Поддержка C# 8 Async Streams в ReSharper 2019.1

Андрей Карпов, JetBrains ReSharper Team
E-mail: andrew.karpov@jetbrains.com
Twitter: akarpov89

C# 5: async/await

```
public async Task<int> GetAnswerAsync()
{
  await Task.Delay(1000);
  return 42;
}
```

C# 6: await в catch/finally

```
var resource = await GetResourceAsync();
try
{
  await UseResourceAsync(resource);
}
catch (Exception e)
{
  await LogExceptionAsync(e);
}
finally
{
  await resource.ReleaseAsync();
}
```

C# 7: task-like типы

```
public async ValueTask<int> GetAnswerAsync()
{
  if (_isAlreadyCalculated)
    return _cachedResult;

  return await CalculateAnswer();
}
```

C# 7.1 Async Main

```
public static async Task Main()
{
  await PrintAnswerAsync();
}
```

Что нового в C# 8?

- Async Disposables
- Async Streams

Demo

```
public interface IDisposable
{
  void Dispose();
}
```

Оператор using

```
using (var resource = GetResource())
{
  // use resource
}
```

Оператор using

```
var resource = GetResource();
try
{
  // use resource
}
finally
{
  resource?.Dispose();
}
```

```
public interface IAsyncDisposable
{
  ValueTask DisposeAsync();
}
```

Оператор await using

```
await using (var resource = GetResource())
{
    // use resource
}
```

Оператор await using

```
var resource = GetResource();
try
{
  // use resource
}
finally
{
  await resource?.DisposeAsync();
}
```

Demo

Оператор foreach

```
foreach (var x in xs)
{
  // use item
}
```

Оператор foreach

```
var enumerator = xs.GetEnumerator();
try
{
  while (enumerator.MoveNext())
  {
    var x = enumerator.Current;
  }
}
finally
{
  enumerator?.Dispose();
}
```

```
public interface IEnumerable<out T> : IEnumerable
{
  IEnumerator<T> GetEnumerator();
}

public interface IEnumerable
{
  IEnumerator GetEnumerator();
}
```

```
public interface IEnumerator<out T>
  : IDisposable, IEnumerator
{
  T Current { get; }
}

public interface IEnumerator
{
  bool MoveNext();
  object Current { get; }
  void Reset();
}
```

Оператор await foreach

```
await foreach (var x in xs)
{
  // use item
}
```

Оператор await foreach

```
var enumerator = xs.GetAsyncEnumerator();
try
{
  while (await enumerator.MoveNextAsync())
  {
    var x = enumerator.Current;
  }
}
finally
{
  await enumerator.DisposeAsync();
}
```

```
public interface IAsyncEnumerable<out T>
{
  IAsyncEnumerator<T> GetAsyncEnumerator(
    CancellationToken cancellationToken = default
  );
}
```

```
public interface IAsyncEnumerator<out T>
  : IAsyncDisposable
{
    ValueTask<bool> MoveNextAsync();

    T Current { get; }
}
```

Demo

Сгенерированный CancellationToken

```
async IAsyncEnumerable<int> GetAsyncEnumerable()
{
  await Task.Delay(1000, cancellationToken);
  yield return 42;
}
```

CancellationToken по атрибуту

```
async IAsyncEnumerable<int> GetAsyncEnumerable(
  [Cancellation] CancellationToken
                 cancellationToken = default)
{
  await Task.Delay(1000, cancellationToken);
  yield return 42;
}
```

DotNext

Yield at me, 'cause I'm awaiting: асинхронные итераторы в C# 8

RU / День 2 / 14:00 / Зал 2 В избранное

> Комментарий Программного комитета:
>
> *Новая фича C# — как упростить себе жизнь в асинхронном мире.*

Язык C# продолжает интенсивно развиваться. Готовящаяся к выходу новая версия добавляет поддержку асинхронных итераторов. Что это такое? Для чего это нужно? Как это работает? В докладе мы ответим на все эти вопросы, разберём нововведения в BCL, сравним новые возможности с уже существующими средствами и, конечно же, заглянем под капот компилятора.

Q&A