1

@saadjamilakhtar

ADVANCED TECHNIQUES

WORKING WITH JAVASCRIPT PROMISES

NEXT

2

@saadjamilakhtar

Promise Chaining

PROMISE CHAINING ALLOWS EXECUTION OF A SEQUENCE OF ASYNCHRONOUS OPERATIONS ONE AFTER ANOTHER. BY RETURNING A NEW PROMISE FROM THE `*then()*` METHOD.

```
getUser()
.then(user=>getUserPosts(user.id))
.then(posts=>processPosts(posts))
.catch(error=>console.error(error));
```

3

@saadjamilakhtar

Promise.all

PROMISE.ALL ALLOWS YOU TO HANDLE MULTIPLE PROMISES CONCURRENTLY AND GET THEIR RESULTS AS AN ARRAY. USEFUL FOR WAITING ON SEVERAL ASYNC OPERATIONS TO FINISH.

```
const promise1 = fetch('api.com/data1');
const promise2 = fetch('api.com/data2');

Promise.all([promise1, promise2])
  .then(resp =>
   Promise.all(resp.map(res=>res.json())))
  .then(data => console.log(data))
  .catch(error => console.error(error));
```

4

@saadjamilakhtar

Promise.race

PROMISE.RACE RETURNS A NEW PROMISE THAT RESOLVES OR REJECTS WHEN THE FASTEST PROMISE IN THE ITERABLE RESOLVES OR REJECTS, USEFUL FOR HANDLING QUICK RESULTS OR TIMEOUTS.

```
const promise1 = fetch('api.com/data1');
const promise2 = fetch('api.com/data2');

Promise.race([promise1, promise2])
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error));
```

5

@saadjamilakhtar

ERROR HANDLING

PROPER ERROR HANDLING IS VITAL IN ASYNCHRONOUS OPERATIONS. USE `*catch()*` TO HANDLE ERRORS AND FOR `*finally()*` CLEANUP LOGIC IN PROMISES.

```
fetch('https://api.example.com/data')
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(error => console.error(error))
  .finally(() => console.log('Cleanup'));
```

@saadjamilakhtar

Was this post helpful?

Your thoughts?