ORIGAMID

JAVASCRIPT ASSÍNCRONO

Promises

new Promise()

`Promise` é uma função construtora de promessas. Existem dois resultados possíveis de uma promessa, ela pode ser resolvida, com a execução do primeiro argumento, ou rejeitada se o segundo argumento for ativado.

```
const promessa = new Promise(function(resolve, reject) {
  let condicao = true;
  if(condicao) {
    resolve();
  } else {
    reject();
  }
});

console.log(promessa); // Promise {<resolved>: undefined}
```

resolve()

Podemos passar um argumento na função `resolve()`, este será enviado junto com a resolução da Promise.

```
const promessa = new Promise(function(resolve, reject) {
  let condicao = true;
  if(condicao) {
    resolve('Estou pronto!');
  } else {
    reject();
  }
});

console.log(promessa); // Promise {<resolved>: "Estou pronto!"}
```

reject()

Quando a condição de resolução da promise não é atingida, ativamos a função reject para rejeitar a mesma. Podemos indicar no console um erro, informando que a promise foi rejeitada.

```
const promessa = new Promise(function(resolve, reject) {
  let condicao = false;
  if(condicao) {
    resolve('Estou pronto!');
  } else {
    reject(Error('Um erro ocorreu.'));
  }
});

console.log(promessa); // Promise {<rejected>: Error:...}
```

then()

O poder das Promises está no método `then()` do seu protótipo. O Callback deste método só será ativado quando a promise for resolvida. O argumento do callback será o valor passado na função resolve.

```
const promessa = new Promise(function(resolve, reject) {
  let condicao = true;
  if(condicao) {
    resolve('Estou pronto!');
  } else {
    reject(Error('Um erro ocorreu.'));
  }
});

promessa.then(function(resolucao) {
  console.log(resolucao); // 'Estou pronto!'
});
```

Assíncrono

As promises não fazem sentido quando o código executado dentro da mesma é apenas código síncrono. O poder está na execução de código assíncrono que executará o `resolve()` ao final dele.

```
const promessa = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('Resolvida');
  }, 1000);
});

promessa.then(resolucao => {
  console.log(resolucao); // 'Resolvida' após 1s
});
```

then().then()

O método `then()` retorna outra Promise. Podemos colocar then() após then() e fazer um encadeamento de promessas. O valor do primeiro argumento de cada then, será o valor do retorno anterior.

```
const promessa = new Promise((resolve, reject) => {
  resolve('Etapa 1');
});

promessa.then(resolucao => {
  console.log(resolucao); // 'Etapa 1';
  return 'Etapa 2';
}).then(resolucao => {
  console.log(resolucao) // 'Etapa 2';
  return 'Etapa 3';
}).then(r => r + 4)
.then(r => {
  console.log(r); // Etapa 34
});
```

catch()

O método `catch()`, do protótipo de Promises, adiciona um callback a promise que será ativado caso a mesma seja rejeitada.

```
const promessa = new Promise((resolve, reject) => {
  let condicao = false;
  if(condicao) {
    resolve('Estou pronto!');
  } else {
    reject(Error('Um erro ocorreu.'));
  }
});

promessa.then(resolucao => {
  console.log(resolucao);
}).catch(reject => {
  console.log(reject);
});
```

then(resolve, reject)

Podemos passar a função que será ativada caso a promise seja rejeitada, direto no método then, como segundo argumento.

```
const promessa = new Promise((resolve, reject) => {
  let condicao = false;
  if(condicao) {
    resolve('Estou pronto!');
  } else {
    reject(Error('Um erro ocorreu.'));
  }
});

promessa.then(resolucao => {
  console.log(resolucao);
}, reject => {
  console.log(reject);
});
```

finally()

`finally()` executará a função anônima assim que a promessa acabar. A diferença do finally é que ele será executado independente do resultado, se for resolvida ou rejeitada.

```
const promessa = new Promise((resolve, reject) => {
  let condicao = false;
  if(condicao) {
    resolve('Estou pronto!');
  } else {
    reject(Error('Um erro ocorreu.'));
  }
});

promessa.then(resolucao => {
  console.log(resolucao);
}, reject => {
  console.log(reject);
}).finally(() => {
  console.log('Acabou'); // 'Acabou'
});
```

Promise.all()

Retornará uma nova promise assim que todas as promises dentro dela forem resolvidas ou pelo menos uma rejeitada. A reposta é uma array com as respostas de cada promise.

```
const login = new Promise(resolve => {
  setTimeout(() => {
    resolve('Login Efetuado');
  }, 1000);
});
const dados = new Promise(resolve => {
  setTimeout(() => {
    resolve('Dados Carregados');
  }, 1500);
});

const tudoCarregado = Promise.all([login, dados]);

tudoCarregado.then(respostas => {
  console.log(respostas); // Array com ambas respostas
});
```

Promise.race()

Retornará uma nova promise assim que a primeira promise for resolvida ou rejeitada. Essa nova promise terá a resposta da primeira resolvida.

```
const login = new Promise(resolve => {
  setTimeout(() => {
    resolve('Login Efetuado');
  }, 1000);
});
const dados = new Promise(resolve => {
  setTimeout(() => {
    resolve('Dados Carregados');
  }, 1500);
});

const carregouPrimeiro = Promise.race([login, dados]);

carregouPrimeiro.then(resposta => {
  console.log(resposta); // Login Efetuado
});
```