Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Кафедра программного обеспечения информационных технологий

СЕТИ ЭВМ

Лабораторный практикум
для студентов специальности 1-40 01 01
«Программное обеспечение информационных технологий»
дневной формы обучения

В 2-х частях
Часть 2

Е. В. Мельникова, И. М. Марина

Минск БГУИР 2011

УДК 004.7(076.5)
ББК 32.973.202я73
С33

Р е ц е н з е н т:
заведующий кафедрой вычислительных методов и программирования учреждения образования «Белорусский государственный университет информатики и радиоэлектроники»,
кандидат технических наук А. А. Иванюк

Сети ЭВМ : лаб. практикум для студ. спец. 1-40 01 01 «Программное
С33 обеспечение информационных технологий» днев. формы обуч. В 2 ч. Ч. 2. / Е. В. Мельникова, И. М. Марина. – Минск : БГУИР, 2011. – 29 с.
ISBN 978-985-488-659-6 (ч. 2).

Дано общее представление о межсетевом взаимодействии, рассмотрены методы и алгоритмы программирования сокетов.

Представлены четыре лабораторные работы.

УДК 004.7(076.5)
ББК 32.973.202я73

Часть 1-я. Сети ЭВМ : лаб. практикум для студ. спец. 1-40 01 01 «Программное обеспечение информационных технологий» днев. формы обуч. В 2 ч. Ч. 1. / Д. А. Сурков [и др.]. – Минск : БГУИР, 2006. – 36 с.

ISBN 978-985-488-659-6 (ч. 2)
ISBN 978-985-444-963-0
ISBN 985-444-963-7

Содержание

Краткие теоретические сведения

Архитектура .NET

Компилятор C# специально предназначен для .NET, т. е. код, написанный на C#, будет всегда исполняться на платформе .NET. Этим определяются два важных свойства языка C#:

– архитектура и методология C# отражают архитектуру и методологию .NET. Например, C# основывается на одиночном наследовании классов и имеет систему типов, которая базируется на различии между типами по ссылке и по значению, так же как и в случае .NET;

– во многих случаях специфические особенности языка C# зависят от особенностей .NET или ее базовых классов. Так, все основные типы данных в C# (int, float, string и т. д.) на самом деле являются основными типами классов .NET, которые C# лишь представляют с использованием более удобного синтаксиса.

Компиляция

Перед запуском программа должна быть откомпилирована. Однако в отличие от прежних исполняемых файлов и DLL теперь компилированный код программы не содержит инструкций ассемблера. Вместо этого он содержит инструкции на промежуточном языке Microsoft (MSIL или IL). Промежуточный язык в чем-то похож на байт-код Java. Это низкоуровневый код, который может быть быстро преобразован (откомпилирован JIT) в родной машинный код.

Пакет, внутри которого будет содержаться откомпилированная программа, состоит из некоторого числа сборок. Каждая сборка содержит код на промежуточном языке и метаданные, описывающие типы данных и методы внутри сборки. Метаданные содержат также простой хэш, который строится на основе содержимого сборки и может быть использован для проверки ее целостности; информацию о версиях; сведения о том, какие сборки будут вызываться данной сборкой; и, возможно, информацию о том, какие привилегии потребуются для выполнения кода сборки.

Исполнение

Во время выполнения программы среда исполнения .NET загружает первую сборку, ту, что содержит точку входа основной программы. Среда использует хэш для проверки целостности сборки и метаданные для того, чтобы просмотреть определенные типы и убедиться, что среда сможет выполнить сборку. Правильно разработанные коммерческие приложения должны явно указывать, какие привилегии .NET им могут потребоваться (например, понадобится ли приложению доступ к файловой системе, реестру и т. д.). В этом случае CLR (общая среда исполнения) обратится к политике безопасности системы и учетной записи, под которой выполняется программа, и проверит, может ли она предоставить необходимые привилегии. Если код не запрашивает права явно, они будут предоставлены ему по первому требованию, если, конечно, это возможно.

На этом этапе CLR выполнит еще одно действие для проверки так называемой безопасности кода по типу памяти (memory type safety). Код считается безопасным по типу памяти только в том случае, если он обращается к памяти способами, которые может контролировать CLR. Безопасный по типу памяти код гарантированно не будет пытаться прочесть или записать в память, не принадлежащую ему. Это важно, так как .NET имеет механизм (так называемые области приложений), позволяющий нескольким приложениям выполняться в одном процессе. При этом необходимо гарантировать, что никакое приложение не будет пытаться обратиться к памяти другого приложения. Если CLR не будет уверена в том, что код безопасен по типу памяти, то в зависимости от местной политики безопасности она может даже отказать в исполнении кода.

Затем CLR выполняет код. Она создает процесс, в котором будет исполняться код, и отмечает область приложения, в которой размещается главный поток приложения. В некоторых случаях программа может потребовать поместить ее в уже имеющийся процесс запущенного ранее кода, тогда CLR создаст для нее только новую область приложения.

CLR берет первую часть кода, которая требуется для исполнения, и компилирует ее с промежуточного языка на язык ассемблера, после чего выполняет ее из соответствующего потока программы. Каждый раз, когда в процессе исполнения встречается новый метод, не исполнявшийся ранее, он компилируется в исполняемый код. Процесс компиляции происходит только один раз. Как только метод откомпилирован, его адрес заменяется адресом компилированного кода. Таким образом, производительность не ухудшается, так как компилируются только те участки кода, которые действительно используются. Этот процесс носит название компиляции just-in-time. Отметим, что JIT-компилятор может в зависимости от параметров компиляции, указанных в сборке, оптимизировать код в процессе компиляции, например, путем подстановки некоторых методов (inline) вместо их вызовов.

Во время выполнения кода CLR следит за использованием памяти. На основе этих наблюдений она в определенные моменты будет останавливать программу на короткий промежуток времени (обычно миллисекунды) и запускать сборщика мусора, который проверит переменные программы и выяснит, какие из областей памяти активно используются программой, для того чтобы освободить неиспользуемые участки.

CLR также производит загрузку сборок по мере необходимости, в том числе COM-компонентов, используя службы .NET COM interop.

Преимущества исполнения программы как управляемого кода

Из приведенного описания уже очевидны некоторые преимущества исполнения управляемого кода:

– прежде всего улучшается безопасность. Благодаря тому что сборка написана на промежуточном языке, среда исполнения .NET может проверить, что собирается делать код. Реальное преимущество здесь в том, что становится более безопасно выполнять код, полученный, например, по Интернету. Политика безопасности .NET для этого кода устанавливается таким образом, чтобы он получал права на выполнение лишь тех задач, которые он предположительно должен выполнять. .NET поддерживает как функциональную безопасность (базирующуюся на сущности процесса, исполняющего код), так и безопасность на основе кода (базирующуюся на степени доверия к самому коду);

– присутствие сборщика мусора освобождает программиста от необходимости написания кода для освобождения используемой памяти. Это также означает, что отсутствует риск утечки памяти для всех переменных, обслуживаемых сборщиком мусора;

– новая концепция областей приложений означает, что различные приложения, которые необходимо изолировать друг от друга, но которые в то же время должны общаться друг с другом, могут быть размещены в одном процессе. Это дает огромный выигрыш в производительности.

Необходимо отметить также следующие преимущества:

– имеет место совместимость языков, которая позволяет легко использовать код, написанный на разных языках;

– самоописываемая структура сборок практически исключает ошибки, которые возникают из-за проблем версий или проблем, связанных с перезаписыванием другими приложениями совместно используемых сборок. Тем самым значительно экономится время и стоимость разработки;

– базовые классы .NET предлагают интуитивную, простую объектную модель, которая заметно упрощает вызовы функциональных возможностей Windows (в смысле упрощения исходного кода) по сравнению с тем, что было в прошлом.

На первый взгляд кажется, что будет иметь место потеря производительности, так как часть процесса компиляции осуществляется во время исполнения программы. Microsoft свои возражения обуславливает тем, что в долгосрочном периоде это не будет являться проблемой, а JIT-компиляторы не ухудшают, а даже улучшают производительность. Улучшение происходит потому, что JIT-компилятор «знает», на каком процессоре будет исполняться код. Эта информация недоступна для традиционных компиляторов.

Промежуточный язык

Промежуточный язык и Java в своей основе имеют одну и ту же идею – это языки низкого уровня с простым синтаксисом (основанным на числовых кодах, а не на тексте), который может быть быстро оттранслирован в родной

машинный код. Целью байт-кода Java является обеспечение платформенной независимости. Обладание четко определенным, универсальным синтаксисом байт-кода означало бы, что файл, содержащий инструкции байт-кода, мог бы быть размещен на любой платформе, например, UNIX, Linux или Windows, а во время выполнения могла бы быть легко выполнена заключительная стадия компиляции, и код мог бы быть запущен на этой платформе.

Следовательно, при написании исходного кода его можно было бы компилировать в байт-код Java, который может быть выполнен, где угодно.

Промежуточный язык использует ту же концепцию, но более амбициозно. Важно то, что промежуточный язык компилируется в процессе выполнения программы, в то время как байт-код Java интерпретируется. Это означает, что большая часть потерь производительности, связанных с интерпретацией байт-кода Java, не затрагивает IL.

Код на промежуточном языке компилируется в родной машинный код, а не интерпретируется.

Целью промежуточного языка является не просто платформенная независимость, а языковая независимость в объектно-ориентированной среде. Идея заключается в том, что должна существовать возможность компиляции кода с любого языка, и компилированный код должен быть совместим с кодом, откомпилированным с других языков.

Эта совместимость достигается в .NET, так как с ее помощью можно писать код, который компилируется в промежуточный язык, на C++, VB.NET или C#. Компиляторы для других языков, включая COBOL, Eiffel и Perl, должны быть вскоре выпущены сторонними производителями. В настоящее время платформенная независимость является только возможной, так как во время создания среда .NET была доступна лишь для Windows, хотя сейчас ведутся активные разговоры о том, чтобы импортировать .NET на другие платформы. Из-за требований языковой независимости и совместимости промежуточный язык гораздо сложнее байт-кода Java.

Конечно, языковая независимость имеет некоторые практические ограничения. В частности, промежуточный язык неизбежно реализует некоторую определенную методологию программирования, а это означает, что языки, целью которых является IL, должны быть совместимы с этой методологией. Конкретный путь, который был выбран Microsoft, заключается в следовании классическому объектно-ориентированному программированию с классами и наследованием. Поэтому классы и наследование определены внутри промежуточного языка.

Традиционное объектно-ориентированное программирование

Для того чтобы понять принципы промежуточного языка, необходимо знать принципы классического объектно-ориентированного программирования (ООП). Эти принципы известны программистам на Visual C++ и Java/J++ (так как оба языка основаны па концепциях ООП).

Идея классического объектно-ориентированного программирования состоит в том, что код организуется в виде классов. Каждый класс определяет некоторый отдельный модуль, с которым можно выполнять какие-то действия. Например, в GUI Windows классы могут включать в себя ListBox, TextBox или Form.

Классы в некотором смысле могут рассматриваться как типы данных, например, целое или число с плавающей точкой. Различие в том, что класс является более сложным, он состоит из нескольких простых типов данных, хранящихся вместе, а также функций (или методов), которые могут быть вызваны для этого класса. Класс является «типом данных пользователя» еще и потому, что его можно определить в исходном коде. Объектно-ориентированное программирование основано на идее, что приложение будет создавать экземпляры объектов, а затем вызывать некоторые из их методов, так что в конце концов объекты начнут взаимодействовать друг с другом.

Промежуточный язык поддерживает так называемое одиночное наследование, т. е. класс может быть напрямую унаследован только от одного класса, но число производных классов от данного не ограничено (базовый класс, от которого производится наследование, в свою очередь может быть производным от другого класса). В конечном итоге образуется древовидная структура классов. В случае промежуточного языка все классы являются производными от класса, известного как Object. Некоторые из базовых классов .NET показаны на рисунке. Заметим, что чем ниже мы спускаемся по иерархии классов, тем более узко определены конкретные классы.

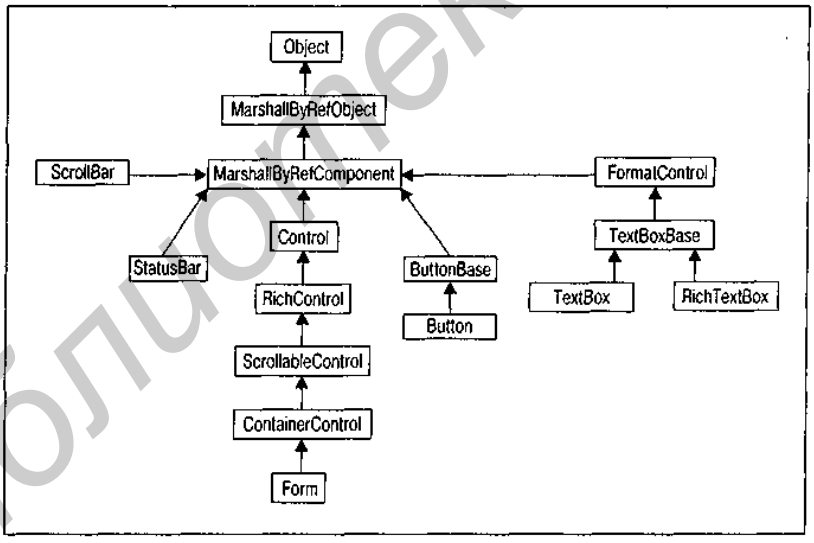


Рис. Базовые классы .NET

Помимо классического объектно-ориентированного программирования, промежуточный язык использует идею интерфейсов, которые были впервые реализованы в Windows с применением COM. Интерфейсы .NET – это не то же самое, что интерфейсы COM: им не требуется поддерживать инфраструктуру COM, в частности, они не наследуются от Iunknown и не имеют связанных GUID. Однако в их основе лежит общая с COM идея, заключающаяся в том, что классы, реализующие данный интерфейс, обязаны содержать реализации методов и свойств, которые определены этим интерфейсом.

Общая спецификация языка (CLS)

Общая спецификация языка работает с общей системой типов для достижения совместимости языков. CLS является минимальным набором стандартов, которые должны поддерживаться всеми компиляторами, предназначенными для .NET. CLS необходима по той причине, что IL очень богатый язык, и поэтому возможен случай, когда производители некоторых компиляторов предпочтут ограничить возможности конкретного компилятора поддержкой лишь некоторого подмножества функций, предлагаемых IL и CTS. Никаких проблем не будет возникать, пока компилятор поддерживает то, что определено в CLS.

Например, CTS определяет два типа данных для 32-битных целых чисел: Int32 (целое со знаком) и UInt32 (целое без знака). C# определяет эти типы как int и uint соответственно. С другой стороны, VB.NET определяет только тип Int32, для которого используется ключевое слово Integer.

Другой пример – чувствительность к регистру символов. IL чувствителен к регистру символов, что позволяет чувствительным к регистру языкам C++ и C# определять переменные, отличающиеся только регистром символов: EmployeeName и employeeName. Разработчики постоянно пользуются этой возможностью при выборе имен переменных. VB.NET, однако, не чувствителен к регистру символов. CLS решает эту проблему, указывая, что совместимый с CLS код не должен содержать двух имен, отличающихся только регистром символов. Таким образом, код VB.NET может работать с совместимым с CLS кодом.

Этот пример показывает два важных аспекта CLS. Во-первых, конкретные компиляторы не обязаны быть настолько мощными, чтобы поддерживать все особенности .NET, – это должно стимулировать разработку компиляторов с других языков программирования для .NET. Во-вторых, предоставляется гарантия того, что если класс ограничен только CLS-совместимыми функциями, он сможет использовать код, написанный на любом языке. Например, если требуется, чтобы код был CLS-совместимым, нельзя возвращать значения Ulnt32, так как этот тип не является частью CLS. Разумеется, можно возвращать и значения типа UInt32, однако в этом случае не гарантируется, что код будет работать со всеми языками.

Не совместимый с CLS код допустим. Однако в этом случае не гарантируется, что откомпилированный код будет полностью независим от языка.

Привлекательность этой идеи заключается в том, что ограничение на использование CLS-совместимых особенностей действительно только для тех элементов, которые могут быть видны извне данной сборки: для открытых и защищенных членов классов и открытых классов. Внутри закрытых определений классов можно писать, какой угодно не совместимый с CLS код – это не является проблемой, так как код других сборок в любом случае не сможет получить доступ к этой части кода.

CLS практически не затрагивает код на C#, поскольку C# имеет лишь несколько не совместимых с CLS особенностей.

Библиотека базовых классов .NET

Вероятно, одним из самых главных достоинств управляемого кода, помимо упрощения процесса написания кода, является возможность использования библиотеки базовых классов .NET.

Базовые классы .NET представляют собой большую коллекцию классов управляемого кода. Они были созданы Microsoft и позволяют выполнять практически любые задачи, которые ранее были доступны благодаря Windows API. Эти классы следуют той же объектной модели, основанной на одиночном наследовании, что используется промежуточным языком. Это означает, что можно как создать экземпляр класса, определенного в библиотеке базовых классов .NET, так и определить класс, производный от данного.

Замечательной особенностью базовых классов .NET является то, что они просты в использовании и самодокументированы. Например, для запуска потока необходимо вызвать метод Start () класса Thread. Для открытия файла нужно вызвать метод Open () класса File. Для того чтобы сделать неактивным TextBox, необходимо присвоить значение false свойству Enabled объекта TextBox. Идея самодокументированных классов знакома разработчикам Visual Basic и Java, чьи библиотеки также просты в применении.

Возможно, это будет большим облегчением для программистов на C++, которые вынуждены иметь дело с такими функциями API, как GetDIBits (), RegisterWndClassEx () и isEqual IID (), а также с целой плеядой функций для обработки дескрипторов Windows. С другой стороны, разработчики на C++ всегда могут получить доступ к Windows API, в то время как разработчики на Visual Basic и Java ограничены в применении функциональности Windows на низком уровне. Новым в базовых классах .NET является то, что они сочетают в себе легкость использования библиотек Visual Basic и Java с полным описанием функций API.

Согласно источникам в Microsoft большая часть базовых классов .NET была написана на C#!

Типы по ссылке и по значению

Как и любой другой язык программирования, промежуточный язык имеет некоторое число заранее определенных простых типов данных. Одним из свойств промежуточного языка является то, что он различает типы по ссылке и по значению. Типы по значению – это типы, для которых переменная хранит свои данные непосредственно в памяти, в то время как переменная ссылочного типа содержит адрес участка памяти, в котором хранятся данные. Эта модель в значительной степени повторяет модель типов по ссылке и значению, используемую в Java.

В терминологии C++ ссылочные типы могут рассматриваться как способ доступа к переменной посредством указателя, а наилучшей аналогией для ссылочных типов Visual Basic будут Objects (объекты), доступ к которым в VB всегда осуществляется по ссылке. Промежуточный язык содержит также спецификации, касающиеся хранения данных: экземпляры типов по ссылке

всегда хранятся в области памяти, известной как куча, в то время как типы по значению обычно хранятся в стеке, хотя в случае, если типы по значению объявлены как поля внутри типов по ссылке, они будут храниться в куче.

Строгий контроль типов

Одним из аспектов промежуточного языка является то, что он основан на строгом контроле типов. Это означает, что каждая переменная имеет один определенный тип данных (в промежуточном языке нет, например, типа данных Variant, как в Visual Basic и языках сценариев). В частности, промежуточный язык обычно не допускает выполнения операций, которые могут привести к различной трактовке того, на какой тип данных ссылается та или иная ссылка.

Разработчикам на C++ приходится постоянно преобразовывать тип указателя. Это имеет большое значение для производительности, но нарушает безопасность типов. Поэтому такая операция допускается только при определенных обстоятельствах лишь в некоторых языках, предназначенных для создания управляемого кода. В частности, указатели (в противоположность ссылкам) разрешены только в некоторых фрагментах кода в C#, но никак не в VB. Использование указателей в коде немедленно нарушит проверку безопасности типов памяти, выполняемую CLR. И хотя требование безопасности типов ухудшает производительность, все же в большинстве случаев выгоды, получаемые от использования служб, предоставляемых .NET (например область приложений), и основывающиеся как раз на безопасности типов, будут перекрывать это ухудшение.

Точно так же разработчикам на VB не придется думать о типах переменных, поскольку VB автоматически выполнит все необходимые преобразования. Если в каком-то фрагменте кода вместо типа String будет передан тип Integer, VB преобразует Integer в String. Философия IL и .NET состоит в том, что удобство неявных преобразований перевешивается связанными с этим проблемами безопасности типов и, в частности, потенциальными трудноуловимыми ошибками времени выполнения, возникающими из-за неверных типов данных. Поэтому при написании кода, предназначенного для .NET, эти преобразования придется целенаправленно выполнять вручную.

Существует несколько серьезных причин, по которым необходим строгий контроль типов, потому как без него некоторые части .NET не будут работать:

– в первую очередь среда исполнения общего языка полагается на способность проверки кода для определения того, какие операции необходимо выполнить перед непосредственным запуском кода. Это важно как с точки зрения предоставления привилегий доступа, так и с точки зрения проверки того, что код не может повредить другой код, который выполняется в другой области приложений, но в том же адресном пространстве. Промежуточный язык был разработан для облегчения проверок, и совершенно очевидно, что

если бы присутствовали какие-то сомнения насчет типов данных в коде, такие проверки не были бы возможны;

– для того чтобы выяснить, какую память надо освободить, сборщик мусора должен уметь определять тип данных, содержащийся в каждой ячейке памяти (иначе он не сможет узнать, какой объем памяти занимает переменная). Опять же, любые неоднозначности в типах данных вызовут проблемы, и среда исполнения .NET начнет работать неправильно;

– совместимость языков (одна из основ платформы .NET) базируется на хорошо определенном и цельном наборе типов данных.

Система типов, используемая промежуточным языком, известна как общая система типов (CTS).

Лабораторная работа №1

Цель работы: обзор пространств имен для работы с сетью, **System.Net**, пространства имен System.Net.Sockets.

Пространство имен System.Net

Пространство имен System.Net содержит сетевые классы для поиска IP-адресов, сетевой аутентификации, разрешений, отправки и получения данных.

Чтобы получить IP-адрес из DNS-имени хоста или получить имя хоста из IP-адреса, можно использовать класс Dns. Класс DnsPermission представляет разрешение, необходимое для поиска имени. DnsPermissionAttribute – это класс атрибута, позволяющий отмечать сборки, классы и методы, нуждающиеся в этих полномочиях.

IP-адреса обрабатываются в классе IPAddress. У одного хоста может быть несколько IP-адресов и алиасов. Вся эта информация содержится в классе IPHostEntry. Когда необходимо найти имя, класс Dns возвращает объект типа IPHostEntry.

Абстрактные базовые классы, предназначенные для отправки запросов на сервер и получения ответов, называются WebRequest и WebResponse. В пространстве имен System. Net имеется несколько специальных реализаций этих классов для HTTP и доступа к файлам: HttpWebRequest, HttpWebResponse, FileWebRequest и FileWebResponse.

Делегат Пространство имен System.Net предоставляет простой программный интерфейс для многих протоколов, используемых в современных сетях. Классы WebRequest и WebResponse образуют основу так называемых сменных протоколов – реализацию сетевых служб, которая позволяет разрабатывать приложения, использующие ресурсы Интернета, не заботясь о конкретных деталях отдельных протоколов.

Иерархия пространства имен

Некоторые классы

Authorization содержит аутентификационное сообщение для веб-сервера.

Dns предоставляет функции по простому разрешению вопросов, связанных с именами доменов.

DnsPermission управляет правами доступа к серверам службы имен доменов (DNS) в сети.

DnsPermissionAttribute устанавливает разрешение на запрос сведений с серверов доменных имен (DNS).

FileWebRequest предоставляет реализацию файловой системы класса WebRequest.

FileWebResponse предоставляет реализацию файловой системы класса WebResponse.

HttpVersion определяет номера версий протокола HTTP, поддерживаемых классами HttpWebRequest и HttpWebResponse.

HttpWebRequest предоставляет ориентированную на HTTP-протокол реализацию класса WebRequest.

HttpWebResponse предоставляет ориентированную на HTTP-протокол реализацию класса WebResponse.

IPAddress предоставляет IP-адрес (IP – Internet Protocol).

WebClient предоставляет общие методы обмена данными с ресурсом, заданным URI (класс не наследуется).

WebPermission управляет правами доступа к интернет-ресурсам HTTP.

WebProxy содержит параметры HTTP-прокси для класса WebRequest.

WebRequest предоставляет отклик в URI (абстрактный класс).

WebResponse предоставляет отклик от URI (абстрактный класс).

Пространство имен System.Net.Sockets

Классы

NetworkStream обеспечивает основной поток данных для доступа к сети.

Socket реализует интерфейс сокетов Berkeley.

SocketException – исключение, которое создается при возникновении ошибки на сокете.

TcpClient обеспечивает клиентские подключения для сетевых служб протокола TCP.

TcpListener ожидает подключения от TCP-клиентов сети.

UdpClient обеспечивает сетевые службы протокола UDP.

Пространство имен System.Net.Sockets предоставляет управляемую реализацию интерфейса Windows Sockets (Winsock) для тех разработчиков, которые нуждаются в надежном управлении доступом к сети.

Класс Socket играет важную роль в сетевом программировании, обеспечивая функционирование как клиента, так и сервера. Главным образом вызовы методов этого класса выполняют необходимые проверки, связанные с безопасностью, в том числе проверяют разрешения системы безопасности.

Некоторые важные свойства класса System. Net.Sockets.Socket:

AddressFamily дает семейство адресов сокета – значение из перечисления Socket AddressFamily.

Available возвращает объем доступных для чтения данных.

Blocking дает или устанавливает значение, показывающее, находится ли сокет в блокирующем режиме.

Connected возвращает значение, информирующее, соединен ли сокет с удаленным хостом.

ProtocolType – тип протокола сокета.

SocketType – тип сокета.

Пример программы, которая осуществляет пинг сети.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.NetworkInformation;
class Program
   {
      static void Main(string[] args)
      {
         Ping pingSender = new Ping();
         PingOptions options = new PingOptions();
         options.DontFragment = true;
         string data = "11111111111111111";
         byte[] buf = Encoding.ASCII.GetBytes(data);
         int timeout = 120;
         string adr = "192.168.33.";
         for (int i = 0; i < 255; i++)
         {
            PingReply reply = pingSender.Send(adr + Convert.ToString(i), timeout, buf,
options);
            if (reply.Status == IPStatus.Success)
            {
               Console.WriteLine("IP адрес: {0} - Активен", reply.Address.ToString());
               Console.WriteLine("Время жизни: {0}", reply.Options.Ttl);
               Console.WriteLine("Не фрагментировано: {0}",
reply.Options.DontFragment);
               Console.WriteLine("Размер буфера: {0}", reply.Buffer.Length);
               Console.WriteLine("************************************");
            }
            else
            {
               Console.WriteLine("IP адрес: {0}  - Не активен", adr + Convert.ToString(i));
               Console.WriteLine("************************************");
            }
         }
         Console.ReadLine();
      }
   }
```

Лабораторная работа №2

Цель работы: ознакомиться с классами HttpWebRequest, HttpWebResponse, WebProxy.

Краткие теоретические сведения

HttpWebRequest

Класс HttpWebRequest предоставляет поддержку свойств и методов, определенных в классе WebRequest, а также дополнительных свойств и методов, позволяющих пользователю взаимодействовать с серверами непосредственно, используя протокол HTTP.

Класс HttpWebRequest предоставляет общие значения HTTP-заголовка, посылаемые в Интернет-ресурс в виде свойств, устанавливаемых методами или системой; ниже приводится полный список заголовков. Другие заголовки устанавливаются в свойстве Headers в виде пар имя–значение.

Здесь перечисляются HTTP-заголовки, устанавливаемые свойствами, методами или системой.

Accept (принять) устанавливается свойством Accept.

Connection (подключение) устанавливается свойством Connection, свойством KeepAlive.

Content-Length (длина содержимого) устанавливается свойством ContentLength.

Content-Type (тип содержимого) устанавливается свойством ContentType.

Expect (ожидать) устанавливается свойством Expect.

Date (дата) устанавливается равным текущей дате.

Host (хост) устанавливается равным текущим данным о хосте.

If-Modified-Since (если изменен с момента) устанавливается свойством IfModifiedSince.

Range (диапазон) устанавливается методом AddRange.

Referer (отправитель) устанавливается свойством Referer.

Transfer-Encoding (передача-кодировка) устанавливается свойством TransferEncoding (свойство SendChunked должно иметь значение «true»).

User-Agent (пользователь-агент) устанавливается свойством UserAgent.

Класс HttpWebRequest регистрируется автоматически. Перед использованием URIs, начинающихся с http:// или https://, не требуется вызывать RegisterPrefix для регистрации System.Net.HttpWebRequest.

HttpWebResponse

Данный класс предоставляет поддержку применений свойств и методов класса WebResponse, характерных для HTTP-протокола. Класс HttpWebResponse используется для построения автономных клиентских HTTP-приложений, которые посылают HTTP-запросы и получают HTTP-ответы.

Никогда не следует создавать экземпляр класса HttpWebResponse напрямую. Вместо этого используйте экземпляр, возвращаемый в результате вызова метода HttpWebRequest.GetResponse.

Сведения обычных заголовков, возвращаемых из интернет-ресурса, предоставляются как свойства класса. Полный перечень заголовков представлен ниже. Другие заголовки можно прочитать в виде пар имя–значение из свойства Headers.

Здесь приводятся обычные HTTP-заголовки, доступ к которым можно получить через свойства класса HttpWebResponse:

Content-Encoding (содержимое-кодировка) ContentEncoding;

Content-Length (длина содержимого) ContentLength;

Content-Type (тип содержимого) ContentType;

Last-Modified (дата последнего изменения) LastModified;

Server (сервер) Server.

Содержимое ответа от интернет-ресурса возвращается в виде объекта Stream посредством вызова метода GetResponseStream.

WebProxy

Содержит параметры HTTP-прокси для класса WebRequest.

Любые члены этого типа с модификаторами public static (Shared в Visual Basic) могут безопасно работать в многопоточных операциях. Потокобезопасность членов экземпляров не гарантируется.

Класс WebProxy содержит параметры прокси, используемые экземпляром WebRequest для переопределения настроек прокси в GlobalProxySelection.

Класс WebProxy является базовой реализацией интерфейса IWebProxy.

WebProxy.Credentials-свойство

Возвращает или задает учетные данные, предоставляемые прокси-серверу для проверки подлинности.

Значение свойства

ICredentials, содержащий учетные данные, предоставляемые прокси-серверу для проверки подлинности.

Реализации

IWebProxy.Credentials содержит учетные данные для проверки подлинности, передаваемые прокси-серверу в ответ на код состояния HTTP 407 (проверка подлинности прокси).

Пример имитации работы браузера

```
HttpWebRequest myHttpWebRequest = (HttpWebRequest)
HttpWebRequest.Create("http://tut.by");
HttpWebResponse myHttpWebResponse =
(HttpWebResponse)myHttpWebRequest.GetResponse();
```

В данном примере в первой строке создается экземпляр класса HttpWebRequest для работы с сервером http:// tut.by. Во второй строке создается экземпляр класса HttpWebResponse, который получает результат выполнения GET-запроса к серверу http:// tut.by. Метод GET – это значение по умолчанию.

Для того чтобы вывести полученный результат в консоль, достаточно добавить еще несколько строчек кода:

```
StreamReader myStreamReader = new
StreamReader(myHttpWebResponse.GetResponseStream(), Encoding.GetEncoding(1251));
Console.WriteLine(myStreamReader.ReadToEnd());
Console.ReadKey();
```

Чтобы экземпляр класса HttpWebRequest использовал прокси-сервер, необходимо указать параметры прокси-сервера в свойстве Proxy:

myHttpWebRequest.Proxy = new WebProxy("127.0.0.1", 8888);

В данном случае 127.0.0.1 – это локальный адрес компьютера, а 8888 – порт, который прослушивает прокси-сервер.

```
HttpWebRequest myHttpWebRequest =
(HttpWebRequest)HttpWebRequest.Create("http://tut.by");
myHttpWebRequest.Proxy = new WebProxy("127.0.0.1", 8888);
HttpWebResponse myHttpWebResponse =
(HttpWebResponse)myHttpWebRequest.GetResponse();
```

Браузер передает гораздо больше заголовков серверу, главным образом это информация о себе (User-Agent), Cookies, предпочитаемый язык (Accept-Language) и тип данных (Accept). Все эти заголовки также можно создать программно через коллекцию Headers либо одноименные свойства объекта HttWebRequest. Объем кода придется увеличить, но сложнее он от этого не станет:

```
HttpWebRequest myHttpWebRequest = (HttpWebRequest)HttpWebRequest.Create("http://
tut.by ");
myHttpWebRequest.Proxy = new WebProxy("127.0.0.1", 8888);
myHttpWebRequest.UserAgent = "Mozila/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1; MyIE2;";
myHttpWebRequest.Accept = "image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*";
myHttpWebRequest.Headers.Add("Accept-Language", "ru");
HttpWebResponse myHttpWebResponse =
(HttpWebResponse)myHttpWebRequest.GetResponse();
```

Если теперь посмотреть на отчет прокси-сервера, то там будут видны все заголовки, которые передали. Таким образом, сервер будет «думать», что на сайт пришел пользователь с браузером Internet Explorer, для которого предпочтительный язык – русский (ru). В данном примере не передавались Cookies, поскольку для начала их надо получить.

Пример авторизации на сайте

Для примера попробуем пройти авторизацию на сайте http://yandex.ru.

Прежде всего нужно сформировать запрос, причем сначала на главную страницу «Яндекс», поскольку именно там выдаются первые Cookies, без которых дальнейшая работа в данном направлении попросту может завести в тупик. Запрос будет выполняться методом GET:

```
HttpWebRequest myHttpWebRequest =
(HttpWebRequest)HttpWebRequest.Create("http://yandex.ru");
myHttpWebRequest.Proxy = new WebProxy("127.0.0.1", 8888);
myHttpWebRequest.UserAgent = "Mozila/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1; MyIE2;";
myHttpWebRequest.Accept = "image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*";
myHttpWebRequest.Headers.Add("Accept-Language", "ru");
HttpWebResponse myHttpWebResponse =
(HttpWebResponse)myHttpWebRequest.GetResponse();
string sCookies = "";
if (!String.IsNullOrEmpty(myHttpWebResponse.Headers["Set-Cookie"]))
{
  sCookies = myHttpWebResponse.Headers["Set-Cookie"];
}
```

Куки мы получаем из заголовка Set-Cookies и записываем их в переменную sCookies. Далее надо пройти процедуру авторизации на «Яндекс», делать это будем уже методом POST:

```
HttpWebRequest myHttpWebRequest =
(HttpWebRequest)HttpWebRequest.Create("http://passport.yandex.ru/passport?mode=auth");
myHttpWebRequest.Proxy = new WebProxy("127.0.0.1", 8888);
myHttpWebRequest.Method = "POST";
myHttpWebRequest.Referer = "http://yandex.ru";
myHttpWebRequest.UserAgent = "Mozila/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1; MyIE2;";
myHttpWebRequest.Accept = "image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*";
myHttpWebRequest.Headers.Add("Accept-Language", "ru");
myHttpWebRequest.ContentType = "application/x-www-form-urlencoded";
// передаем куки, полученные в предыдущем запросе
if (!String.IsNullOrEmpty(sCookies))
{
  myHttpWebRequest.Headers.Add(HttpRequestHeader.Cookie, sCookies);
}
// ставим False, чтобы при получении кода 302 не делать автоматический редирект
myHttpWebRequest.AllowAutoRedirect = false;
// передаем параметры
TimeSpan ts = DateTime.Now - new DateTime(1970, 1, 1);
string sQueryString = "retpath=http%3A%2F%2Fmail.yandex.ru%2F&timestamp=" +
```

```
Math.Floor(ts.TotalSeconds).ToString() +
"&login=" + sLogin + "&passwd=" + sPassword;
byte[] ByteArr = System.Text.Encoding.GetEncoding(1251).GetBytes(sQueryString);
myHttpWebRequest.ContentLength = ByteArr.Length;
myHttpWebRequest.GetRequestStream().Write(ByteArr, 0, ByteArr.Length);
// делаем запрос
HttpWebResponse myHttpWebResponse =
(HttpWebResponse)myHttpWebRequest.GetResponse();
```

Очень важно не забыть у свойства AllowAutoRedirect указать значение False, если этого не сделать, то при получении кода 301/302, HttpWebRequest автоматически перейдет на указанную в заголовках страницу, при этом могут потеряться важные данные, например, куки, и следовательно, сервер не пустит нас на нужную страницу. Также не забудьте указать в строке запроса (переменная sQueryString) параметры login и passwd. Если вы сделаете все правильно, то сервер возвратит код 302, в противном случае – код 200 со страницей, содержащей информацию об ошибке, либо форму авторизации.

Далее надо получить адрес страницы, на которую перенаправляет сервер, эта информация содержится в заголовке Location. Все последующие запросы будут выполняться методом GET:

```
        string sLocation = myHttpWebResponse.Headers["Location"];
// получам куки
sCookies = "";
if (!String.IsNullOrEmpty(myHttpWebResponse.Headers["Set-Cookie"]))
{
  sCookies = myHttpWebResponse.Headers["Set-Cookie"];
}
// формируем запрос
myHttpWebRequest =
(HttpWebRequest)HttpWebRequest.Create("http://passport.yandex.ru" + sLocation);
myHttpWebRequest.Proxy = new WebProxy("127.0.0.1", 8888);
myHttpWebRequest.Referer = "http://passport.yandex.ru/passport?mode=auth";
myHttpWebRequest.UserAgent = "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1; MyIE2;";
myHttpWebRequest.Accept = "image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*";
myHttpWebRequest.Headers.Add("Accept-Language", "ru");
myHttpWebRequest.ContentType = "text/plain";
if (!String.IsNullOrEmpty(sCookies))
{
  myHttpWebRequest.Headers.Add(HttpRequestHeader.Cookie, sCookies);
}
// выполняем запрос
myHttpWebResponse = (HttpWebResponse)myHttpWebRequest.GetResponse();
```

Следующая страница, которую возвратит сервер, будет содержать информацию о том, что браузер не поддерживает автоматический редирект, и

предложение нажать на ссылку для входа в ящик. Редирект производится при помощи JavaScript, а также через мета-тэги. Будем использовать JavaScript, хотя принципиального значения это не имеет. Собственно адрес, на который нужно перейти, был в самом первом шаге, когда отправлялся POST-запрос, на него и будем сейчас переходить. Здесь очень важно не потерять куки, поскольку в них содержится идентификатор сессии (Session), если куки потеряны, то сервер перенаправляет на страницу авторизации.

```
myHttpWebRequest = (HttpWebRequest)HttpWebRequest.Create("http://mail.yandex.ru/");
myHttpWebRequest.Proxy = new WebProxy("127.0.0.1", 8888);
myHttpWebRequest.Referer ="http://passport.yandex.ru" & sLocation;
myHttpWebRequest.UserAgent = "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1; MyIE2;";
myHttpWebRequest.Accept = "image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*";
myHttpWebRequest.Headers.Add("Accept-Language", "ru");
myHttpWebRequest.ContentType = "text/plain";
if (!String.IsNullOrEmpty(sCookies))
{
  myHttpWebRequest.Headers.Add(HttpRequestHeader.Cookie, sCookies);
}
// выполняем запрос
myHttpWebResponse = (HttpWebResponse)myHttpWebRequest.GetResponse();
// выводим результат в консоль
StreamReader myStreamReader = new
StreamReader(myHttpWebResponse.GetResponseStream(), Encoding.GetEncoding(1251));
Console.WriteLine(myStreamReader.ReadToEnd());
Console.WriteLine("Нажмите любую клавишу для завершения.");
Console.ReadKey();
```

Если запустить данный код и посмотреть на отчет прокси-сервера, то будет видно, что при первом запросе сервер выдает код 200, затем 302 и опять два раза 200, т. е. все точно так же, как если бы была пройдена процедура авторизации через браузер. В окне консоли должно появиться содержимое страницы сервиса работы с почтовым ящиком на «Яндекс.Ру».

Лабораторная работа №3

Цель работы: ознакомиться с понятием репликация базы данных.

Краткие теоретические сведения

Репликация (синхронизация) – процесс приведения данных электронных таблиц двух БД в идентичное состояние.

Репликацию можно классифицировать следующим образом.

По направлению репликации. Если данные изменяются только в одной из БД, а в другой данные только хранятся и не подвергаются изменениям, то такая репликация называется однонаправленной, или односторонней. Если же данные могут изменяться и вводиться на всех БД, то такой вид репликации называется мультинаправленной, или многосторонней.

По времени проведения сеанса репликации. Если данные должны быть засинхронизированны немедленно после изменений, то такая репликация называется репликация реального времени. Если же процесс репликации запускается по какому-либо событию во времени или по указанию администратора БД, то такой вид репликации называется отложенной репликацией.

По способу передачи информации во время процесса репликации. Если соединение серверов, хранящих распределенные БД, происходит при помощи программы клиента, которая с одной стороны подсоединяется к своему серверу, а с другого конца имеет прямую связь с БД другого сервера и может подключиться напрямую к данным другого сервера для прямого изменения и анализа реплицируемых данных с обоих концов, имея при этом гарантированный устойчивый канал связи (ADSL, выделенный канал, двупроводная линия Dial-Up и пр.), то такой вид синхронизации называется прямым. Если же канал неустойчивый и не гарантирует устойчивую связь без падений во время процесса синхронизации и данные приходится передавать цельными пачками, при этом принимающая сторона во время закачки и анализа данных не имеет немедленной возможности опросить источник при возникновении, на ее взгляд, сомнительных моментов, а решение принимать в любом случае нужно, то эта синхронизация – недетерминированная, или вероятностная.

По способу анализа реплицируемой информации. Если ядро алгоритма работает по принципу сравнения записей одной таблицы с записями другой, и на основании этого принимается решение о синхронизации, то такой процесс называется репликацией по текущему состоянию. Если в базе предусмотрен журнал вносимых изменений в БД, и алгоритм репликации переносит изменения по дельтам изменений, накопленным в журнале, это – дельта репликация.

Проблема синхронизации уникальных идентификаторов

Основа любой учетно-финансовой автоматизированной системы – объекты, хранящиеся в справочниках БД. А для справочника главное – это

поддержание уникальности своих ID-идентификаторов, как минимум. Только в данном случае эту таблицу можно называть справочником. Обычно ID записи создаются хранимым генератором уникальной последовательности целых чисел, и при помощи триггера BeforeInsert производится вставка новой записи в справочник. А теперь попытайтесь перенести все сказанное на распределенную базу, в которой центральная БД и все удаленные БД работают независимо друг от друга, и, соответственно, должны уметь генерировать уникальные ID своих справочников, не консультируясь друг с другом.

Можно ввести в каждую таблицу дополнительное поле – номер БД, в которой эта запись была создана впервые (DBID). При этом очевидно, ID уже не будет являться первичным ключом, вместо этого первичным ключом будет пара (DBID, ID). Следует заметить, что из-за этого данное решение не слишком привлекательно.

Можно сделать первичным ключом строку специального формата, например XXXX-YYYY-ZZZZZZZZ, где XXXX – это идентификатор базы данных, где запись была создана впервые, YYYY – идентификатор таблицы, ZZZZZZZZ – идентификатор записи внутри конкретной таблицы конкретной БД. Такое решение является хорошо масштабируемым, позволяет поместить в ID много дополнительной информации, но у него тоже есть минусы. Во-первых, некоторая избыточность информации. Во-вторых, более сложный механизм генерации таких ID.

Для InterBase лучшим вариантом является следующий – ID для всех таблиц генерируется обычным триггером, выбирающим значения из генератора. При этом начальное значение генератора различно для разных БД, за счет чего обеспечится уникальность ID по всем БД. Данный подход характерен для InterBase. Применение его для других СУБД может быть ограничено невозможностью задать начальное значение для счетчика автоинкрементных полей.

Лабораторная работа №4

Цель работы: ознакомиться с понятием RSS-протокол, технологией XML DOM.

Краткие теоретические сведения

RSS – формат передачи веб-контента. Название технологии – «Really Simple Syndication», т. е. «по-настоящему простая передача информации».

Все файлы RSS обязаны соответствовать спецификации XML 1.0. На высшем уровне документ RSS представляет собой элемент <rss> с обязательным атрибутом version, указывающим версию RSS. Если документ соответствует этой спецификации, значение этого атрибута должно быть «2.0». Дочерний элемент <rss> – один элемент <channel>, который включает информацию о канале (метаданные) и его содержимое.

Обязательные элементы канала:

– title – название канала, по которому люди будут ссылаться на сервис. Если у вас есть веб-сайт, который содержит ту же информацию, что и RSS-файл, элемент title канала должен совпадать с title заглавной веб-страницы;

– linkURL веб-сайта, связанного с каналом;

– description – фраза или предложение для описания канала.

Необязательные элементы канала:

– language – язык, на котором написан канал. Позволяет сборщикам, на пример, объединить на одной странице все сайты на итальянском языке;

– copyright – информация об авторском праве на канал;

– managingEditor – адрес электронной почты человека, ответственного за редакторский текст;

– webMaster – адрес электронной почты человека, ответственного за технические аспекты канала;

– pubDate – дата публикации текста в канале;

– lastBuildDate – время последнего изменения содержимого канала;

– category – указывает одну и более категорию, к которой относится канал. Следует тем же правилам, что и элемент category в <item>.(<category>Newspapers</category>);

– generator – строка определяет использованную для создания канала программу;

– docsURL – указывает на документацию для использованного в файле RSS формата (например, на данную страницу). Это для людей, которые могут столкнуться через 25 лет с RSS-файлом на веб-сервере и захотят узнать, что это такое;

– cloud – указывает веб-сервис, поддерживающий интерфейс rssCloud;

– ttl – время жизни; количество минут, на которые канал может кешироваться перед обновлением с ресурса.(<ttl>60</ttl>);

– image – изображение GIF, JPEG или PNG, которое может отображаться с каналом;

– rating – рейтинг канала PICS;

– textInput – поле текстового ввода, которое может отображаться с каналом;

– skipHours – совет сборщикам каналов – какие часы можно пропустить;

– skipDays – совет сборщикам каналов – какие дни можно пропустить.

Технология DOM

DOM (от англ. Document Object Model — «объектная модель документов») – это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому документов, а также изменять содержимое, структуру и оформление документов.

Модель DOM не накладывает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого содержит элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями родитель – потомок.

Изначально различные браузеры имели собственную модель DOM, не совместимую с остальными. Для того чтобы обеспечить взаимную и обратную совместимость, специалисты международного консорциума W3C классифицировали эту модель по уровням, для каждого из которых была создана своя спецификация. Все эти спецификации объединены в общую группу, носящую название W3C DOM.

Уровни W3C DOM:

Уровень 0

Включает в себя все специфические модели DOM, которые существовали до появления Уровня 1, например, document.images, document.forms, document.layers и document.all. Необходимо обратить внимание, что эти модели формально не являются спецификациями DOM, опубликованными W3C, а скорее являются информацией о том, что существовало до начала процесса стандартизации.

Уровень 1

Базовые функциональные возможности DOM (HTML и XML) в документах, такие как получение дерева узлов документа, возможность изменять и добавлять данные.

Уровень 2

Является текущим уровнем спецификаций DOM.

Поддержка так называемого пространства имен XML <--filtered views--> и событий.

Уровень 3

Состоит из шести различных спецификаций:

DOM Level 3 Core;

DOM Level 3 Load and Save;

DOM Level 3 XPath;

DOM Level 3 Views and Formatting;

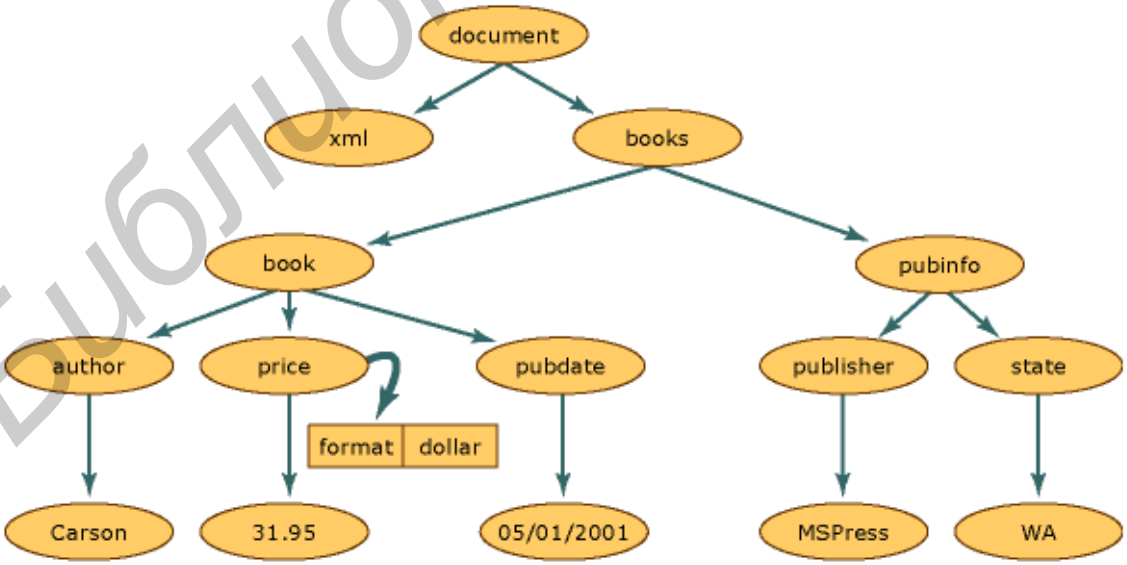
Level 3 Requirements; и
DOM Level 3 Validation.
Эти спецификации являются дополнительными расширениями DOM.

XML DOM

Модели объектов-документов (DOM) XML служат для доступа и обновления содержимого, стиля и структуры XML-документов через программируемый интерфейс. При этом берется форма модели объекта, состоящая из объектов, свойств, методов и событий, с помощью которых можно управлять компонентами XML-документа. DOM позволяет читать, управлять и изменять документ XML. Класс **XmlReader** также читает XML (доступ только для чтения). Данные XML-документа сохраняются в виде иерархической древовидной структуры, отражающей структуру самого документа.

```
<?xml version="1.0"?>
 <books>
  <book>
     <author>Carson</author>
     <price format="dollar">31.95</price>
     <pubdate>05/01/2001</pubdate>
  </book>
  <pubinfo>
     <publisher>MSPress</publisher>
     <state>WA</state>
  </pubinfo>
 </books>
```

Структура XML документа



Пример XML-файла:

```
<?xml version="1.0" encoding="windows-1251" ?>
- <rss version="2.0">
  - <channel>
      <title>TUT.BY: Новости ТУТ: Главные новости</title>
      <link>http://news.tut.by/</link>
      <description>Новости TUT.BY</description>
    + <image>
      <lastBuildDate>Sat, 01 Mar 2008 17:57:44 +0200</lastBuildDate>
    - <item>
        <title>Андрей Костицын установил новый белорусский рекорд НХЛ</title>
        <link>http://news.tut.by/104426.html</link>
        <description>29 февраля в поединке "Баффало" и "Монреаля", завершившемся крупной п
          отметился 19-й заброшенной шайбой в нынешнем чемпионате Национальной хоккейно
          рекордный результат Владимира Цыплакова.</description>
        <pubDate>Sat, 01 Mar 2008 10:01:00 +0200</pubDate>
        <guid>http://news.tut.by/104426.html</guid>
      </item>
    + <item>
    + <item>
    + <item>
    + <item>
    + <item>
    + <item>
    + <item>
    + <item>
    + <item>
    + <item>
    + <item>
    </channel>
  </rss>
```

DOM служат для доступа к любому компоненту XML-документа, включая элементы, атрибуты, команды обработки, примечания и элементы описания. Любой XML-документ можно загрузить в DOM. При этом происходит его чтение и сохранение в виде логической модели узлов.

Программируемый интерфейс DOM позволяет перемещаться по структуре документа в приложении и управлять ее узлами. Каждый узел имеет свой тип согласно константам перечисляемого типа XML DOM, которые также определяют родительский и дочерний узлы для каждого типа узла. Для большинства XML-документов общими типами являются элемент, атрибут и текст. Атрибуты занимают особое место в модели объекта, поскольку не являются дочерними узлами и рассматриваются как свойства элемента.

Ниже описаны некоторые объекты DOM, используемые при роботе с XML-документом, а также представляемые типы узлов XML.

XMLDOMDocument – представляет XML-документ в целом и содержит свойства и методы, позволяющие просматривать, осуществлять поиск и изменять содержимое и структуру XML-документа.

XMLDOMNode – представляет отдельный узел в структуре документа и является базовым объектом доступа к данным в XML DOM, включающим в себя поддержку типов данных, пространств имен и XML-схем.

XMLDOMNodeList – представляет коллекцию узлов и позволяет осуществлять операции итерационного и индексного доступа над активной коллекцией IXMLDOMNode.

XMLDOMElement – представляет элемент XML-документа.

XMLDOMAttribute – представляет атрибут XML-документа.

Класс XmlDocument

Если необходимо XML-файл в память для его обработки, для этого создается новый класс XMLDocument, который представляет собой два конструктора с типом доступа public:

```
public XmlDocument();
public XmlDocument(XmlNameTable);
```

Следующая часть кода демонстрирует, как можно загрузить XML-файл:

```
XmlDocument doc = new XmlDocument();
doc.Load(filename) ;
```

Метод Load всегда работает синхронно, документ полностью находится в памяти и готов к дальнейшей обработке с помощью методов и свойств, объявленных в классе. Класс XmlDocument использует XML-reader (читателя) внутри, чтобы выполнить любую прочитанную операцию и построить заключительную структуру дерева первоисточника.

Методы класса XmlDocument:

CloneNode – создает дубликат документа;

CreateAttribute – создает атрибут с заданным именем;

CreateCDataSection – создает CDATA секцию с заданным именем;

CreateComment – создает комментарий с заданными данными;

CreateDocumentType – создает DOCTYPE элемент;

CreateElement – создает элемент;

CreateNode – создает запись с заданным типом;

CreateXmlDeclaration – создает стандартную XML-декларацию;

ImportNode – импортирует записи в другие документы;

Load – загружает XML данные в указанный источник;

LoadXml – загружает XML данные в указанную строку;

Save – сохраняет текущий документ в указанное место.

Литература

1. MSDN Library. Раздел Networking and Directory Services – Network Protocols – NetBIOS, раздел Networking and Directory Services – Network Management – Windows Networking (WNet).
2. Справочник по командам Windows: http://winchanger.whatis.ru/file/prog.zip/.
3. Windows: Сети: Изучение TCP/IP: http://www.oszone.net/windows/lan/dns/default.htm/.
4. Закер, К. Компьютерные сети. Модернизация и поиск неисправностей / К. Закер; пер. с англ. – СПб. : БХВ-Петербург, 2004. – 1008 с.

Св. план 2010, поз. 51

Учебное издание

Сети ЭВМ

Лабораторный практикум
для студентов специальности 1-40 01 01
«Программное обеспечение информационных технологий»
дневной формы обучения

В 2-х частях
Часть 2

Мельникова Елена Владимировна
Марина Ирина Михайловна

Редактор Е. Н. Батурчик
Корректор А. В. Тюхай
Компьютерная верстка М. В. Гуртатовская

Подписано в печать 27.04.2011. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Отпечатано на ризографе. Усл. печ. л. 1,86.
Уч.-изд. л. 1,7. Тираж 100 экз. Заказ 196.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6