

O navegador fornece um sistema de coordenadas para como organizar elementos em uma página. O comportamento padrão é essencialmente um layout onde os elementos, sem quaisquer outros atributos de posição especificados, serão simplesmente exibidos na página em um fluxo padrão. Neste contexto, a coordenada de base é o canto superior esquerdo da janela, que pode ser entendida como coordenada (x, y) (0,0). Isso é chamado de layout estático. CSS fornece alguns mecanismos onde você pode substituir o layout padrão da página. Isto é conseguido especificando o comportamento de posição desejado com a propriedade de posição (**position**). Uma vez definida a propriedade de posição, outras propriedades CSS, como superior, esquerda, inferior ou direita, são definidas. Em um layout estático, os elementos não responderão à parte superior, esquerda, inferior ou direita da propriedades. O tipo de posicionamento deve ser especificado.

Dica de exame

Para o exame, certifique-se de que entende que cada elemento HTML é uma caixa e cada caixa inicia seu próprio novo sistema de coordenadas. **Se você colocar um elemento div na página em (50px, 50px), quaisquer elementos colocados dentro dele não serão colocados em uma coordenada começando em (50px, 50px) apenas porque é aí que o elemento div é. Os elementos filho dentro do div começam na coordenada (0,0), que é o canto superior esquerdo do próprio div.** Todos os elementos filhos são posicionados em relação ao recipiente no qual eles são colocados.

A propriedade de posição permite que você especifique uma de três opções diferentes: **fixo**, **relative** ou **absolute**. Com posicionamento **fixed**, os elementos são colocados em relação à janela do navegador. Com Posicionamento **relative**, os elementos são posicionados em relação à sua

posição em fluxo normal. Com posicionamento **absolute**, o elemento é posicionado em relação ao seu primeiro elemento pai. Você começará com uma imagem colocada em uma página dentro de um elemento div. A Figura 4-26 mostra esse elemento no fluxo estático.

FIGURA 4-26 Uma imagem na sua posição padrão dentro de um elemento div

Dica de exame

As propriedades esquerda e direita iniciam as suas medições a partir da extremidade mais caixa. Para o exame, tenha em mente que, se houver margens ou preenchimento especificado, isso também influenciará a posição do objeto.

Aplicando o seguinte estilo à imagem, você é capaz de reposicionar a imagem dentro do elemento div. A saída deste é mostrada na Figura 4-27.

```
img {  
  position: fixed;  
  left: 25px;  
  top: 25px;  
}
```

FIGURA 4-27 Uma imagem reposicionada definindo as propriedades CSS superior e esquerda

Usando o posicionamento **relativo**, você pode ajustar o layout dos elementos em relação a onde eles estariam no fluxo estático normal. Para demonstrar isso, você copiará o elemento da imagem muitas vezes para mostrar a imagem da flor muitas vezes dentro da div. Isso é mostrado no fluxo estático na Figura 4-28.

FIGURA 4-28 Uma imagem duplicada em fluxo estático

Agora, você moverá todas as imagens para a esquerda por 25 px. Isto será feito especificando posicionamento relativo e -25 px para a propriedade à esquerda. Como isso demonstra, você pode especificar números negativos para as propriedades de esquerda ou superior. A Figura 4-29 mostra a saída do seguinte código:

```
img:nth-child(1n+0) {  
  position: relative;  
  left: -25px;  
  top: 25px;
```



```
}
```

FIGURA 4-29 A utilização da propriedade esquerda com posicionamento relativo.

Como mostrado na Figura 4-29, todos os elementos da imagem foram movidos para a esquerda por 25 px. Usando posicionamento relativo, você pode realmente fazer seus elementos HTML se sobrepõem. Enquanto o código acima move todas as imagens acima por 25 px, se você fosse modificar o código de tal forma que cada elemento foi movido proporcionalmente mais para fazê-los sobreposição, você pode criar alguns efeitos agradáveis. Lembre-se que os elementos são movidos com base em onde eles teriam estado em fluxo normal. Uma vez que as imagens no centro são 100 px umas das outras, você precisará mover a segunda imagem, terceiro e imagem adicional pela mesma quantidade que o seu vizinho movido, mais a quantidade de sobreposição desejada. O código a seguir mostra isso:

```
img:nth-child(2) {  
  position: relative;  
  left: -25px;  
}  
img:nth-child(3) {  
  position: relative;  
  left: -50px;  
}  
img:nth-child(4) {  
  position: relative;  
  left: -75px;  
}
```

Este código produz a saída na Figura 4-30. Cada elemento após o primeiro é movido sobre o suficiente para que ele sobrepõe seu lado esquerdo vizinho por 25 px. Observe que o valor da propriedade esquerda é incrementalmente maior para explicar o fato de que seu vizinho também se moveu.

FIGURA 4-30 O uso do posicionamento relativo para sobrepor imagens

O uso do posicionamento absoluto permite que você especifique a localização de um elemento de forma que ele seja removido do fluxo normal da página. Ou seja, o resto da página flui como se esse elemento não existisse. O elemento pode ser considerado como pairando sobre ou sob o conteúdo que está no fluxo normal da página. Com esta abordagem, também é possível sobrepor elementos.

Dica de exame

Quando se sobrepõem elementos usando o posicionamento absoluto, o CSS fornece uma propriedade z-index. Isso permite especificar em que ordem os elementos devem ser empilhados na página ao longo do eixo z (a terceira dimensão!).

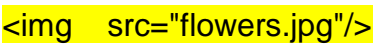
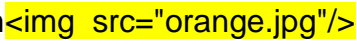
O código a seguir demonstra como usar o posicionamento absoluto de uma imagem sobre um bloco de texto. O texto abaixo da imagem processa em seu fluxo normal. A imagem não afeta nada no fluxo normal do documento. A saída deste código é mostrada na Figura 4-31.

```
img {  
  position: absolute;  
  left: 215px;  
  top: 100px;  
  height: 50px;  
  width: 50px;  
}
```

FIGURA 4-31 O uso do posicionamento absoluto para sobrepor uma imagem sobre o fluxo normal

A propriedade final disponível para usar para posicionar elementos é a propriedade float. A propriedade float move automaticamente um elemento para a esquerda ou para a direita do conteúdo circundante. Isso é mais comumente usado para colocar as imagens em linha com o texto para forçar o texto a envolver a imagem. Com base no exemplo acima, você moverá o elemento img de acordo com o texto, como mostrado aqui:

```
<p style="width: 200px;margin-left: 200px;">
```

Lorem ipsum dolor sit amet,  consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim  ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio...

</p>

Neste elemento parágrafo, há duas imagens em linha com o texto. Uma imagem colocada desta forma irá empurrar o texto para fora do caminho para criar espaço para si mesmo. Isso é mostrado na Figura 4-32.

FIGURA 4-32 Imagens embutidas em linha com o texto em fluxo normal

A propriedade `float` permite que você especifique como deseja que o elemento se posicione com os outros elementos ao seu redor. Você pode especificar para `float: left` ou `float: right`. Isso moverá o elemento de imagem para a esquerda ou para a direita, respectivamente, e permitirá que o texto flua em torno dele sem problemas. O código a seguir demonstra como especificar isso:

```
img.flower {  
  float: left;  
  left: 215px;  
  top: 100px;  
  height: 50px;  
  width: 50px;  
}  
img.orange {  
  float: right;  
  left: 215px;  
  top: 100px;  
  height: 50px;  
  width: 50px;
```

}

A saída do código acima é mostrada na Figura 4-33.

FIGURA 4-33 Imagens em linha com o texto com suas propriedades de flutuação especificadas

Agora, com a propriedade float especificada, o texto está fluindo suavemente em torno das imagens.

Experimento de pensamento

Criando um elemento HTML em movimento Neste experimento de pensamento, aplique o que aprendeu sobre esse objetivo. Você pode encontrar uma resposta a esta pergunta na seção "Respostas" no final deste capítulo. Crie os estilos e scripts para fazer com que um objeto HTML se mova pela página. Isso é visto em muitas páginas da Web hoje com controles de banner ou ticker que exibem informações movendo-se através da tela. Usando o que você sabe sobre como configurar e alterar a posição de um objeto com CSS, crie um elemento HTML em movimento.

Resumo do objetivo

- Cada elemento HTML é uma caixa e tem as propriedades de uma caixa, como altura e largura.
- CSS3 permite que você altere o tamanho de uma caixa especificando uma nova altura e largura.
- CSS3 permite que você defina as propriedades da caixa das seguintes maneiras:
 - A propriedade border-style permite que você especifique uma linha sólida ou tracejada para a borda.
 - A propriedade border-color permite que você especifique a cor da borda.
 - A propriedade border-spacing permite especificar a quantidade de espaço entre Elementos adjacentes.
 - A propriedade border-width permite que você especifique uma espessura para a borda.
 - Cada lado da caixa pode ser feito de forma diferente.

■ CSS3 fornece uma maneira de definir o padding e **margem** que uma caixa deve ter relativa para elementos adjacentes. Isso pode ser configurado de forma diferente para cada lado da caixa.

■ Um elemento pode ser tornado **transparente** ou **parcialmente transparente**, ajustando a **propriedade opacidade**.

■ Um elemento pode conter uma imagem de fundo configurando sua propriedade de imagem de plano de fundo.

■ O CSS3 fornece a capacidade de criar efeitos de sombra especificando a propriedade **caixa-sombra**.

■ CSS3 fornece a capacidade de clip de imagens usando a propriedade de clipe para mostrar apenas uma parte de uma imagem.

■ O CSS3 pode ser usado para estabelecer a posição de um elemento como fixo, absoluto ou relativo.

■ As propriedades CSS esquerda e superior podem ser usadas para alterar a posição de um elemento.

Revisão objetiva

Responda às seguintes perguntas para testar seu conhecimento da informação neste objetivo. Você pode encontrar as respostas a essas perguntas e explicações de por que cada opção de resposta está correta ou incorreta na seção "Respostas" no final deste capítulo.

1. Qual das opções a seguir não é uma maneira válida de alterar o tamanho de um elemento?
 - A. `div{height: 50px; width: 50%;}`
 - B. `div{height: 50px; width: 50px;}`
 - C. `div{height: 50cm; width: 50px;}`
 - D. `div{height: 50ft; width: 50ft;}`**
2. Qual das opções a seguir formará com êxito a borda de um elemento `div`?
 - A. `border-top: 5px dotted blue;`**
`border-right: 5px solid green;`
`border-left: 3px dashed red;`
`border-bottom: 10px double black;`
 - B. `border-sides: 5px solid green;`

- C. border-all: 1px solid black;
- D. border: full red;

3. Ao olhar da borda externa de um elemento HTML e mover para o interior

Borda, em que ordem ocorre o preenchimento, margem e borda?

- A. padding, border, margin
- B. margin, border, padding**
- C. border, padding, margin
- D. margin, padding, border

4. Qual das seguintes afirmações aplicará uma sombra de caixa à direita e à parte inferior borda de um elemento div?

- A. box-shadow: gray 5px 5px;**
- B. box-shadow: gray -5px 5px;
- C. box-shadow: gray 5px -5px;
- D. box-shadow: gray -5px -5px;

5. Qual das seguintes opções colocará um elemento em relação à janela do navegador?

- A. absolute
- B. fixed**
- C. relative

Objetivo 4.3: Criar um layout de conteúdo flexível

Organizar conteúdo na página sempre foi um desafio no design da página da web. O conceito de padrões de design para separar layout de conteúdo / lógica existe há muito tempo em outros espaços de desenvolvimento. No entanto, tem sido menos facilmente disponíveis no espaço HTML.

O CSS3 introduz novos conceitos como o **layout de grid** e o **layout de flexbox** (grid layout and flexbox layout) que fornecem mecanismos para alcançar essa separação adequada.

Este objetivo abrange como:

- Implementar um layout usando um modelo de caixa flexível
- Implementar um layout usando multi-colunas
- Implementar um layout usando posicionamento, flutuação e exclusões

- Implementar um layout usando alinhamento de grade
- Implementar um layout usando regiões, agrupamento e aninhamento

Implementar um layout usando um modelo de caixa flexível

A flexbox é uma construção CSS3 que fornece uma maneira de estabelecer elementos que fluem. Fluxo significa que os elementos irão fluir da esquerda para a direita, também conhecido como horizontal, ou para cima e para baixo, também conhecido como vertical. Para começar com uma flexbox, você precisa criar um elemento container e dar-lhe um nome. Use um elemento <div> e nomeie-o como mostrado neste código:

```
<div id="flexbox1">  
</div>
```

Com este bloco de código, você tem o começo de um flexbox. Tudo o que resta fazer é criar o CSS para indicar que o contêiner é de fato um flexbox. O seguinte CSS consegue isso:

```
#flexbox1 {  
display: flexbox;  
border: 1px solid black;  
margin-top: 100px;  
min-height: 15px;  
}
```

Com esse HTML e CSS no lugar, você pode executar a página e ver um recipiente especificado para ter um layout flexbox. A saída é mostrada na Figura 4-34.

FIGURA 4-34 Um elemento div configurado como uma flexbox

Todos os elementos dentro de uma caixa flexível são chamados itens flexbox. Você pode especificar que o layout flexbox é executado horizontalmente ou verticalmente. Você precisará estar familiarizado com alguns dos principais estilos que podem ser aplicados a uma flexbox e como o navegador irá interpretá-los. A Tabela 4-7 descreve os estilos importantes relacionados ao fluxo dos elementos filho.

TABELA 4-7 estilos CSS disponíveis para uma caixa flexível

Estilo	Opção	Descrição
flex-direction	Column	Flui os elementos filho da flexbox através do eixo vertical de cima para baixo .
	row (default)	Flui os elementos filhos da flexbox ao longo do eixo horizontal da esquerda para a direita .
	column-reverse	Renderiza os elementos filho ao longo do eixo vertical do final da parte de trás para o topo .
	row-reverse	Renderiza os elementos filhos ao longo do eixo horizontal a partir da extremidade inversa da direita para a esquerda .
flex-pack	End	Renderiza os elementos filhos da extremidade em relação à direção do conjunto de eixos de layout.
	Start	Renderiza os elementos filho desde o início em relação à direção do conjunto de eixos de layout.
	center	Renderiza os elementos filho centrados no eixo de layout.

	distribute	Espaços uniformemente os elementos filho ao longo do eixo de layout.
--	------------	--

Dica de exame

A flexbox é orientada com base na direção do flex. A direção do flex é baseada no eixo de layout. Se o layout da flexbox for coluna, o eixo de layout será vertical. Se o layout flexbox for linha, o eixo de layout será horizontal. Para o exame, isso é importante para entender, a fim de saber como outras propriedades na malha flex será processado.

Agora adicione alguns elementos filho ao seu layout flexbox:

```
<div id="flexbox1">
<div></div>
<div></div>
<div></div>
</div>
```

Adicionar elementos `<div>` vazios não será suficiente para mostrar conteúdo. Você também precisa adicionar alguns estilos para que o layout flexbox mostre seus elementos filho. Aqui estão os estilos a serem adicionados:

```
#flexbox1 > div {
min-width: 80px;
min-height: 80px;
border: 1px solid black;
margin: 5px;
}
#flexbox1 > div:nth-child(1) {
background-color: green;
}
#flexbox1 > div:nth-child(2) {
background-color: yellow;
}
#flexbox1 > div:nth-child(3) {
background-color: red;
}
```

Este código produz a saída na Figura 4-35.

FIGURA 4-35 Uma caixa flexível com elementos de conteúdo configurados

O fluxo padrão dentro da flexbox é exibir os elementos da esquerda para a direita a partir da borda esquerda. Você pode experimentar manipular as propriedades para exibir as caixas coloridas em uma ordem diferente. Por exemplo, e se você queria exibir as caixas no mesmo local no eixo de layout, mas em ordem inversa? Você poderia fazer algo como isto:

```
#flexbox1 {  
  display: flexbox;  
  flex-direction: row-reverse;  
  border: 1px solid black;  
  margin-top: 100px;  
  min-height: 15px;  
}
```

A saída deste código é mostrada na Figura 4-36.

FIGURA 4-36 Conteúdo Flexbox em ordem inversa ao longo do mesmo eixo

Aqui você mudou a direção para `row-reverse`. Isso altera a ordem das caixas. Quando você executa isso, você vê que não só inverteu a ordem das caixas, mas também as moveu para a extremidade direita. Isso ocorre porque a direção flex controla o que é considerado os pontos de origem no eixo de layout. Ao inverter a direção, você indicou que o início do layout está agora na extremidade direita eo layout está na extremidade esquerda. Para evitar inverter a ordem, você precisará especificar a propriedade `flex-pack` para indicar o final do eixo de layout:

```
#flexbox1 {  
  display: flexbox;  
  flex-flow: row-reverse;  
  flex-pack: end;  
  border: 1px solid black;  
  margin-top: 100px;  
  min-height: 15px;  
}
```

Com esta atualização, as caixas estão agora a mostrar na mesma ordem (vermelho, amarelo, verde), mas alinhado à esquerda como pretendido. A saída é demonstrada na Figura 4-37.

FIGURA 4-37 Conteúdo Flexbox na mesma ordem, mas alinhado com a extremidade esquerda

Há também algumas funcionalidades adicionais que fornece a capacidade para cada elemento filho para se flexionar para ocupar a quantidade de espaço que é especificado. Se você queria que uma caixa ocupasse 15% do espaço, outra ocupasse 25% e a última ocupasse qualquer espaço deixado sem deixar espaço em branco entre elas, implementaria a propriedade flex. A propriedade flex é especificada em cada um dos elementos filhos para designar a quantidade de espaço cada deve ocupar.

Primeiro, você precisará alterar as propriedades do contêiner de layout para ter a seguinte aparência:

```
#flexbox1 {  
  display: flexbox;  
  border: 1px solid black;  
  margin-top: 100px;  
  min-height: 15px;  
}  
#flexbox1 > div {  
  min-width: 80px;  
  min-height: 80px;  
  border: 1px solid black;  
  margin: 5px;  
}  
#flexbox1 > div:nth-child(1) {  
  background-color: green;  
  flex: 2;  
}  
#flexbox1 > div:nth-child(2) {  
  background-color: yellow;  
  flex: 15;  
}
```

```
#flexbox1 > div:nth-child(3){  
background-color: red;  
flex:3;  
}
```

No código CSS atualizado, a propriedade flex é adicionada a cada um dos elementos filhos. A propriedade flex leva um parâmetro que é um valor relativo. Esta não é uma medida hardcoded Em pixels ou polegadas. É relativo ao valor conforme especificado entre todos os elementos filhos. Neste caso, o primeiro elemento irá conter 10 por cento do espaço, o último elemento terá 15 por cento do espaço, e o elemento do meio irá realizar qualquer espaço é deixado. Para fazer isso, você precisa calcular os valores relativos que seriam necessários para gerar essas proporções. O restante dos cálculos para processar a saída será tratado pelo navegador. Porque estamos falando de porcentagens, é lógico que toda a largura do espaço é 100 por cento. A maneira fácil de mostrar isso é colocar em cada elemento a sua respectiva percentagem, uma vez que o tamanho relativo seria o que você está procurando. Para ilustrar o ponto, você pode fatorar as unidades em algo menor, dividindo por 20. Isso faria 10 por cento igual 2 partes de 100 por cento e 15 por cento igual 3 partes de 100 por cento. A seção restante usará 15 partes, ou 75 por cento de 100 por cento. A Figura 4-38 mostra a saída com valores.

FIGURA 4-38 Distribuição do conteúdo do Flexbox usando a propriedade flex

A ordem dos itens flexbox também pode ser especificada explicitamente usando a propriedade order nos itens flexbox. Um exemplo disto está listado aqui:

```
#flexbox1 > div:nth-child(1) {  
background-color: green;  
flex-order: 2;  
}  
#flexbox1 > div:nth-child(2) {  
background-color: yellow;  
flex-order: 1;  
}  
#flexbox1 > div:nth-child(3) {
```

```
background-color: red;
flex-order: 3;
}
```

A Figura 3-39 demonstra que os itens flexbox são exibidos na ordem especificada em vez da ordem padrão, que estaria na ordem em que os itens são listados no HTML.

FIGURA 4-39 Definindo explicitamente a sequência dos itens flexbox com a propriedade order

Outra propriedade que é importante entender é a opção de envolvimento. Assim como o envoltório de texto, flex-wrap fornece a capacidade de especificar o que o navegador deve fazer no caso de o conteúdo dentro do flexbox exceder o espaço disponível da própria flexbox. Nesse caso, você pode especificar que a flexbox deve ser envolvida ou não. Um exemplo de empacotamento é mostrado a seguir. Primeiro atualize seu flexbox para ter uma largura fixa e, em seguida, adicione a propriedade flexbox wrap:

```
#flexbox1 {
display: flexbox;
flex-flow: row;
flex-wrap: wrap;
border: 1px solid black;
margin-top: 100px;
min-height: 15px;
width: 200px;
}
```

Para demonstrar a funcionalidade de empacotamento, você precisa adicionar mais alguns itens flexbox para que eles possam transbordar a flexbox:

```
<div id="flexbox1">
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
</div>
```

Adicione esses estilos adicionais para definir cores de plano de fundo para os itens flexbox extra:

```
#flexbox1 > div:nth-child(1) {  
background-color: green;  
}  
#flexbox1 > div:nth-child(2) {  
background-color: yellow;  
}  
#flexbox1 > div:nth-child(3) {  
background-color: red;  
}  
#flexbox1 > div:nth-child(4) {  
background-color: purple;  
}  
#flexbox1 > div:nth-child(5) {  
background-color: blue;  
}
```

A Figura 4-40 demonstra a funcionalidade de envolvimento da flexbox.

FIGURA 4-40 Uso da propriedade flex-wrap para envolver automaticamente os itens flexbox

Dica de exame

No exame, você pode ver a versão abreviada do que você acabou de fazer com o invólucro. A propriedade de fluxo flexível suporta especificando o estilo de recorte como o segundo parâmetro: flex-flow: row wrap.

Implementação de um layout usando multi-coluna

CSS3 fornece a capacidade de criar um layout usando colunas. Isso fornece uma aparência ao conteúdo, como o que você pode ver em um artigo de jornal onde o conteúdo se encaixa até a próxima coluna. Como com as outras técnicas de layout, a técnica multi-coluna começa especificando um recipiente para armazenar as colunas. Nesse caso, um elemento de artigo será usado. A seguinte marcação é adicionada à página:

```
<html>
```

```

<head>
<style>
#c_lo {
width:80%;
height: 400px;
border: 1px solid black;
column-count: 5;
column-rule-color: black;
column-rule-style: dashed;
column-rule-width: 2px;
column-gap: 5px;
}
</style>
</head>
<body>
<article id="c_lo">
</article>
</body>
</html>

```

Esta página HTML contém um bloco de estilo para configurar o layout de várias colunas. O corpo contém um elemento de artigo que será configurado para processar as várias colunas. Algumas propriedades de várias colunas já estão definidas no elemento article. A Tabela 4-8 detalha as propriedades de várias colunas.

TABELA 4-8 Propriedades de várias colunas

Descrição - Propriedade
Column-count - Especifica o número de colunas
Column-gap - Especifica a quantidade de espaço a colocar entre as colunas
Column-rule-color - Especifica a cor da regra vertical desenhada entre colunas
Column-rule-style - Especifica o tipo de regra vertical a desenhar entre colunas , por exemplo, sólido ou linha tracejada

Column-rule-width	- Especifica a largura da regra vertical desenhada entre as colunas
Column-rule	- Uma forma abreviada de especificar a cor, estilo e largura da regra vertical entre as colunas
Column-span	- Especifica quantas colunas o elemento deve abranger; Valores possíveis são um Número de colunas ou todos; O valor padrão é 1
Width-column	- Especifica a largura que as colunas devem ser
Columns	- Um método abreviado para especificar o número de colunas e sua largura

Dica de exame

Sempre que você vir uma propriedade com o mesmo nome prefixo como com a propriedade de regra de coluna, você pode saber que existe um mecanismo abreviado para especificar as propriedades mais comuns em uma linha.

Depois de analisar a Tabela 4-8, agora você sabe que o elemento artigo no código anterior bloco é configurado para ter cinco colunas, cada 5 px separados. A regra vertical entre colunas usará uma linha tracejada que é preto e 2-px de largura. É comum que os artigos tenham um título. Normalmente, o título irá abranger todas as colunas do artigo. Para obter esse efeito, você precisará usar a propriedade column-span em um elemento para indicar que ele deve renderizar em várias colunas. Para conseguir isso, adicione o seguinte código para o CSS:

```
<html>
<head>
<style>
#c_lo {
width: 80%;
height: 400px;
border: 1px solid black;
column-count: 5;
column-rule-color: black;
```



```

column-rule-style: dashed;
column-rule-width: 2px;
column-gap: 5px;
}
hgroup {
column-span: all;
text-align:center;
}
</style>
</head>
<body>
<article id="c_lo">
<hgroup>
<h1>My Blog Article</h1>
</hgroup>
<p>
...
</p>
</article>
</body>
</html>

```

Você adicionou um elemento hgroup, elementos de estilo adicionados para especificar que o texto hgroup deve ser centralizado e especificou o valor all para a propriedade column-span.

Essa marcação produz a saída mostrada na Figura 4-41.

FIGURA 4-41 Usando a técnica de layout de várias colunas para exibir texto em um formato de artigo

Implementação de um layout usando posicionamento, flutuação e Exclusões

Usar position e float para posicionar elementos foi discutido na seção Objetivo 4.3 intitulada "Aplicar estilos para estabelecer e alterar a posição de um elemento." Nesta seção, você vai examinar como eles podem ser usados no

layout geral. Float é um mecanismo pelo qual o conteúdo circundante fluirá suavemente em torno do elemento com sua propriedade flutuante especificada para float: left ou float: right. Esquerda e direita são as únicas duas opções disponíveis para a propriedade float. As exclusões fornecem uma maneira de superar essa limitação com float. As exclusões são alcançadas especificando o fluxo de envoltório de propriedade CSS3.

REQUISITOS IMPORTANTES PARA A VERSÃO DO EXPLORADOR DA INTERNET

Neste momento, a propriedade wrap-flow é implementada apenas no Internet Explorer 10+. Portanto, esse elemento requer que o prefixo -ms seja aplicado a ele.

A propriedade wrap-flow suporta uma variedade de opções. Essas opções estão na tabela 4-9. Você usará a maioria dessas opções nos exemplos a seguir.

TABELA 4-9 Valores disponíveis para a propriedade de fluxo de envolvimento

Valor	Descrição
Auto	Este é o valor padrão. O item de exclusão será superior ao elemento inline.
Both	A exclusão forçará o elemento inline a enrolar sem problemas em ambos os lados.
Start	A exclusão forçará os elementos inline a serem envolvidos somente na borda de a borda final estará vazia.
End	A exclusão forçará o elemento inline a envolver somente na extremidade final e a borda inicial estará vazia.
Maximum	A exclusão forçará o elemento inline a envolver apenas no lado com o maior espaço disponível.

Clear	A exclusão forçará o conteúdo inline a envolver apenas na parte superior e inferior e sair as bordas de início e fim são vazias.
-------	--

Com a compreensão dos valores fornecidos pela tabela anterior, agora você pode aplicar esses conceitos em suas páginas. O código abaixo demonstrará o efeito de definir a propriedade de fluxo de envolvimento para ambos:

```
<html>
<head>
<style>
p {
width: 80%;
padding-left: 50px;
}
img {
position: absolute;
height: 100px;
width: 150px;
-ms-wrap-flow: both;
}
</style>
</head>
<body>
<p>
Lorem ipsum dolor sit ...
debitis. Modus elaboraret temporibus no sit. At invidunt
splendide qui, ut pro choro iisque democritum. Partem timeam graecis ea vis,
utamur
feugiat ...
</p>
</body>
</html>
```

Este código produz a saída na Figura 4-42.

FIGURA 4-42 Configurando a propriedade wrap-flow para o valor de ambos para fazer o envoltório de texto inline em ambos os lados da imagem

O texto em linha se encaixa muito bem em torno do início (margem esquerda da imagem) eo final (Borda direita da imagem). O texto é bastante próximo da imagem. A propriedade wrap-margin pode ser especificada para fornecer uma margem em torno da imagem. Adicione esta propriedade ao seu código CSS da seguinte forma:

```
img {  
    position: absolute;  
    height: 100px;  
    width: 150px;  
    -ms-wrap-flow: both;  
    -ms-wrap-margin: 15px;  
}
```

A saída deste código é mostrada na Figura 4-43.

FIGURA 4-43 Configurando a propriedade wrap-margin para criar uma margem em torno da exclusão

No caso em que você deseja ter o envoltório de conteúdo em linha apenas à esquerda ou à direita, você especificará o início ou o final, respectivamente, como o valor para a propriedade de fluxo de envoltório. A Figura 4-44 demonstra a saída se você alterar o valor para iniciar. Definir o valor para terminar terá o efeito oposto.

FIGURA 4-44 Definindo a propriedade wrap-flow para o valor start para ter o content wrap somente à esquerda

A Figura 4-45 mostra o efeito se a propriedade wrap-flow estiver definida como clear. Ambos os lados do exclusão são limpas e o texto envolve somente ao longo da parte superior e inferior.

FIGURA 4-45 Configurando a propriedade de fluxo de envoltório para o valor claro para que o conteúdo envolva somente ao longo da parte superior e inferior

Implementação de um layout usando alinhamento de grade

O recurso de layout de grade do CSS3 fornece uma maneira de definir o conteúdo da página da Web muito parecido com uma tabela HTML, mas usando apenas CSS para obter os resultados. Isso fornece mais flexibilidade e código mais sustentável.

Para demonstrar a capacidade do layout de grade com o CSS3, você criará uma página da Web Layout que se parece com o da Figura 4-46.

FIGURA 4-46 A página da Web que você criará usando o recurso de layout de grade no CSS3.

Tradicionalmente, para alcançar esse layout, você teria provavelmente criado uma marcação HTML, como o seguinte código:

```
<table border="1" style="width:100%; height: 85%">
  <tr>
    <td colspan="3">
  </td>
</tr>
<tr>
  <td rowspan="3">
</td>
  <td rowspan="3">
</td>
  <td>
</td>
</tr>
<tr>
  <td>
</td>
  <td>
</td>
</tr>
<tr>
  <td>
</td>
  <td>
</td>
</tr>
```

```
</tr>
<tr>
<td colspan="3">
</td>
</tr>
</table>
```

Usando uma tabela HTML funciona e tem trabalhado por um tempo muito longo. No entanto, as melhores práticas hoje sugerem separar o conteúdo do layout. Isso é visto em muitos outros espaços de desenvolvimento de software e design no contexto de padrões de design. Neste exemplo, o layout está correto no HTML. Você vai reconstruir a página de modo que o conteúdo pode ser fornecido independente do layout e todo o layout será definido no código CSS. O layout de grade é totalmente definido em CSS. Para criar a grade, as propriedades CSS precisarão ser aplicadas aos elementos HTML. Assim como com a flexbox, a grade é baseada em ter a exibição especificada para um contêiner. Você usará um elemento div simples para seu contêiner de grade. Crie a div a seguir e aplique os estilos como indicado:

```
<style>
#mainGrid {
display: grid;
}
</style>
<div id="mainGrid">
</div>
```

Você designou seu div com o id de mainGrid para ser o recipiente para um layout de grade, especificando sua exibição para ser grade. No entanto, executando isso no navegador não vai muito mais do que um resultado. Você precisa definir a estrutura da grade. Neste exemplo, você criará um layout de grade simples de quatro por dois. Adicione os elementos filho ao div como mostrado e atualize os estilos como mostrado para fornecer informações ao recipiente sobre como você gostaria que ele processasse as células filho.

```
#mainGrid {
display: grid;
grid-columns: 150px 150px 150px 150px;
```

```
grid-rows: 75px 75px;
}
<div id="mainGrid">
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
</div>
```

NOTA UTILIZANDO ATUALIDADE

Alguma grande taquigrafia está disponível para especificar linhas e colunas em um caso onde você tem padrões. No exemplo desta seção, o seguinte CSS fornece as mesmas instruções:

```
#blogPage {
display: grid;
grid-columns: 15% 1fr 25%;
grid-rows: (20%)[5];
width: 90%;
height: 95%;
border: 1px dotted black;
margin: auto;
}
```

Você adicionou duas propriedades adicionais de CSS ao mainGrid. Essas propriedades são auto-explicativo. Como o nome sugere, a propriedade `grid-columns` permite especificar como muitas colunas estão em sua grade. A propriedade `grid-rows` permite especificar quantas linhas estão na sua grade. Os valores de propriedade aceitos são atributos de tamanho separados por espaços. O renderer interpretará cada valor como uma nova coluna ou linha.

Se você executar este HTML no navegador, a saída será menos do que emocionante. Uma razão é que os elementos `div` estão vazios. Vá em frente e coloque algum texto em cada `div` e examine a saída que você recebe no navegador. Se tudo correr como esperado, seus elementos `div` exibirão todos em cima uns dos outros. Eles são, para todos os efeitos, ignorando suas instruções CSS anteriores fornecido ao contêiner ao layout em um estilo de quatro por dois. Bem, eles não estão realmente ignorando as instruções. Lembre-se, as instruções foram para o contentor `mainGrid` sobre como você queria que ele colocasse seus elementos filhos. **Você ainda precisa dar a cada elemento filho sua própria identidade dentro da grade.** Ou seja, o **contêiner sabe que precisa configurar um quatro-por-dois** Grid, mas não sabe qual elemento vai onde dentro dessa grade de quatro por dois. É só flutua todos eles em cima uns dos outros. Agora, você especificará os elementos da grade deve viver no espaço relativo disponível dentro da grade. Você aplicará algumas cores de forma que a página renderizada demonstre facilmente os conceitos.

```
#mainGrid > div:nth-child(1){
  grid-column: 1;
  grid-row:1;
  background-color: blue;
}
#mainGrid > div:nth-child(2){
  grid-column:2;
  grid-row:1;
  background-color: aqua;
}
#mainGrid > div:nth-child(3){
  grid-column: 3;
  grid-row:1;
  background-color: red;
}
#mainGrid > div:nth-child(4){
  grid-column: 4;
  grid-row:1;
  background-color: green;
```



```

}
#mainGrid > div:nth-child(5){
grid-column: 1;
grid-row:2;
background-color: magenta;
}
#mainGrid > div:nth-child(6){
grid-column: 2;
grid-row:2;
background-color: yellow;
}
#mainGrid > div:nth-child(7){
grid-column: 3;
grid-row:2;
background-color: orange;
}
#mainGrid > div:nth-child(8){
grid-column: 4;
grid-row:2;
background-color: olive;
}

```

Com este código, você instruiu os elementos onde eles devem ser posicionados dentro da grade. Quando o processador processa esses elementos, ele agora saberá perguntando a cada elemento onde ele deve ir e como colocá-los. Agora, a página exibida no navegador se parece com a Figura 4-47. Cada elemento é colocado de acordo com a posição da coluna e da linha especificada para ele: elementos azuis, aqua, vermelhos e verdes respectivamente nas colunas 1 a 4, linha 1; Magenta, amarelo, laranja e azeitona respectivamente nas colunas 1 a 4, linha 2.

FIGURA 4-47 A página da Web com todas as colunas e linhas da grade exibida

Agora, com a fundação no lugar, você pode dar um passo adiante para criar uma grade que corresponda ao layout da tabela especificado anteriormente.

Para fazer isso, você precisa apenas um par mais propriedades CSS que irá

fornecer a funcionalidade equivalente como o `colspan` eo `rowspan`. Além disso, seus estilos fornecerão algumas propriedades de dimensionamento para garantir que a grade se expanda na página como você gostaria que ela fosse. Isto é opcional, claro, no mundo real. Você pode querer ter conteúdo forçar o dimensionamento de sua grade ou você pode querer especificar explicitamente o tamanho.

A listagem de código completo é especificada aqui:

```
html, body {  
  height: 100%;  
  width: 100% ;  
}  
#blogPage {  
  display: grid;  
  columns: 15% 1fr 25%;  
  grid-rows: 20% 20% 20% 20% 20%;  
  width: 90%;  
  height: 95%;  
  border: 1px dotted black;  
  margin: auto;  
}  
#blogPage > header {  
  grid-column: 1;  
  grid-column-span: 3;  
  grid-row: 1;  
  border: 1px dotted black;  
}  
#blogPage > footer {  
  grid-column: 1;  
  grid-row: 5;  
  grid-column-span: 3;  
  border: 1px dotted black;
```

```
}  
#blogPage > article {  
  grid-column: 2;  
  grid-row: 2;  
  grid-row-span: 3;  
  border: 1px dotted black;  
}  
#blogPage > #calendar {  
  grid-column: 3;  
  grid-row: 3;  
  border: 1px dotted black;  
}  
#blogPage > #blogRoll {  
  grid-column: 3;  
  grid-row: 4;  
  border: 1px dotted black;  
}  
#blogPage > #aboutMe {  
  grid-column: 1;  
  grid-row: 2;  
  grid-row-span: 3;  
  border: 1px dotted black;  
}  
#blogPage > #bloghistory {  
  grid-column: 3;  
  grid-row: 2;  
  border: 1px dotted black;  
}  
</style>  
<body>
```

```

<div id="blogPage">
<header>My Blog Header</header>
<article>My Blog's Main Body</article>
<footer>My Blog Footer</footer>
<aside id="calendar">A calendar</aside>
<aside id="blogRoll">My favorite blogs</aside>
<aside id="aboutMe">Who am I?</aside>
<aside id="bloghistory">My blog history</aside>
</div>
</body>

```

Uma vez que você está produzindo uma página da web real que irá realizar conteúdo real, o apropriado Tags semânticas estão sendo usadas. Você pode ver de olhar para a parte HTML do código que não há nenhuma indicação para layout. Você definiu apenas uma série de seções para a página a ser exibida. Toda a implementação do layout acontece no CSS. Você notará a adição de duas propriedades CSS ao seu repertório: grid-row-span e grid-column-span.

Como seus nomes sugerem, você pode esperar que eles se comportem da mesma maneira que os atributos HTML colspan e rowspan. Eles dizem ao navegador para colocar a coluna ou a linha de modo que ela se espera o número especificado de colunas ou linhas. Quando você executar este código no navegador, você obterá a saída exibida na Figura 4-48.

FIGURA 4-48 O layout da página web usando apenas o layout de grade CSS3

A implementação de um layout usando regiões, agrupamento e regiões de aninhamento

É uma nova construção CSS3 que permite que você tenha fluxo de conteúdo por várias regiões em uma página da Web. Isso poderia fornecer alguns cenários muito interessantes. Para começar, você precisará de uma página HTML com regiões definidas nele. O HTML a seguir fornece o ponto de partida para o estabelecimento de regiões.

```

<body>
<div class="regionLayout">
<div id="region1"></div>c
<div id="region2"></div>
<div id="region3"></div>
...

```

```
<div id="region-n"></div>

</div>

</body>
```

OBSERVE O APOIO DE NAVEGADOR PARA REGIÕES

As regiões estão atualmente em fase experimental de desenvolvimento. Neste momento, há suporte limitado para o navegador. Esta seção destaca apenas os principais objetivos de design desse recurso.

O layout de uma página da Web usando regiões requer duas coisas: uma fonte de conteúdo e as regiões que serão o destino de conteúdo. O HTML acima descreve as regiões. O conteúdo pode vir de outra página através de um iframe ou outro elemento na própria página (embora isso atualmente não funcione em qualquer navegador). Ao adicionar um iframe à página, você pode definir o iframe src para o conteúdo que será renderizado nas regiões:

```
<iframe src="content_source.html"/>
```

Com a fonte de conteúdo estabelecida no HTML, existem agora apenas duas coisas com necessidade de ocorrer. CSS é usado para controlar a funcionalidade do conteúdo da origem para o destino. As novas propriedades CSS chamadas `flow-into` e `flow-from` são usadas para atribuir a função dos elementos HTML no layout da região. A propriedade de fluxo de entrada é atribuído um valor para armazenar o conteúdo. Esse valor pode ser qualquer coisa como neste exemplo:

```
.content_source{
  flow-into: myflow;
}
```

Em seguida, o destino do conteúdo é definido em uma classe como esta:

```
.content_regions{
  flow-from: myflow;
}
```

Contanto que o mesmo nome for usado no `flow-into` e no `flow-from`, trabalharão juntos. Isso é chamado de fluxo nomeado. Todos os elementos que formam as regiões para fonte de conteúdo no mesmo fluxo nomeado é chamada uma cadeia de região. Você pode ter várias fontes e várias cadeias de região. O nome atribuído às propriedades `flow-into` e `flow-from` é usado para coordenar qual fonte de conteúdo vai para quais regiões.

Dica de exame

Uma vez que o recurso regiões ainda é experimental, o exame não é susceptível de cobrir este tópico. No entanto, este é apenas no momento da escrita e poderia mudar a qualquer momento. Seja familiar e continue verificando atualizações com respeito à prontidão desta construção do CSS no mundo real.

Experimento de pensamento

Combinando layouts nesta experiência de pensamento, aplique o que você aprendeu sobre esse objetivo. Você pode encontrar uma resposta a esta pergunta na seção "Respostas" no final deste capítulo. Estenda a página do blog de layout de grade atual para que a área de conteúdo principal use um layout de coluna. Cada layout é muito poderoso por si só para servir a sua finalidade específica. É importante, porém, saber que você pode criar alguns layouts muito poderosos, combinando-os onde faz sentido.

Resumo do objetivo

- ■ O Flexbox permite que você elabore elementos de forma fluida.
- ■ O layout de várias colunas permite separar o conteúdo em um número fixo de colunas, como um layout de jornal.
- ■ O fluxo de texto em torno de elementos como imagens pode ser controlado usando **wrap-flow** Layouts.
- O layout de grade fornece a melhor maneira de separar o layout do conteúdo especificando explicitamente em CSS onde cada elemento deve ser exibido dentro de uma grade predefinida.

Revisão objetiva

Responda às seguintes perguntas para testar seu conhecimento da informação neste objetivo. Você pode encontrar as respostas a essas perguntas e explicações de por que cada opção de resposta está correta ou incorreta na seção "Respostas" no final deste capítulo.

1. Qual dos seguintes layouts é baseado em colunas e linhas?
 - A. flexbox
 - B. multi-column
 - C. grid layout**
 - D. exclusions
2. Qual dos seguintes itens é falso sobre um layout flexbox?
 - A. A direção dos elementos numa flexbox pode ser controlada com a direção flexível propriedade.
 - B. O layout dos elementos pode ser configurado ao longo do eixo de layout usando o pacote flex propriedade.
 - C. Elementos em um flexbox são chamados itens flexbox.
 - D. Elementos em uma flexbox podem ser definidos em linhas e colunas.**

3. Qual dos valores de propriedade a seguir para wrap-flow permitirá que o texto envolva ao longo ambos os lados de um elemento?

- A. both
- B. all
- C. left and right
- D. cross

Objetivo 4.4: Criar uma interface de usuário animada e adaptável

O site do dia moderno é uma experiência interativa para o usuário final. CSS3 fornece muitos mecanismos para aplicar um toque profissional para a interação do usuário final. Isso é conseguido através da capacidade de **animar e transformar objetos**. Ao adicionar esses recursos ricos em suas páginas da web, você realmente traz a experiência para o próximo nível. Além disso, existe a oportunidade de criar uma interface de usuário **responsiva**. **Uma interface de usuário responsiva é aquela que pode se adaptar automaticamente com base no tamanho da tela disponível**. Finalmente, a capacidade de **ocultar** e **desativar controles** fornece a capacidade de personalizar ainda mais a interface do usuário com CSS.

Este objetivo abrange como:

- Animar objetos aplicando transições CSS
- Aplicar transformações 3-D e 2-D
- Ajustar IU com base em consultas de mídia
- Ocultar ou desativar controles

Animando objetos aplicando transições CSS

As transições fornecem um mecanismo para alterar o estilo de um objeto de tal forma que a mudança ocorre de forma gradual e visível. Você tem a capacidade de controlar qual propriedade de estilo é alterada e quanto tempo leva para concluir sua transição de um estilo para outro. **Uma transição começa quando a propriedade especificada é alterada**. Em sua forma mais simples, o código a seguir transita duas propriedades de um elemento div: a margem esquerda e a cor de fundo. A transição é tomada em um segundo. A transição ocorrerá quando o mouse estiver pairando sobre o elemento div.

OBSERVAÇÃO VEJA-O PARA SI MESMO

Uma vez que este é um livro e transições são efeitos visuais que envolvem a mudança de propriedades gradualmente, capturas de tela não demonstram a funcionalidade muito bem. Você deve tentar o código em suas próprias páginas da web para se familiarizar com os resultados dos estilos e scripts.

O efeito visual é que esta div, que começa com um fundo cinza, vai desaparecer da vista para a direita. O código a seguir demonstra isso:

```
<html>
<head>
<style>
div {
width: 100px;
height: 100px;
background-color: gray;
margin-left: 250px;
margin-top: 250px;
transition: background-color 1s, margin-left 1s ;
}
div:hover {
margin-left: 350px;
background-color: white;
}
</style>
</head>
<body>
<div>
</div>
</body>
</html>
```

Com este código CSS no lugar, quando o usuário move o mouse sobre o elemento div, ele se moverá para a direita e sua cor de fundo mudará para branco. Como o plano de fundo da página também é branco, ele fornece o efeito de desaparecer. Você precisa entender quais propriedades estão sendo usadas

para atingir esse efeito. Neste código específico, você está usando uma propriedade abreviada chamada **transição** que permite especificar uma lista separada por vírgulas de propriedades CSS e um período de tempo para a transição da propriedade especificada a ser realizada. Essas propriedades também podem ser indicadas separadamente usando as várias propriedades CSS na Tabela 4-9.

TABELA 4-9 Propriedades de transição CSS3

Nome da Propriedade	Descrição
Transition-property	Especifica a propriedade à qual uma transição será aplicada
Transition-duration	Especifica o tempo que a transição deve levar do início ao fim
Transition-delay	Especifica quanto tempo de espera a partir do momento em que a propriedade é alterada antes de iniciar a transição

Dica de exame

Ao usar as propriedades individuais **transition-***, você pode especificar apenas uma propriedade para transição. Com a abreviação de transição, você pode especificar uma lista separada por vírgulas.

Outra propriedade que existe para controlar a velocidade das transições é a **transição-timing-function**. Esta propriedade permite que você tenha um pouco mais controle sobre a velocidade da transição. Com a propriedade **transition-timing-function**, você pode especificar alguns efeitos diferentes para o momento da transição. Os valores possíveis são especificados na Tabela 4-10.

TABELA 4-10 Valores para transição-temporização-função

Valor	Descrição
-------	-----------

ease	O valor padrão que aplica o efeito de tal forma que ele começa lento, acelera, então termina lento.
Linear	Faz a transição constante do início ao fim
Ease-in	Faz com que a transição tenha um início lento.
Ease-out	Faz com que a transição tenha um acabamento lento.
Easy-in-out	Faz com que a transição tenha um início lento e um acabamento lento.
Cubicbezier	Permite definir valores. Isso leva quatro parâmetros que são valores de 0 e 1 .

Aplicando transformações 3-D e 2-D

Usando CSS você também é capaz de aplicar transformações aos elementos em sua página da web. Nesta seção, você verá como aplicar transformações tridimensionais aos seus elementos.

Seguindo com o mesmo div da seção anterior, você aplicará as transformações tridimensionais (3-D) listadas na Tabela 4-11. As transformações bidimensionais (2-D) são cobertas no Objetivo 1.3, "Aplicar transforma".

TABELA 4-11 Transformações tridimensionais

Transformação	Efeito
Translate	Move o objeto de seu local atual para um novo local
scale	Altera o tamanho do objeto
Rotate	Gira o objeto ao longo do eixo x, do eixo y e / ou do eixo z
Matrix	Permite que você combine todas as transformações em um comando

Como você pode ver, as transformações 3-D são os mesmos valores de propriedade como as transformações 2-D. **A adição é que cada propriedade**

agora permite que você invoque a transformação através do eixo z em vez de apenas o eixo x eo eixo y. Além disso, existem propriedades abreviadas disponíveis, como `translate3d` e `rotate3d`.

Para demonstrar o uso de uma transformação 3-D, você vai olhar para a transformação girar. O código a seguir aplica uma rotação 3-D do elemento `div`.

```
div {  
    transform: rotateX(30deg) rotateY(30deg) rotateZ(30deg);  
}
```

Quando a página é carregada, todos os elementos `div` na página serão girados 30 graus ao longo de cada eixo. A transformada acima pode ser expressa desta forma também:

```
transform: rotate3d(1,1,1, 30deg);
```

Neste caso, `rotate3d` leva os primeiros parâmetros a especificar em qual eixo girar. Um valor de zero indica que não há rotação nesse eixo, enquanto que um valor de 1 indica uma rotação nesse eixo. Os parâmetros estão em ordem de eixo x, eixo y, eixo z. O último parâmetro especifica o número de graus a girar.

Quando a página é carregada, você verá a saída na Figura 4-49.

FIGURA 4-49 Um elemento `div` sendo girado em 3-D

Você pode experimentar com cada transformação. A saída é semelhante à do 2-D com a exceção de que os efeitos são aplicados ao longo do eixo z também. Além disso, você pode ver que você ainda pode usar as funções 3-D para conseguir efeitos 2-D. Tudo depende dos parâmetros que você especificar.

Ajustando a IU com base em consultas de mídia

No mundo moderno, o tamanho da tela é uma variável que agora você tem que lidar com ao construir páginas da web. Com muitas pessoas acessando a Internet a partir de diferentes dispositivos, como telefones inteligentes, tablets e desktops, não há garantia de que sua página se encaixará bem na tela e, como resultado, pode não ser amigável. Isto é onde o conceito de consultas de mídia é capaz de ajudar. Com o uso de consultas de mídia, você pode criar uma interface de usuário responsiva que será ajustada automaticamente como o tamanho das alterações de página Web renderizadas disponíveis. Usando informações do navegador, você é capaz de determinar como apresentar seu conteúdo para que ele oferece uma experiência amigável em qualquer dispositivo.

A sintaxe da consulta de mídia é tão simples como adicionar o seguinte ao seu arquivo CSS:

```
@media screen and (max-width: 800px){  
}
```

Esse código aplicará todos os estilos dentro da consulta de mídia à página quando a largura da tela não for maior que 800 px. Para obter um layout diferente para diferentes tamanhos de tela ou dispositivos, é necessário especificar uma consulta de mídia para as diferentes faixas de tamanhos. Para explorar isso, use o layout do blog que foi criado no Objetivo 4.3 usando um layout de grade. O layout padrão do blog é mostrado na Figura 4-48. No entanto, à medida que o tamanho da tela fica menor, o blog fica compactado até o ponto em que pode não ser legível ou, dependendo da quantidade de conteúdo, exigirá uma rolagem desajeitada em um dispositivo. Para acomodar os diferentes tamanhos de tela, seu código CSS pode incluir consultas de mídia. O código a seguir adiciona uma consulta de mídia para aplicar o layout padrão a telas maiores, como desktops ou laptops:

```
@media screen and (min-width: 1200px) {  
  #blogPage {  
    display: -ms-grid;  
    grid-columns: 15% 1fr 25%;  
    grid-rows: (20%)[5];  
    width: 90%;  
    height: 95%;  
    border: 1px dotted black;  
    margin: auto;  
  }  
  #blogPage > header {  
    grid-column: 1;  
    grid-column-span: 3;  
    grid-row: 1;  
    border: 1px dotted black;  
  }  
  #blogPage > footer {  
    grid-column: 1;  
    grid-row: 5;  
  }  
}
```

```
grid-column-span: 3;
border: 1px dotted black;
}
#blogPage > article {
grid-column: 2;
grid-row: 2;
grid-row-span: 3;
border: 1px dotted black;
}
#blogPage > #calendar {
grid-column: 3;
grid-row: 3;
border: 1px dotted black;
}
#blogPage > #blogRoll {
grid-column: 3;
grid-row: 4;
border: 1px dotted black;
}
#blogPage > #aboutMe {
grid-column: 1;
grid-row: 2;
grid-row-span: 3;
border: 1px dotted black;
}
#blogPage > #bloghistory {
grid-column: 3;
grid-row: 2;
border: 1px dotted black;
}
```

```
}
```

Isso produz a saída na Figura 4-48. Desde que a tela tenha pelo menos 1,200 px de largura, este layout será aplicado. No entanto, como a tela fica menor, em um tablet, por exemplo, a interface do usuário começa a ficar menos amigável. Para acomodar uma tela de tablet, você pode ajustar o layout um pouco adicionando o seguinte código CSS à página:

```
@media screen and (max-width: 1199px) and (min-width: 401px) {
```

```
#blogPage {  
  display: -ms-grid;  
  grid-columns: 75% 1fr;  
  grid-rows: (20%)[6];  
  width: 90%;  
  height: 95%;  
  border: 1px dotted black;  
  margin: auto;  
}
```

```
#blogPage > header {  
  grid-column: 1;  
  grid-column-span: 2;  
  grid-row: 1;  
  border: 1px dotted black;  
}
```

```
#blogPage > footer {  
  grid-column: 1;  
  grid-row: 6;  
  grid-column-span: 2;  
  border: 1px dotted black;  
}
```

```
#blogPage > article {  
  grid-column: 1;  
  grid-row: 3;  
  grid-row-span: 3;
```

```

border: 1px dotted black;
}
#blogPage > #calendar {
grid-column: 2;
grid-row: 4;
border: 1px dotted black;
}
#blogPage > #blogRoll {
grid-column: 2;
grid-row: 5;
border: 1px dotted black;
}
#blogPage > #aboutMe {
grid-column: 1;
grid-row: 2;
grid-column-span: 2;
border: 1px dotted black;
}
#blogPage > #bloghistory {
grid-column: 2;
grid-row: 3;
border: 1px dotted black;
}
}
}

```

Com este código, você pode reestruturar o layout da grade com base no tamanho de tela diferente. A saída desse código produz uma interface de usuário mostrada na Figura 4-50.

FIGURA 4-50 O layout do blog é ajustado para o tamanho da tela do tablete

Agora você tem a tela do tamanho do desktop e a tela do tamanho do tablet com boa aparência. O próximo a explicar é a tela do telefone inteligente menor. Adicione o seguinte código ao seu CSS para colocar uma consulta de mídia para o tamanho de tela menor:

```
@media screen and (max-width: 400px) {
```

```
#blogPage {  
  display: grid;  
  grid-columns: 50% 50%;  
  grid-rows: 15% 15% 1fr 15% 15%;  
  width: 90%;  
  height: 95%;  
  border: 1px dotted black;  
  margin: auto;  
}
```

```
#blogPage > header {  
  grid-column: 1;  
  grid-column-span: 2;  
  grid-row: 1;  
  border: 1px dotted black;  
}
```

```
#blogPage > footer {  
  grid-column: 1;  
  grid-row: 5;  
  grid-column-span: 2;  
  border: 1px dotted black;  
}
```

```
#blogPage > article {  
  grid-column: 1;  
  grid-row: 3;  
  grid-column-span: 2;  
  border: 1px dotted black;  
}
```

```
#blogPage > #calendar {  
  grid-column: 2;
```



```

grid-row: 2;
border: 1px dotted black;
}
#blogPage > #blogRoll {
grid-column: 1;
grid-row: 4;
border: 1px dotted black;
}
#blogPage > #aboutMe {
grid-column: 1;
grid-row: 2;
border: 1px dotted black;
}
#blogPage > #bloghistory {
grid-column: 2;
grid-row: 4;
border: 1px dotted black;
}
}

```

Como o tamanho da tela começa a ser tão pequeno quanto seria em um telefone inteligente, a interface do usuário será processado como mostrado na Figura 4-51.

FIGURA 4-51 O layout do blog é ajustado para o tamanho de tela de um telefone inteligente

OBSERVAÇÃO FUNCIONA?

A consulta de mídia não exige que um smartphone ou tablet esteja em uso, somente que a porta de visualização da página está dentro dos parâmetros da consulta. Para testar esse código, basta redimensionar a janela do navegador além dos tamanhos definidos para ver a página alterar seu layout dinamicamente.

Até agora, o CSS está funcionando muito bem. O único problema é que você tem um monte de código CSS em uma única página. Normalmente, um site tem várias páginas com estilos compartilhados entre páginas diferentes. Como resultado, você provavelmente estará ligando um arquivo CSS externo para sua

página HTML. O elemento de link também suporta consultas de mídia, o que por sua vez permite compartilhar um arquivo CSS em várias páginas. Por exemplo, você pode querer alterar o tamanho da fonte do texto em todo o site com base no tamanho da porta de exibição. Para conseguir isso, mova o CSS de cada consulta de mídia para seu próprio arquivo CSS e link em seus arquivos CSS da seguinte maneira:

```
<link rel="stylesheet" media="screen and (min-width: 1200px)" href="Desktop.css"/>
```

```
<link rel="stylesheet" media="screen and (max-width: 1199px) and (min-width: 401px)" href="tablet.css"/>
```

```
<link rel="stylesheet" media="screen and (max-width: 400px)" href="phone.css"/>
```

Com o CSS vinculado desta forma, você pode adicionar e modificar os estilos para as diferentes portas de exibição centralmente para todo o seu site.

Ocultando ou desativando controles

A capacidade de modificar o posicionamento da interface do usuário usando consultas de mídia como mostrado na última seção é muito útil. Além disso, alguns layouts podem não funcionar em algumas portas de exibição. Neste caso, convém concluir os controles de **ocultar** ou **desativar** os controles. Os elementos HTML são visíveis por padrão. No entanto, eles podem ser tornados invisíveis definindo a propriedade CSS de visibilidade como mostrado no código a seguir:

```
.myhiddenelements {  
    visibility: hidden;  
}
```

Ao definir a visibilidade como oculta, o controle não é visível para o usuário final da página da Web. Ao ocultar um elemento usando a propriedade de **visibility**, o layout geral ainda se comporta como se o elemento estivesse lá. Se você preferir ter o elemento oculto eo layout se comportar como se ele não estivesse lá, a propriedade de **display** deve ser usada como mostrado no código a seguir:

```
.myhiddenelements {  
    display: none;  
}
```

Com a propriedade **display**, o elemento não é visível eo layout não é afetado por ele.

Se você não quiser ocultar o elemento, mas apenas torná-lo desativado para que o usuário possa vê-lo, mas não pode executar nenhuma ação nele, você precisa adicionar o atributo diretamente ao elemento HTML. Como tal, você

pode definir uma classe CSS que você pode aplicar a qualquer elemento que você deseja desativar:

```
.disableIt
```

Agora que você tem uma classe CSS chamada `.disableIt`, você pode aplicar esta classe a qualquer elemento que você deseja desabilitar. Neste caso, você deseja desativar um elemento de botão, então aplique a classe ao elemento de botão como mostrado aqui:

```
<button id="myButton" class="disableIt" >My Button</button>
```

A última etapa é criar algum JavaScript que localize todos os controles com essa classe atribuída a ele e adicione o atributo `disabled` para eles. O código a seguir demonstra isso:

```
$( "document" ).ready(function (e) {  
    $(".disableIt").attr("disabled", "disabled");  
});  
</script>
```

Este script tem o mesmo efeito líquido que colocar o atributo diretamente no elemento do botão como mostrado aqui:

```
<button id="myButton" disabled="disabled">My Button</button>
```

Quando você tem muitos elementos que você gostaria de desativar, é muito mais fácil criar uma classe CSS, aplicá-la aos elementos e, em seguida, usando jQuery, aplicar o atributo `disabled` a todos eles.

Experimento de pensamento

Combinando efeitos Neste experimento de pensamento, aplique o que aprendeu sobre esse objetivo. Você pode encontrar a resposta a esta pergunta na seção "Respostas" no final deste capítulo. Considere combinar transições com transformações. Individualmente, transições e transformações fornecem efeitos interessantes para a página HTML. Considere que tipo de efeitos podem ser alcançados usando-os juntos.

Resumo do objetivo

■ Os elementos HTML podem ser manipulados com transições usando a propriedade de transição,

Transição-duração e propriedades de CSS de atraso de transição.

■ Os elementos podem ser manipulados em espaço 2-D e 3-D com efeitos como traduzir, dimensionar, girar e matriz.

■ As consultas de mídia permitem que você tenha uma interface de usuário dinâmica e responsiva com base no tamanho e tipo da porta de exibição.

■ A propriedade de visibilidade esconde um controle, mas mantém sua posição no layout geral. Usando a propriedade de exibição para ocultar um controle remove-lo do layout também.

Revisão objetiva

Responda às seguintes perguntas para testar seu conhecimento da informação neste objetivo. Você pode encontrar as respostas a essas perguntas e explicações de por que cada opção de resposta está correta ou incorreta na seção "Respostas" no final deste capítulo.

1. Qual das seguintes afirmações esconderá um elemento, mas manterá seu espaço no fluxo geral?

- A. `display: none;`
- B. `visibility: hidden;`
- C. `display: inline;`
- D. `visibility: visible;`

2. Consultas de mídia são mais adequadas para que finalidade?

A. Definindo a prioridade das referências de folhas de estilos em uma página da Web

B. Criando uma interface de usuário responsiva com base no tamanho de tela da porta de exibição

C. Modificando a porta de exibição para ajustar corretamente o conteúdo da página

D. Conexão com folhas de estilo de terceiros para alterar o layout

3. Qual das seguintes propriedades da função de sincronização de transição faz com que a transição seja iniciada Lento, acelerar e, em seguida, terminar lento?

- A. `ease`
- B. `ease-in`
- C. `ease-out`
- D. `ease-in-out`

Objetivo 4.5: Localizar elementos usando seletores CSS e jQuery

Objetivo 1.2, "escrever código que interage com controles de interface do usuário," cobriu o uso de seletores de documento para localizar elementos HTML

por seu nome exclusivo. Nesta seção, você explora técnicas mais avançadas para encontrar elementos através do uso de seletores CSS e jQuery.

Este objetivo abrange como:

- Definir seletores de elementos, estilos e atributos
- Escolha o seletor correto para fazer referência a um elemento
- Localizar elementos usando **pseudo-elementos** e **pseudo-classes**

Definindo seletores de elementos, estilos e atributos

O CSS usa seletores que você define em um arquivo CSS ou bloco de estilo para identificar quais elementos em uma página da Web os estilos definidos devem ser aplicados. Isso pode ser feito **especificando o próprio elemento como o seletor**. Por exemplo, o seguinte CSS é um seletor de elemento para o elemento div:

```
div{  
...  
}
```

Qualquer div no escopo da declaração terá os estilos aplicados a ele. **Outro possível seletor é um seletor de classe**. Para usar um seletor de classe, você define um nome de classe personalizado no arquivo CSS. Este pode ser **qualquer nome prefixado com um ponto**. Em seguida, qualquer elemento que tenha essa classe atribuída a ele por meio do atributo **class** terá os estilos definidos aplicados. Por exemplo:

```
.class{  
....  
}
```

Outra maneira de usar o CSS para selecionar elementos específicos na página é usar **a seleção de atributos**. Isto é conseguido **especificando um tipo de elemento seguido de um atributo específico**. Por exemplo, se você tiver um formulário da web que precisa ser preenchido, você pode atribuir campos obrigatórios com uma borda vermelha em torno das caixas de texto. O código a seguir atinge isso para todos os elementos que possuem o atributo especificado:

```
input[required] {  
border: 1px red solid;  
}
```

Há outras possibilidades para o uso de seletores de atributo. Estes são delineados em Tabela 4-12.

TABELA 4-12 Capacidades do seletor de atributos

Selector de atributo	Descrição
=	Especifica que um atributo é igual a um valor específico. Por exemplo, o URL de uma âncora é um URL de especificação.
~ =	Especifica uma lista de palavras separada por espaços como os valores de um atributo.
^ =	Especifica que o atributo tem um valor começando com o texto especificado.
\$ =	Especifica que o atributo tem um valor que termina com o texto especificado.
* =	Especifica que o atributo tem um valor que contém o texto especificado.

Escolhendo o seletor correto para fazer referência a um elemento

Escolher o seletor correto para fazer referência a um elemento é uma consideração importante. Você precisa garantir que você organize seus seletores e seus elementos de modo que somente os elementos desejados sejam afetados pelos estilos definidos. Por exemplo, o estilo a seguir afeta todos os elementos article:

```
article{  
    border-color: 1px solid red;  
}
```

Se você não quiser afetar todos os artigos, mas apenas o artigo mais recente, deve distingui-los, talvez adicionando uma classe CSS personalizada à definição e atribuindo isso somente ao novo artigo:

```
article.newest{  
    border-color: 1px solid red;  
}
```

Ao especificá-lo desta forma, você está certo de que nem todo artigo na página é afetado pelo estilo. Este tópico é abordado em mais detalhes no Objetivo 4.6 na seção "Referenciando elementos corretamente."

Encontrando elementos usando pseudo-elementos

Pseudo-classes e pseudo-elementos fornecem algumas maneiras muito poderosas para adicionar estilos aos elementos. As pseudo-classes permitem que você aplique estilos a um elemento com base em seu estado, sua interação

com o usuário ou sua posição no documento. Pseudo-elementos permitem inserir conteúdo na página em locais relativos aos elementos aos quais o CSS está sendo aplicado. Você examinará cada uma das pseudo-classes e pseudo-elementos comuns nesta seção.

:Link, :visited, e :hover

Estas são as pseudo-classes mais usadas, usadas com mais frequência com o elemento **âncora**, fornecendo um link clicável para o usuário da página da web. Com essas pseudo-classes, você pode controlar quais estilos são aplicados a um hyperlink nos diferentes estados. Por exemplo, o seguinte CSS altera a cor do link com base em seu estado:

```
a:link {  
    color: green;  
}  
a:hover {  
    color: red;  
}  
a:visited {  
    color: black;  
}
```

Neste exemplo, o link por padrão será verde. Quando um usuário move o mouse sobre o link, a cor do link mudará para vermelho. Se o usuário não clicar no link e, em seguida, sair dele, o link voltará a verde. No entanto, se o usuário clicar no link, ele se tornará um link visitado e será alterado para preto.

:checked

A pseudo-klasse checked: permite aplicar estilos a elementos que estão em um estado marcado. Os elementos que suportam esta pseudo-klasse são caixas de verificação (checkbox) e botões (radio buttons) de opção. A quantidade de estilo que você pode aplicar aos elementos padrão é mínima. No entanto, existem amplos recursos para personalizar esses elementos usando CSS. O exemplo a seguir mostra como ocultar uma caixa de seleção quando um usuário clica nele.

```
input[type="radio"]:checked  
input[type="checkbox"]:checked {  
    display: none; }
```

:required

A pseudo-classe: **required** permite aplicar estilos a qualquer elemento na página que tenha o atributo **required**. Esta é uma maneira conveniente para realçar campos obrigatórios em um formulário. O seguinte CSS demonstra aplicar estilos para todos os controles de entrada necessários:

```
input:required {  
  border: 2px solid red;  
}
```

Todos os controles de entrada necessários terão agora uma borda vermelha para destacar isso para o usuário.

:enabled and :disabled

As pseudo-classes: **enabled** e: **disabled** permitem que você controle o estilo com base no seu estado **habilitado** ou **desabilitado**. Por padrão, os controles desativados geralmente são cinza claro. Com essas pseudo-classes, você pode controlar como o elemento é exibido em qualquer estado. O código a seguir demonstra isso:

```
input:disabled {  
  background-color: blue;  
}  
  
input:enabled {  
  background-color: white;  
}
```

Se um controle estiver habilitado, o plano de fundo será branco; Caso contrário controles desabilitados serão azul.

:first-child

O pseudo-elemento **first-child** aplica os estilos especificados à primeira instância do elemento que ocorre em uma lista, por exemplo, o elemento do primeiro parágrafo neste HTML:

```
<div>  
<p>Lorem Ipsum ...</p>  
<p>Lorem Ipsum ...</p>  
<p>Lorem Ipsum ...</p>
```



```
<p>Lorem Ipsum ...</p>
```

```
</div>
```

O seguinte CSS irá alterar a cor do texto para verde no elemento do primeiro parágrafo:

```
p:first-child {  
  color:green;  
}
```

:first-letter

O pseudo-elemento :first-letter irá **alterar o estilo da primeira letra no elemento especificado**. Continuando com o exemplo HTML acima, o seguinte CSS irá aumentar o tamanho da primeira letra em cada elemento de parágrafo:

```
p::first-letter {  
  font-size: xx-large;  
}
```

:before and :after

Os pseudo-elementos **before** e **after** adicionam o conteúdo especificado **na frente ou depois do seletor de elementos indicado**. Portanto, o código a seguir adicionaria ** à frente e ao final do elemento de parágrafo:

```
p::before {  
  content: '**';  
}  
  
p::after {  
  content: '**';  
}
```

:first-line

O pseudo-elemento de **first-line** **altera os estilos da primeira linha de um elemento de texto**. O seguinte CSS fará a primeira linha do texto dentro do elemento parágrafo verde e maior:

```
p::first-line {  
  color:green;  
  font-size: x-large;  
}
```

Experimento de pensamento

Usando jQuery com pseudo-classes Neste experimento de pensamento, aplique o que você aprendeu sobre esse objetivo. Você pode encontrar uma resposta a esta pergunta na seção "Respostas" no final deste capítulo. Considere como você pode usar jQuery para selecionar elementos usando pseudo-classes e pseudo-elementos. Usando apenas jQuery, aplique estilos CSS ao primeiro parágrafo de qualquer grupo de elementos de parágrafo.

Resumo do objetivo

■ ■ Pseudo-elementos e pseudo-classes fornecem um mecanismo avançado para pesquisar elementos HTML em uma página e aplicar estilos.

■ ■ Usando pseudo-elementos e pseudo-classes você pode mudar o estilo de um elemento com base nas ações do usuário.

■ ■ Usando pseudo-elementos e pseudo-classes você pode ganhar controle granular sobre partes do texto em um bloco de texto.

Revisão objetiva

Responda às seguintes perguntas para testar seu conhecimento da informação neste objetivo. Você pode encontrar as respostas a estas perguntas e explicações de por que cada resposta ch oice está correta ou incorreta na seção "Respostas" no final deste capítulo.

1. Qual dos seguintes é um seletor de classe CSS?

A. `.code`

B. `#code`

C. `div[code]`

D. `:code`

2. Qual das seguintes opções é um seletor de atributos?

A. `.required`

B. `#required`

C. `input[required]`

D. `:required`

3. Qual das seguintes afirmações alteraria o estilo de um elemento de âncora quando o mouse é movido sobre ele?

A. `a:link`

B. `a:mouseover`

C. `a:beforeclick`

D. `a:hover`

Objetivo 4.6: Estruturar um arquivo CSS usando seletores CSS

Os arquivos CSS podem se tornar grandes e complexos. Estruturá-los de uma forma organizada vai torná-los mais fáceis de manter e também saber quais os seletores são mais adequados para ser usado para fazer referência ao HTML em sua página.

Este objetivo abrange como:

- ■ Elementos de referência corretamente
- Implementar herança
- ■ Substituir herança usando **! Importante**
- ■ Estilo de um elemento baseado em pseudo-elementos e pseudo-classes

Referenciando elementos corretamente

CSS é usado para aplicar estilos a elementos em uma página HTML. Para fazer isso, o CSS tem que saber quais os elementos para aplicar os estilos. Existem algumas maneiras de fazer referência a elementos do CSS.

Isso é conhecido como a sintaxe do seletor. Isso foi demonstrado ao longo do capítulo. Esta seção explicará especificamente como referenciar elementos do CSS. A principal consideração é garantir que você faça referência a elementos tais que os estilos afetam apenas os elementos que você deseja. Em CSS complexo grande, pode ficar complicado.

Os elementos **podem ser referenciados a partir do CSS pelo seu nome de elemento**. Por exemplo:

```
p{...}
```

```
article{...}
```

```
div{...}
```

Neste código, os estilos são aplicados a todos os elementos de parágrafo, elementos do artigo e div Elementos. O próximo método para selecionar elementos é através do uso de **classes** como mostrado aqui:

```
.bold{...}
```

```
.largeTitle{...}
```

Neste código, os estilos são aplicados apenas a elementos HTML que têm o respectivo atributo de classe atribuído estes nomes de classe. Nomes de elementos e classes podem ser combinados para estreitar ainda mais o marcador:

```
p.largeTitle{...}
```

Este código aplica os estilos apenas aos elementos de parágrafo que têm a classe largeTitle Atribuído ao atributo de classe.

O método mais granular para referenciar elementos HTML do CSS é usando o id ou nome do elemento:

```
#nameBox{...}
```

Este código aplica os estilos especificados apenas ao elemento único na página com o especificado. Em alguns casos, convém aplicar o mesmo estilo a muitos elementos de tipos diferentes. Nesse caso, você pode agrupá-los e definir os estilos apenas uma vez:

```
p, H1, H2 {...}
```

Neste exemplo, todos os três elementos HTML observados terão os estilos definidos aplicados a eles.

Implementando herança

Alguns estilos aplicados a um elemento pai são automaticamente herdados por elementos filhos. Por exemplo, se uma série de elementos de parágrafo estiver dentro de um elemento de artigo, e os estilos de fonte e de texto forem aplicados ao artigo, todos os elementos de parágrafo herdarão automaticamente os estilos de fonte e de texto também. O código a seguir demonstra esse conceito:

```
<style>
```

```
div {
```

```
font-family: sans-serif;
```

```
color: green;
```

```
}
```

```
</style>
```

And with the following HTML:

```
<div>
```

```
hello div world.
```

```
<p>
```

Hello paragraph world.

</p>

</div>

Tanto o div quanto o parágrafo terão os estilos de fonte e cor aplicados a eles porque o elemento de parágrafo não tem nenhum de seus próprios estilos definidos. Se você atribuir estilos ao elemento de parágrafo para substituir os estilos div, seria possível impedir a herança dos estilos, como mostrado aqui:

<style>

div {

font-family: sans-serif;

color: green;

}

p {

font-family: serif;

color: blue;

}

</style>

Substituindo herança usando !Importante

CSS para grandes sites pode ser complicado. Sites grandes podem ter CSS provenientes de diferentes fontes. Poderia estar em cada página e referenciado externamente. Bibliotecas externas são cada vez mais comuns como especialistas em toda a comunidade criaram temas que podem ser importados em seus aplicativos da web. Com todo esse estilo vindo de fontes diferentes, a herança de estilos pode ser complicada. Em alguns casos, você pode apenas precisar substituir competindo estilos completamente com seu próprio estilo desejado. Este é o lugar onde a palavra-chave **importante** vem dentro considere o seguinte exemplo CSS simples:

p {

font-family: serif;

color: blue;

}

p {

color: purple;

```
}  
  
p{  
color: yellow;  
}
```

Neste código CSS, você tem três estilos concorrentes para o elemento de parágrafo. Como o navegador processa isso baseado no último estilo que ele lê para um elemento ao qual ele se aplica. Assim, neste caso, o texto em todos os parágrafos será amarelo. No entanto, se você quiser substituir esse comportamento e forçar os elementos de parágrafo na página para ser roxo, basta adicionar a palavra-chave **!important** para o estilo que você deseja ser aplicado:

```
p{  
font-family: serif;  
color: blue;  
}  
  
p{  
color: purple !important;  
}  
  
p{  
color: yellow;  
}
```

Os elementos do parágrafo tornarão roxo e não amarelo. A notação **important** diz ao analisador que dê essa prioridade de estilo. Este é um exemplo simplista, mas o conceito é o mesmo se os estilos estão em uma única página como esta ou se eles vêm de uma variedade de fontes externas com estilos conflitantes.

Styling um elemento baseado em pseudo-elementos e pseudo-classes

A seção Objetivo 4.5, "Encontrando elementos usando pseudo-elementos e pseudo-classes" demonstrou o uso de pseudo-elementos e pseudo-classes como seletores. Além disso, essa seção pode ser referenciada para como aplicar estilos a elementos com base no uso dos seletores pseudo-classe e pseudo-elemento.

Experimento de pensamento

Combinando seletores de pseudo-elementos Nesta experiência de pensamento, aplique o que você aprendeu sobre esse objetivo. Você pode encontrar uma resposta a esta pergunta na seção "Respostas" no final deste capítulo. Considere como você pode obter estilos muito específicos aplicados aos seus elementos HTML através da combinação de pseudo-classes e pseudo-elementos. Por exemplo, como você mudaria a aparência da primeira letra do primeiro parágrafo de cada grupo de parágrafos na página, mas somente quando o usuário paira sobre ela? Experimente outras combinações de seletores para obter efeitos específicos.

Resumo do objetivo

- Referenciar elementos corretamente leva consideração cuidadosa de como você irá estruturar seu CSS e seus elementos HTML.

- Os seletores podem ser aninhados e unidos para se tornarem mais específicos.

- Os elementos HTML herdam estilos automaticamente de seus elementos pai.

- O CSS é processado de cima para baixo, de modo que o **último estilo processado ganha se entrar em conflito com outras declarações de estilo**.

- **! Importante** pode ser usado para garantir que o estilo desejado seja renderizado quando houver uma declaração CSS concorrente.

Revisão objetiva

Responda às seguintes perguntas para testar seu conhecimento da informação neste objetivo. Você pode encontrar as respostas a essas perguntas e explicações de por que cada opção de resposta está correta ou incorreta na seção "Respostas" no final deste capítulo. As perguntas de revisão usam a seguinte lista HTML (os números de linha são apenas para referência):

1. <html>
2. <body>
3. <div>
4. <hgroup>
5. <h1></h1>
6. <h2></h2>
7. </hgroup>
8. </div>
9. <div>
10. <section>

11. <article>
12. <h1></h1>
13. <p></p>
14. <p></p>
15. </article>
16. <article>
17. <h1></h1>
18. <p></p>
19. <aside></aside>
20. <p></p>
21. </article>
22. </section>
23. </div>
24. <div>
25. <footer>
26. <p></p>
27. <p></p>
28. </footer>
29. </div>
30.</body>
31.</html>

1. Referenciando a listagem HTML, como você classificaria apenas o primeiro parágrafo dentro do elemento de rodapé para ter um tamanho de fonte menor?

A.

```
footer p:first-child {  
font-size: x-small;  
}
```

B.

```
footer p.first-child {  
font-size: x-small;
```



```
}
```

C.

```
Footer:p:first-child {
```

```
font-size: x-small;
```

```
}
```

D.

```
Footer=>p:first-child {
```

```
font-size: x-small;
```

```
}
```

2. Referenciando a lista HTML, como você aplicaria uma nova fonte a todos os elementos H1? Além disso, os elementos <h1> em um artigo devem ser itálico.

A.

```
h1 {
```

```
font-family: 'Courier New';
```

```
article h1 {
```

```
font-style:italic;
```

```
}
```

```
}
```

B.

```
h1 {
```

```
font-family: 'Courier New';
```

```
}
```

```
article h1 {
```

```
font-style:italic;
```

```
}
```

C.

```
h1 {
```

```
font-family: 'Courier New';
```

```
font-style:italic;
```

```
}
```

```
article h1 {  
  font-style:italic;  
}  
D. h1 {  
  font-family: 'Courier New';  
}  
article, h1 {  
  font-style:italic;  
}
```

3 - Fazendo referência à listagem HTML anterior, escreva o código CSS para aplicar uma lado que é 100 pixels de altura e 50 pixels de largura. Além disso, fornecer um efeito de sombra e ligeiramente distorcer o elemento para a direita 5 graus.