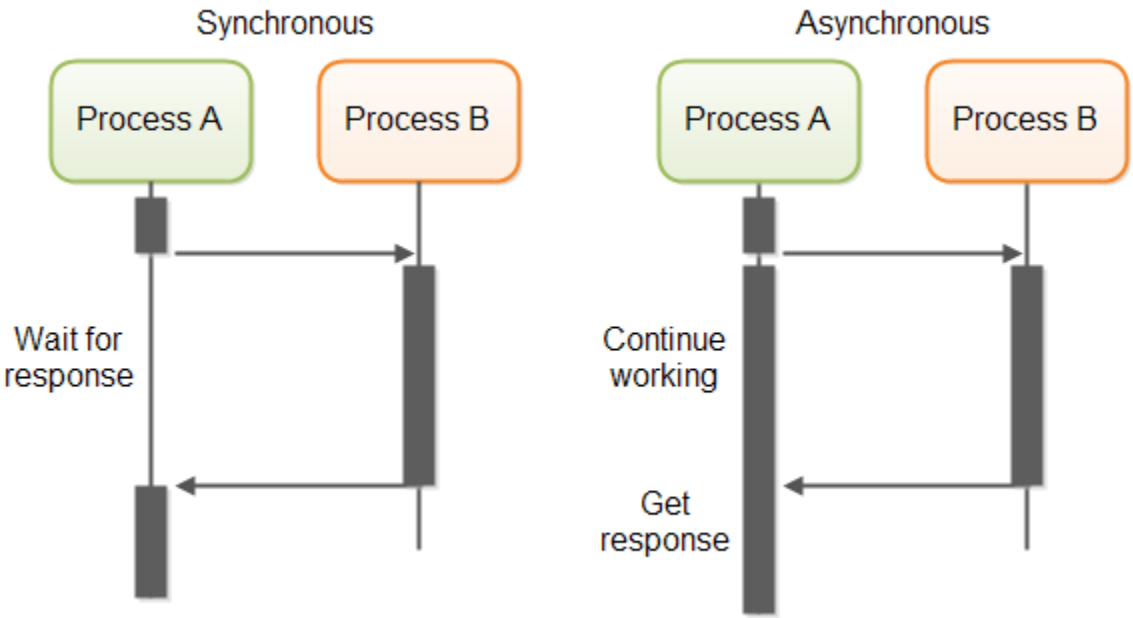

HENRY

Promises

Promises


Synchronous
Process A
Process B
Wait for response
Asynchronous
Process A
Process B
Continue working
Get response

¿Qué es una Promise?

> *“ Una promesa representa el eventual resultado de una operación asincrónica.*
>
> - The Promise A+ Spec

Las Promesas son Objetos

Es un objeto que representa y gestiona el lifecycle de una respuesta futura.

El objeto mantiene dentro suyo:

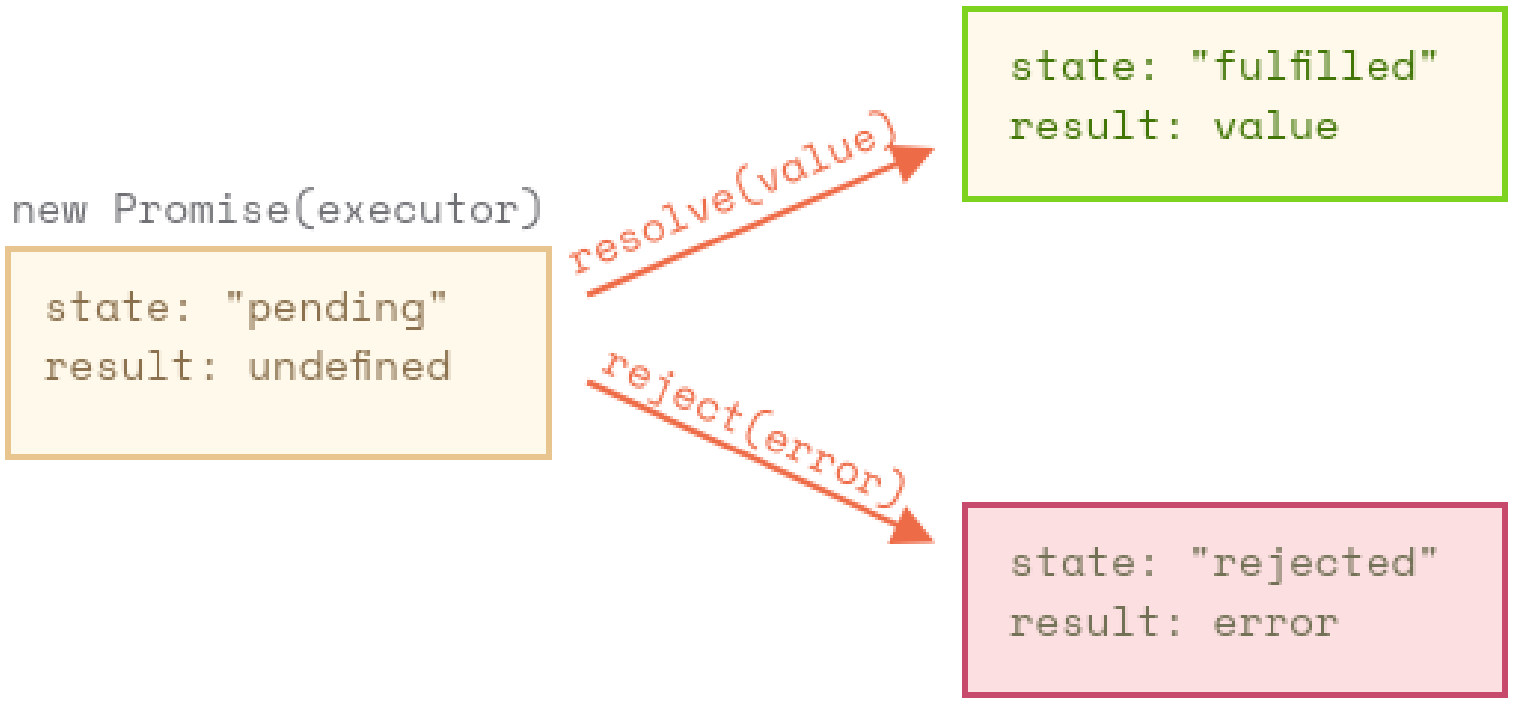
status(pending, fulfilled, rejected)
information (value or a reason)

Sólo podemos acceder al método:

.then()

Las Promesas son Objetos

El estado de las promesas sólo cambia una vez.



unaPromise.then(succesfullHandler, errorHandler)

Promises

Un vez que una promesa se fullfilea, o es rechazada. Ejecuta sus handlers

Si le agregamos un handler nuevo, lo ejecuta con el mismo valor al que se fullfileó.

```
aPromise.then(handler, errorHandler);

// se fullfilea la promesa, por ejempllo con un valor (sin error);
// se ejecuta el handler.

aPromise.then(handler2);

// si agregamos un nuevo handler, se ejecuta con el mismo valor!
```

Creando Promesas

```
1
2  var promise = new Promise(function(resolve, reject) {
3    // Hacer cosas acá dentro, probablemente asincrónicas.
4
5    if (/* Todo funcionó como esperabamos*/) {
6      resolve("Jooya!");
7    }
8    else {
9      reject(Error("Algo se rompió"));
10   }
11 });
```

Callback Hell

```
const verifyUser = function(username, password, callback){
    dataBase.verifyUser(username, password, (error, userInfo) => {
        if (error) {
            callback(error)
        }else{
            dataBase.getRoles(username, (error, roles) => {
                if (error){
                    callback(error)
                }else {
                    dataBase.logAccess(username, (error) => {
                        if (error){
                            callback(error);
                        }else{
                            callback(null, userInfo, roles);
                        }
                    })
                }
            })
        }
    })
};
```

Callback Hell

```
const verifyUser = function(username, password) {
   database.verifyUser(username, password)
       .then(userInfo => dataBase.getRoles(userInfo))
       .then(rolesInfo => dataBase.logAccess(rolesInfo))
       .then(finalResult => {
           //do whatever the 'callback' would do
       })
       .catch((err) => {
           //do whatever the error handler needs
       });
};
```

Encadenado Promesas

.then() retorna una promesa! por eso las podemos encadenar!

```
var primerMetodo = function() {
   var promise = new Promise(function(resolve, reject){
      setTimeout(function() {
         console.log('Terminó el primer método');
         resolve({num: '123'}); //pasamos unos datos para ver como los manejamos
      }, 2000); // para simular algo asincronico hacemos un setTimeOut de 2 s
   });
   return promise;
};


var segundoMetodo = function(datos) {
   var promise = new Promise(function(resolve, reject){
      setTimeout(function() {
         console.log('Terminó el segundo método');
         resolve({nuevosDatos: datos.num + ' concatenamos texto y lo pasamos'});
      }, 2000);
   });
   return promise;
};

var tercerMetodo = function(datos) {
   var promise = new Promise(function(resolve, reject){
      setTimeout(function() {
         console.log('Terminó el tercer método');
         console.log(datos.nuevosDatos); //imprimos los datos concatenados
         resolve('hola');
      }, 3000);
   });
   return promise;
};

primerMetodo()
   .then(segundoMetodo)
   .then(tercerMetodo)
   .then(function(datos){
               console.log(datos); //debería ser el 'hola' que pasamos en tercerMetodo
   });
```

< DEMO />