.NET平台编程语言
异步特性演变

赵劼 - 2012.2

关于我

- 赵劼 / 老赵 / Jeffrey Zhao / 赵姐夫
- 日写代码三百行，不辞长作程序员
- 博客：http://blog.zhaojie.me/
- 微博：@老赵
- F#, JavaScript, Scala, C#, Python, .NET, Mono...
- 痛恨Java语言

内容

- C# 1.0
- C# 2.0
- F#
- C# 5

C# 1.0

两种简单模型

- Begin / End 模型
- 事件模型
- 两种模型都基于回调函数
  - “异步”所在

Begin / End 模型

```
delegate AsyncCallback(IAsyncResult);

interface IAsyncResult {
    object AsyncState { get; }
    ...
}

void BeginXyz(arg1, arg2, ..., AsyncCallback, state);

TResult EndXyz(IAsyncResult);
```

事件模型

```
class XyzCompletedEventArgs : EventArgs {
    Exception Error { get; }
    TResult Result { get; }
}

class Target {
    event EventHandler<XyzCompletedArgs> XyzCompleted;
    void XyzAsync(arg1, arg2, ...);
}
```

实现异步Transfer方法

示例一

C# 1.0原生异步编程

破坏了代码的局部性

- 线性代码表达更清晰，更符合习惯
- 异步会强迫拆分代码逻辑
  - 不能使用 if / using / while / for ...
- 难以
  - 组合异步操作
  - 异常处理
  - 取消操作

C# 2.0

“yield” for Iterators

```
IEnumerable<int> Numbers() {
    yield return 0;
    yield return 1;
    yield return 2;
}
```

“yield” for Iterators

```
IEnumerable<int> Numbers() {
```

MoveNext()

```
    yield return 0;
    yield return 1;
    yield return 2;
}
```

“yield” for Iterators

```
IEnumerable<int> Numbers() {
```

MoveNext() ⟵ `yield return 0;`

MoveNext() ⟵ `yield return 1;`

```
        yield return 2;

    }
```

“yield” for Iterators

```
IEnumerable<int> Numbers() {
```

MoveNext() ⟵ `yield return 0;`

MoveNext() ⟵ `yield return 1;`

MoveNext() ⟵ `yield return 2;`

```
}
```

“yield” for Iterators

```
IEnumerable<int> Numbers() {
```

MoveNext()

```
    yield return 0;
```

MoveNext()

```
    yield return 1;
```

MoveNext()

```
    yield return 2;
```

MoveNext()

```
}
```

示例二

C# 2.0的yield异步编程

“yield” 之与异步编程

- 带来新的异步编程模式
- 保持代码局部性
  - 优势：支持 if / using / while / for ...
  - 不完美：不支持 try...catch
- 可用来实现 Fibers：轻量的计算单元

F#

F# 编程语言

- 微软研究院Don Syme设计
- 强类型，静态类型
- 函数式编程语言，包含面向对象特性
- 面向业界及教育
  - 开源（Apache 2.0）
  - 微软提供跨平台支持

并发的挑战

- 状态共享 - 不可变性
- 代码局部性 - async { ... }
- I/O并行 - async { ... }
- 扩展至集群 - 使用 async { ... } 的代理

什么是 async { ... }

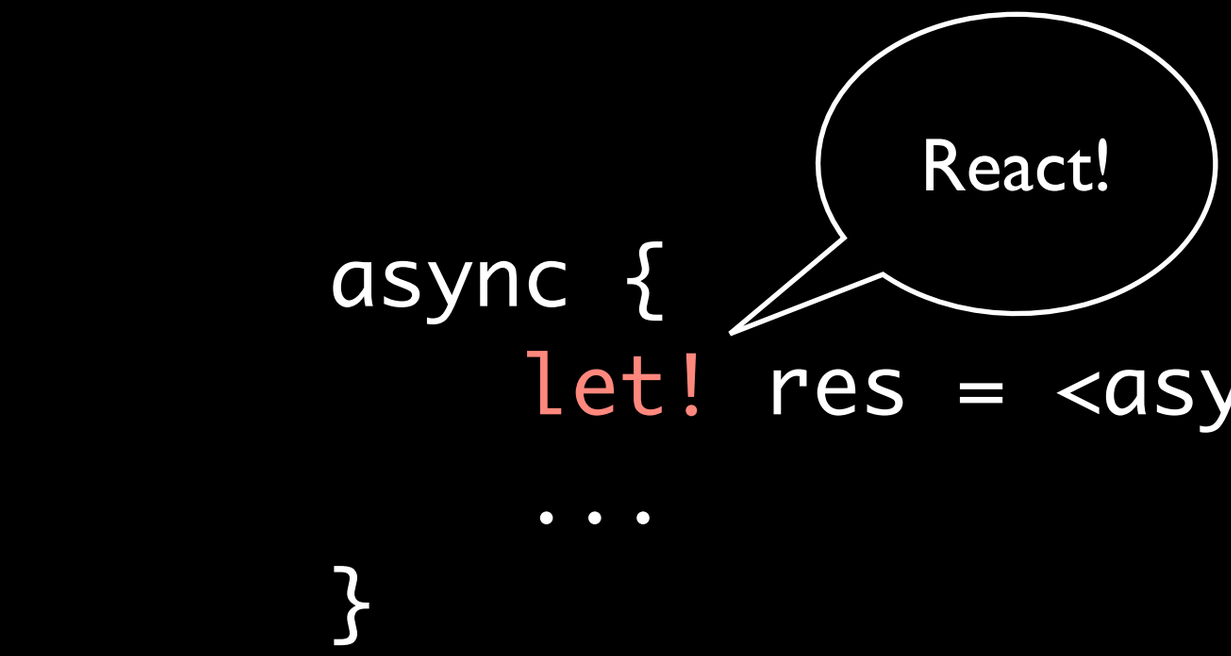
... the principle we go by is, don't expect to see a particular concurrency model put into C# because there're many different concurrency model ... it's more about finding things are common to to all kinds of concurrency ...

- Anders Hejlsberg

异步工作流

```
async {
    let! res = <async work>
    ...
}
```

异步工作流



异步工作流

React!

```
async {
    let! res = <async work>
    ...
    ...
}
```

HTTP 响应
UI 事件
Timer 回调
查询结果
Web Servcie 答复
I/O 完成
代理消息

async { ... } 工作方式

```
async {
    let! img = AsyncRead "http://..."
    printfn "loaded!"
    do! AsyncWrite img @"c:\..."
    printfn "saved!" }
```

async { ... } 工作方式

```
async {
    let! img = AsyncRead "http://..."
    printfn "loaded!"
    do! AsyncWrite img @"c:\..."
    printfn "saved!" }
```

=

```
async.Delay(fun ->
   async.Bind(AsyncRead "http://...", (fun img ->
      printfn "loaded!"
      async.Bind(AsyncWrite img @"c:\...", (fun () ->
         printfn "saved!"
         async.Return())))))
```

示例三

F# 异步工作流

C# 5

源代码

```
async Task<XElement> GetRssAsync(string url) {
    var client = new WebClient();
    var task = client.DownloadStringTaskAsync(url);
    var text = await task;
    var xml = XElement.Parse(text);
    return xml;
}
```

编译结果

```
Task<XElement> GetRssAsync(string url) {
    var $builder = AsyncMethodBuilder<XElement>.Create();
    var $state = 0;
    TaskAwaiter<string> $a1;
    Action $resume = delegate {
        try {
            if ($state == 1) goto L1;
            var client = new WebClient();
            var task = client.DownloadStringTaskAsync(url);
            $state = 1;
            $a1 = task.GetAwaiter();
            if ($a1.BeginAwait($resume)) return;
        L1: var text = $a1.EndAwait();
            var xml = XElement.Parse(text);
            $builder.SetResult(xml);
        }
        catch (Exception $ex) { $builder.SetException($ex); }
    };
    $resume();
    return $builder.Task;
}
```

示例四

C# 5 异步编程支持

示例五

Jscex：JS异步编程

Q & A

谢谢