C# 5.0

Взгляд в будущее

Специально для **TulaDev.NET**

Язык формирует наш способ мышления и
определяет то, о чем мы можем мыслить.

Б. Л. Ворф

rookee

О чем поговорим

- Я расскажу об одном интересном нововведении, которое планируется сделать в C# 5.0
- Объясню внутреннюю мотивацию разработчиков .NET
- Рассмотрим примеры ситуаций, в которых полезны новые возможности
- Разберем конкретный пример

История развития C#

- **C# 1.0 Managed code**

rookee

История развития C#

- C# 1.0 Managed code
- **C# 2.0 Iterators / Generics / Anonymous**

История развития C#

- C# 1.0 Managed code
- C# 2.0 Iterators / Generics / Anonymous
- **C# 3.0 LINQ / Lambda**

rookee

История развития C#

- C# 1.0 Managed code
- C# 2.0 Iterators / Generics / Anonymous
- C# 3.0 LINQ / Lambda
- **C# 4.0 Dynamic / PLINQ**

rookee

История развития C#

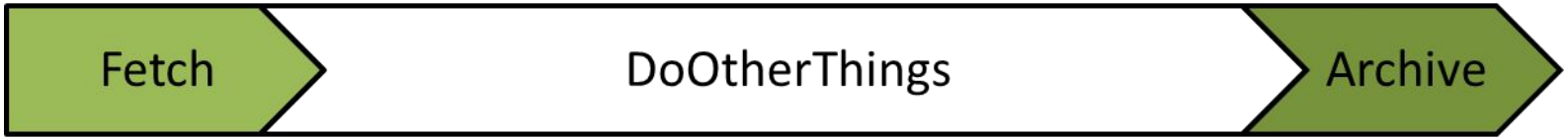
- C# 1.0 Managed code
- C# 2.0 Iterators / Generics / Anonymous
- C# 3.0 LINQ / Lambda
- C# 4.0 Dynamic / PLINQ
- **C# 5.0 TAP (Task-based asynchronous pattern)**

rookee

Зачем нужна асинхронность?

- ```
  var document = FetchDocument(url);
  ArchiveDocument(document);
  ```


- ```
  FetchDocumentAsync(url, document => {
   ArchiveDocument(document);
  });
  ```



rookee

Зачем нужна асинхронность?

```
FetchDocumentAsync(url1, document => {
 ArchiveDocument(document);
});
FetchDocumentAsync(url2, document => {
 ArchiveDocument(document);
});
FetchDocumentAsync(url3, document => {
 ArchiveDocument(document);
});
```

- Асинхронность позволяет рационально использовать ресурсы процессора в пределах одного потока

rookee

Асинхронность против многопоточности

- Асинхронность позволяет производить параллельную обработку в одном потоке
- Следовательно асинхронный код не будет использовать все ядра вашего процессора
- Но, так как поток один, то вы можете делать параллельную обработку без блокировок (lock)

rookee

Где нужна асинхронность?

- В первую очередь в обработке событий пользовательского интерфейса
- SilverLight (JavaScript, например, — язык только с поддержкой асинхронности)
- При программировании мультиплексоров и аналогичных им алгоритмов, использующих медленные внешние ресурсы: сеть, диски, другие накопители

rookee

Сопрограммы

- «Подпрограмма является частным случаем сопрограммы». Д. Кнут.
- Пример программы, использующей 2 сопрограммы:

```
var q := new queue
coroutine produce
  loop while q is not full
        create some new items
        add the items to q
  yield to consume
coroutine consume
  loop while q is not empty
        remove some items from q
        use the items
  yield to produce
```

При первом вызове сопрограммы, выполнение начинается со стартовой точки сопрограммы.

Однако при последующих вызовах выполнение продолжается с точки последнего возврата.

Как это выглядит в 5.0?

- **Возьмем код**

```
var document = FetchDocument(url);
ArchiveDocument(document);
```

- **Его можно преобразовать в асинхронный вариант**

```
Task<document> task = FetchDocumentAsync(url);
task.ContinueWith(document => {
 ArchiveDocument(document);
});
```

- **В C# 5.0 мы можем использовать другой синтаксис**

```
ArchiveDocument(await FetchDocumentAsync(url));
```

Как это выглядит в 5.0?

- Первая магия C# 5.0 - `await task`
  Означает конструкция буквально следующее:
  - Если таск, для которого выполняется `await` еще не завершен, то установить остаток текущего метода как продолжение (continuation) таска
  - Затем вернуть управление вызывающему методу
  - Продолжение метода будет вызвано таском, когда он будет завершен

Как это выглядит в 5.0?

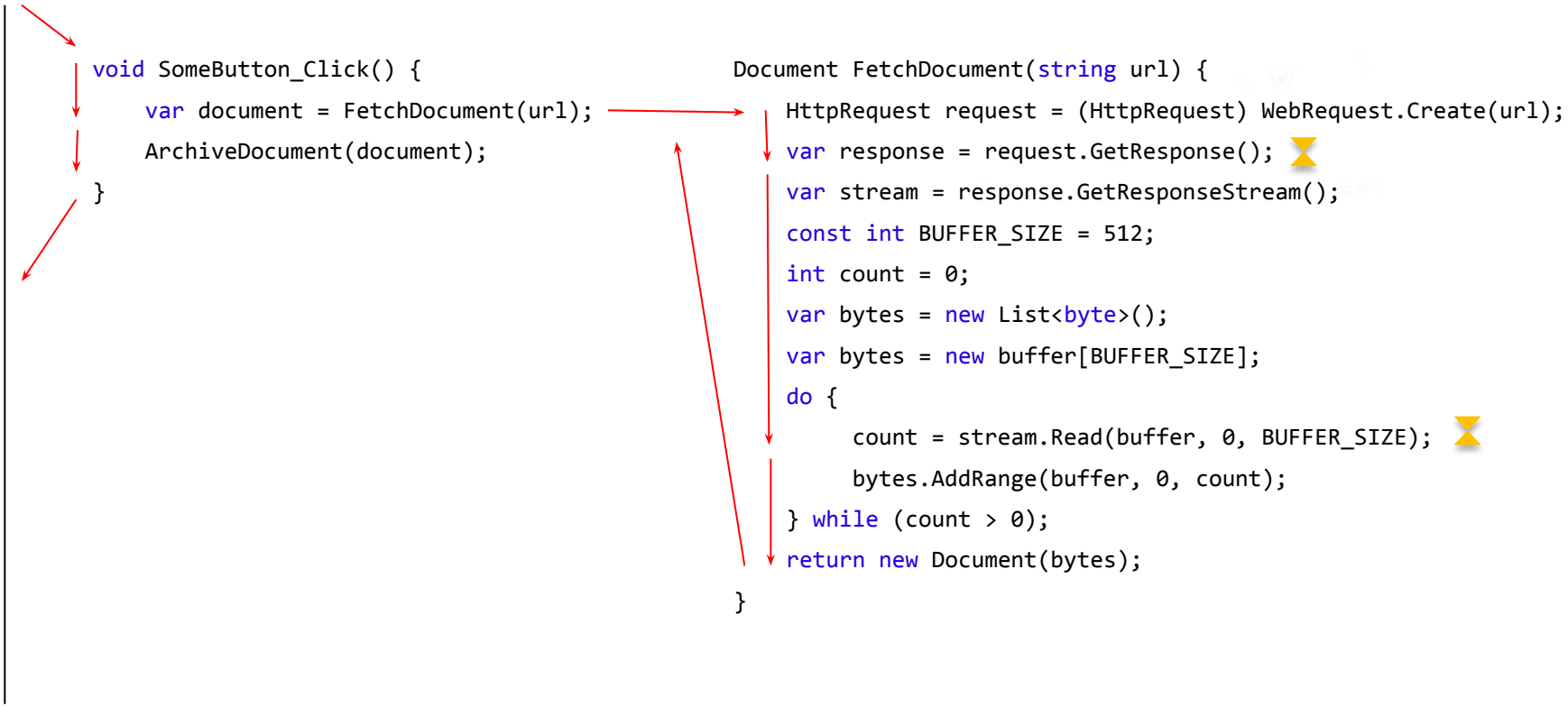
- Вторая магия C# 5.0 – модификатор метода `async`
  Означает буквально следующее:
  - Этот метод содержит конструкции `await`, так что компилятор должен преобразовать его таким образом, чтобы асинхронные операции могли продолжить выполнение этого метода
  - Метод может быть продолжен не с произвольной точки, а только с тех точек, которые запоминают конструкции `await`

Как это выглядит в 5.0?

- Конструкция `await` может быть указана только внутри `async` метода
- Методы `async` могут возвращать только `void`, `Task` или `Task<T>`
- При этом `void` и `Task` методы не могут возвращать ничего внутри своей реализации. То есть содержать только `return;`
- `Task<T>` методы могут внутри содержать только конструкцию `return <something of type T>;`

Синхронный пример

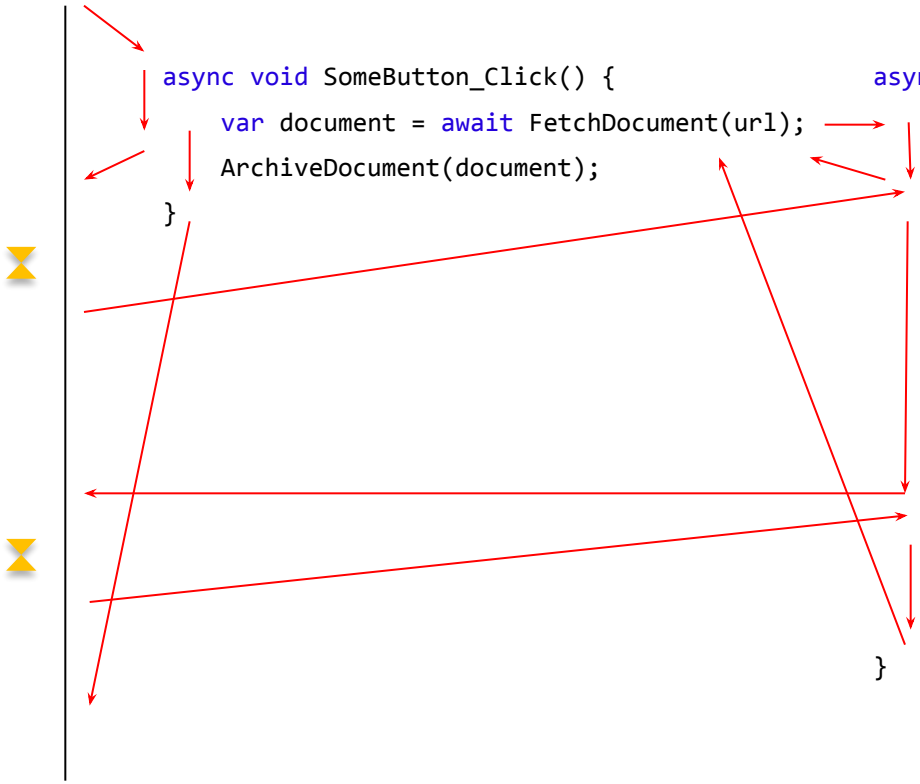
Thread



TulaDev.NET

Асинхронный пример

Thread



```
async void SomeButton_Click() {
    var document = await FetchDocument(url);
    ArchiveDocument(document);
}
```

```
async Task<Document> FetchDocument(string url) {
    HttpRequest request = (HttpRequest) WebRequest.Create(url);
    var response = await request.GetResponseAsync();
    var stream = response.GetResponseStream();
    const int BUFFER_SIZE = 512;
    int count = 0;
    var bytes = new List<byte>();
    var bytes = new buffer[BUFFER_SIZE];
    do {
        count = await stream.ReadAsync(buffer, 0, BUFFER_SIZE);
        bytes.AddRange(buffer, 0, count);
    } while (count > 0);
    return new Document(bytes);
}
```

TulaDev.NET

rookee

Лучше один раз увидеть...

Давайте рассмотрим
пример в VisualStudio

Что еще нужно знать?

- Конструкция `await` применима не только для `Task`. Но и для любого выражения, для которого определен метод `GetAwaiter()`, возвращающий тип, для которого можно вызвать методы `BeginAwait(Action)` и `EndAwait()`
- Асинхронный синтаксис сохраняет интуитивно понятный способ обработки исключений для вашего кода
- Предварительная версия компилятора C# 5.0 пока еще совсем предварительная. То есть в ней может поменяться все, вплоть до синтаксиса

Источник информации

http://msdn.com/vstudio/async

—Здесь можно скачать предварительную версию Async CTP (Community Technology Preview)

—Есть ссылки на документацию, примеры и обсуждения в блогах

—Много видео с выступлениями дизайнеров C# 5.0

http://nsentinel.blogspot.com/

—Можно подробно прочитать о TAP в C# 5.0 по-русски

Контактная информация

sergey.shebanin@re-actor.ru

ICQ: 198779172

http://www.tuladev.net/