David Shariff
MANAGE THE UI PLATFORM TEAM @ AMAZON.COM

Futures and Promises in JavaScript

With JavaScript usage constantly on the increase, asynchronous event-driven applications are becoming more and more popular. However, a common issue many developers face is with result-dependent operations being used in an asynchronous environment.

You will often end up with something like:

```
doA(function(aResult) {
    // do some stuff inside b then fire callback
    doB(aResult, function(bResult) {
        // ok b is done, now do some stuff in c and fire callba
        doC(bResult, function(cResult) {
            // finished, do something here with the result from
        });
    });
});
```

Since each step requires the previous steps result, you will regularly see a pattern where people start nesting the callback functions within each other's callbacks. These nested callbacks become difficult to maintain, understand and follow in

larger asynchronous applications. Simple async flow such as do (A + B + C) then do D becomes an increasingly complex task.

A solution to use in this situation is the *Promise / Futures* pattern, which represents the result of a callback that has not happened yet. The concept is quite simple, instead of a function blocking and waiting to complete before returning the result, it simply returns immediately when invoked with an object that *promises* the future computation / result. This results in a non-blocking behaviour:

```
doA()
    .then(function() { return doB(); })
    .then(function() { return doC(); })
    .done(function() { /* do finished stuff here */ });
```

Writing your code using the *Promise / Future* pattern gives you most of the benefits of using nested callbacks, along with a cleaner, more structured code that is easier to maintain, understand and follow in most asynchronous environments.

Promises / Futures are not the ultimate solution, and there are dozens upon dozens of other solutions that all have their own benefits and drawbacks, each which should be explored in their own right for different situations.

Related Posts...

1 JavaScript Inheritance Patterns

2 Identifier Resolution and Closures in the JavaScript Scope Chain

3 What is the Execution Context & Stack in JavaScript?

4 Chaining Variable Assignments in JavaScript: Words of Caution

5 JavaScript's 'this' Keyword

Comments

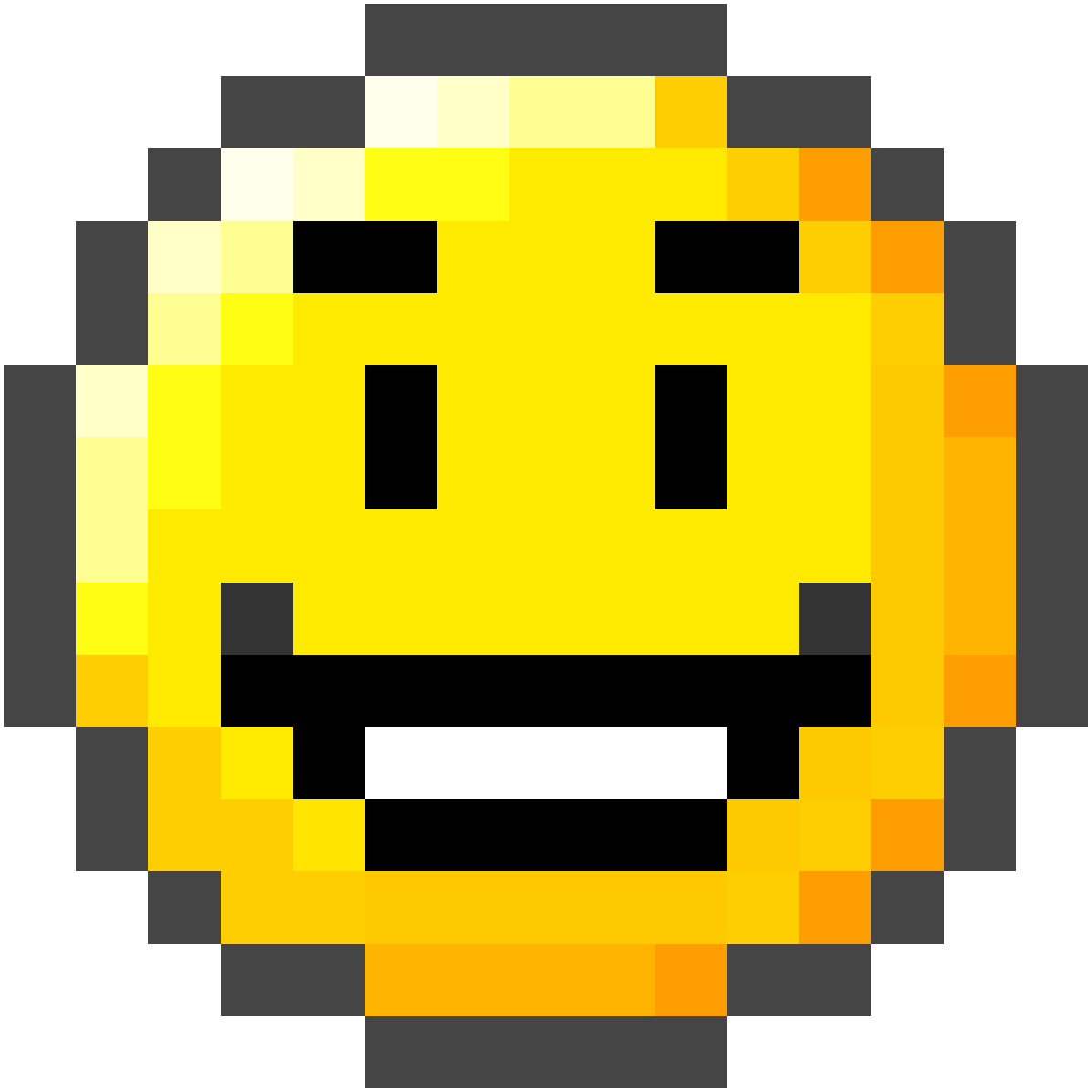
Kalyan ***said*** **on 31/03/2013 at 8:12 am:**

Hi
Thanks for giving the detailed information on Promises in JavaScript, they are very useful in writing the Windows Apps. Some code snippets on this topic can be found in below post
http://www.techbubbles.com/javascript/using-promises-in-javascript/

Reply ↓

Rupali Gangarde ***said*** **on 10/06/2016 at 1:33 am:**

Helpful article.
Thanks



Reply ↓

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

[Spam Check] What is: * + 8 = ten

Comment

Post Comment

← READ MORE