Chapter 5:
Setting a Time Limit for Async Tasks

Async JavaScript Seminar

Tim Dronkers & Josh Keller

Promise.race( )

Definitions

- Promise.race() is a static method in the Promise class
- Syntax: Promise.race(iterable);
- Return Value: a pending promise that *asynchronously* yields the value of the first promise in the given iterable to fulfill OR reject (MDN)
- Designed to race several promises against each other and return the result of the promise that “settles” first (Kelhini)

Promise.race( )

Basic Example

```
const promiseResolve = new Promise((resolve, _) => {
  setTimeout(resolve, 500, 'SUCCESS!');
});

const promiseReject = new Promise((_, reject) => {
  setTimeout(reject, 1000, 'Error...');
});

Promise.race([promiseResolve, promiseReject])
  .then(response => console.log(response)) // 'SUCCESS!'
  .catch(error => console.error(error));
```

```
const promiseResolve = new Promise((resolve, _) => {
  setTimeout(resolve, 1000, 'SUCCESS!');
});

const promiseReject = new Promise((_, reject) => {
  setTimeout(reject, 500, 'Error...');
});

Promise.race([promiseResolve, promiseReject])
  .then(response => console.log(response))
  .catch(error => console.error(error)); // 'Error...'
```

Promise.race( )

vs. Promise.any( )

race( )

```
const promiseResolve = new Promise((resolve, _) => {
  setTimeout(resolve, 1000, 'SUCCESS!');
});

const promiseReject = new Promise((_, reject) => {
  setTimeout(reject, 500, 'Error...');
});

Promise.race([promiseResolve, promiseReject])
  .then(response => console.log(response))
  .catch(error => console.error(error)); // 'Error...'
```

Promise.race( ) rejects after 1/2 second

any( )

```
const promiseResolve = new Promise((resolve, _) => {
  setTimeout(resolve, 1000, 'SUCCESS!');
});

const promiseReject = new Promise((_, reject) => {
  setTimeout(reject, 500, 'Error...');
});

Promise.any([promiseResolve, promiseReject])
  .then(response => console.log(response)) // 'SUCCESS!'
  .catch(error => console.error(error));
```

Promise.any( ) fulfills after 1 second

Promise.race( )

Gotchas

An empty iterable causes the returned promise to be stuck in a pending state:

```
const emptyPromise = Promise.race([]);

console.log(emptyPromise); // Promise { <pending>}

setTimeout(function() {
  console.log('The stack is now empty...'); // '...'
  console.log(emptyPromise); // Promise { <pending>}
});
```

Promise.race( )

Gotchas

If the iterable contains one or more non-promise values or already settled promises, then promise.race will resolve to the first of these values found in the iterable:

```
const fastPromise = new Promise((resolve, _) => {
  setTimeout(resolve, 0 , 'a' );
});
const alreadyFulfilled = Promise.resolve('b');
const nonPromise = 'c';

const strangeRace = Promise.race([
  fastPromise,
  alreadyFulfilled,
  nonPromise
]);

setTimeout(function() {
  console.log('The stack is now empty...');
  strangeRace.then(value => console.log(value)); // 'b'
});
```

Promise.race( )

Use Case 1: Use Cached Data After Timeout

By racing an async task such as an API call against a promise that is going to be *rejected* after a set-period of time, we effectively create a time limit for the async task.

Promise.race( )

Cont.

```
API call successful. Here is your fresh data:

{
  userId: 1,
  id: 1,
  title: 'delectus aut autem',
  completed: false
}
```

```javascript
function loadFromCache() {
  const data = {
    "userId": 1,
    "id": 1,
    "title": 'delectus aut autem',
    "completed": false,
    "source": 'this is from cache'
  };
  return new Promise((resolve) => {
    resolve(data)
  });
}

function fetchNewOrCached() {
  const timeOut = 1000;
  const cache = loadFromCache()
    .then((data) => {
      return new Promise ((_, reject) => {
        setTimeout(() => reject(data), timeOut); // reject
      });
    });

  const freshData = fetch('https://jsonplaceholder.typicode.com/todos/1');
  return Promise.race([cache, freshData]);
}

fetchNewOrCached()
  .then((response) => {
    console.log('API call successful. Here is your fresh data:');
    // json() method is part of the response interface of the Fetch API
    response.json().then((data) => console.log(data));
  })
  .catch((error) => {
    console.log('Time Limit Exceeded. Here is the cached data:');
    console.log(error);
  });
```

Promise.race( )

Cont.

```
Time Limit Exceeded. Here is the cached data:

{
  userId: 1,
  id: 1,
  title: 'delectus aut autem',
  completed: false,
  source: 'this is from cache'
}
```

```
function loadFromCache() {
  const data = {
    "userId": 1,
    "id": 1,
    "title": 'delectus aut autem',
    "completed": false,
    "source": 'this is from cache'
  };
  return new Promise((resolve) => {
    resolve(data)
  });
}

function fetchNewOrCached() {
  const timeOut = 100;
  const cache = loadFromCache()
    .then((data) => {
      return new Promise ((_, reject) => {
        setTimeout(() => reject(data), timeOut); // reject
      });
    });

  const freshData = fetch('https://jsonplaceholder.typicode.com/todos/1');
  return Promise.race([cache, freshData]);
}

fetchNewOrCached()
  .then((response) => {
    console.log('API call successful. Here is your fresh data:');
    // json() method is part of the response interface of the Fetch API
    response.json().then((data) => console.log(data));
  })
  .catch((error) => {
    console.log('Time Limit Exceeded. Here is the cached data:');
    console.log(error);
  });
```

Promise.race( )

Cont.

```
API call successful. Here is your fresh data:
Time Limit Exceeded. Here is the cached data:
TypeError: response.json is not a function...
```

```
function loadFromCache() {
  const data = {
    "userId": 1,
    "id": 1,
    "title": 'delectus aut autem',
    "completed": false,
    "source": 'this is from cache'
  };
  return new Promise((resolve) => {
    resolve(data)
  });
}

function fetchNewOrCached() {
  const timeOut = 100;
  const cache = loadFromCache()
    .then((data) => {
      return new Promise ((resolve, _) => {
        setTimeout(() => resolve(data), timeOut); // resolve
      });
    });

  const freshData = fetch('https://jsonplaceholder.typicode.com/todos/1');
  return Promise.race([cache, freshData]);
}

fetchNewOrCached()
  .then((response) => {
    console.log('API call successful. Here is your fresh data:');
    // json() method is part of the response interface of the Fetch API
    response.json().then((data) => console.log(data));
  })
  .catch((error) => {
    console.log('Time Limit Exceeded. Here is the cached data:');
    console.log(error);
  });
```

Promise.race( )

Use Case: Batching Async Requests