Массивы и строки

# Статичные массивы.



Статичные массивы, как и все остальное в .NET – объекты, это не набор элементов, лежащих друг за другом, как в С. Причем это Reference Type. Этот объект содержит следующую информацию о массиве:

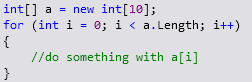
* Количество измерений массива
* Нижняя граница всех измерений
* Длина всех измерений
* Тип элементов массива
* Прочее, в том числе индекс синхронизации и ссылку на объект типа.



При таком объявлении переменной массива, кроме самой переменной также создается новый тип, который так и называется «Int32[]». В случае, если создается массив с ненулевым стартовым индексом, то название у типа будет другим – «Int32[\*]». Каждый такой новый тип наследуется от *System.Array*, который в свою очередь наследуется от *System.Object*.

Внутренне CLR различает 2 типа массивов – одномерный начинающиеся с 0 и все остальные. Такое разделение сделано, поскольку именно такой тип массивов самый используемый, и потому его значительно оптимизировали, добавив специально для него несколько инструкций в IL-язык. Поэтому следует использовать именно такие массивы, если только не требуется в конкретной ситуации.

При обращении к элементу массива происходит проверка на то, чтобы индекс не вышел за пределы массива. Впрочем, не стоит считать, что это сильно влияет на работу программы. В действительности, JIT-компилятор часто оптимизирует эту работу, прежде всего в циклах. Если же время работы критично, можно добавить оператор unsafe, как checked/unchecked для арифметических операций, и в этом случае никаких проверок не будет. Однако скомпилированные таким образом программы не будут проходить тест на безопасность.



Рассмотрим пример. Обратите внимание, что здесь для проверки условий выхода из цикла используется свойство массива, то есть фактически вызов метода. Однако вызов этого метода будет происходить не каждый раз при каждой итерации. JIT-компилятор оптимизирует эту часть кода. Он сделает вызов метода только один раз. После чего сохранит результат в локальной переменной. Во всех последующих вызовах будет происходить проверка именно с этой переменной. Так что нет нужды оптимизировать эту часть кода своей собственной переменной. JIT-компилятор сделает это сам. Более того, своей оптимизацией вы нарушите другой метод оптимизировать этот код.

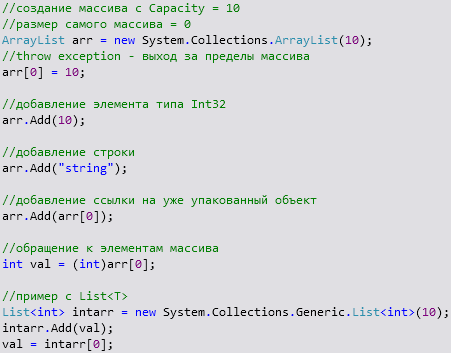
Проверку того, что индекс не выходит за рамки массива, JIT-компилятор тоже оптимизирует. В действительности, вместо проверки на каждой итерации, компилятор сделает следующее: он проверит, что нижняя граница массива (вызовет метод *a.GetLowerBound(0)*) меньше или равна нулю и что верхняя граница (*a.GetUpperBound(0)*) больше или равна значению (*a.Length -1*). Если обе проверки пройдут, то компилятор сгенерирует код без проверок на выход за пределы массива. То есть эта проверка произойдет только один раз во время компиляции.

Как видите, для массивов с несколькими измерениями или имеющих ненулевой стартовый индекс такая проверка не пройдет, поэтому каждое обращение к элементам будет проверяться на выход за пределы массива.

# Динамические массивы.

Для динамических массивов используются класс *System.Collections.ArrayList* или набор классов-дженериков *System.Collections.Generic.List<T>*. Их отличие в том, что *ArrayList* – это массив элементов типа *Object*, а для *List* можно задавать тип элементов. Это отличие хорошо в первую очередь, если тип элементов – Value Type, в этом случае не элементы не будут находиться в массиве в упакованном виде. Второе преимущество *List* в том, что такой вариант более надежный для проверки типов.

Внутри этих классов используется статичный массив, в котором и находятся элементы. Размер этого статичного массива может не совпадать с размером динамического. За его размер отвечает свойство *Capacity*. Если при добавлении нового элемента в динамический массив размера статичного массива достаточно для нового элемента, то просто количество используемых элементов увеличивается. Если же размера недостаточно, то создается новый массив с большим размером. Такое частое выделение памяти сильно снижает производительность. Поэтому размер статичного массива увеличивается с резервом на будущие увеличения размера массива. Также очень полезно самому управлять этим размером, используя свойство *Capacity*. Если заранее известно примерное количество элементов (причем, лучше задавать с небольшим резервом), то лучше его задать сразу в конструкторе массива. Можно использовать и ведение статистики, сохраняя размеры динамических массивов куда-нибудь, чтобы потом использовать эти значения при новом создании массивов.



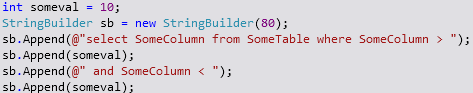
Рассмотрим пример. Здесь используются оба типов массивов. В *arr* находятся ссылки на объекты, причем могут ссылаться на объекты различных типов. Если эти объекты Value Type, то они упаковываются. Несколько элементов могут указывать на один и тот же объект. При обращении к элементу массива необходимо явно приводить его к нужному типу. Если тип Value Type, то при этом еще происходит распаковка.

Массив *intarr* действует аналогично, но объекты Value Type не упаковываются (хотя и оказываются в куче, как часть объекта массива), не требуется явное преобразование типа при обращении к элементам массива.

# Строки.

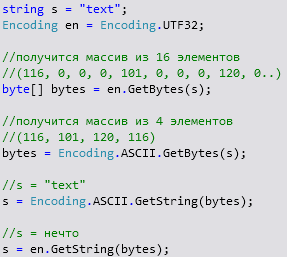
В .NET всего 2 типа строка – собственно *System.String* и *System.Text.StringBuilder*. Причем, как видно из названия, S*tringBuilder* используется только для построения строки.

*System.String* – Reference Type, поэтому он позволяет эффективно управлять строками через ссылки. При присвоении одной переменной типа *String* другой будет скопирована только ссылка, таким образом, в памяти будет только один объект самой строки. Однако в связи с тем, что несколько переменных могут ссылаться на один и тот же объект строки, появилась необходимость сделать строки неизменяемыми. *String* является неизменяемым объектом. Сам класс содержит большое количество методов, меняющих строки, но каждый из этих методов сначала создает копию строки и потом работает с этой копией. Первоначальная строка не изменяется. В результате каждое изменение в строке приводит к созданию нового объекта строки (и соответственно, выделению памяти), а старый (если он не продолжает использоваться кем-то еще) становится мусором.



Однако задача преобразования строки в несколько шагов – довольно распространенная, поэтому сделан специальный класс только для редактирования строки – S*tringBuilder*. Он содержит практически тот же набор средств редактирования строки, что и *String*, плюс специфические, например, *String* не может редактировать отдельный символ (*a[i] = ‘1’*). Но в отличие от *String* все изменения происходят на самом объекте без дополнительного выделения памяти, если только после изменения будет хватать количества элементов внутреннего массива. В отношении внутреннего массива *Char* будут действовать те же рекомендации, что и для динамического массива. Там так же необходимо регулировать размер массива с помощью свойства *Capacity*. Также следует иметь в виду, что после вызова функции *ToString*, любые изменения в объекте *StringBuilder* приведут к созданию нового объекта внутреннего массива. Поэтому следует избегать вызова этой функции до окончания сборки строки.

# Кодировка.

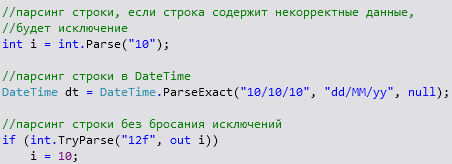


Все строки, а также отдельные символы (*System.Char*) в .NET являются 16-битными Unicode. Однако другие кодировки существуют, например в файлах или при передачи данных по сети или как-то ещё. Для взаимодействия с другими кодировками нужно переводить строку в массив байт (и обратно) с использованием класса *System.Text.Encoding*.



Кроме определенных в константах типов кодировок, можно создать объект и других кодировок.

# Парсинг из строки.



В .NET есть набор методов, которые парсят объект из строки. Для таких сложных объектов, как дата/время, можно задавать формат, по которому парсится строка, а также параметры культуры и языка (Например, выставлять «г.» в дате). Простой вызов функции *Parse* бросает исключение, если формат строки оказался неверным. Если парсинг вызывается часто и при этом может быть большое количество строк с ошибками, то лучше использовать функции *TryParse*, которая возвращает в булевой переменной результат – удался парсинг или нет.