Promise

This is a new technology, part of the ECMAScript 2015 (ES6) standard.
This technology's specification has been finalized, but check the compatibility table for usage and implementation status in various browsers.

`Promise` é um objeto usado para processamento assíncrono. Um `Promise` (*de "promessa"*) representa um valor que pode estar disponível agora, no futuro ou nunca.

Sintaxe

```
new Promise(/* executor */ function(resolve, reject) { ... });
```

Parâmetros

executor
: Uma função que recebe dois argumentos `resolve` and `reject`. Esta função é executada imediatamente pela implementação do `Promise`, passando as funções `resolve` and `reject`.O executor é chamado antes que o construtor de `Promise` retorne o objeto criado.

  As funções `resolve` e `reject`, quando chamadas, *resolvem* (em caso de sucesso) ou *rejeitam* (quando ocorre um erro) a promessa, respectivamente. O executor começa o trabalho assíncrono que, quando concluído, chama uma das funções `resolve` ou `reject` para definir o estado da promessa.

Descrição

Um **`Promise`** representa um proxy para um valor que não é necessariamente conhecido quando a promessa é criada. Isso permite a associação de métodos de tratamento para eventos da ação assíncrona num caso eventual de sucesso ou de falha. Isto permite que métodos assíncronos retornem valores como métodos síncronos: ao invés do valor final, o método assíncrono retorna uma *promessa* ao valor em algum momento no futuro.

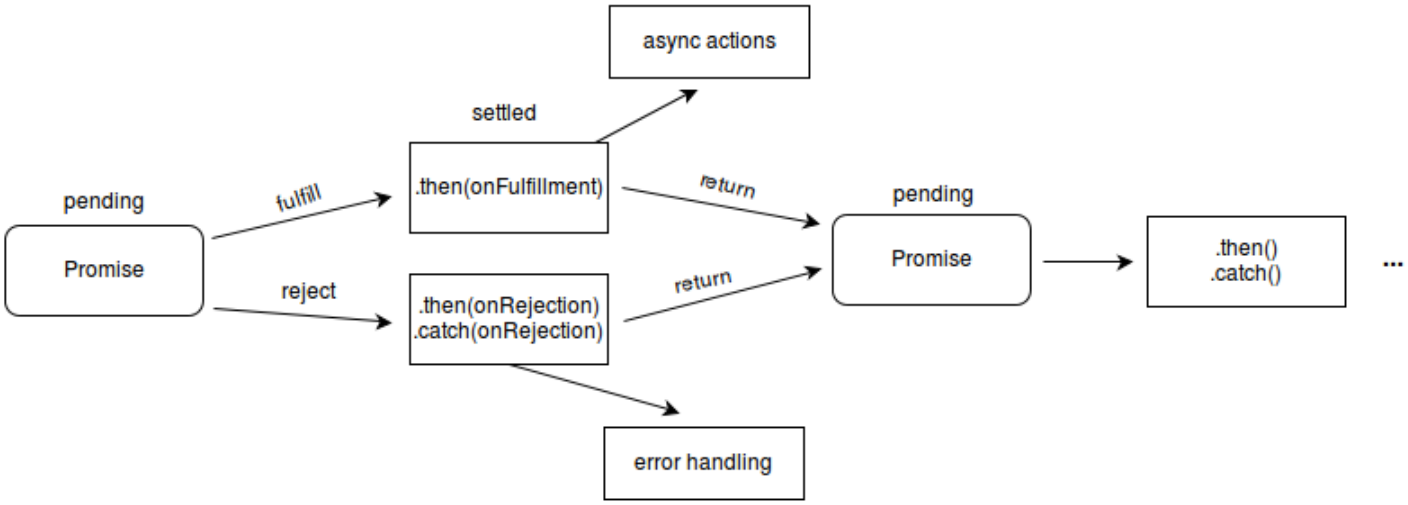
Um **`Promise`** está um destes estados:

- *pending (*pendente*)*: Estado inicial, que não foi realizada nem rejeitada.
- *fulfilled (*realizada): sucesso na operação.
- *rejected (*rejeitado): falha na operação.
- settled (estabelecida): Que foi realizada ou rejeitada.

Uma promessa pendente pode se tonar *realizada* com um valor ou *rejeitada* por um motivo (erro). Quando um desses estados ocorre, o método `then` do `Promise` é chamado, e ele chama o método de tratamento associado ao estado (`rejected` ou `resolved`). (Se a promessa foi realizada ou rejeitada quando o método de tratamento

correspondente for associado, o método será chamado, deste forma não há uma condição de competição entre uma operação assíncrona e seus manipuladores que estão sendo associados.)

Como os métodos `Promise.prototype.then` e `Promise.prototype.catch` retornam promises, eles podem ser encadeados — uma operação chamada *composição*.



Propriedades

`Promise.length`

Propriedade length cujo valor é 1 (número de argumentos do método construtor).

`Promise.prototype`

Representa o protótipo para o método construtor da `Promise`.

Métodos

`Promise.all(lista)`

Retorna uma promise que é resolvida quando todas as promises no argumento *lista* forem resolvidas ou rejeitada assim que uma das promises da lista for rejeitada. Se a promise retornada for resolvida, ela é resolvida com um array dos valores das promises resolvidas da lista. Se a promise for rejeitada, ela é rejeitada com o motivo da promise que foi rejeitada na lista. Este método pode ser útil para agregar resultados de múltiplas promises.

`Promise.race(lista)`

Retorna uma promise que resolve ou rejeita assim que uma das promises do argumento lista resolve ou rejeita, com um valor ou o motivo daquela promise.

`Promise.reject(motivo)`

Retorna um objeto `Promise` que foi rejeitado por um dado motivo.

`Promise.resolve(valor)`

Retorna um objeto `Promise` que foi resolvido com um dado valor. Se o valor possui um método `then (thenable)`, a promise retornada "seguirá" este método, adotando esse estado eventual; caso contrário a promise retornada será realizada com o valor. Geralmente, se você quer saber se um valor é uma promise ou não, utilize `Promise.resolve(valor)` e trabalhe com a valor de retorno que é sempre uma promise.

Protótipo `Promise`

Propriedades

`Promise.prototype.constructor`
Returns the function that created an instance's prototype. This is the `Promise` function by default.

Métodos

`Promise.prototype.catch(onRejected)`
Appends a rejection handler callback to the promise, and returns a new promise resolving to the return value of the callback if it is called, or to its original fulfillment value if the promise is instead fulfilled.

`Promise.prototype.then(onFulfilled, onRejected)`
Appends fulfillment and rejection handlers to the promise, and returns a new promise resolving to the return value of the called handler, or to its original settled value if the promise was not handled (i.e. if the relevant handler `onFulfilled` or `onRejected` is not a function).

Exemplos

Criando uma Promise

Este pequeno exemplo mostra o mecanismo de uma `Promise`. O método `testPromise()` é chamado cada vez que `<button>` é clicado. Isso cria uma promise que resolverá, usando `window.setTimeout()`, o contador de promise `promiseCount` (iniciando em 1) a cada 1 a 3s randomicamente. O construtor `Promise()` é usado para criar a promise.

A realização da promise é simplesmente registrada, por meio de configuração na função callback de realização usando `p1.then()`. Alguns logs mostram como a parte síncrona do método é desacoplada da conclusão assíncrona da promise.

```
var promiseCount = 0;
function testPromise() {
  var thisPromiseCount = ++promiseCount;

  var log = document.getElementById('log');
  log.insertAdjacentHTML('beforeend', thisPromiseCount +
      ') Started (<small>Sync code started</small>)<br/>');

  // Criamos uma nova promise: prometemos a contagem dessa promise (após aguardar
  var p1 = new Promise(
    // a função resolve() é chamada com a capacidade para resolver ou
    // rejeitar a promise
    function(resolve, reject) {
      log.insertAdjacentHTML('beforeend', thisPromiseCount +
          ') Promise started (<small>Async code started</small>)<br/>');
      // Isto é apenas um exemplo para criar assincronismo
      window.setTimeout(
        function() {
```

```
            // Cumprimos a promessa !
            resolve(thisPromiseCount)
          }, Math.random() * 2000 + 1000);
      });

    // definimos o que fazer quando a promise for realizada
    p1.then(
      // apenas logamos a mensagem e o valor
      function(val) {
        log.insertAdjacentHTML('beforeend', val +
            ') Promise fulfilled (<small>Async code terminated</small>)<br/>');
      });

    log.insertAdjacentHTML('beforeend', thisPromiseCount +
        ') Promise made (<small>Sync code terminated</small>)<br/>');
}
```

Este exemplo é executado pelo click do botão. Você precisa de uma versão de navegedor com suporte a `Promise`. Clicando algumas vezes no botão num curto intervalo de tempo, você verá as diferentes promises sendo realizadas uma após a outra.

Make a promise!

Carregando uma imagem com XHR

Outro simples exemplo usando `Promise` e `XMLHTTPRequest` para carregar imagens está disponível no repositório GitHub MDN promise-test. Você também pode vê-lo em ação. Cada passo é comentado e lhe permite acompanhar de perto a arquitetura de Promise e XHR.

Especificações

Especificação	Status	Comentário
domenic/promises-unwrapping	Draft	Standardization work is taking place here.
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Promise' in that specification.	ST Standard	Initial definition in an ECMA standard.

Compatibilidade de Navegador

Desktop Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	32	24.0 (24.0) as `Future` 25.0 (25.0) as `Promise` behind a flag[1] 29.0 (29.0) by default	Não suportado	19	7.1

[1] Gecko 24 tem uma implementação experimental de `Promise`, sob o nome inicial de `Future`. Isto foi renomeado para o nome final no Gecko 25, mas desabilitado por padrão sob a flag `dom.promise.enabled`. Bug 918806 habilitado suporte a Promise por padrão no Gecko 29.

Veja também

- Promises/A+ specification
- Jake Archibald: JavaScript Promises: There and Back Again
- Domenic Denicola: Callbacks, Promises, and Coroutines – Asynchronous Programming Patter in JavaScript
- Matt Greer: JavaScript Promises ... In Wicked Detail
- Forbes Lindesay: promisejs.org