JavaScript Promises

Pulak (@pulakb)
http://pulakonline.com/

Discussions

- Callbacks
- JavaScript Promises
  - jQuery
  - ‘q’ library
  - AngularJs
  - Node.js

- What is a Callback function?
- How Callback function work?
- ‘Callback hell’

What is a Callback function?

- A callback function (say, Y) is a function that is passed to another function (say, X) as a parameter, and Y gets executed inside X.

- JavaScript uses callback functions to handle asynchronous control flow.

```
1  $( "#dataRow" ).on( "click", function() {
2      alert('hello');
3  });
```

How Callback function work?

```
function writeCode(callback) {
    //do something
    callback();
    //.....
}

function introduceBugs() {
   //.... Make bugs
}

writeCode(introduceBugs)
```

How Callback function work?

```
fs = require('fs');

fs.readFile ('f1.txt','utf8',function(err,data) {
    if (err) {
        return console.log(err);
    }
    console.log(data);
});
```

How Callback function work?

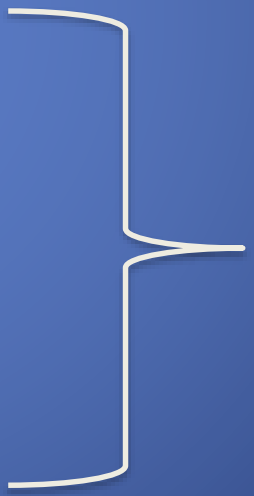
```
fs = require('fs');

fs.readFile('f1.txt','utf8',function(err,data){
    if (err) {
        return console.log(err);
    }
    console.log(data);
});
```

How Callback function work?

```
 fs = require('fs');

fs.readFile('f1.txt','utf8',
  function(err,data){
    if (err) {
        return console.log(err);
    }
    console.log(data);
  }
);
```



Equivalent formatting

‘Callback hell’

- Code complexity
- Modularity
- Maintainability
- Tightly coupled interface

‘Callback hell’

When working with callbacks, nesting of functions can make reading and understanding the code very difficult

```
step1(function (value1) {
    step2(value1, function(value2) {
        step3(value2, function(value3) {
            step4(value3, function(value4) {
                // Do something with value4
            });
        });
    });
});
```

Promises

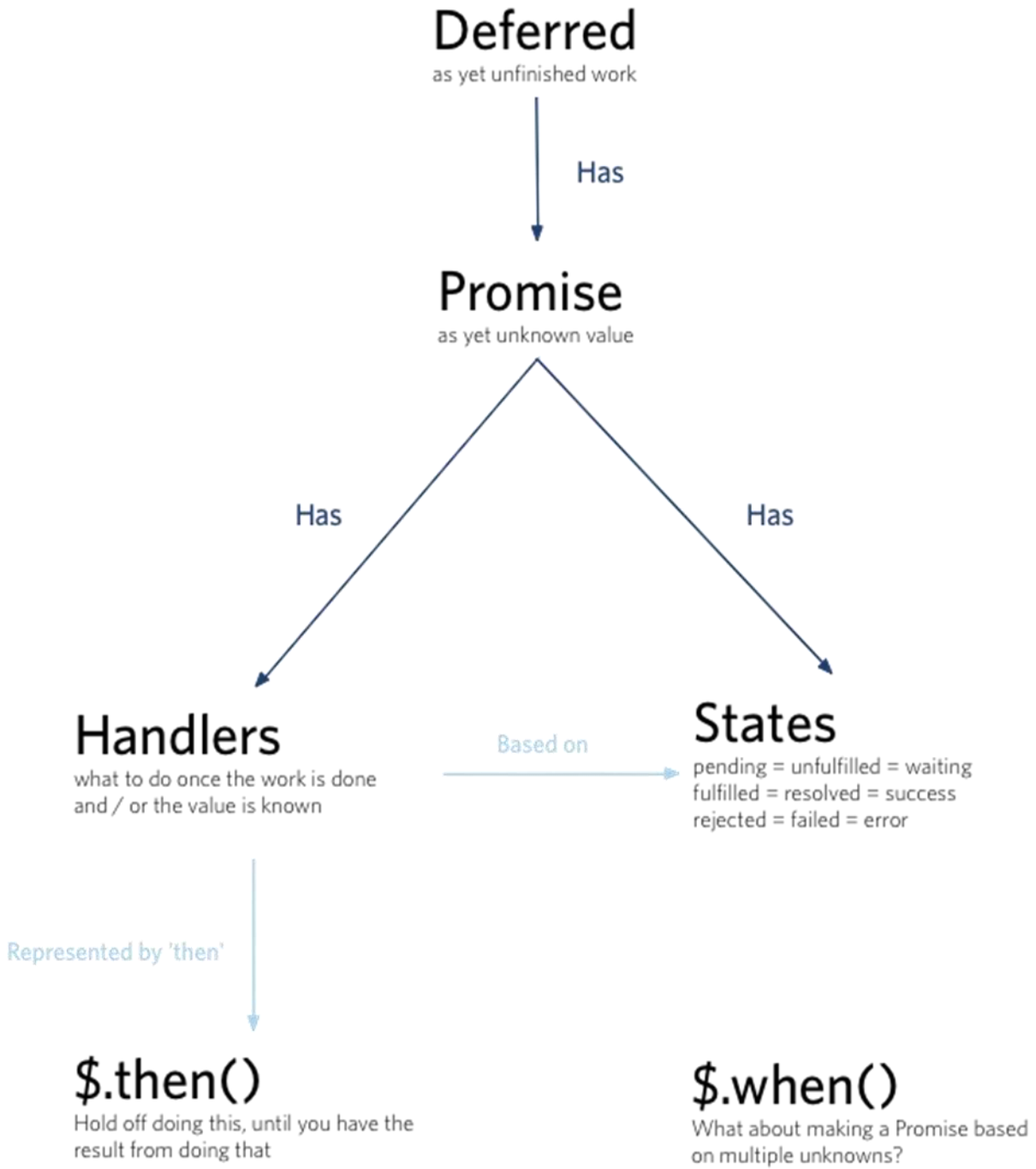
- What is Promise?
- What is Deferred?
- Deferred & Promise
- What are the use cases of Promise?
- What does Promises guarantee?
- Why promises are awesome?

What is Promise?

- A promise is an object that represents the return value or the thrown exception that the function may eventually provide.
- In other words, a promise represents a value that is not yet known.
  *A promise is an **asynchronous value***.
- The core idea behind promises is that a promise represents the result of an asynchronous operation.
- A Promise has 3 possible states
  - Pending
  - Fulfilled
  - Rejected

Deferred & Promise

- A **deferred** (object) represents a work that is not yet finished.
- A **promise** (property) is a placeholder for a result which is initially unknown.
- **Every deferred has a promise which functions as a proxy for the future result**.
- From a semantic perspective this means that instead of calling a function (*callback*), we are able to return a value (*promise*).


Deferred
as yet unfinished work
Has
Promise
as yet unknown value
Has
Has
Handlers
what to do once the work is done
and / or the value is known
Based on
States
pending = unfulfilled = waiting
fulfilled = resolved = success
rejected = failed = error
Represented by 'then'
$.then()
Hold off doing this, until you have the
result from doing that
$.when()
What about making a Promise based
on multiple unknowns?

What are the use cases of Promise?

- An AJAX request and callbacks(a single request, parallel/chained requests)
- An asynchronous loading of assets and actions to perform
- Animation
- Modal User Interaction

What does Promise guarantee?

promiseForResult.then(onFulfilled, onRejected);

- Only one of **onFulfilled** or **onRejected** will be called.
- **onFulfilled** will be called with a single fulfillment value (⇔ return value).
- **onRejected** will be called with a single rejection reason (⇔ thrown exception).
- If the promise is already settled, the handlers will still be called once you attach them.
- The handlers will always be called asynchronously.

What does Promise guarantee?

- At first, a Promise is in a pending state. Eventually, it's either resolved or rejected.
- Once a Promise is resolved or rejected, it'll remain in that state forever, and its callbacks will never fire again
- Promises can be chained

Why promises are awesome

- Cleaner method signatures
- Uniform return/error semantics
- Easy composition
- Easy sequential/parallel join
- Always async
- Exception-style error bubbling

Case 1: Simple Functional Transform

```
var author = getAuthors();
var authorName = author.name;

// becomes

var authorPromise = getAuthors().then(function (author) {
    return author.name;
});
```

Case 2: Reacting with an Exception

```
var author = getAuthors();
if (author === null)
    throw new Error("null author!");
```

becomes

```
var authorPromise = getAuthors().then(function (author) {
    if (author === null)
        throw new Error("null author!");
    return author;
});
```

Case 3: Handling an Exception

```
try {
    updateAuthor(data);
} catch (ex) {
    console.log("There was an error:", ex);
}

// becomes

var updatePromise = updateAuthor(data).then(undefined, function (ex) {
    console.log("There was an error:", ex);
});
```

Case 4: Rethrowing an Exception

```
try {
    updateAuthor(data);
} catch (ex) {
    throw new Error("Updating author failed. Details: " + ex.message);
}

// becomes

var updatePromise = updateAuthor(data).then(undefined, function (ex) {
    throw new Error("Updating author failed. Details: " + ex.message);
});
```

Async Case: Waiting

```
var name = promptForNewAuthorName();
updateAuthor({ name: name });
refreshScreen();

// becomes

promptForNewAuthorName()
    .then(function (name) {
        return updateAuthor({ name: name });
    })
    .then(refreshScreen);
```

jQuery

Q - library

Q - library

```
step1(function (value1) {
    step2(value1, function(value2) {
        step3(value2, function(value3) {
            step4(value3, function(value4) {
                // Do something with value4
            });
        });
    });
});
```

With a promise library, you can flatten the pyramid.

```
Q.fcall(promisedStep1)
.then(promisedStep2)
.then(promisedStep3)
.then(promisedStep4)
.then(function (value4) {
    // Do something with value4
})
.catch(function (error) {
    // Handle any error from all above steps
})
.done();
```

jQuery - q differences

jQuery	Q	Notes
then	then	Q's then, and in fact all of its methods, have different exception-handling behavior, as described above.
done	then	then does not support multiple handlers; use multiple calls to then to attach them.
fail	catch	catch does not support multiple handlers; use multiple calls to catch to attach them.
deferred.promise(method)	deferred.promise(property)	You *must* get the promise part of the deferred; the deferred does not have the promise API.

- https://github.com/kriskowal/q/wiki/Coming-from-jQuery

Node and q library

AngularJS

AngularJS

- Limitations of Promise in Angular

In Angular's $Q implementation, If you don't fire off $scope.$apply(), after resolving, the resolved values will never propagate to your 'then' functions. Sometimes I want to use promises that aren't tied to my Angular $digest cycle.

URLs

Credit goes to all the people for sharing their thoughts:

- http://wiki.commonjs.org/wiki/Promises/A
- http://promisesaplus.com/
- http://promisesaplus.com/differences-from-promises-a
- http://api.jquery.com/category/deferred-object/
- http://sitr.us/2012/07/31/promise-pipelines-in-javascript.html
- http://stackoverflow.com/questions/12160785/jquery-deferred-promise-design-patterns-and-use-cases
- http://stackoverflow.com/questions/6801283/what-are-the-differences-between-deferred-promise-and-future-in-javascript
- http://stackoverflow.com/questions/5436327/jquery-deferreds-and-promises-then-vs-done
- http://blog.mediumequalsmessage.com/promise-deferred-objects-in-javascript-pt1-theory-and-semantics
- https://gist.github.com/domenic/3889970
- http://domenic.me/2012/10/14/youre-missing-the-point-of-promises/
- https://github.com/kriskowal/q
- https://github.com/kriskowal/uncommonjs/blob/master/promises/specification.md
- http://james.padolsey.com/jquery/#v=2.0.3&fn=jQuery.Deferred
- http://www.dwmkerr.com/promises-in-angularjs-the-definitive-guide/
- http://spion.github.io/posts/why-i-am-switching-to-promises.html
- http://blog.ometer.com/2011/07/24/callbacks-synchronous-and-asynchronous/

Q &A