Promises

CODE FELLOWS

401
Advanced
Javascript

Promises

Better way of handling asynchronous behaviors.

Weren't always part of the language.

Q and Bluebird came out, and got so popular, that the concept became part of the language.

Can be easily "chained" and their success/failure protocols are understandable

Where a Callback says "Run this function and then tell me what to do when I'm done" …

A Promise says "I'm going to go ahead and do this thing, and when I'm all done, I'll let you know how it went, and then you can decide how to handle it"

CODE FELLOWS

401
Advanced
Javascript

Promises - Implementation

Promises are a constructor function that has 3 **static methods**:

resolve()	Completes the promise successfully and “returns” a “resolution value”
reject()	Completes the promise unsuccessfully and “returns” an error string
all()	Completes all named promises (takes an array) and process their resolution values In order

Basic Usage Code Flow:

- Create a function that returns a promise
- That function implements either/both a resolve() and a reject() based on success or failure
- When you call a promise, you “listen” for it to finish up.
- Meanwhile, your code continues to run top-down (Promises are asynchronous)
- When it completes, you may **then** work with the resolution value or **catch** any errors

CODE FELLOWS

401
Advanced
Javascript

Creating Promises

2 Ways to create a promise

Create an Instance

```
let getName = () => new Promise( (resolve, reject) =>
{
   setTimeout( ()=> {
      resolve("John");
   }, 0);
});
```

Use the static convenience methods directly to return a promise to a caller that expects one
...sometimes known as “wrapping a value in a promise”

```
getName = () => Promise.resolve(“John”);

--or --

getName = () => Promise.reject( new Error(“Bad Name”));
```

401
Advanced
Javascript

Calling & Using Promises

Use the method “then()” to interact with resolved values
Use the method “catch()” to interact with rejected values

- .then() receives as it’s params, the resolved values from the promise
- .catch() receives as it’s param, the string value in the reject
- Both of these return a promise
- Their return values are wrapped in a resolve and be fed into the next .then()
- If you throw a value, it’ll be wrapped in a reject and fed to the next .catch()
- Failures will fall down to the first catch
- Then the next .then() will pick up … and so on.

```
getName
   .then(name => console.log(name)
   .catch(error => console.log(error);
   .then( //… )
   .then( //… )
   .catch( //… );
```

CODE FELLOWS

401
Advanced
Javascript

Examples

```
Promise.reject(2)
  .then(success)
  .catch(number => number*2)
  .then(success);
```

```
let getName = () => new Promise( (resolve, reject) => {
  setTimeout( ()=> {
      resolve("Johnny");
  }, 0);
});


getName()
  .then( name => name.toUpperCase() )
  .then( success )
  .catch( fail );
```

```
superagent.get('https://www.reddit.com/r/programming.json')
.then(res => res.body)
.then(body => body.data)
.then(data => data.children.map(listing => listing.data.domain))
.then(domains => superagent.get(domains[0]))
.then(res => res.text)
.then(success)
.catch(fail);
```

401
Advanced
Javascript

Appendix: Static Methods

Static Methods live on the constructor, not the instances.

Static Methods

```
function Note(title="") {
   this.title = title;
}

Note.isNote(note) = function() {
   Return !! this.title;
}

Note.prototype.showTitle = function() {
   Return this.title;
}
```

```
Let note = new Note("Hello");
note // {title:"Hello")
note.showTitle; // Hello
note.isNote; // note.isNote is not a
function
Note.isNote(note); // true
```