Promises

https://www.promisejs.org
http://de.slideshare.net/domenicdenicola/callbacks-promises-and-coroutines-oh-my-the-evolution-of-asynchronicity-in-javascript
https://promisesaplus.com
https://developers.google.com/web/fundamentals/getting-started/primers/promises#toc-promisifying-xmlhttprequest
https://github.com/googlesamples/web-fundamentals/tree/gh-pages/fundamentals/getting-started/primers
Kyle Simpson: You Don’t Know JS: ES6 & Beyond

A problem

- Suppose you want to read a file and parse it as JSON.

```
function readJSONSync(filename) {
  var data = fs.readFileSync(filename,’utf8');
  return JSON.parse(data);
}
```

- Why is it a bad idea to write code like this?

A problem

- Suppose you want to read a file and parse it as JSON.

```
function readJSONSync(filename) {
  var data = fs.readFileSync(filename,’utf8');
  return JSON.parse(data);
}
```

- Why is it a bad idea to write code like this?

OK, how about this?

```
function readJSON(filename, callback){
  fs.readFile(filename, 'utf8', function (err, res){
    if (err) {
      return callback(err);
    }
    callback(null, JSON.parse(res));
  });
}
```

- Problems?
  - How do you use this?
  - Callback and return value are confused. (Hard to see and reason about the return value for JSON.parse(res)

```
function readJSON(filename, callback){
  fs.readFile(filename, 'utf8', function (err, res){
    if (err) {
      return callback(err);
    }
    callback(null, JSON.parse(res));
  });
}

readJSON(filename, function(err, bookObj) {
  if (err)
     throw err
}
```

- Other problems?
    - Doesn’t handle errors from JSON.parse

Yielding

- Getting sequencing right with callbacks is also challenging.

```
console.log(“1”);
$.get(“/echo/2”, function(result) {
   console.log(“2”);
});
console.log(“3”);

// 1, 3, 2
```

Promises

- Callback functions have been the main mechanism for managing asynchronous programming
- Callbacks can be hard to trace and reason about.
- Promises are a different type of abstraction for managing asynchronous programming

- New way of thinking about asynchronous functions:
    - Instead of being passed a callback, return a promise

Promises

- “A promise is a future value, a time-independent container wrapped around a value.” (Kyle Simpson)
    - You can reason about a promise whether or not the value has been resolved or not.
- A promise is an asynchronous version of a synchronous function’s return value.
- Promises can be thought of as event listeners where the event fires only once

Terminology

- A promise is an object or function with a then method whose behavior conforms to this specification.
- thenable is an object or function that defines a then method.
- value is any legal JavaScript value (including undefined, a thenable, or a promise).
- exception is a value that is thrown using the throw statement.
- reason is a value that indicates why a promise was rejected.

.then

- The then method registers a callback to receive either a promise's eventual value, or the reason it cannot be fulfilled.
- then returns a Promise!

```
myPromise.then(handleResolve, handleReject);
function handleResolve(data) {
    //handle success
}
function handleReject(error) {
    //handle failure
}
```

Promise API

- Built into ES6, but have existed in different libraries for a while
- Promises/A+ standard
    - Any “thenable” object is treated as a promise and if the standard is followed, promises from different libraries can be chained together
- JQuery promises are a bit different

```
var promise = new Promise(function(resolve, reject) {
  // here is where the real work goes
  if (/* success */) {
    resolve("Stuff worked!");
  }
  else {
    reject(Error("It broke"));
  }
});

promise.then(function(result) {
  console.log(result); // "Stuff worked!"
}, function(err) {
  console.log(err); // Error: "It broke"
});
```

States

- A Promise can be in one of three states:
  - Pending: may transition to fulfilled or rejected state
  - Fulfilled: has a value which must not change
  - Rejected: has a reason which must not change
- The term settled is also used for a promise that has either been fulfilled or rejected.

Back to our readJSON example, assuming readFile has been implemented with promises.

```
function readJSON(filename){
  return readFile(filename, 'utf8').then(JSON.parse);
}
```

readFile returns a promise with the data from the file as its value. This new promise calls

Promisifying old APIs

- Most of the time you will just use libraries that support the promise API and will not create promises using **`new Promise()`**
- However, let’s see how to wrap XMLHttpRequest in a promise

```
function get(url) {
  return new Promise(function(resolve, reject) {
    // Do the usual XHR stuff
    var req = new XMLHttpRequest();
    req.open('GET', url);

    req.onload = function() {
      if (req.status == 200) {
        // Resolve promise with response text
        resolve(req.response);
      }
      else {
        // Otherwise reject with the status text
        reject(Error(req.statusText));
      }
    };

    // Handle network errors
    req.onerror = function() {
      reject(Error("Network Error"));
    };

    // Make the request
    req.send();
  });
}
```

```
get('story.json').then(function(response) {
  console.log("Success!", response);
}, function(error) {
  console.error("Failed!", error);
})
```

Chaining

```
var myPromise = new Promise(function(resolve, reject) {
    // A mock async action using setTimeout
    setTimeout(function() { resolve(10); }, 3000);
});
myPromise.then(function(num) {
    console.log('first then: ', num); return num * 2;
})
.then(function(num) {
    console.log('second then: ', num); return num * 2;
})
.then(function(num) {
    console.log('last then: ', num);
});
```

Catch

```
new Promise(function(resolve, reject) {
 // A mock async action using setTimeout
 setTimeout(function() { reject('error!'); }, 3000);
})
.then(function(e) { console.log('done', e); })
.catch(function(e) { console.log('catch: ', e); });

// From the console:
// 'catch: error!'
```

Promise.all

- Sometimes you want to wait for a number of events to happen, but only want to proceed when all are completed

```
Promise.all([promise1,
              promise2]).then(function(results) {
  // Both promises resolved
})
.catch(function(error) {
  // One or more promises was rejected
});
```

Example

- https://github.com/googlesamples/web-fundamentals/blob/gh-pages/fundamentals/getting-started/primers/async-all-example.html

Promise.all

Array of strings

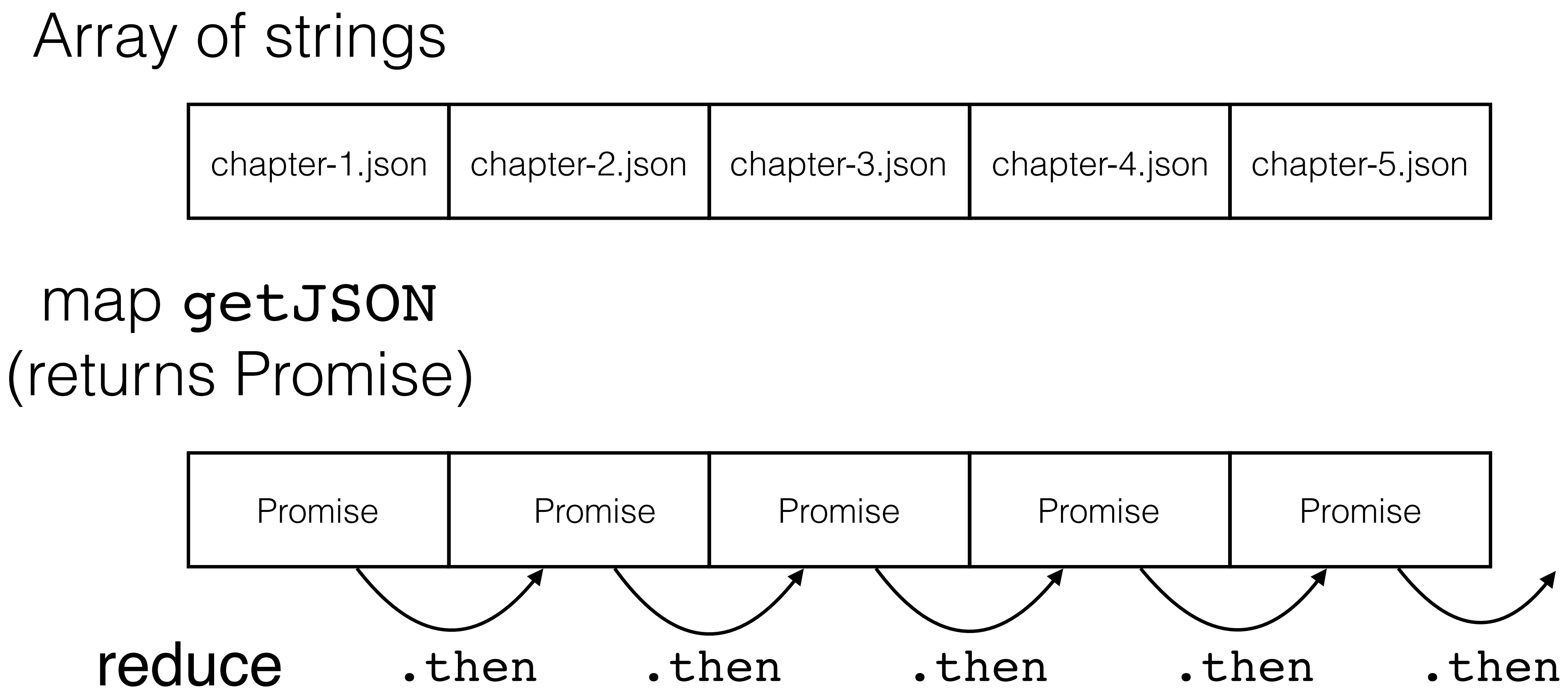
chapter-1.json	chapter-2.json	chapter-3.json	chapter-4.json	chapter-5.json

map **`getJSON`**
(returns Promise)

Promise	Promise	Promise	Promise	Promise

`then` is resolved when all are fulfilled (or first error)

Promise with reduce chaining



Web Sockets

- We already have a TCP connection
  - or the ability to create one
- Why not use it for two-way communication?
  - Server and client can push messages to each other

Protocol

- Start with an HTTP request:

```
GET ws://websocket.example.com/ HTTP/1.1
Origin: http://example.com
Connection: Upgrade
Host: websocket.example.com
Upgrade: websocket
```

- Use wss scheme for HTTPS

Protocol

- Server response

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Wed, 16 Oct 2013 10:07:34 GMT
Connection: Upgrade
Upgrade: WebSocket
```

- Handshake is complete and initial HTTP connection is replaced by a WebSocket connection using the same TCP connection

Advantages

- Server can push to client
- Can transfer data without overhead of traditional HTTP messages
- Chat example