ES6 Promise

Promise是什么

`Promise`本意是`承诺`，在程序中的意思就是承诺我`过一段时间`会给你一个结果。 什么时候会用到`过一段时间`？答案是异步操作，异步是指可能比较长时间才有结果的才做，例如网络请求、读取本地文件等。

`Promise`最初打算以`Futures`的命名纳入DOM的规范中，后来改名为`Promise`，并纳入了`ECMAScript`规范中。

如果只纳入DOM规范中，那么`Promise`只能在浏览器端使用，并且由于兼容性问题可能并不会应用的特别好，但是如果纳入`ECMAScript`中，那么不仅仅能在浏览器中使用`Promise`，在`Node.js`中也能够使用`Promise`了。

从原来`Futures`这个名字也能够理解到，是未来的意思，也就是能够预料到未来发生的事情。

在`JavaScript`语言层面上，就是能够对一些异步的操作(异步的操作的结果在**未来**才会发生)，进行可预知的处理。

状态机的概念

`Promise`本质上是一个有限状态机，所谓状态机是会根据特定的条件按照一定的顺序进行转换，而且过程不可逆。

每个`Promise`实例都有三种状态，`Pending`，`Resolve`，`Reject`

在状态机中，需要某些条件才能将自己的状态触发

从初始状态到成功：Pending -----> Resolve

从初始状态到失败：Pending -----> Reject

Promise 最基本使用

如果构造一个Promise

```
let promiseInstance = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve('This is resolve!');
    }, 1000);
});
```

- 构造一个 `Promise` 实例需要给 `Promise` 构造函数传入一个函数。
    - 传入的函数需要有两个形参，两个形参都是 `function` 类型的参数。
        - 第一个形参运行后会让 `Promise` 实例处于 `resolve` 状态，所以我们一般给第一个形参命名为 `resolve` ，一般约定 `resolve` 是成功的状态。
        - 第一个形参运行后会让 `Promise` 实例处于 `reject` 状态，所以我们一般给第一个形参命名为 `reject` ，一般约定 `reject` 是失败的状态

写一个实例

```
let promiseInstance = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve('This is resolve!');
    }, 1000);
});

promiseInstance
    .then((value) => {
        console.log(value);
    })
    .catch((err) => {
        console.error(err);
    })
```

Promise作为函数的返回值

```
function ajaxPromise (queryUrl) {
  return new Promise((reslove, reject) => {
    let xhr = new XMLHttpRequest();
    xhr.open('GET', queryUrl, true);
    xhr.send(null);
    xhr.onreadystatechange = () => {
      if (xhr.readyState === 4) {
        if (xhr.status === 200) {
          reslove(xhr.responseText);
        } else {
          reject(xhr.responseText);
        }
      }
    }
  });
}

ajaxPromise('https://api.github.com/')
  .then((value) => {
    console.log(value);
  })
  .catch((err) => {
    console.error(err);
  });
```

API

Promise实例上的方法

- `Promise.prototype.then()` Promise实例变为resolve状态后会执行
- `Promise.prototype.catch()` Promise实例变为reject状态后会执行

Promise构造函数上的方法

- `Promise.all(Array)` 返回一个Promise，等待参数中所有的Promise都处于resolve状态后会触发返回的Promise实例的resolve状态
- `Promise.race(Array)` 返回一个Promise，参数中任意一个Promise变成resolve状态就会触发
- `Promise.resolve()` 立刻返回一个resolve状态的实例
- `Promise.reject()` 立刻返回一个reject状态的实例

Promise.prototype.then()

demo:

```
let let promiseInstance = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve('This is resolve!');
    }, 1000);
});

promiseInstance
    .then((value) => {
        console.log(value);
    });
```

- `then` 方法运行的时机：当 `Promise` 实例处于 `resolve` 状态时。
- `then` 方法的参数：由 `Promise` 构造时传入的参数(是一个函数)的第一个参数(也是一个函数)运行时传入的第一个参数。

Promise.prototype.catch()

与 `Promise.prototype.then()` 原理类似，只是是由 `reject` 状态转移触发的。

从本质上将 `resolve` 和 `reject` 没有什么原理上的不同，只是 `Promise` 实例的两种状态而已，只是大家习惯上将 `resolve` 当成成功的状态，把 `reject` 当成失败的状态。

所以一般 `then` 方法处理成功的操作， `catch` 处理错误的操作。

这只是大家的一种约定，有限状态机还是那个有限状态机。

Promise.all(Array)

> 所有

- **参数**：接受一个数组，数组内都是 `Promise` 实例
- **返回值**：返回一个 `Promise` 实例，这个 `Promise` 实例的状态转移取决于参数的 `Promise` 实例的状态变化。当参数中所有的实例都处于 `resolve` 状态时，返回的 `Promise` 实例会变为 `resolve` 状态。如果参数中任意一个实例处于 `reject` 状态，返回的 `Promise` 实例变为 `reject` 状态。

Promise.race(Array)

> 竞速

- **参数**：接受一个数组，数组内都是 `Promise` 实例
- **返回值**：返回一个 `Promise` 实例，这个 `Promise` 实例的状态转移取决于参数的 `Promise` 实例的状态变化。当参数中任何一个实例处于 `resolve` 状态时，返回的 `Promise` 实例会变为 `resolve` 状态。如果参数中任意一个实例处于 `reject` 状态，返回的 `Promise` 实例变为 `reject` 状态。

Promise.resolve()

返回一个 `Promise` 实例，这个实例处于 `resolve` 状态。

根据传入的参数不同有不同的功能：

- 值(对象、数组、字符串等)：作为 `resolve` 传递出去的值
- `Promise` 实例：原封不动返回
- `thenable` 对象(类Promise对象，具有then方法)：将会转换成 `Promise` 对象，例如jQuery中$.ajax返回的Deferred对象转换成Promise对象
- 无参数：那就只返回一个 `resolve` 状态的 `Promise` 对象，没有传递值

Promise.reject()

返回一个 `Promise` 实例，这个实例处于 `reject` 状态。

- 参数一般就是抛出的错误信息。

Promise的链式调用

- 每次调用返回的都是一个新的Promise实例
- 链式调用的参数通过返回值传递
- `catch` 内部运行正常会到紧接着的 `then` 中