
PROMISE IS A BIG WORD. IT EITHER MAKES SOMETHING OR IT BREAKS EVERYTHING
Lifehack

A *promise* represents the eventual result of an asynchronous operation. The primary way of interacting with a promise is through its *then* method, which registers callbacks to receive either a promise's eventual value or the reason why the promise cannot be fulfilled.

Nat

C ri


"INTERNET EXPLORER HAS STOPPED WORKING."
FALSE. THAT WOULD MEAN THAT INTERNET EXPLORER WAS WORKING TO BEGIN WITH.

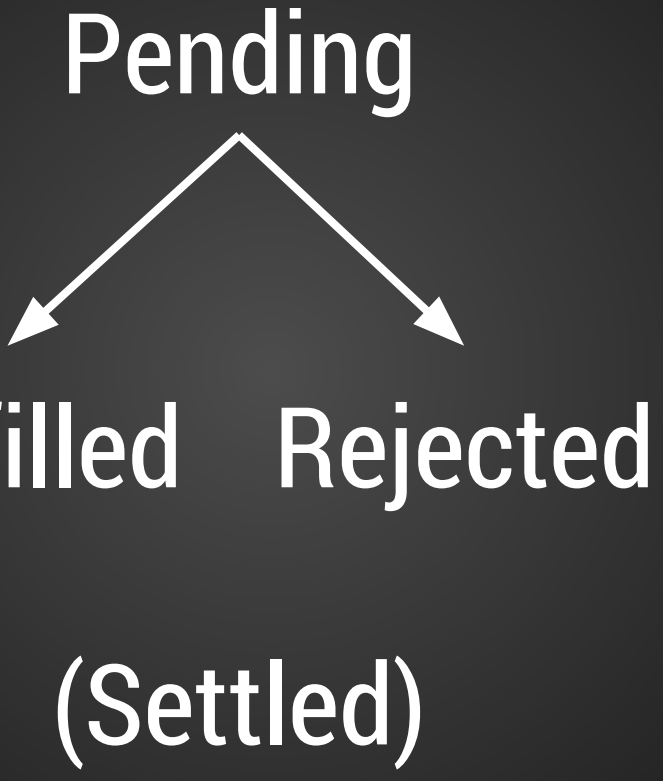
Promises/A+

Q / when / WinJS / RSVP.js

~~jQuery Deffereds / Promise~~

http://promisesaplus.com/

State



```
var img1 = document.querySelector('.img-1');

img1.addEventListener('load', function() {
  // woo yey image loaded
});

img1.addEventListener('error', function() {
  // argh everything's broken
});
```

Ett promise lyckas eller misslyckas **en gång.**

Ett promise kan **inte byta** från att vara lyckat till misslyckat eller vice versa.

Om ett promise har lyckats eller misslyckats och du lägger till callbacks senare anropas rätt callback.

```
var myPromise = new Promise(function(resolve, reject) {
  // do a thing, possibly async, then…

  if (/* everything turned out fine */) {
    resolve("Stuff worked!");
  }
  else {
    reject(Error("It broke"));
  }
});
```

Then

Catch

Resolve

Reject

All

Race

Then

```
myPromise.then(function(myString) {
  // myString = “Stuff worked!”
}, function(err) {
  // Err = Error(“It broke”)
});
```

“Thenables”

```
var jsPromise = Promise.resolve($.ajax('/whatever.json'));
```

```
function(response, statusText, xhrObj) { }
```

Catch

```
myPromise.then(null, function(err) {
  // Err = Error(“It broke”)
});

myPromise.catch(function(err) {
  // Err = Error(“It broke”)
});
```

Resolve / Reject

```
Promise.resolve("Lyckades!");

Promise.reject("Misslyckades!");
```

Resolve / Reject

```
var promise = new Promise(function (resolve, reject) {
  resolve(JSON.parse("detta är inte JSON!"));
});

promise.then(function () {
  console.log("OK!");
}, function () {
  console.log("Error!");
});

// Error!
```

All

```
var p1 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 500, "one");
});

var p2 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 100, "two");
});

Promise.all([p1, p2]).then(function(value) {
  // value === [“one”, “two”]
});
```

Race

```
var p1 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 500, "one");
});

var p2 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 100, "two");
});

Promise.race([p1, p2]).then(function(value) {
  // value === "two"
});
```

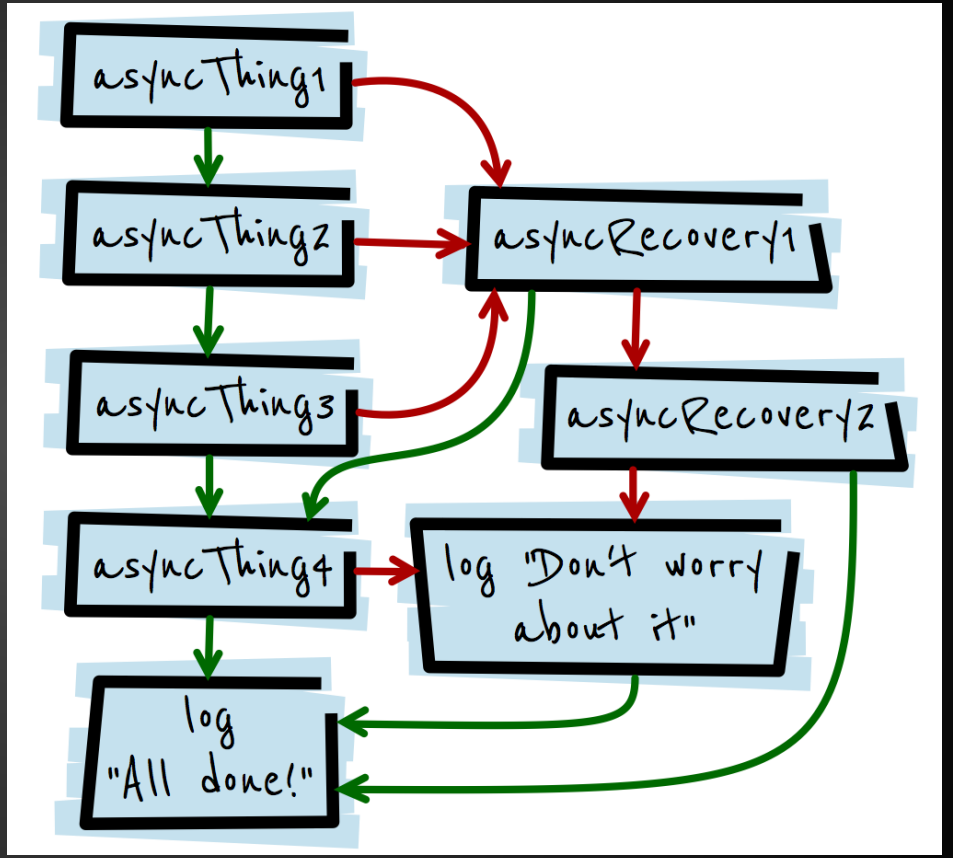
Chaining

```
Promise.resolve(5).then(function (num) {
  return num * 2;
}).then(function (num) {
  return Promise.resolve(num + 5);
}).then(function (num) {
  // num === 15
});
```

KOM IHÅG RETURN!

```
var p = Promise.resolve(asyncThing());

for (var i = 0; i < 10; i++) {
  p = p.then(/* gör något med resultat */);
}

p.then(/* gör tillslut något */);
```

```
asyncThing1().then(function() {
  return asyncThing2();
})

.then(function() {
  return asyncThing3();
})

.catch(function(err) {
  return asyncRecovery1();
})

.then(function() {
  return asyncThing4();
}, function(err) {
  return asyncRecovery2();
})

.catch(function(err) {
  console.log("Don't worry about it");
})

.then(function() {
  console.log("All done!");
});
```



https://github.com/jayway/javascript-promises-lab