Geekbang› 极客邦科技 InfoQ

QCon 全球软件开发大会 【北京站】2016

More than async

monad-based language extensions / 杨博

International Software Development Conference

More than async

monad-based language extension

杨博

杨博

- CEO、共同创始人、项目经理、游戏架构师、 游戏设计师、桌游吧老板、译者、游戏客服……

杨博

- ~~CEO、共同创始人、项目经理、游戏架构师、游戏设计师、桌游吧老板、译者、游戏客服……~~

杨博

- ~~CEO、共同创始人、项目经理、游戏架构师、游戏设计师、桌游吧老板、译者、游戏客服……~~
- Thoughtworks Consultant

杨博

- ~~CEO、共同创始人、项目经理、游戏架构师、游戏设计师、桌游吧老板、译者、游戏客服……~~
- Thoughtworks Consultant
- open-source contributor for <u>each</u> project

目录

- 为什么要有async?
- 实现async的不同机制
  - level 0: callback
  - level 1: generator
  - level 2: async/await
  - level 3: monad

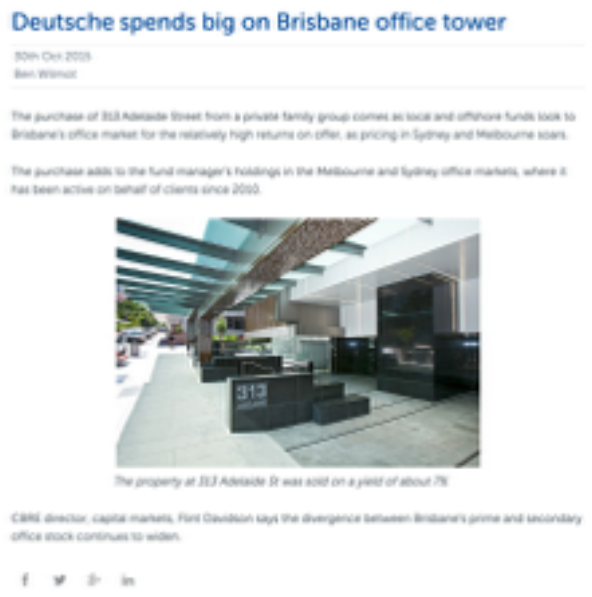
目录

- 为什么要有async?
- 实现async的不同机制
  - level 0: callback
  - level 1: generator
  - level 2: async/await
  - level 3: monad

为什么要有async?
- 问题描述

实现一个单页面微服务:

- 上游RESTful API
  - 文章主内容 - Wordpress API
  - 广告 - 分类信息 API
- 组装上游数据渲染页面



Deutsche spends big on Brisbane office tower

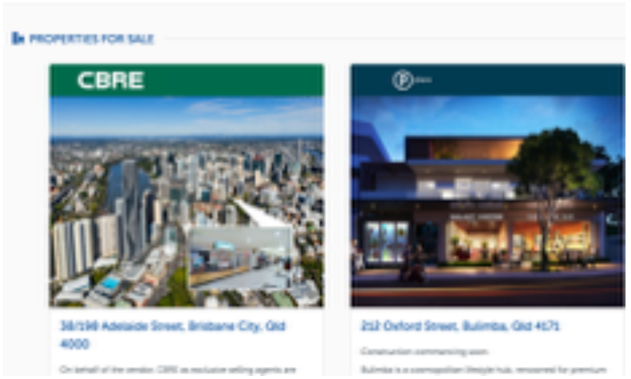
30th Oct 2015
Ben Wilmot

The purchase of 313 Adelaide Street from a private family group comes as local and offshore funds look to Brisbane's office market for the relatively high returns on offer, as pricing in Sydney and Melbourne soars.

The purchase adds to the fund manager's holdings in the Melbourne and Sydney office markets, where it has been active on behalf of clients since 2010.

The property at 313 Adelaide St was sold on a yield of about 7%

CBRE director, capital markets, Flint Davidson says the divergence between Brisbane's prime and secondary office stock continues to widen.



PROPERTIES FOR SALE

CBRE

38/198 Adelaide Street, Brisbane City, Qld 4000

On behalf of the vendor, CBRE as exclusive selling agents are

212 Oxford Street, Bulimba, Qld 4171

Construction commencing soon.

Bulimba is a cosmopolitan lifestyle hub, renowned for premium

为什么要有async?

- 同步实现

```
function getNewsArticle(){
    var post = wordpressApi.getPostbyCategory('news');
    var listings = syndicationApi.getListings(post.district);
    var article = {'content': post.content, 'listings': listings};
    return article;
}
```

为什么要有async?

- 同步实现

```
function getNewsArticle(){
    var post = wordpressApi.getPostbyCategory('news');
    var listings = syndicationApi.getListings(post.district);
    var article = {'content': post.content, 'listings': listings};
    return article;
}
```

优点

为什么要有async?

- 同步实现

```
function getNewsArticle(){
    var post = wordpressApi.getPostbyCategory('news');
    var listings = syndicationApi.getListings(post.district);
    var article = {'content': post.content, 'listings': listings};
    return article;
}
```

优点	简单直接 易于理解

为什么要有async?

- 同步实现

```
function getNewsArticle(){
    var post = wordpressApi.getPostbyCategory('news');
    var listings = syndicationApi.getListings(post.district);
    var article = {'content': post.content, 'listings': listings};
    return article;
}
```

优点	简单直接 易于理解
缺点	

为什么要有async?

- 同步实现

```
function getNewsArticle(){
    var post = wordpressApi.getPostbyCategory('news');
    var listings = syndicationApi.getListings(post.district);
    var article = {'content': post.content, 'listings': listings};
    return article;
}
```

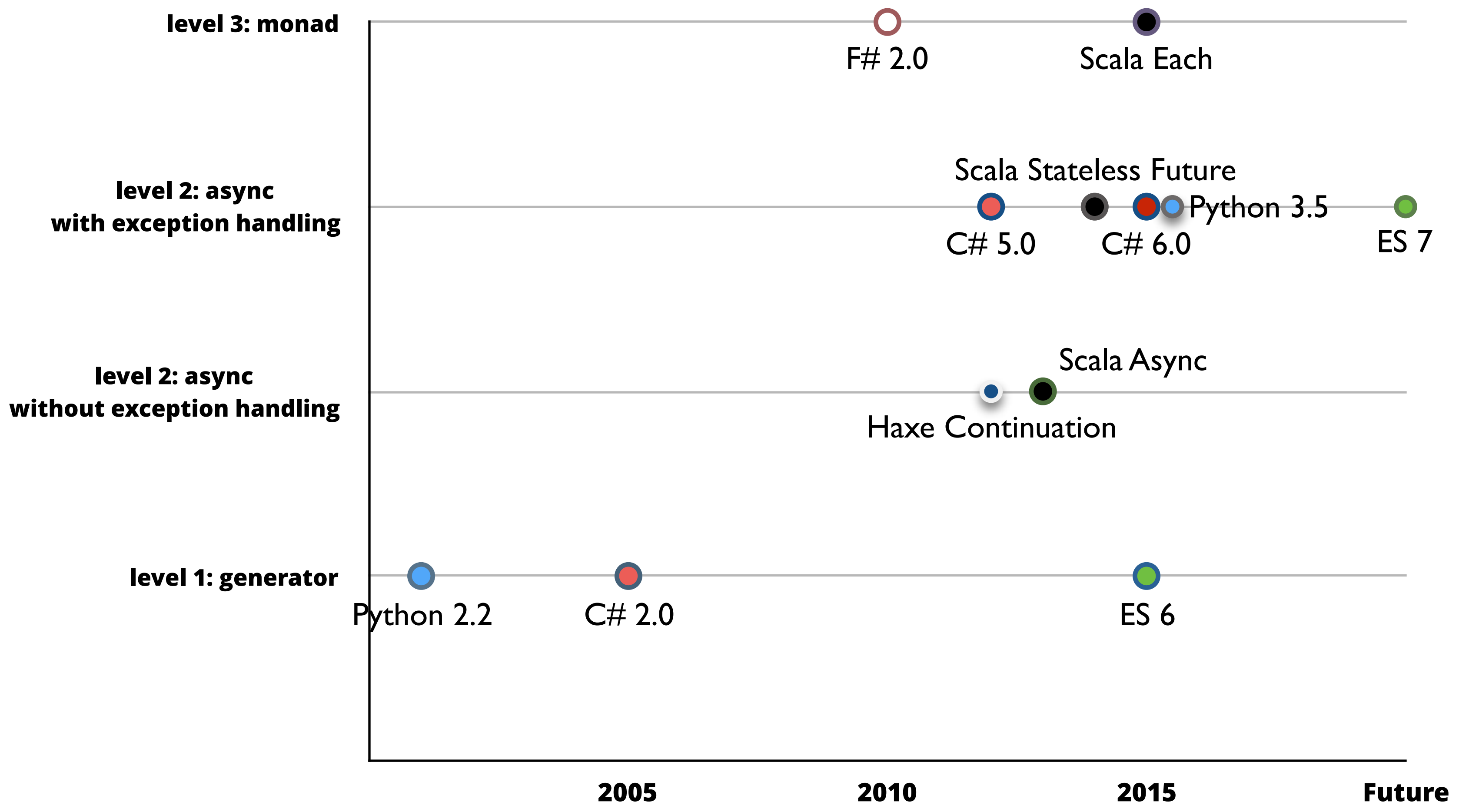
优点	简单直接 易于理解
缺点	并发低 阻塞线程

目录

- 为什么要有async?
- 实现async的不同机制
  - level 0: callback
  - level 1: generator
  - level 2: async/await
  - level 3: monad

实现async的不同机制

- level 1 至 level 3 在各种语言中的现状



目录

- 为什么要有async?
- 实现async的不同机制
  - level 0: callback
  - level 1: generator
  - level 2: async/await
  - level 3: monad

level 0: callback
- Continuation-passing style (CPS)

```
function getNewsArticle(callback) {
    wordpressApi.getPostbyCategory('news', function (post) {
        syndicationApi.getListings(post.district, function (listings) {
            var article = {'content': post.content, 'listings': listings};
            callback(article);
        });
    });
}
```

level 0: callback
- CPS with exception handling

```
function getNewsArticle(callback) {
    wordpressApi.getPostbyCategory('news', function (err, post) {
        if (err) {
            callback(err)
            return
        }
        syndicationApi.getListings(post.district, function (err, listings) {
            if (err) {
                callback(err)
                return
            }
            var article = {'content': post.content, 'listings': listings};
            callback(null, article);
        });
    });
}
```

level 0: callback
- Promise and then

```
function getNewsArticle(callback) {
    wordpressApi.getPostbyCategory('news').then(function (post) {
            syndicationApi.getListings(post.district).then(function(listings){
                    var article = {
                        'content': post.content,
                        'listings': listings
                    };
                    callback(article);
                }
            );
        }
    );
}
```

目录

- 为什么要有async?
- 实现async的不同机制
  - level 0: callback
  - level 1: generator
  - level 2: async/await
  - level 3: monad

目录

- 为什么要有async?
- 实现async的不同机制
  - level 0: callback
  - level 1: generator
  - level 2: async/await
  - level 3: monad

目录

- 为什么要有async?
- 实现async的不同机制
  - level 0: callback
  - level 1: generator
  - level 2: async/await
  - level 3: monad

level 1: generator
- ECMAScript 2015

```
function* idMaker(){
  var index = 0;
  while(true)
    yield index++;
}

var gen = idMaker();

console.log(gen.next().value); // 0
console.log(gen.next().value); // 1
console.log(gen.next().value); // 2
```

level 1: generator
- C# 2.0 and Unity

```
// Get the latest webcam shot from outside "Friday's" in Times Square
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public string url = "http://images.earthcam.com/ec_metros/ourcams/
fridays.jpg";
    IEnumerator Start() {
        WWW www = new WWW(url);
        yield return www;
        Renderer renderer = GetComponent<Renderer>();
        renderer.material.mainTexture = www.texture;
    }
}
```

目录

- 为什么要有async?
- 实现async的不同机制
  - level 0: callback
  - level 1: generator
  - level 2: async/await
  - level 3: monad

level 2: async/await

```
async function getNewsArticle() {
    let post = await wordpressApi.getPostbyCategory('news');
    let listings = await syndicationApi.getListings(post.district);
    let article = {'content': post.content, 'listings': listings};
    return article;
}
```

level 2: async/await

- 定义一个异步（或同步）操作函数

level 2: async/await

- 定义一个异步（或同步）操作函数

js direct style	`function op() {`
	`    //???`
	`}`

level 2: async/await

- 定义一个异步（或同步）操作函数

js direct style	`function op() {`
	`    //???`
	`}`

js CPS	`function op(callback) {`
	`    //???`
	`}`

level 2: async/await

- 定义一个异步（或同步）操作函数

js direct style	`function op() { //??? }`
ES7	`async function op() { //??? }`
js CPS	`function op(callback) { //??? }`

js direct style

```
function op() {
    //???
}
```

ES7

```
async function op() {
    //???
}
```

js CPS

```
function op(callback) {
    //???
}
```

level 2: async/await

- 定义一个异步（或同步）操作函数

js direct style

```
function op() {
    //???
}
```

js CPS

```
function op(callback) {
    //???
}
```

ES7

```
async function op() {
    //???
}
```

Haxe

```
@:async function op() {
    //???
}
```

C#

```
async Task op()
{
  //???
}
```

F#

```
let op() = async {
  //???
}
```

Scala each

```
def op = monadic[Future] {
  //???
}
```

level 2: async/await

- 执行一个操作

level 2: async/await

- 执行一个操作

js direct style	`var result = op();`

level 2: async/await

- 执行一个操作

js direct style	`var result = op();`

js CPS	`op(function(result) {`
	`    //???`
	`});`

level 2: async/await

- 执行一个操作

js direct style	`var result = op();`
ES7	`var result = await op();`

js CPS

```
op(function(result) {
    //???
});
```

level 2: async/await

- 执行一个操作

js direct style	`var result = op();`
ES7	`var result = await op();`
Haxe	`var result = @await op();`
F#	`let! result = op()`

js CPS	`op(function(result) {`
	`    //???`
	`});`

C#	`var result = await op();`
Scala each	`val result = op().each`

level 2: async/await

- 顺序执行多个操作

level 2: async/await

- 顺序执行多个操作

js direct style	`var result1 = op1();`
	`var result2 = op2(result1);`

level 2: async/await
- 顺序执行多个操作

js direct style

```
var result1 = op1();
var result2 = op2(result1);
```

js CPS

```
op1(function(result1) {
    op2(result1, function(result2) {
        //???
    });
});
```

level 2: async/await

- 顺序执行多个操作

js direct style	`var result1 = op1();`
	`var result2 = op2(result1);`

ES7	`var result1 = await op1();`
	`var result2 = await op2(result1);`

js CPS

```
op1(function(result1) {
    op2(result1, function(result2) {
        //???
    });
});
```

level 2: async/await

- 顺序执行多个操作

js direct style

```
var result1 = op1();
var result2 = op2(result1);
```

js CPS

```
op1(function(result1) {
    op2(result1, function(result2) {
        //???
    });
});
```

ES7

```
var result1 = await op1();
var result2 = await op2(result1);
```

Haxe

```
var result1 = @await op1();
var result2 = @await op2(result1);
```

C#

```
var result1 = await op1();
var result2 = await op2(result1);
```

F#

```
let! result1 = op1()
let! result2 = op2(result1)
```

Scala each

```
val result1 = op1().each
val result2 = op2(result1).each
```

level 2: async/await

- 把多个操作的结果加起来

level 2: async/await

- 把多个操作的结果加起来

js direct style	`var r = op1() + op2();`

level 2: async/await

- 把多个操作的结果加起来

js direct style	`var r = op1() + op2();`

js CPS	see code below

```
op1(function(result1) {
    op2(function(result2) {
        var r = result1 + result2;
    });
});
```

level 2: async/await
- 把多个操作的结果加起来

js direct style	`var r = op1() + op2();`
ES7	`var r = await op1() + await op2();`

js CPS

```
op1(function(result1) {
    op2(function(result2) {
        var r = result1 + result2;
    });
});
```

level 2: async/await

- 把多个操作的结果加起来

js direct style

```
var r = op1() + op2();
```

js CPS

```
op1(function(result1) {
    op2(function(result2) {
        var r = result1 + result2;
    });
});
```

ES7

```
var r = await op1() + await op2();
```

Haxe

```
var r = @await op1() + @await op2();
```

C#

```
var r = await op1() + await op2();
```

F#

```
let! result1 = op1()
let! result2 = op2()
let r = result1 + result2
```

Scala each

```
val r = op1().each + op2().each
```

level 2: async/await

- 循环执行操作

level 2: async/await

- 循环执行操作

js direct style	`var array = [3, 5, 7];`
	`for (var i = 0; i < array.length; i++) {`
	`    op(array[i]);`
	`}`

level 2: async/await

- 循环执行操作

js direct style	`var array = [3, 5, 7];`
	`for (var i = 0; i < array.length; i++) {`
	`    op(array[i]);`
	`}`

js CPS

```
var array = [3, 5, 7];
function loop(i) {
    op(array[i], function() {
        if (i < array.length) {
            loop(i + 1);
        }
    });
}
loop(0);
```

level 2: async/await

- 循环执行操作

js direct style

```
var array = [3, 5, 7];
for (var i = 0; i < array.length; i++)
{
    op(array[i]);
}
```

ES7

```
var array = [3, 5, 7];
for (let element of array) {
    await op(element);
}
```

js CPS

```
var array = [3, 5, 7];
function loop(i) {
    op(array[i], function() {
        if (i < array.length) {
            loop(i + 1);
        }
    });
}
loop(0);
```

level 2: async/await

- 循环执行操作

js direct style

```
var array = [3, 5, 7];
for (var i = 0; i < array.length; i++)
{
    op(array[i]);
}
```

js CPS

```
var array = [3, 5, 7];
function loop(i) {
    op(array[i], function() {
        if (i < array.length) {
            loop(i + 1);
        }
    });
}
loop(0);
```

ES7

```
var array = [3, 5, 7];
for (let element of array) {
    await op(element);
}
```

Haxe

```
var array = [3, 5, 7];
for (element in array) {
    @await op(element);
}
```

C#

```
var array = new int [] { 3, 5, 7 };
foreach (int element in array) {
    await op(element);
}
```

F#

```
let list = [ 3; 5; 7 ]
for element in list do
    do! op(element)
```

Scala each

```
val list = List(3, 5, 7)
for (element <- list) {
  op(element).each
}
```

level 2: async/await

- 循环执行操作并把结果存进数组

level 2: async/await

- 循环执行操作并把结果存进数组

JS direct style	`var all = []; var array = [3, 5, 7]; for (var i = 0; i < array.length; i++) { all.push(op(array[i])); }`

```
var all = [];
var array = [3, 5, 7];
for (var i = 0; i < array.length; i++) {
    all.push(op(array[i]));
}
```

level 2: async/await

- 循环执行操作并把结果存进数组

JS direct style

```
var all = [];
var array = [3, 5, 7];
for (var i = 0; i < array.length; i++) {
    all.push(op(array[i]));
}
```

JS CPS

```
var array = [3, 5, 7];
function loop(i, result) {
    op(array[i], function(currentValue) {
        if (i < array.length) {
            loop(i + 1,
result.concat([currentValue]));
        } else {
            var all = result;
            ???
        }
    });
}
loop(0, []);
```

level 2: async/await
- 循环执行操作并把结果存进数组

JS direct style

```
var all = [];
var array = [3, 5, 7];
for (var i = 0; i < array.length; i++) {
    all.push(op(array[i]));
}
```

ES7

```
var array = [3, 5, 7];
var all = [ for (let element of array)
await op(element) ];
```

JS CPS

```
var array = [3, 5, 7];
function loop(i, result) {
    op(array[i], function(currentValue) {
        if (i < array.length) {
            loop(i + 1,
result.concat([currentValue]));
        } else {
            var all = result;
            ???
        }
    });
}
loop(0, []);
```

level 2: async/await

- 循环执行操作并把结果存进数组

JS direct style

```
var all = [];
var array = [3, 5, 7];
for (var i = 0; i < array.length; i++) {
    all.push(op(array[i]));
}
```

ES7

```
var array = [3, 5, 7];
var all = [ for (let element of array)
await op(element) ];
```

Haxe

```
var array = [3, 5, 7];
var all = [ for (element in array)
@await op(element) ];
```

F#

```
let list = [ 3; 5; 7 ]
let all = seq { for element in list do
yield! op(element) }
```

JS CPS

```
var array = [3, 5, 7];
function loop(i, result) {
    op(array[i], function(currentValue) {
        if (i < array.length) {
            loop(i + 1,
result.concat([currentValue]));
        } else {
            var all = result;
            ???
        }
    });
}
loop(0, []);
```

C#

```
var all = new List<int>();
var array = new int [] { 3, 5, 7 };
foreach (int element in array) {
    builder.Add(await op(element));
}
```

Scala each

```
val list = List(3, 5, 7)
val all = (for { i <- list.monadicLoop
} yield op(i).each).underlying
```

目录

- 为什么要有async?
- 实现async的不同机制
  - level 0: callback
  - level 1: generator
  - level 2: async/await
  - level 3: monad

level 3: monad
- Future monad

```
import com.thoughtworks.each.Monadic._
import scalaz.std.scalaFuture._

// Returns a Future of the sum of the length of each string in each parameter Future,
// without blocking any thread.
def concat(future1: Future[String], future2: Future[String]): Future[Int] =
monadic[Future] {
  future1.each.length + future2.each.length
}
```

level 3: monad
- Option monad

```
import com.thoughtworks.each.Monadic._
import scalaz.std.option._

def plusOne(intOption: Option[Int]) = monadic[Option] {
  intOption.each + 1
}
assertEquals(None, plusOne(None))
assertEquals(Some(16), plusOne(Some(15)))
```

level 3: monad
- List monad

```
import com.thoughtworks.each.Monadic._
import scalaz.std.list._

def plusOne(intSeq: List[Int]) = monadic[List] {
  intSeq.each + 1
}
assertEquals(Nil, plusOne(Nil))
assertEquals(List(16), plusOne(List(15)))
assertEquals(List(16, -1, 10), plusOne(List(15, -2, 9)))
```

level 3: monad
- Binding monad

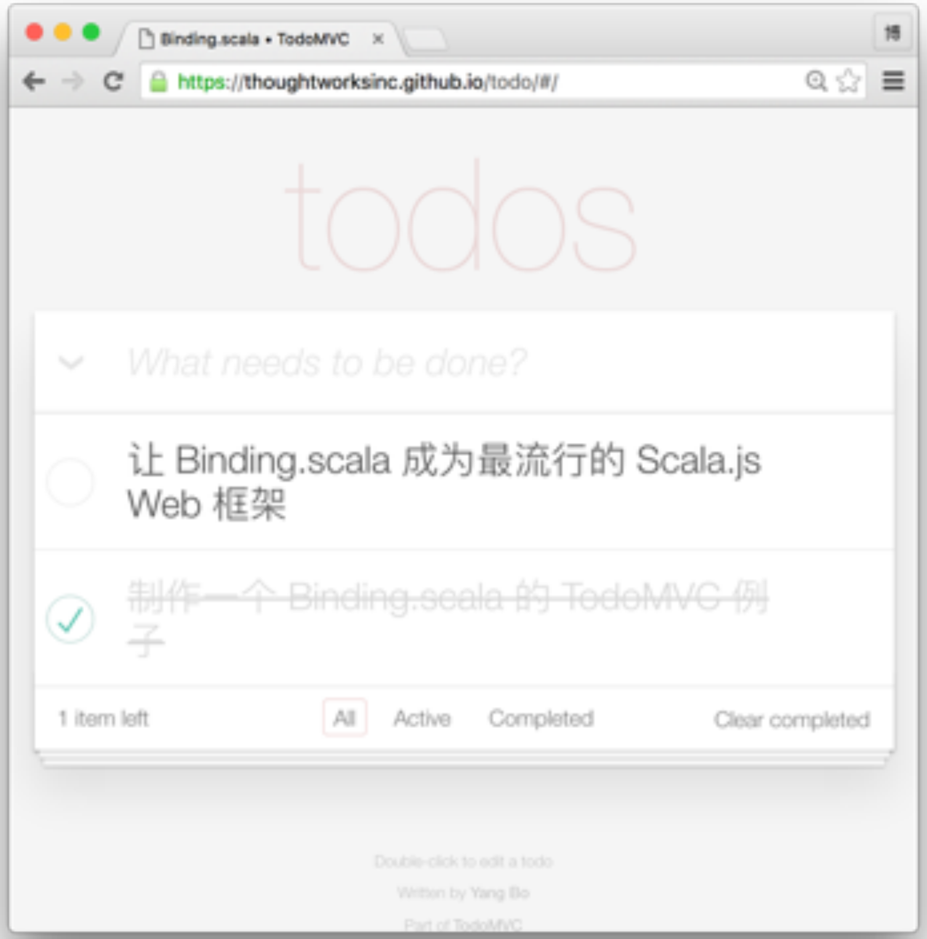
```
val target = Var("World")
val hello = monadic[Binding] {
  "Hello, " + target.each + "!"
}
hello.watch()

assert(hello.get == "Hello, World!")
target := "Each"
assert(hello.get == "Hello, Each!")
```

level 3: monad
- Binding monad

```
@dom
override def render = {
  val value = Var("")
  <div>
    <input onchange={ event: Event => value := dom.currentTarget.value }/>
    Your input value is { value.each }
  </div>
}
```

DEMO URL:
https://thoughtworksinc.github.io/Binding.scala/#5

sslevel 3: monad
- Binding monad



让 Binding.scala 成为最流行的 Scala.js Web 框架

~~制作一个 Binding.scala 的 TodoMVC 例子~~

DEMO URL:
https://thoughtworksinc.github.io/todo

结论

	代表语言和框架	支持高并发	是否避免 callback hell?	是否易于理解?	支持其他DSL
同步编程	所有语言	否	是	是	否
level 0: callback	所有现代语言	是	否	否	否
level 1: generator	Python 2.2, C# 2.0, ES 6	是	是	否	否
level 2: async	Python 3.5, C# 5.0, ES 7	是	是	是	否
level3: monad	Scala/Each, F# 2.0	是	是	是	是

各种语言 async 实现的功能对比

	ECMAScript 7	Scala Each	F# 2.0	C# 5.0	Haxe-continuation	Python 3.5
Exception Support	yes	yes	yes	yes	no	yes
Runnable in Browser	yes	yes	no	no	yes	no
Implemented as a …	external complier	library	built-in feature	built-in feature	library	built-in feature
Monad Support	no	yes	yes	no	no	no

扩展阅读

- JavaScript / Babel
- Python / Generators
- Python / Tasks and coroutines
- C# / yield
- C# / async
- Haxe / Continuation
- Scala / Each
- Scala / Scalaz
- Scala / Binding.scala
- F# / Computation Expressions

THANKS!