



SergeyT 2 апреля 2012 в 21:48

Замыкания на переменных цикла в C# 5

.NET

Многие разработчики языков программирования, библиотек, да и классов простых приложений стремятся к интуитивно понятному интерфейсу создаваемых классов. Скотт Мейерс еще полтора десятка лет назад сказал о том, чтобы мы стремились разрабатывать классы (библиотеки, языки), которые легко использовать правильно, и сложно использовать неправильно.

Если говорить о языке C#, то его разработчики подходят к вопросам «юзабилити» весьма основательно; они спокойно могут пожертвовать «объектной чистотой» в угоду здравому смыслу и удобству использования. Одним из немногих исключений из этого правила является замыкание на переменной цикла, той самой фици, которая ведет себя не так, как считают многие разработчики. При этом количество недовольства и недопонимания настолько много, что в 5-й версии языка C# это поведение решили изменить.

Итак, давайте рассмотрим пример кода, который показывает проблему замыкания на переменной цикла:

```
var actions = new List<Action>();
foreach(var i in Enumerable.Range(1, 3))
{
    actions.Add(() => Console.WriteLine(i));
}

foreach(var action in actions)
{
    action();
}
```

Большинство разработчиков разумно предполагают, что результатом выполнения этого кода будет “1 2 3”, поскольку на каждой итерации цикла мы добавляем в список анонимный метод, который выводит на экран новое значение **i**. Однако если запустить этот фрагмент кода в VS2008 или VS2010, то мы получим “3 3 3”. Эта проблема настолько типична, что некоторые тулы, например, ReSharper, выдает предупреждение в строке **actions.Add()** о том, что мы захватываем изменяемую переменную, а Эрик Липперт настолько задолбался отвечать всем, что это фица, а не баг, что решил изменить существующее поведение в C# 5.0.

Чтобы понять, почему данный фрагмент кода ведет себя именно так, а не иначе, давайте рассмотрим, во что компилятор разворачивает этот код (я не буду слишком сильно углубляться в детали работы замыканий в языке C#, за подробностями обращайтесь к заметке “Замыкания в языке C#”).

В языке C# захват внешних переменных осуществляется «по ссылке», и в нашем случае это означает, что переменная **i** исчезает из стека и становится полем специально сгенерированного класса, в который затем помещается и тело анонимного метода:

```
// Упрощенная реализация объекта-замыкания
class Closure
```

Реклама

ЧИТАЮТ СЕЙЧАС

Эпл испортила мне ноутбук почти сразу после покупки и до сих пор не собирается чинить

👁 45,1k 💬 273

Картинка, которая одновременно является кодом на Javascript

👁 5,9k 💬 8

Играем в DOOM на тесте на беременность. Что? Да

👁 19,3k 💬 34

Германия, или Туда и Обратно — 2

👁 5,8k 💬 79

Поисковик «Спутник» прекратил работу

👁 9,5k 💬 34

“Чистый код”: пять ключевых моментов из обязательной к прочтению книги для программистов

👁 3,2k 💬 22

От ASCII к Unicode: как появились кириллические домены и адреса электронной почты

[Мераност](#)

Редакторский дайджест

Присылаем лучшие статьи раз в месяц

Электрпочта

```

{
    public int i;
    public void Action()
    {
        Console.WriteLine(i);
    }
}
var actions = new List<Action>();

using (var enumerator = Enumerable.Range(1, 3).GetEnumerator())
{
    // int current;
    // создается один объект замыкания
    var closure = new Closure();
    while(enumerator.MoveNext())
    {
        // current = enumerator.Current;
        // и он используется во всех итерациях цикла foreach
        closure.i = enumerator.Current;
        var action = new Action(closure.Action);
        actions.Add(action);
    }
}

foreach (var action in actions)
{
    action();
}

```

Поскольку внутри цикла используется один объект **Closure**, то после завершения первого цикла, **closure.i** будет равно **3**, а поскольку переменная **actions** содержит три ссылки на один и тот же объект **Closure**, то не удивительно, что при последующем вызове методов **closure.Action()** мы получим на экране "3 3 3".

Изменения в C# 5.0

Изменения в языке C# 5.0 не касаются замыканий как таковых и мы, как замыкались на переменные (и не делаем копии значений), так и замыкаемся. На самом деле, изменения касаются того, во что разворачивается цикл **foreach**. Замыкания в языке C# реализованы таким образом, что для каждой области видимости (scope), в которой содержится захватываемая переменная, создается собственный экземпляр класса замыкания. Именно поэтому, для того, чтобы получить желаемое поведение в предыдущих версиях языка C#, достаточно было написать следующее:

```

var actions = new List<Action>();
foreach(var i in Enumerable.Range(1, 3))
{
    var tmp = i;
    actions.Add(() => Console.WriteLine(tmp));
}

```

Если вернуться к нашему упрощенному примеру с классом **Closure**, то данное изменение приводит к тому, что создание нового экземпляра **Closure** происходит внутри цикла **while**, что приводит к сохранению нужного значения переменной **i**:

```

using (var enumerator = Enumerable.Range(1, 3).GetEnumerator())
{
    int current;
    while(enumerator.MoveNext())
    {

```

```

current = enumerator.Current;
// Теперь для каждой итерации цикла мы создаем
// новый объект Closure с новым значением i
var closure = new Closure {i = current};
var action = new Action(closure.Action);
actions.Add(action);
}
}

```

В C# 5.0 решили изменить цикл **foreach** таким образом, чтобы на каждой итерации цикла переменная **i** создавалась вновь. По сути, в предыдущих версиях языка C# в цикле **foreach** была лишь одна переменная цикла, а начиная с C# 5.0, используется новая переменная для каждой итерации.

Теперь исходный цикл **foreach** разворачивается по-другому:

```

using (var enumerator = Enumerable.Range(1, 3).GetEnumerator())
{
    // В C# 3.0 и 4.0 current объявлялась здесь
    //int current;
    while (enumerator.MoveNext())
    {
        // В C# 5.0 current объявляется заново для каждой итерации
        var current = enumerator.Current;
        actions.Add(() => Console.WriteLine(current));
    }
}

```

Это делает временную переменную внутри цикла **foreach** излишней (поскольку ее добавил для нас компилятор), и при запуске этого кода мы получим ожидаемые "1 2 3".

Кстати, обратите внимание, что это изменение касается только цикла **foreach**, поведение же цикла **for** никак не изменилась и при захвате переменной цикла, вам все еще нужно самим создавать временную переменную внутри каждой итерации.

Дополнительные ссылки

1. Eric Lippert [Closing over loop variable considered harmful](#)
2. Eric Lippert [Closing over loop variable, part two](#)
3. [Замыкания в языке C#](#)
4. [Visual C# Breaking Changes in Visual Studio 11 Beta](#)

Теги: [.net](#), [CSharp 5](#), [closures](#)

Хабы: [.NET](#)

↑ +38 ↓ 133 👁 31,2k 💬 46 ➦ Поделиться



225,5

Карма

0,0

Рейтинг

Сергей Тепляков @SergeyT

Пользователь

ПОХОЖИЕ ПУБЛИКАЦИИ

10 июля 2015 в 16:16

Генерация LINQ to SQL кода для SQLite в .NET (C#)

↑ +2 👁 13,7k 📖 46 💬 8

21 мая 2010 в 15:12

Как вы видите .NET / C# 5.0?

↑ +5 👁 5k 📖 6 💬 76

27 апреля 2009 в 12:59

О разворачивании строк в .Net/C# и не только

↑ +73 👁 23,7k 📖 86 💬 57

СРЕДНЯЯ ЗАРПЛАТА В IT

113 000 ₺/мес.

Средняя зарплата по всем IT-специализациям на основании 4 324 анкет, за 2-ое пол. 2020 года

[Узнать свою зарплату](#)

Реклама

Комментарии 46

 **kekekeks** 2 апреля 2012 в 21:52 🏷 📖 ↑ +2 ↓

Большинство разработчиков разумно предполагают, что результатом выполнения этого кода будет "1 2 3"

Я бы не сказал, что разумно. Скорее неразумно, ибо такое предположение можно сделать лишь при полном непонимании того, как работают замыкания.

 **Nagg** 2 апреля 2012 в 21:57 🏷 📖 🔄 ⬆ ↑ +1 ↓

Очень странное нововведение, а вдруг в существующем коде где-то использовалось замыкание на current в благих целях?

 **Nagg** 2 апреля 2012 в 21:58 🏷 📖 🔄 ⬆ ↑ 0 ↓

промахнулся :-).

 **kekekeks** 2 апреля 2012 в 21:59 🏷 📖 🔄 ⬆ ↑ 0 ↓

В текущем виде оно приводит к неопределённому поведению (точнее, поведение зависит от времени, когда будет выполнен код замыкания), так что не думаю, что его кто-то в здравом уме мог использовать рассчитывая на текущее поведение.

ЧТО ОБСУЖДАЮТ

Сейчас

Вчера

Неделя

Варианты использования конфигурации в ASP.NET Core

👁 1,4k 💬 1

Германия, или Туда и Обратно — 2

👁 5,8k 💬 79

США может ввести санкции в отношении SMIC, крупнейшего производителя микросхем Китая

👁 4,6k 💬 28

Анонс Nvidia Ampere – Как Хуанг всех приятно удивил

👁 21,6k 💬 73

С ножовкой — на лазер, с голыми руками — на чип. Как DIY-авторы создают свои шедевры на Хэбре

[Мераност](#)

 **semmaxim** 2 апреля 2012 в 22:06 # 0 1 0

Нет никакого неопределённого поведения. Оно не зависит от времени. В любой момент у нас чёткая зависимость от значения переменной-итератора цикла.

 **kekekeks** 2 апреля 2012 в 22:10 # 0 0 0

В общем, я так и не увидел живого примера того, зачем надо замыкаться на переменную `foreach`-цикла в расчёте на то, что значение изменится. Пока что только придирки к формулировкам.

 **SergeyT** 2 апреля 2012 в 22:17 # 0 4 0

Несколько сотен постов на `stackoverflow` говорят о том, что очень-но многие разработчики считают именно «1 2 3» естественным результатом.

Естественно, после того, как вы узнали как это дело *устроено*, то все вопросы отпадают. Но ведь помимо нас с вами, есть еще пара миллионов индусов (это я про мировоззрение, а не рассовую принадлежность), которые с нами не согласятся, поскольку они не знают да и не хотят знать, как это дело внутри устроено.

 **kekekeks** 2 апреля 2012 в 22:18 # 0 0 0

Ну я не спорю, что всё правильно сделали, просто такое поведение вытекает из общей логики работы циклов и замыканий.

 **SergeyT** 2 апреля 2012 в 22:20 # 0 0 0

Это вытекает из того, какое количество переменных существует в цикле `foreach`: одна на весь `foreach` или на каждую итерацию создается новая переменная.

 **kekekeks** 2 апреля 2012 в 22:25 # 0 0 0

Какому циклу первому учат в школе? `for`. В `for` переменная одна на весь цикл, есть явные конструкции управления её изменением и выходом из цикла. Следовательно, логичным было бы предположить, что в `foreach` так же, а конструкции скрыты от глаз (это если не читать стандарт ЕСМА, на одних предположениях из «логики вещей»). Логика тех, кто предполагает, что переменная каждый раз новая, мне не ясна.

 **SergeyT** 2 апреля 2012 в 22:30 # 0 0 0

А цикл `for` никак не изменился. В нем как была одна переменная на весь цикл, так и осталась.

Здесь был вопрос компромисса: что лучше, пожертвовать согласованностью с циклом `for` или устранить одно из самых назойливых непониманий языка С#. При создании С# 2.0 решили отдать предпочтение согласованности с циклом `for`, но со временем решили, что этой согласованностью стоит пожертвовать.

 **kekekeks** 2 апреля 2012 в 22:31 # 0 0 0

Эм. Где я сказал, что цикл `for` изменился?

 **SergeyT** 2 апреля 2012 в 22:37 # 0 0 0

Нет, нигде. Так, на всяк случай напомнил.



justserega 3 апреля 2012 в 07:24



↑ +5 ↓

Несколько сотен постов на stackoverflow говорят о том, что очень-но многие разработчики считают именно «1 2 3» естественным результатом.

Теперь будет полторы тысячи вопросов от тех же индусов — почему в foreach все работает «правильно», а в цикле for «мистика».



Shedal 3 апреля 2012 в 12:07



↑ +1 ↓

Zanuda mode=on

Индус — не расовая принадлежность, а вероисповедание. Индус — это приверженец индуизма. А расовая принадлежность это индиец.

Извините :)



mlurker 4 апреля 2012 в 23:13



↑ 0 ↓

Поскольку большинство разработчиков не представляют себе как работает .net внутри, то утверждение про разумность имеет смысл.



kekekeks 4 апреля 2012 в 23:25



↑ 0 ↓

Эм. А ничего что переменная объявлена вне скобочной конструкции? Т. е. должна быть одна на все итерации того, что внутри фигурных скобок?



Deranged 9 апреля 2012 в 12:42



↑ 0 ↓

На мой взгляд «неправильный» код выглядит куда более естественно, чем код со временной переменной. Можно понимать что он неправильный, и чувствовать, что всё равно хотелось бы писать так. Тут Липперт молодец.

Вообще в C# еще несколько таких мест.



semmaxim 2 апреля 2012 в 21:54



↑ 0 ↓

Мда... Мне как то изначальный вариант казался естественным... Это ж ломает обратную совместимость.



kekekeks 2 апреля 2012 в 21:57



↑ +1 ↓

На самом деле не ломает. В цикле foreach всегда объявляется новая переменная. А поскольку раньше на неё никто не делал замыканий, то новая реализация ничего не ломает.



semmaxim 2 апреля 2012 в 22:05



↑ 0 ↓

Ну Вы так ловко ответили прям за всех...



kekekeks 2 апреля 2012 в 22:07



↑ +3 ↓

Ну приведите пример того, зачем нужно замыкаться на постоянно изменяющуюся переменную перечислителя цикла, рассчитывая при этом, что она будет изменена? Мне сложно представить ситуацию, когда такое может вообще понадобиться.

Впервые эту проблему поднял Эрик в конце 2009-го года и уже тогда обсуждались решения, как это можно пофиксить. Сейчас было принято решение, что нет никакого разумно корректного кода, который бы использовал эту фичу именно таким образом (т.е. чтобы намеренно замкнулся на переменную цикла, в надежде получить сотню одинаковых результатов).

Таким образом, в новой версии будет двойственное поведение переменных в такой конструкции:

```
var actions = new List<Action>();
var jj = 0;
foreach(var i in Enumerable.Range(1, 3))
{
    actions.Add(() => Console.WriteLine(i, jj));
    jj = jj + 1;
}

foreach(var action in actions)
{
    action(); //даст для новой версии(5): 1 3    2 3    3 3
}
```

Скорее всего массовое непонимание усугубляет тот факт, что в цикле в этом языке создаётся окружение, а не в функции, как, например, в javascript и perl.

Но, впрочем, и замыкания в функциях в javascript по умолчанию никто не понимает, начинают понимать после того как напишут нечаянно замыкание, не зная об этом.

Поведение будет таким же «согласованным» как и ранее, но для этого нужно понимать две вещи: (1) экземпляр класса замыкания для каждой области видимости и (2) каждая итерация цикла foreach содержит новую переменную i.

В данном случае это означает, что для переменной jj будет создан один объект замыкания, который будет шариться всеми остальными объектами замыкания для каждой итерации.

Т.е. этот код можно рассматривать таким образом:

```
var actions = new List<Action>();
var jjClosure = new jjClosure();
jjClosure.jj = 0;

foreach(var i in Enumerable.Range(1, 3))
{
    // внутренний объект замыкания содержит ссылку на внешний объект замыкания
    var iClosure = new iClosure() {i = i, jjClosure = jjClosure};

    // () => Console.WriteLine(i, jj) теперь перешло в метод Action замыкания iClosure.
    // Тело iClosure.Action выглядит так:
    // Console.WriteLine(this.i, this.jjClosure.jj);
    actions.Add(iClosure.Action);
}
```

В результате чего мы и получим 1 3, 2 3, 3 3, поскольку всеми iClosure будет использоваться один и тот же объект jjClosure.

 **VasilioRuzanni** 3 апреля 2012 в 01:38    +1 

Эх, а ранее Липперт приводил вполне обоснованные аргументы, почему они не собираются делать это. Я лично и сам не знаю, как именно лучше — «чистота конструкции» или «интуитивная юзабельность в общем случае» — меня не напрягает то, как это есть сейчас. Да, про «Access to Modified Closure» знали не все. Видимо, Эрика всерьез достали с этим :)

По идее это не должно сломать существующий код, так что ужасного в этом решении ничего нет. Хотя личное ощущение, что это в чистом виде хак (да, не «синтэкс шуга», а именно хак).

 **rroyter** 7 апреля 2012 в 08:35      0 

Раньше аргумент у него был один: не ломать старый код. Но он признавался, что они хотели это починить ещё в C# 3.0.







 **zvulon** 3 апреля 2012 в 04:24    0 

Сегодня написал подобный код, но мне правда решарпер подсказал, что не все в порядке. обратил мое внимание на то, что я переменную цикла в замыкание засунул. Я заменил тело цикла на вызов функции (ну там передача по значению). Вобщем будет удобнее в C#5.

 **Shaddix** 3 апреля 2012 в 09:02    0 

Эта фича будет «компилироваться» только в .net 4.5?

А то если будет в .net 4, то возможна ситуация, что один и тот же код скомпиленный в VS2010 (со старым поведением) не работает, а в VS11 — работает. Совместное использование студий будет тогда проблематично.

 **SergeyT** 3 апреля 2012 в 09:35      +2 

Это фича именно компилятора языка C# 5.0, причем не важно, на какую платформу мы таргетимся, т.е. при запуске этого кода из VS11 будет результат «1 2 3», даже если мы таргетимся на .NET Framework 3.5.

Поэтому ситуация, когда из VS10 будет один результат, а из VS11 — другой, вполне возможна. В данном случае решением является использования «общего» подхода, которое подойдет для всех версий — т.е. явное использование темповой переменной внутри цикла.

 **IlVin** 3 апреля 2012 в 12:06    0 

В Mono 2.10.5 результат — 3, 3, 3

Чую — путаница будет с этими замыканиями...

Но идея мне кажется здоровой — всегда подсознательно не понимал почему в циклах «int» пишут в скобках, а не где-нибудь еще...

Если введут эту фичу и для for, то в моей голове «все сложится»

Старое поведение:

```
int i;
for( i=0; i<10; i++ )
{
    /* Здесь работаем с одной и той же переменной */
}
```


Новое поведение:

```
for( int i=0; i<10; i++ )
{
  /* А здесь для каждой итерации генерируется новая переменная */
}
```

Осталось только дождаться когда Липперт соберется с духом и возьмет на себя ответственность за обратную совместимость... :)



deilux 3 апреля 2012 в 13:39



↑ +1 ↓

Переменную цикла пишут в скобках, потому что она локальная для этого цикла: дело не в том, одна она или на каждый чих новая, а в области видимости. Вне цикла её не должно быть



IlVin 3 апреля 2012 в 13:47



↑ 0 ↓

Спасибо за напоминание — это я усвоил, когда изучал циклы.
И даже могу сказать почему этой фишки никогда не будет в for — все дело в обратной совместимости с такими конструкциями:

```
for( int i=0; i<10; i++ )
{
  if( true ) i+=3;
}
```



mayorovp 3 апреля 2012 в 16:17



↑ 0 ↓

Вот только внутри if должно стоять не true, а более сложное условие.



mayorovp 3 апреля 2012 в 13:46



↑ 0 ↓

Ответьте мне, пожалуйста, каким образом конструкция i++ может быть применима к переменной, генерируемой для каждой итерации заново?



IlVin 3 апреля 2012 в 14:04



↑ 0 ↓

Нда — неправильно в комментариях мысль написал. Надо было писать:

Новое поведение:

```
for( int i=0; i<10; i++ )
{
  /* А здесь для каждой итерации создается новый экземпляр Closure,
  что приводит к сохранению нужного значения переменной i */
}
</sorce>
```



mayorovp 3 апреля 2012 в 14:21



↑ 0 ↓

```
for (int i=0; i<10; i++)
{
  Action act = () => i--; // Так будет делать нельзя - переменная
  i стала вдруг привязанной к итерации
  i--; // а так по-прежнему можно?
}
```

Вам это странным не кажется?



IIVin 3 апреля 2012 в 14:25 # 📖 🔍 ↻

↑ 0 ↓

Хм...

А как Ваш пример работает в цикле `foreach`?

Неужели мы получили еще одни грабли?



mayorovp 3 апреля 2012 в 16:16 # 📖 🔍 ↻

↑ 0 ↓

А цикле `foreach` мой пример не работает.

Я не могу придумать ни одной ситуации, в которой понадобилось бы изменять переменную цикла.

Потому в новой версии языка `foreach` и переделали, а `for` оставили.



yorick_kiev_ua 3 апреля 2012 в 13:50 # 📖

↑ 0 ↓

А для чего вообще сделали такое поведение, как в 4.0? Это концепция или побочный эффект?



mayorovp 3 апреля 2012 в 14:22 # 📖 🔍 ↻

↑ 0 ↓

Побочный эффект на пересечении двух концепций.

Читайте статью внимательнее.



Enrey 4 апреля 2012 в 11:46 # 📖

↑ 0 ↓

Мне жутко не нравится то, что если код, написанный на C# 5 (не нужны временные переменные) откомпилировать на C# 4 (нужны временные переменные) то будет непредсказуемое поведение :(



zvulon 4 апреля 2012 в 23:11 # 📖 🔍 ↻

↑ 0 ↓

а зачем это делать?

в общем случае код просто не скомпилируется.



Seeker 4 апреля 2012 в 12:10 # 📖

↑ -1 ↓

Спасибо за статью. Вообще в первый раз услышал про замыкания.

Странно, что об этом не написал Рихтер)



SergeyT 5 апреля 2012 в 09:42 # 📖 🔍 ↻

↑ 0 ↓

На самом деле, у Рихтера об этом все написано, включая магию генерации классов замыканий при захвате внешних переменных из анонимного метода. Просто страшных терминов, типа `closure` у него нет.

См. главу 17. Delegates раздел Syntactical Shortcut #3: No Need to Wrap Local Variables in a Class Manually to Pass Them to a Callback Method



lthilgwau 12 июня 2013 в 21:51 # 📖

↑ 0 ↓

Отличная статья, спасибо! Жаль, что разработчики C# идут на поводу горе-программистов... Для меня лично раньше всё было очевидно, а сейчас уже нет. Тем более, что `for` остался прежним. Такой подход идёт вразрез с логикой...

Только [полноправные пользователи](#) могут оставлять комментарии. [Войдите](#), пожалуйста.

САМОЕ ЧИТАЕМОЕ

Неделя

Месяц

Эпл испортила мне ноутбук почти сразу после покупки и до сих пор не собирается чинить

 +37
  45,1k
  17
  273

Google продвигает новый стандарт WebBundles — потенциально опасную для веба технологию «упаковки» веб-сайтов

 +63
 31,8k
 52
 74

Играем в DOOM на тесте на беременность. Что? Да

 +34
 19,3k
 34
 34

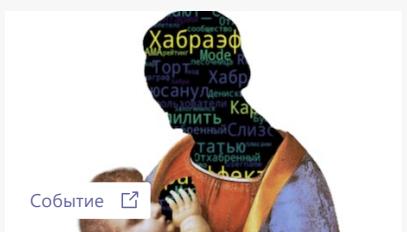
USB4: все тот же USB?

 +33
 19,1k
 60
 57

Как проверить нейросетью 20 000 домашнихек

Мегапост

МИНУТОЧКУ ВНИМАНИЯ



Прокачка техноавторов от Хабра: конкурс и мастер-классы



Успеть за 2 месяца. Как прокачаться с ментором до QA Jun+

Разместить

Ваш аккаунт	Разделы	Информация	Услуги
Войти	Публикации	Устройство сайта	Реклама
Регистрация	Новости	Для авторов	Тарифы
	Хабы	Для компаний	Контент
	Компании	Документы	Семинары
	Пользователи	Соглашение	Мегапроекты
	Песочница	Конфиденциальность	Мерч