The Power of Promises and Parallel XQuery

James Wright

About me

- XQuery User/Advocate
- Located: Denver, Colorado
- Email: james.jw@hotmail.com
- Twitter: @jamesthewright
- Github: https://github.com/james-jw

Methods of Parallelism

- System Specific functions
 - Marklogic's spawn
 - Exist-db's util:eval-async
- Batch process spawning
- Optimizations
- Scheduling

What is the problem?

- Complex
- Multi layered
- Error prone
- Fault intolerant
- Inefficient

The Promise Pattern?

- 1. Mechanism for deferring work
- 2. Pipeline architecture
- 3. Fault tolerant
- 4. Pattern for resource contention management
- 5. Easy to implement
- 6. Adaptable

xq-promise

- A small module for BaseX
 - Requires an additional .jar in the BaseX classpath
- Implements the promise pattern
 - As seen in JQuery.js, Q.js
- Enables parallel processing capabilities with XQuery in BaseX

Find it on Github!

https://github.com/james-jw/xq-promise

Install with EXPackage:

- Run:
 - "https://raw.githubusercontent.com/jamesjw/xq-promise/master/dist/xq-promise-0.8.2-beta.xar"! repo:install()

defer

- Defers a function of work
- Returns a promise function

```
import module namespace p = 'https://github.com/james-jw/xq-promise';
let $greet := function($name) { 'Hello ' || $name }
let $promise := p:defer($greet, 'world')
return
$promise
```

function Promise#0

Defer continued

```
import module namespace p = 'https://github.com/james-jw/xq-promise';
let $greet := function($name) { 'Hello ' || $name }
let $promise := p:defer($greet, 'world')
return
$promise()
```

Hello world

Callbacks

Mechanism for pipelining

Events:

- then
- done
- fail
- always

2/17/2016

then

- Called on success
- Has ability to alter the pipeline result
 - Usually does

Hello world!

2/17/2016

done

- Also called on success
- Does not alter pipeline result.
- Useful for logging or troubleshooting

Hello world!

Greet finished: Hello world!

fail

- Called on failure
 - Initial work or callback throws an exception
- Provided a map(*) with error details:

```
map {
    'code': 'error code',
    'description': 'error description',
    'value': 'error value',
    'module': 'file',
    'line': 'line number',
    'column': 'column number',
    'additional': map {
        'deferred': 'Function item which failed. Can be used to retry the request',
        'arguments': 'The arguments provided to the failed deferred.'
    }
}
```

fail ... continued

- Acts like the 'catch' clause
- If an error is thrown the entire query ceases
- If an empty or non-empty sequence is returned, the error is ignored
 - Pipeline continues with the value or empty sequence returned

2/17/2016

fail ... continued

```
Failed to greet: {
  "additional": {
    "arguments": [
      "world"
    ],
    "deferred": function (anonymous)#1
},
  "module": "C:/Program Files (x86)/BaseX/repo/promise/file2",
  "code": "FOER0000",
  "value": null,
  "line": 2,
  "column": 42,
  "description": "Halted on error()."
}
```

Demo

always

- Called on success or failure
- Has no affect on pipeline results
- Useful in logging

Multiple callbacks

- All callback events accept multiple callbacks
- Called in order they were added

when

- Combines two or more:
 - Deferred objects
 - Zero arity functions
- Returns a single new deferred
- Enables complex pipeline logic

2/17/2016

when ... continued

Hello world | "Hello mars"

The Power of Promise and Parallel Execution

- We now understand
 - How to defer work
 - How to add callbacks
 - How to combine work

2/17/2016

Fork-join

- Consumes a sequence of one or more:
 - Deferred objects
 - Zero arity functions

```
import module namespace geo = "http://expath.org/ns/geo";
import module namespace p = 'https://github.com/james-jw/xq-promise';
declare namespace gml = 'http://www.opengis.net/gml';

let $counties := db:open('DetailedCounties')//gml:Polygon
let $area :=
for $county in $counties return
p:defer(geo:area(?), $county)
return

sum($area => p:fork-join())
```

Demo

Fork

- Similar to defer
- Forks the work immediately
- Returns a 'locked' promise

```
import module namespace geo = "http://expath.org/ns/geo";
import module namespace p = 'https://github.com/james-jw/xq-promise';
declare namespace gml = 'http://www.opengis.net/gml';

let $counties := db:open('DetailedCounties')//gml:Polygon
let $area :=
for $county in $counties return
    p:fork(geo:area(?), $county)
    return
    sum($area ! .())
```

Demo

Map - reduce

```
1 import module namespace geo = "http://expath.org/ns/geo";
 2 import module namespace p = 'https://github.com/james-jw/xq-promise';
 3 declare namespace gml= 'http://www.opengis.net/gml';
 5 let $counties := db:open('DetailedCounties')//gml:Polygon
 6 let $area :=
       for tumbling window $gml in $counties
       start $s at $spos when true()
 8
       end at $epos when $epos - $spos = 1000
10
     return
         p:when($gml ! p:fork(geo:area(?), .))
11
            => p:then(sum(?))
12
13 return
     sum($area ! .())
14
```

Demo

Implementation

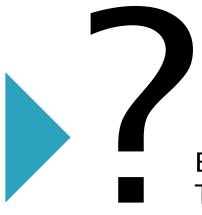
- XqPromise Module
- XqDeferred Promise object
- XqForkJoinTask -
 - Java 7's ForkJoinPool

2/17/2016

Current Limitations

- ▶ BETA!!
- Database updates in a fork
- XQuery Transform clause in a fork
- Resolved in custom BaseX build:
 - Caused by contention with update list

2/17/2016



Email: james.jw@hotmail.com

Twitter: @jamesthewright

Github: https://github.com/james-jw