DZone

Common Promise Mistakes

by Can Ho MVB · Aug. 18, 16 · Web Dev Zone

Below are some Promise mistakes I found during a code review session:

> **The `Promise` object is used for deferred and asynchronous computations. A `Promise` represents an operation that hasn't completed yet, but is expected in the future.**
>
> **MDN: Promise**

---

1. Mistake: Put Try/Catch in Promise Definition

```
function doSomething(){
    return new Promise(function(resolve, reject){
        try {
            doSomethingSync();
            doSomethingAsync('some data', function(err, data){
                if (err){
                    return reject(err);
                };
                resolve(data);
            });
        }catch(e){
            reject(e);
        }
     });
}
```

Promise grabs all errors (even typo errors) by wrapping all code in a try/catch so that any exception thrown during execution will be caught and converted to a rejected promise. In the **doSomething** function, the try/catch block is unnecessary.

Better

```
function doSomething(){
    return new Promise(function(resolve, reject){
        doSomethingSync();
        doSomethingAsync('some data', function(err, data){
            if (err){
                return reject(err);
            };
            resolve(data);
        });
    });
}
```

2. Mistake: Promise Hell

Promises are one of the ways to solve callback hell. But using Promises incorrectly can cause ‘**Promise hell**’.

```
authenticateUser('user1').then(function(user){
    getPosts(user).then(function(posts){
        showPosts(posts).then(function(){
            console.log('done!');
        });
    });
});
```

In the above code, we’re trying to authenticate user ‘user1’ and then get that user’s posts and finally show these posts on the home page. Nesting **getPosts** and **showPosts** as in the above code causes ‘Promise hell’. To fix this, we need to un-nest our code by returning **getPosts** promise from the first **then** and handle it by the second **then**.

```
authenticateUser('user1')
  .then(function(user){
      return getPosts(user);
  })
  .then(function(posts){
      return showPosts(posts);
  })
  .then(function(){
      console.log('done!');
  });
```

Even better

```
authenticateUser('user1')
  .then(getPosts)
  .then(showPosts)
  .then(function(){
       console.log('done!');
   });
```

3. Mistake: Not Utilizing Promise.all

In some cases, we need to fetch some resources from the server before doing some actions with these resources. Not using the **Promise.all** utility can cause deeply nested promises:

```
getProduct('p1')
  .then(function(p1){
    getProduct('p2')
      .then(function(p2) {
        getProduct('p3')
          .then(function(p3) {
            return compare(p1, p2, p3);
          });
      });
});
```

This code can be improved by using **Promise.all**

```
Promise.all([getProduct('p1'), getProduct('p2'), getProduct('p3')])
       .then(function(products){
          return compare(products[0], products[1], products[2]);
       });
```

Even better (I updated this based on my comment in response to Greg's)

```
Promise.all([getProduct('p1'), getProduct('p2'), getProduct('p3')])
       .then(function([p1, p2, p3]){
          return compare(p1, p2, p3);
       });
```

4. Mistake: Always Creating Unnecessary Promises

```
function doSomething() {
    return new Promise(function(resolve, reject) {
        fetchData('resource1')
          .then(function(resource) {
             var data = process(resource);
             resolve(data);
          })
          .catch(function(err) {
              reject(err);
          });
    });
}
```

In the above code,the main purpose of the returned Promise is to capture and return the data (and error) from fetchData. The above code can be vastly improved like this:

```
function doSomething() {
    return fetchData('resource1')
              .then(function(resource) {
                  return process(resource);
              });
}
```

5. Mistake: Trying to Make Sync → Async by Creating Promises

```
function isEmail(email) {
      return new Promise(function(resolve, reject) {
          if (/\S+@\S+\.\S+/.test(email)) {
              resolve(email);
          } else {
              reject(‘Invalid email’);
          }
      });
 }
 function createUser(req, resp) {
    var user = ...;
    isEmail(user.email)
        .then(function(){
            return createUser(user);
        })
        .then(function(){
           resp.status(200).end();
        })
        .catch(function(err) {
          ...
        });
}
```

In the function **isEmail** above, Promise is overused when it is trying to make sync code –> async by creating a new Promise which makes the code slower as the email validation code is deferred to the next job while the validation code can be executed immediately.

```
function isEmail(email) {
    return /\S+@\S+\.\S+/.test(email);
}
```

```
function createUser(req, resp) {
    var user = ...;
    if (isEmail(user.email)) {
        createUser(user)
            .then(function(){
                resp.status(200).end();
            });
    } else {
        resp.status(400).end();
    }
}
```

Topics: NEW, PROMISE, REVIEW, FUNCTION, NESTED

Published at DZone with permission of Can Ho, DZone MVB. See the original article here.
Opinions expressed by DZone contributors are their own.