Telerik Academy

Promises and Asynchronous Programming

Callback-oriented asynchrony, CommonJS Promise/A, Promises in Q, Promises in jQuery

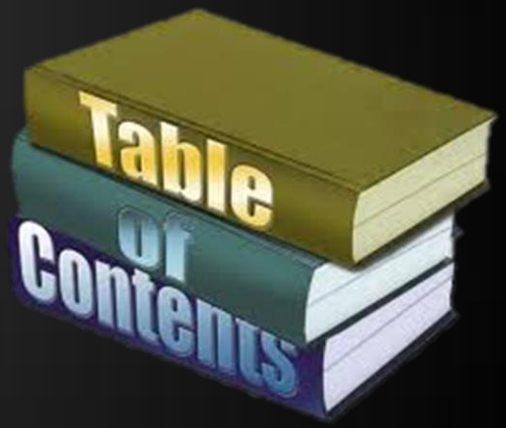
Telerik Software Academy

Learning & Development Team

http://academy.telerik.com

Telerik Academy

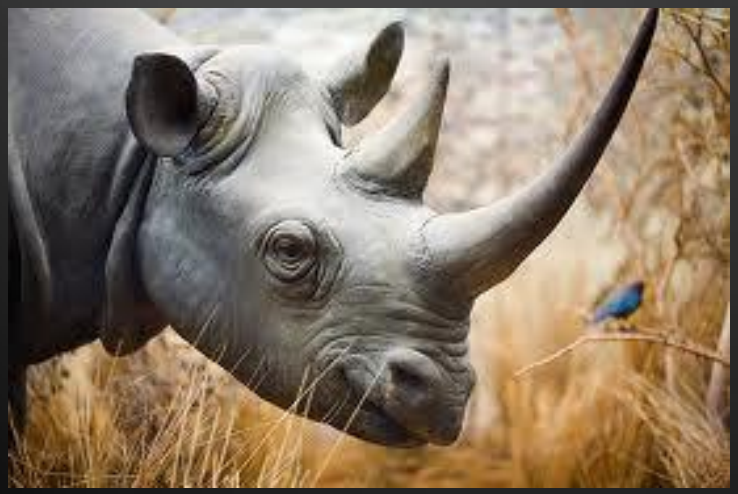
Table of Contents

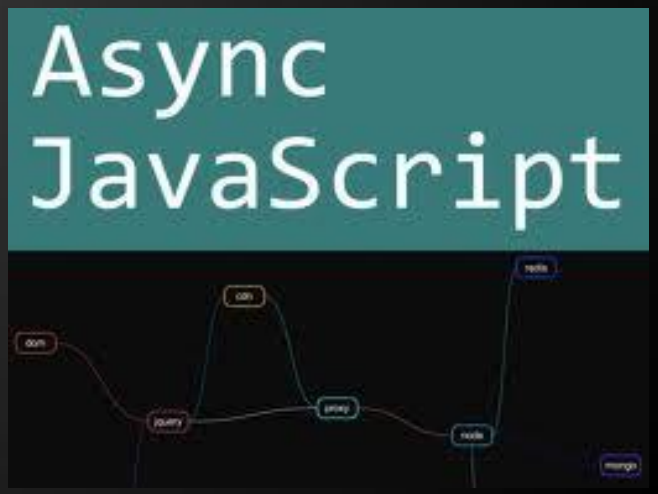


Telerik Academy

Asynchrony in JavaScript

How to do it





Asynchrony in JavaScript

- JavaScript is single-threaded
  - Long-running operations block other operations
- Asynchronous operations in JavaScript
  - Break up long operations into shorter ones
    - So other operations can "squeeze in"
  - Delayed execution
    - Postpone heavy operations to the end of the event loop
    - To give event handlers the ability to respond

Asynchrony in JavaScript (2)

- Browsers provide some asynchronous APIs
  - Web workers
  - AJAX
  - Geolocation
  - CSS3 animations, etc.
- All of the above require callbacks
  - Functions to call at some point
    - When beginning to do work
    - After the work is done to transmit values

Telerik Academy

Callback-oriented Programming

with JavaScript

Callback-oriented Programming

- Callback function
  - A function object passed to another function
  - The other function can call the passed one
  - The other function can give arguments
- Examples of callbacks:
  - Event handlers are sort-of callbacks
  - setTimeout and setInterval take a callback argument
  - Some OOP patterns use callbacks for _super

Telerik Academy

Simple Callback

Live Demo

Callback-oriented Programming (2)

- Callback-oriented programming
  - Functions get passed to each other
  - Each functions calls the passed ones
    - To continue the work
    - To process values
  - Inversion of control principle ("don't call us, we'll call you")
  - Problems:
    - "Return" values by passing to other functions
    - Heavily nested functions are hard to understand
    - Errors and exceptions are a nightmare to process

Telerik Academy

Callback with Value Needed by Other Method

Live Demo

Telerik Academy

Using Browser-provided Async APIs

How to access browser APIs asynchronously

Telerik Academy

Using Browser-provided Asynchronous APIs

- How do asynchronous browser APIs work?
  - JavaScript runs in one thread of the browser
  - The browser can create other threads
    - For its own needs, including async APIs
- How do we use asynchronous APIs with JS?
  - Request some browser API
    - Pass arguments for what you want
  - Provide callback methods to execute when the API has processed your request

Telerik Academy

Using Browser-provided Asynchronous APIs

- Using the Geolocation API
  - Locating the device takes time
  - To request the current position
    - Call navigator.geolocation.getCurrentPosition
    - Pass in a success and error handler
    - i.e. pass in callback functions
    - Process the data
    - Visualize it accordingly

Telerik Academy

Callback-based usage of the Geolocation API

Live Demo

Telerik Academy

Summary on callback-based usage of Geolocation

- We need some function nesting
  - We want to have good function cohesion
  - Provide separate functions for different operations
- What will happen with a larger application?
  - Lots of levels of nesting
  - Nightmarish error-handling
    - Errors are easy to get lost
    - Handling needs to happen in inappropriate places

Telerik Academy

Promises

The evolution of Callback-oriented programming

(switch on your imagination)

Promises

- A promise is an object which represents an eventual (future) value
  - Methods "promise" they will return a value
    - Correct or representing an error
- A Promises can be in one of three states:
  - Fulfilled (resolved, succeded)
  - Rejected (an error happened)
  - Pending (unfulfilled yet, still being computed)

Promises

- Promise objects can be used in code as if their value is known
  - Actually we attach code which executes
    - When the promise is fulfilled
    - When the promise is rejected
    - When the promise reports progress (optionally)
- Promises are a pattern
  - No defined implementation, but strict requirements
  - Initially described in CommonJS Promises/A

Promises

- More specifically:
  - Each promise has a .then() method accepting 3 parameters:
    - Success, Error and Progress function
    - All parameters are optional
  - So we can write:

```
promiseMeSomething()
.then(function (value) {
        //handle success here
}, function (reason) {
        //handle error here
});
```

- * Provided promiseMeSomething returns a promise

Promises

- Each .then() method returns a promise in turn
  - Meaning promises can be chained:

```
asyncComputeTheAnswerToEverything()
        .then(addTwo)
        .then(printResult, onError);
```

- Promises enable us to:
  - Remove the callback functions from the parameters and attach them to the "result"
  - Make a sequence of operations happen
  - Catch errors when we can process them

Promises

- The full and modern description of Promises:
  - CommonJS Promises/A+
    http://promises-aplus.github.io/promises-spec/
  - An improvement of the Promises/A description
  - Better explanation of border cases
  - Several libraries fulfill the A+ spec:
    - A notable example is Kris Kowal's Q library
      https://github.com/kriskowal/q

Telerik Academy

The Q Promise Library

A rich CommonJS Promises/A+ library

The Q Promise Library

- Some ways of using the Q module:
  - Option 1: Download it from the Q repository
    1. Add the q.js or q.min.js file to your project
    2. Reference it with a <script> tag
    3. The Q library will create a global Q object you can use
  - Option 2: Using NuGet in Visual Studio
    1. Open the Package Manager Console
    2. Type `Install-Package Q`
    3. Go to step 2 in the previous option

Telerik Academy

Getting Started with Q

Live Demo

The Q Promise Library

- Creating Promises with Q
  - We can make a regular function into a Promise
  - i.e. we take the return value as the value of the function
  - Using the function `Q.fcall()`
  - First parameter is the function to call
  - The following parameters are passed into the called function
  - The return value of .fcall() is a promise with the function's return value

Telerik Academy

Creating Promises from Function Values

Live Demo

The Q Promise Library

- Promises from callback-based functions
  - Often we need to wrap a callback in a promise
  - We can use the Deferred object in Q

```
var deferred = Q.defer()
```

  - Deferred is an object which can
    - self-fulfill itself with some argument:

```
deferred.resolve(result)
```

    - Or self-reject itself with an error:

```
deferred.reject(reason)
```

    - Get the promise which will be fulfilled/rejected:

```
deferred.promise
```

Telerik Academy

Creating Promises from Callback-based Functions

Live Demo

The Q Promise Library

- The .then() method in the Q library
  - Follows the specification
  - Success, error and progress handlers
  - Value returned from the promise is passed to the success handler
  - Errors in the promise are passed to the error handler
  - Any progress data the promise reports is passed to the progress handler

Telerik Academy

Full-featured .then() in Q

Live Demo

The Q Promise Library

- Chaining promises
  - Each .then() method returns a new promise
  - The value of the promise is:
    - The return value of the success handler, if the previous promise is fulfilled
    - The error data if the previous promise failed

Telerik Academy

Promise Chaining in Q

Live Demo

The Q Promise Library

- Error propagation
  - Errors are propagated up the promise chain
  - The first error handler processes the error
    - All promises after the error are in the rejected state
    - No success handler will be called
- .done() function
  - Good practice to place at the end of chain
  - If no error handler is triggered, done will throw an exception

Telerik Academy

Error Propagation in Q

Live Demo

The Q Promise Library

- Use `Q.all([promise1, promise2, …)` to get a promise for a collection of promises:
  - Fulfilled when all promises are fulfilled
    - Success handler gets the results as an array
  - Rejected if any promise is rejected
    - Error handler gets the error of the first rejected
- Use `.spread()` instead of `.then()` to spread the array of results into `arguments` of success handler

Telerik Academy

Collections of Promises

Live Demo

The Q Promise Library

- We now know the basics of the Q library and Promises
  - There's a lot more functionality in Q
    - E.g. each promise instance method has a 'static' counterpart:
    - `promise.then(…)` and `Q.then(promise, …)`
  - Read the documentation
- We will re-write the Geolocation example
  - Without callbacks
  - With promises and promise chains

Telerik Academy

Geolocation with Q Promises

Live Demo

Telerik Academy

Promises in jQuery

Creation, Usage, Specifics

Promises in jQuery

- jQuery supports CommonJS Promises/A
  - Since jQuery 1.5
  - * almost (details here)
    - .then() didn't return a promise until jQuery 1.8
      - .pipe() was used
    - Errors in handlers don't propagate up
- Generally, jQuery promises look and feel the same as Q promises
  - Use them the same way, but be cautious

Promises in jQuery

- The jQuery.Deferred object
  - Extended, mutable promise
    - Just like in Q
  - Can resolve and reject itself with arguments
  - Can retrieve an immutable promise object
    - Which in fact will be resolved/rejected

```
var d = jQuery.Deferred(); //$.Deferred()
d.resolve(result); //resolves the deferred,
calling success handlers
d.reject(reason); // rejects the deferred,
calling error handlers
d.promise(); //note: here promise is a function
```

Telerik Academy

Creating and Using Promises in jQuery

Live Demo

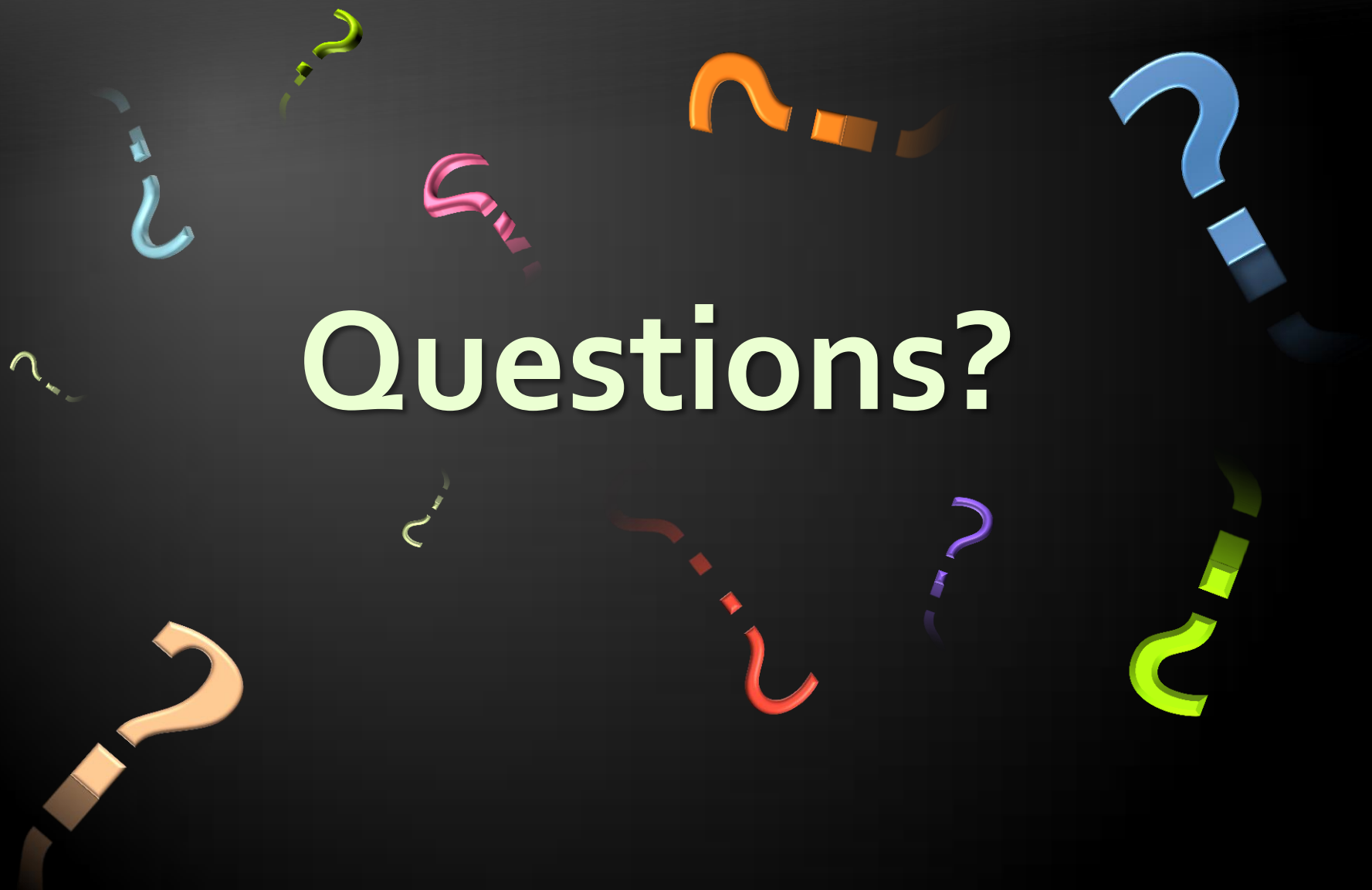
Promises in jQuery

- Specifics of error propagation in jQuery
  - Calling reject (from deferred) works as expected
    - Only error handlers are called
  - Errors in success/error handlers are not propagated
    - Thrown exceptions will not be processed by error handlers in the chain

```
promiseMeSomething().then(function(){
        invalid code //throws an exception
}).then(function(){}, function(err){
        //this error handler will not be called
})
```

Telerik Academy

Promises and Asynchronous Programming



Telerik Academy

Free Trainings @ Telerik Academy

- "Web Design with HTML 5, CSS 3 and JavaScript" course @ Telerik Academy
  - html5course.telerik.com

- Telerik Software Academy
  - academy.telerik.com

Telerik Academy

- Telerik Academy @ Facebook
  - facebook.com/TelerikAcademy

- Telerik Software Academy Forums
  - forums.academy.telerik.com