Common Patterns Using Promises

Neal Lindsay

@neall

hello@testdouble.com

what is a promise?

patterns

anti-patterns

stunning conclusion!

a container you get immediately 📦

for a value you get eventually 🔜 🔢

promises are for asynchronous data

~~event handlers?~~

~~streaming data?~~

each “then” only happens once

2010 - promises/A

2011 - jQuery “deferreds”

2012 - promises/A+

2013 - browser promises

```
new Promise(function(resolve, reject) {
  if (thingsGoWell) {
    resolve('value');
  } else {
    reject(Error('reason'));
  }
});
```

```
Promise.resolve('value');

Promise.reject(Error('reason'));
```

```
p.then(
  function(value){…},
  function(reason){…}
);
```

don't call it a callback

(it’s been here for years)

data handler

error handler

.then() returns a new promise

```
np = p.then(…);
```

$$p \neq np$$

(dumb math joke)

how does this new promise resolve?

1. falls through if no handler

```
p.then(f)

p.then().then(f)
```

1. falls through if no handler
2. rejected with any error thrown
3. resolved with a value returned
4. except… if a promise returned, “becomes” that promise

```
np =
  p.then(function(value) {
    return Promise.resolve(3 + value);
  });
```

patterns

chaining vs fanning

chaining

```
get('/dog')
   .then(JSON.parse)
   .then(function(dog) {…});
```

fanning

```
p = get('/dog').then(JSON.parse);

p.then(alert);
p.then(playFetch);
```

```
cache = {};
getUserWithCache = function(id) {
  return cache[id] =
    cache[id] || getUser(id);
};

getUserWithCache(3).then(displayName);
// later
getUserWithCache(3).then(displayAge);
```

naming

value: plain old noun

```
p.then(function(dog) {…});
```

error: error

```
p.then(
  undefined,
  function(error) {}
);
```

```
p.then(
  undefined,
  function(error) {…}
);

p.catch(
  function(error) {…}
);
```

```
p.then(
  undefined,
  function(error) {…}
);

p['catch'](
  function(error) {…}
);
```

function that returns a promise:

imperative verb phrase

```
guessFavoriteColor()
  .then(function(color) {…});
```

promise: past-tense verb phrase

```
gotScruffy = getDog('scruffy');
gotScruffy
  .then(function(scruffy) {…});
```

name handler functions for readability

```
getDog('scruffy')
  .then(function(dog) {
    return getPerson(dog.ownerId);
  })
  .then(function(person) {
    return person.name;
  })
  .then(alert);
```

```
getOwner = function(obj) {
  return getPerson(obj.ownerId);
};
readName = function(obj) {
  return obj.name;
};

getDog('scruffy')
  .then(getOwner)
  .then(readName)
  .then(alert);
```

use higher-order functions

```
multiplyBy = function(x) {
  return function(y) {
    return x * y;
  };
};

timesThree = multiplyBy(3);

alert(timesThree(2)); // 6
```

```
getOwner = function(obj) {
  return getPerson(obj.ownerId);
};
readName = function(obj) {
  return obj.name;
};

getDog('scruffy')
  .then(getOwner)
  .then(readName)
  .then(alert);
```

```
readProperty = function(propertyName) {
  return function(obj) {
    return obj[propertyName];
  };
};

getDog('scruffy')
  .then(readProperty('ownerId'))
  .then(getPerson)
  .then(readProperty('name'))
  .then(alert);
```

recover from errors

```
getDog('scruffy')
  .then(alert);
```

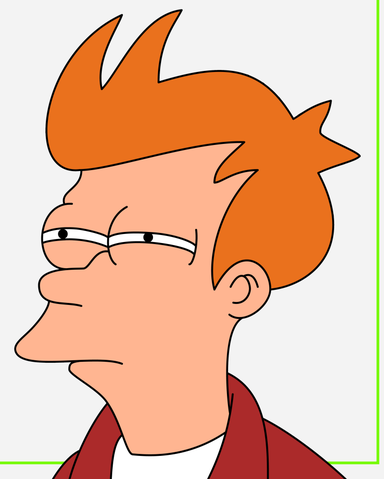
```
var getDefaultDog = function() {
  return getDog('fluffy');
}

getDog('scruffy')
  .catch(getDefaultDog)
  .then(alert);
```

```
var getEmergencyDog = function() {
  return {name: 'claude'};
};

getDog('scruffy')
  .catch(getDefaultDog)
  .catch(getEmergencyDog)
  .then(alert);
```

anti-patterns



returning data
not in a promise

```
var getEmergencyDog = function() {
  return {name: 'claude'};
};

getDog('scruffy')
  .catch(getDefaultDog)
  .catch(getEmergencyDog)
  .then(alert);
```

```
var getEmergencyDog = function() {
  return Promise.resolve({name: 'claude'});
};

getDog('scruffy')
  .catch(getDefaultDog)
  .catch(getEmergencyDog)
  .then(alert);
```

unnecessary function wrapping

```
get('/dogs')
  .then(function(response) {
    return JSON.parse(response);
  })
  .then(doSomething);
```

```
get('/dogs')
    .then(JSON.parse)
    .then(doSomething);
```

except...

```
getDog('scruffy')
  .then(window.alert); // works

getDog('scruffy')
  .then(console.log); // TypeError!
```

```
getDog('scruffy')
  .then(function(dog) {
    console.log(dog)
  }); // works, but meh :-/
```

```
getDog('scruffy')
  .then(console.log.bind(console));
```

synchronous code

after

promise code

```
getDog('scruffy').then(alert);

alert('Good dog!');
```

```
getDog('scruffy')
  .then(alert)
  .then(function() {
    alert('Good dog!');
  });
```

inappropriately chaining queries

```
getDog('scruffy')
  .then(goWalkies);
```

can you walk my dog too?

```
getDog('scruffy')
  .then(goWalkies);

getDog('fluffy')
  .then(goWalkies);
```

```
getDog('scruffy')
  .then(function (scruffy) {
    return new Promise(function(resolve) {
      getDog('fluffy')
        .then(function(fluffy) {
          resolve([scruffy, fluffy]);
        });
    });
  })
  .then(goWalkies);
```

```
Promise
  .all([
    getDog('scruffy'),
    getDog('fluffy'),
    getEmergencyDog()
  ])
  .then(goWalkies);
```

```
Promise
  .all([
    getDog('scruffy').then(getOwnerForDog),
    getDog('fluffy').then(getOwnerForDog),
    getEmergencyDog().then(getOwnerForDog)
  ])
  .then(renegotiateWalkingRates);
```

conclusion

chain vs fan

naming

move inline functions into named variables

replace named functions with generated functions

return promises
even for some static
data for consistency

don't wrap functions for no reason

but do use function.bind() when needed

keep synchronous code above promise code

don't chain expensive queries when you can avoid it

thank you

questions?