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/james-jw/xq-promise/master/dist/xq-promise-0.8.2-beta.xar" ! repo:install()

defer

- Defers a function of work
- Returns a promise function

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

- function Promise#0

Defer continued

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

- Hello world

Callbacks

- Mechanism for pipelining

Events:

- then
- done
- fail
- always

then

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

```
1 import module namespace p = 'https://github.com/james-jw/xq-promise';
2 let $greet := function($name) { 'Hello ' || $name }
3 let $add-enthusiasm := function($phrase) { $phrase || '!' }
4 let $promise := p:defer($greet, 'world')
5    => p:then($add-enthusiasm)
6 return
7   $promise()
```

- Hello world!

done

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

```
1 import module namespace p = 'https://github.com/james-jw/xq-promise';
2 let $greet := function($name) { 'Hello ' || $name }
3 let $add-enthusiasm := function($phrase) { $phrase || '!' }
4 let $promise := p:defer($greet, 'world')
5    => p:then($add-enthusiasm)
6    => p:done(trace(?, 'Greet finished: '))
7 return
8   $promise()
```

- 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

fail … continued

```
1 import module namespace p = 'https://github.com/james-jw/xq-promise';
2 let $greet := function($name) { fn:error() }
3 let $add-enthusiasm := function($phrase) { $phrase || '!' }
4 let $promise := p:defer($greet, 'world')
5    => p:then($add-enthusiasm)
6    => p:done(trace(?, 'Greet finished: '))
7    => p:fail(trace(?, 'Failed to greet: '))
8 return
9   $promise()
```

```
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

```
1 import module namespace p = 'https://github.com/james-jw/xq-promise';
2 let $greet := function($name) { fn:error() }
3 let $add-enthusiasm := function($phrase) { $phrase || '!' }
4 let $promise := p:defer($greet, 'world')
5    => p:then($add-enthusiasm)
6    => p:always(trace(?, 'Greet Result: '))
7 return
8   $promise()
```

Multiple callbacks

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

```
1 import module namespace p = 'https://github.com/james-jw/xq-promise';
2 let $greet := function($name) { 'Hello ' || $name }
3 let $add-enthusiasm := function($phrase) { $phrase || '!' }
4 let $quote := function($phrase) { '"' || $phrase || '"'}
5 let $promise := p:defer($greet, 'world')
6    => p:then(($add-enthusiasm, $quote))
7 return
8   $promise()
```

when

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

when … continued

```
1  import module namespace p = 'https://github.com/james-jw/xq-promise';
2  let $greet := function($name) { 'Hello ' || $name }
3  let $quote := function($phrase) { '"' || $phrase || '"'}
4  let $p1 := p:defer($greet, 'world')
5  let $p2 := p:defer($greet, 'mars')
6      => p:then($quote)
7  let $promise := p:when(($p1, $p2))
8      => p:then(string-join(?, ' | '))
9  return
10   $promise()
```

- 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

Fork–join

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

```
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';
4
5  let $counties := db:open('DetailedCounties')//gml:Polygon
6  let $area :=
7      for $county in $counties return
8      p:defer(geo:area(?), $county)
9  return
10   sum($area => p:fork-join())
```

Demo

Fork

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

```
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';
4
5  let $counties := db:open('DetailedCounties')//gml:Polygon
6  let $area :=
7      for $county in $counties return
8      p:fork(geo:area(?), $county)
9  return
10   sum($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';
4
5  let $counties := db:open('DetailedCounties')//gml:Polygon
6  let $area :=
7      for tumbling window $gml in $counties
8      start $s at $spos when true()
9      end at $epos when $epos - $spos = 1000
10     return
11       p:when($gml ! p:fork(geo:area(?), .))
12          => p:then(sum(?))
13 return
14   sum($area ! .())
```

Demo

Implementation

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

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

Email: james.jw@hotmail.com
Twitter: @jamesthewright
Github: https://github.com/james-jw