Promisses

Orientadora: Kelly Joany de Oliveira Santos

Reprograma: Turma Backend - Semana 13

Javascript assíncrono

Se você programa há um bom tempo sabe que a primeira forma de criar um código assíncrono no Javascript era utilizando os callbacks da seguinte forma:

```
function asyncFunction(params, callback, err) {
  var response = // requisição assíncrona
  if (response.status === true) {
    callback(response.data);
  } else {
    err(response.data);
  }
}

asyncFunction(
  params,
  function(response) {
    console.log('Sucesso!');
  },
  function(err) {
    console.log('Erro!');
  }
);
```

Promises

Presente de forma nativa no Javascript desde 2015 no ES6, as Promises são classes que são utilizadas pela maioria das bibliotecas de requisições HTTP atualmente como Axios, jQuery, API Fetch, entre outras. A Promise é facilmente identificada por códigos que utilizam o `.then` para callback de sucesso e `.catch` para erros. Vamos ver como criar uma Promise:

```
const divDelayed = (a, b) => new Promise((resolve, reject) => {
  setTimeout(() => {
    if (b == 0) reject('O valor de B não pode ser zero!');

    resolve(a / b);
  }, 2000);
});
```

Promises

O `resolve` é chamado quando temos uma resposta de sucesso ativando o `.then`, e o `reject` é chamado quando temos um erro disparando o `.catch`.

```
divDelayed(10, 0)
   .then(resp => console.log(`Sucesso: ${resp}`))
   .catch(err => console.log(err));

// O valor de B não pode ser zero!
```

Encadeamento

```
promise1()
  .then(resp => resp.promise2())
  .then(resp2 => console.log(resp2));
```

Cascata

Por estarmos tratando de código assíncrono você provavelmente irá utilizar Promises em casos que você precisa esperar pelo resultado para realizar alguma ação e novamente, essa ação que está aguardando pode ser uma Promise

```
api.get('/users/diego3g').then(user => {
  api.get(`/addresses/${user.id}`).then(addresses => {
    console.log(addresses);
  });
});
```

```
api.get('/users/diego3g').then(user => {
  api.get(`/groups/${user.id}`).then(groups => {
    groups.map(group => {
      api.get(`/group/${group.id}`).then(groupInfo => {
        console.log(groupInfo);
      });
    })
  });
});
```

Promises simultâneas

Em alguns casos você necessita aguardar o resultado de múltiplas Promises porém nenhuma depende da outra para executar, nesse caso você pode utilizar o método `Promise.all` para aguardar por todas elas:

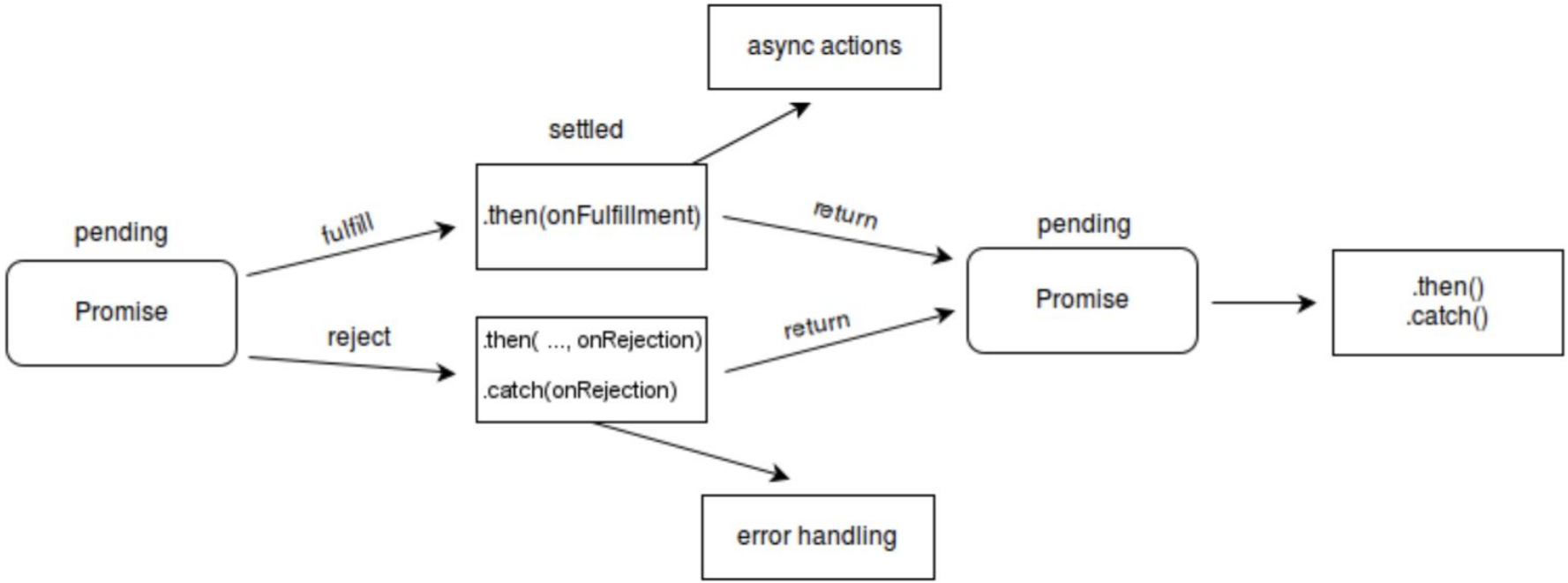
```
Promise.all([
  api.get('/users/diego3g'),
  api.get('/phones/diego3g'),
]).then([user, phones] => {
  console.log(user);
  console.log(phones);
});
```

Corrida

Uma das funcionalidades desconhecidas porém muito útil da Promise é o método `race` que nos permite criar uma corrida entre múltiplas Promises recuperando o resultado da que for mais rápido. Essa corrida é muito útil para adicionar `timeout` à requisições assíncronas:

```
const timeout = ms => new Promise((resolve, reject) =>
  setTimeout(reject, ms, { timeout: true });
);

Promise.race([api.get('/users/diego3g'), timeout(2000)])
  .then(resp => console.log(resp))
  .catch(err => console.log(err));
```


async actions
settled
pending
fulfill
.then(onFulfillment)
return
pending
Promise
.then()
.catch()
Promise
reject
.then(..., onRejection)
.catch(onRejection)
return
error handling

```
// Modelo antigo (ES6)
function fetchUser(user) {
  api.get(`/users/${user}`).then(response => {
    console.log(response);
  });
}

// Novo modelo (ES8)
async function fetchUser(user) {
  const response = await api.get(`/users/${user}`);
  console.log(response);
}
```

```
api.get('/users/diego3g').then(user => {
  api.get(`/groups/${user.id}`).then(groups => {
    groups.map(group => {
      api.get(`/group/${group.id}`).then(groupInfo => {
        console.log(groupInfo);
      });
    })
  });
});
```

```
async function fetchUser() {
  const response = await api.get('/users/diego3g');

  return response;
}

async function fetchGroups() {
  const user = await fetchUser();

  const response = await api.get(`/groups/${user.id}`);

  console.log(response);
}
```

```
// Sintaxe antiga (ES6)

function fetchUser() {
  api.get('/users/diego3g')
    .then(response => console.log(response));
    .catch(err => console.log('Erro:', err));
}

// Nova sintaxe (ES8)

async function fetchUser() {
  try {
    const response = await api.get('/users/diego3g');

    console.log(response);
  } catch (err) {
    console.log('Erro:', err);
  }
}
```