academeez

Promise

Class to help us with async code

Promise

academeez

- Promise is a class that helps us deal with async code
- It implements common api for dealing with pattern of shout / listeners
  - The promise will wrap our async code, and when it happens it will shout out
  - We can attach listeners with a callback that will be called when they hear the shout
- Javascript is filled with async code, it is important to master the patterns and tools for better Javascript programming

Async code for promise

academeez

- Promise is not suitable for all async code.
- If you missed the **Classifying the async code** chapter I recommend seeing that chapter before learning this one.
- Promise is suitable for the following async diagram:

Shouter - Listeners

academeez

- We can divide the promise to 2 parts
  - Shouter - will wrap the async code and emit a single shout when the async code executes
  - Listeners - Will attach a callback that will be called when the shout is emitted

Shout

academeez

- The shout will wrap our async code, and send a shout when the async code is executes
- For example a promise that shout “hello world” after 1 sec
- It is also common that we will not create ourselves the promise, rather use a method which will return a promise.

```
const helloPromise = new Promise((resolve) => {
    setTimeout(() => {
        resolve('Hello Listeners');
    }, 1000);
});
```

resolve / reject

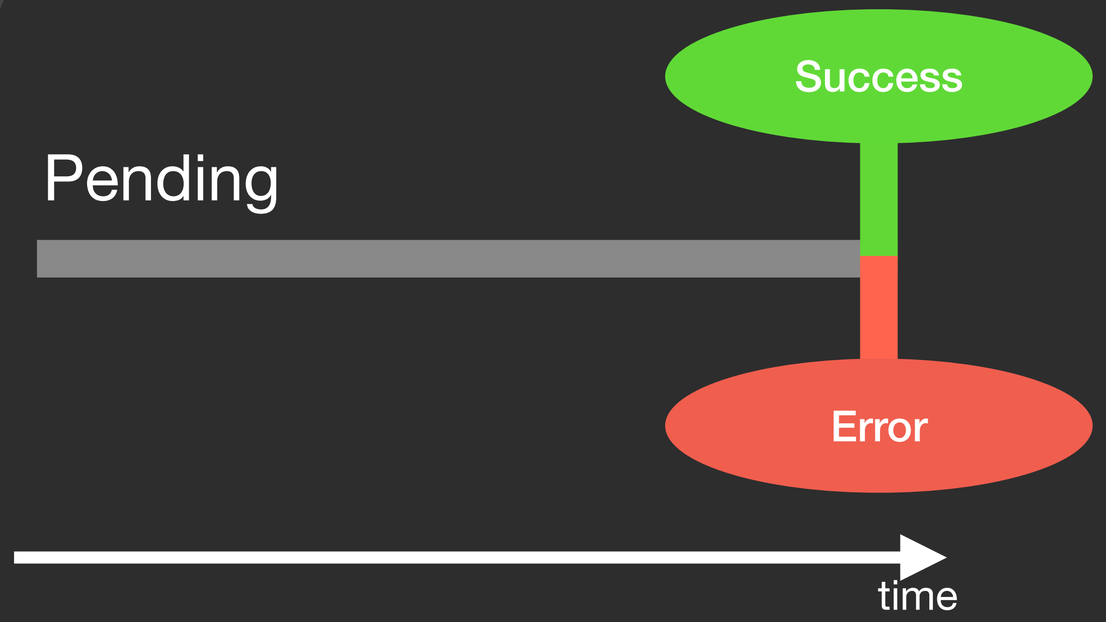
academeez

- We need to pass the promise constructor a function, that function will be called with two callbacks: resolve, reject
- The resolve is used to say: our async code ran successfully and pass data to the listeners
- The reject is used to say: our async code fail, here is the error
- You can call either resolve or reject and only once

```
const helloPromise = new Promise((resolve, reject) => {
    setTimeout(() => {
        // reject(new Error('something happened'));
        resolve('Hello Listeners');
    }, 1000);
});
```

Promise state

- If we can call the **resolve / reject** once this means our promise can be in one of the following states:



Listener

- After we have the promise, we can now attach listeners that will be called when the shout is emitted
- To attach a listener we use the **then** method on the promise and pass a callback as the first argument that will be called when the promise emits a shout
- You can attach multiple listeners the same way

```
helloPromise.then(
    (message) => console.log(message)
);
```

Listen for error

- A listener can also listen for an error, by passing a second callback to the **then**
- A listener can also use the **catch** method on the promise to pass an error callback without the need to add a success callback as well

```
helloPromise.then(
    (message) => console.log(message),
    (err) => console.log(err.message)
);

helloPromise.catch(
    (err) => console.log(err.message)
)
```

Promise chaining

- The **then** method on a **Promise** will return another promise

```
then<TResult1 = T, TResult2 = never>(
    onfulfilled?: ((value: T) => TResult1 | PromiseLike<TResult1>) | undefined | null,
    onrejected?: ((reason: any) => TResult2 | PromiseLike<TResult2>) | undefined | null
): Promise<TResult1 | TResult2>;
```

Promise chaining - example

academeez

- With then manipulating the promise, we can turn a promise containing a certain value to a promise containing another value.
- This manipulation allows us to do **Promise Chaining**

```
helloPromise
    .then(
        url => fetch(url)
    )
    .then(
        res => res.json()
    )
    .catch(() => [])
    .then((dataFromServer) => {
        ...
    })
```

Summary

academeez

- With Promise we can deal with async code that is resolved or rejected once
- We wrap our async code and place it in a function in the Promise constructor
- We call resolve or reject when the async callback happens
- We attach listeners on the promise that will listen for fulfilled or rejected

academeez

Thank You

Next Lesson: async await