МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «КубГУ»)**

**Кафедра вычислительных технологий**

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК Заведующий кафедрой д-р физ.-мат. наук, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.И,Миков

(подпись) (инициалы, фамилия) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2016 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**РАЗРАБОТКА ПРОГРАММНОГО АГЕНТА ДЛЯ**

**ИНФОРМАЦИОННОГО ПОИСКА В СРЕДЕ ИНТЕРНЕТ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Айдоган Ялчын

(подпись, дата)

Факультет компьютерных технологий и прикладной математики Направление 02.03.02 – «Фундаментальная информатика и информационные технологии»

Научный руководитель канд. физ.-мат. н.\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ О.Н.Лапина

(подпись, дата)

Нормоконтролер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.С. Фисун

(подпись, дата)

Краснодар 2016

# РЕФЕРАТ

Выпускная квалификационная работа содержит 55 страниц, 11 рисунков, 14 источников.

Целью работы является разработка автономного программного агента, с помощью которого можно получать информацию по заданным характеристикам с различных сайтов сети интернет.

Результат работы – разработанный автономный агент в форме программного приложения, который проводит анализ популярных сайтов, содержащих в открытом доступе научно-исследовательские статьи по дисциплинам математики и компьютерные науки. Программное приложение производит поиск статей по заданным характеристикам и сохраняет их в базу данных. При появлении новой публикации программный агент добавляет ее в список сохраненных публикаций.

В работе использовался язык программирования С#, библиотека HtmlAgilityPack.

КЛЮЧЕВЫЕ СЛОВА ДАННОЙ РАБОТЫ:

Программные агенты, парсинг сайта, регулярные выражения

# СОДЕРЖАНИЕ

Введение ................................................................................................................. 4

1. Программные агенты ........................................................................................ 6
   1. Основные понятия ...................................................................................... 6
   2. Классификация программных агентов ...................................................... 8
   3. Информационный поиск в среде Интернет ......................................... 10
2. Парсинг сайта ................................................................................................... 13
   1. Парсинг и его фазы .................................................................................... 13
   2. Регулярные выражения ............................................................................. 14
   3. Библиотека HtmlAgilityPack...................................................................... 17
   4. Объектная модель документа (DOM) ...................................................... 18
   5. Многопоточные приложения ................................................................... 22
3. Процесс сериализации ..................................................................................... 26
   1. Сериализация и десериализация .............................................................. 26
   2. Бинарная сериализация (Binary Serialization) ......................................... 28
   3. Простая сериализация (Basic serialization) .............................................. 29
   4. Форматоры (Formatters) ............................................................................ 32
   5. События бинарных форматоров и сериализация .................................... 33
4. Разработка проекта на C# ................................................................................ 35
   1. Анализ сайтов ............................................................................................ 35
   2. Пример работы программы ...................................................................... 38

Заключение ........................................................................................................... 44

Список использованных источников .................................................................. 45

Приложение А Программный код ....................................................................... 46

ВВЕДЕНИЕ

Проблематика интеллектуальных агентов и мультиагентных систем имеет богатое прошлое и была сформирована на основе полученных результатов по распределенному и параллельному искусственному интеллекту. Но лишь с недавнего времени проблематика интеллектуальных агентов и мультиагентных систем выделилась в обособленную область приложений и исследований. Данная тематика содержит широкое направление работ в исследуемой области: компьютерные сети, искусственный интеллект, информационные технологии.

Существует два подхода в построении агентно-ориентированных систем: разработка мультиагентной системы и формирование уникального автономного агента, которые осуществляет взаимодействие только с пользователем.

Мультиагентные системы являются программными комплексами, которые содержат взаимодействующие между собой различные агенты, решающие поставленные задачи, которые невозможно выполнить для одного агента.

Целью выпускной работы является разработка автономного программного агента, с помощью которого можно получать информацию по заданным характеристикам с различных сайтов сети интернет.

Основные задачи:

1. Исследование сети Интернет, поиск сайтов, которые содержат в открытом доступе научную литературу (книги, статьи) по дисциплинам математике и компьютерным наукам: электронные библиотеки, сайты ВУЗов и т.п.
2. Разработка программного агента, который должен анализировать сайты с целью поиска по заданным характеристикам научной публикации.
3. Разработка базы данных для хранения полученной информации. База данных должна хранить список найденной литературы, ссылки на сайты и состоять из различных таблиц – по различной тематике (например, Компьютерные сети, Программирование, Графы, Криптография и т.п.). Если на сайтах появляется новая публикация, необходимо ее данные добавить в соответствующую таблицу базы данных.

Язык программирования C#, с использованием библиотеки HtmlAgilityPack и системы регулярных выражений для задания шаблонов поиска информации.

1. Программные агенты

1.1 Основные понятия

В настоящее время в области агентных технологий работают множество университетов, исследовательских лабораторий, предприятий и промышленных организаций, и их количество постоянно растет.

Список вышеперечисленных организаций включает малоизвестные имена и маленькие коллективы, признанные научные центры (например, фирма GeneralMagic и университет Карнеги Мэллон (CMU)), а также огромные компании (например, AT&T,Apple, BT, HP, Daimler-Benz, DEC, IBM, Microsoft, Lotus, Oracle, Sharp и др.)

Агентные технологии находят практическое применение в следующих областях:

1. управления воздушным движением;
2. управление информационными потоками;
3. информационный поиск;
4. электронная коммерция;
5. обучение;
6. электронные библиотеки; и многие другие приложения.

Мультиагентные системы (*MAC)* и программные агенты, необходимы и полезны по нескольким причинам. Главная причина – агенты являются автономными и могут выполняться от лица пользователя в фоновом режиме при решении различных задач. К наиболее важным задачам следует отнести сбор информации, ее фильтрация и использование для принятия решений.

Таким образом, в основу идеи программных агентов входит делегирование полномочий. Для реализации этой идеи агенту необходимо иметь возможность взаимодействия с пользователем или со своим владельцем для получения соответствующих задач и возвращения полученных результатов, а так же ориентироваться в среде своего выполнения и принимать необходимые для выполнения поставленных задач решения.

Существует два подхода к постарению агентно-ориентированных систем:

1. разработка мультиагентной системы;
2. реализация единственного автономного агента

Автономный агент взаимодействует только с пользователем и реализует весь спектр функциональных возможностей, необходимых в рамках агентно-ориентированной программы. В противоположность этому MAC являются программно-вычислительными комплексами, где взаимодействуют различные агенты для решения поставленных задач, которые недоступны или трудны в силу своей сложности для одного агента.

Такие мультиагентные системы часто называют агентствами, в рамках которых агенты взаимодействуют, кооперируются между собой и договариваются для поиска решения поставленной перед ними задачи.

К определению понятий в данной предметной области существует несколько подходов. Международная ассоциация FIPA (Foundation for intelligent physical agents) является одним из наиболее последовательных в этом вопросе. В каждом документе содержится толковый словарь терминов.

Замечание об отсутствии одного общего мнения по поводу понятия агента и его базисных свойств, стало общим местом практически во всех работах, где даются определения. Каждый автор, используя понятие «агент», определяет своего агента с определенным набором свойств в зависимости от поставленных целей разработки решаемых задач и техники реализации.

В следствие в рамках этого направления появилось большое количество разновидностей агентов, например: мобильные и интеллектуальные агенты, автономные агенты, персональные ассистенты, социальные агенты и множество других, а вместо уникального определения базового агента - большое количество определений производных видов.

Учитывая вышесказанное, понятие «агента» целесообразно понимать как метаимя или класс, который содержит множество подклассов.

Возьмем за основу следующую концепцию: Агент – это программная или аппаратная сущность, которая способна действовать в интересах достижения целей, поставленных перед ним пользователем и/или владельцем.

Будем рассматривать программные агенты как обособленные компоненты, которые действуют от лица пользователя.

1.2 Классификация программных агентов

В настоящее время существует несколько классификаций агентов. При этом можно выделить две большие группы: простые агенты и интеллектуальные агенты. Простые программные агенты должны иметь следующие свойства:

1. Автономное выполнение задач;
2. Взаимодействие с другими агентами и пользователями; 3) Слежение за окружением.

Целесообразное поведение появляется только на уровне интеллектуальных агентов. Так как для него необходимо не только наличие целей функционирования, но и возможность использования достаточно сложных знаний о среде, партнерах и о себе.

Понятно, что интеллектуальные агенты наследуют все характеристики простых типов агентов. Приведем типовой список свойств, которыми должны обладать интеллектуальные агенты:

1. Свойство автономности: агент способен функционировать без вмешательства со стороны владельца и может осуществлять контроль внутреннего состояния и своих действий;
2. Свойство социальности: агент может взаимодействовать и другими агентами, отправлять им сообщения;
3. Свойство реактивности: агент способен адекватно воспринимать окружение и реагировать на его изменения ;
4. Свойство проактивности: агент способен формировать цели и действовать рациональным образом для их достижения ;
5. Наличие базовых знаний: агент обладает знаниями о себе, об окружающей среде, и также о других агентах, которые не меняются, в рамках жизненного цикла агента;
6. Наличие убеждений: агент имеет непостоянную часть базовых знаний, которые могут меняться во времени, при этом агент может не знать об изменениях и продолжать использовать старые данные;
7. Целенаправленность: наличие последовательности состояний выполнение которых приводит к достижению определенной цели;
8. Наличие обязательств: задач, которые агент способен брать на себя по просьбе или поручению других агентов;
9. Наличие намерений: то что агент делает в силу своих обстоятельств намерений, желаний;

Кроме того, в этот перечень можно добавить такие свойства, как рациональность, правдивость, благожелательность, а также мобильность.

Можно выделить три базовых класса архитектуры:

1. архитектуры, основывающиеся на принципах и методах работы со знаниями;
2. архитектуры, которые опираются на поведенческих моделях типа

«стимул-ответная реакция»;

1. архитектуры гибридного типа: опираются на различные поведенческие модели.

Агенты, основывающиеся на архитектуре 1-го типа, являются «разумными» агентами, в качестве основы проектирования и реализации они используют модели и методы из области искусственного интеллекта.

Как альтернативных подход к архитектуре интеллектуальных агентов возникли принципы реактивной архитектуры. Идея реактивных агентов впервые возникла в работах Брукса, выдвинувшего тезис, что интеллектуальное поведение может быть реализовано без символьного представления знаний.

Реактивными называются агенты и архитектуры, где нет представленной модели мира, а функционирование отдельных агентов и всей системы осуществляется по правилам типа ситуация-действие. При этом под ситуацией понимается потенциально сложная комбинация внутренних и внешних состояний.

Многие исследователи считают, что ни первый, ни второй подходы не дают оптимального результата при разработке агентов и MAC. Поэтому попытки их объединения предпринимаются постоянно и уже привели к появлению разнообразных гибридных архитектур. По сути дела, именно гибридные архитектуры и используются в настоящее время во всех, скольконибудь значимых проектах и системах.

1.3 Информационный поиск в среде Интернет

В настоящее время уже ни у кого нет сомнений в том, что Интернет является de facto всемирным хранилищем информации практически во всем аспектам жизни человечества. Так же как и то, что эффективных доступ к этой информации в связи с экспоненциальным ростом объема Интернетресурсов становится все более сложным и трудоемким. И не столько с технической точки зрения, сколько с точки зрения поиска и анализа информации.

С другой стороны, как уже отмечалось, важнейшими из областей практического использования агентных технологий являются сбор информации, ее фильтрация и использование для принятия решений.

Уже сегодня пространство WWW содержит огромное количество

HTML-документов, причем не только тексты, но и видео, графику, звук и т.

д. Отношения между отдельными информационными фрагментами, представленными в сети отражают гипертекстовые связи между Webдокументами и их частями. Браузеры, поддерживающие HTML-стандарты, обеспечивают пользователям навигацию по ссылкам для доступа к документам, распределенным по сети. Однако поиск информации в настоящее время поддержан существенно слабее и в большинстве случаев базируется на использовании ограниченного числа типов машин поиска и ключевых слов.

Самым распространенным и доступным ресурсом для извлечения информации в сети Интернет являются машины поиска. Как правило, при этом используются два типа сетевых роботов: слайдеры (spiders) и индексы (indexes). Слайдеры, называемые также ботами (bots, от робот-robots), перемещаются по Web от сайта к сайту. Некоторые из них перемещаются от сервера к серверу беспорядочно, другие используют приоритеты, например такие, как посещаемость сайта. Спайдер, оказавшись на сайте, посылает отчет поисковой машине и продолжает индексирование. Индексы используются для ускорения поиска и сбора информации. Некоторые поисковые механизмы индексируют содержание страниц полностью, другие — только отдельные их части, например такие, как заголовки документов.

Основными характеристиками машин поиска являются время индексации и поиска, объем индекса, язык запросов пользователя, представление исходных и выходных документов. Важной характеристикой машин поиска является также качество представления результатов. Наиболее популярные поисковые машины в настоящее время - AltaVista (http://www.altavista.com и http: //altavista.ru), Yahoo! [(http://www.yahoo.com)](http://www.yahoo.com/), Exite (http://www.excite.com), Lycos

(http://www.lycos.com), Infoseek (http://www.infoseek. com), WebCrawler (http://www.webcrawler. com) и некоторые другие. Система Яndex является примером полно-текстовой системы поиска с учетом морфологии русского языка.

Поисковые машины, как правило, обеспечивают интерфейс типа меню, с помощью которого пользователь, используя ключевые слова или фразы и логические связки И-ИЛИ-НЕ может создать запрос на поиск информации. Большинство машин поиска находят огромное количество «релевантных» страниц по запросу пользователя. Обычно каждый найденный документ ранжируется по степени его корреляции с запросом. Релевантность каждого документа оценивается с помощью различных технологий, например учета частоты появления на странице искомых слов. Некоторые поисковые механизмы используют дополнительно другие факторы, такие как близость расположения друг к другу искомых терминов или частота посещения страницы

В целом, характеризуя поисковые машины, можно отметить, что это глобальные поисковые механизмы, охватывающие до 90 % ресурсов Интернета. Они не имеют средств анализа информации и не могут настраиваться на предпочтения пользователя. Кроме того, их сетевым роботам становится все труднее справляться с постоянным ростом ресурсов Интернета. Главной задачей машин поиска, является индексация ресурсов глобальной сети, а также поддержка и расширение соответствующих баз данных. Фактически в базах данных машин поиска хранится информация о том, где и что лежит в сети. Поэтому можно считать, что существующие машины поиска обеспечивают низкоуровневый сервис для клиентских поисковых программ более высокого уровня.

1. Парсинг сайта

2.1 Парсинг и его фазы

Нередко пользователям интернета необходимо скопировать большое количество картинок, текста или названий с определенного сайта. Простое копирование (copy-past) совершенно не подходит. Чтобы избавиться от такой монотонной работы, и существует парсинг страниц.

Парсинг – это автоматизированная процедура сбора информации или текстового контента с различных ресурсов в Интернете – сервисов или сайтов. Парсинг сайтов – последовательный синтаксический анализ информации, размещённой на интернет-страницах.

В конечном итоге информация, полученная путем применения парсинга, оформляется в виде базы данных или может быть представлена отдельным файлом.

За сбор, преобразование и анализ собранной информации отвечает специальная программа или скрипт – парсер.

Обычно парсинг требуется, когда нужно быстро найти и обобщить какую-либо конкретную информацию или контент, посвященный определенной тематике. Иногда требуется, чтобы в результате парсинга был предоставлен контент, структурированный определенным образом.

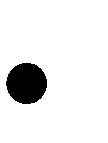
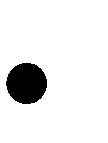
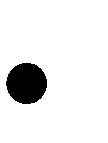
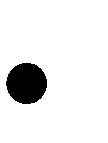
По сути, парсинг это процесс, при котором из моря трудночитаемой информации получается вполне понятный контент нужного формата.

Характер парсинга определяется заданием получить определенную информацию со страниц сайта, параметры анализа заранее задаются. Собранная информация предоставляется в определенном виде. Парсинг проводится на одном из языков программирования.

Процесса парсинга содержит три фазы:

1. сначала это пунктуальный сбор информации, это может быть, например, код интернет-страницы.
2. Затем анализ данных, обработка и преобразование в нужный формат.
3. Наконец – предоставление результата, вывод данных.

Парсинг сайтов является эффективным решением для автоматизации сбора и изменения информации. Веб-приложения для парсинга обычно пишут на C++, Delphi, Perl, Ruby, Python, PHP.

По сравнению с человеком, компьютерная программа-парсер:  быстро обойдёт тысячи веб-страниц;  аккуратно отделит техническую информацию от «человеческой»;  безошибочно отберёт нужное и отбросит лишнее;  эффективно упакует конечные данные в необходимом виде.

На сегодняшний день известны два способа парсинга:

1. с помощью регулярных выражений;
2. с помощью специальных языков запросов элементов (XPath и др.).

(англ. regular expressions) — формальный язык

поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов. По сути, регулярное выражение – это строка-образец (англ. pattern, её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска. Многие современные языки программирования имеют встроенную поддержку регулярных выражений.

2.2 Регулярные выражения

Регулярное выражение – это некий шаблон, составленный из символов и спецсимволов, который позволяет находить подстроки соответствующие этому шаблону в других строках. Спецсимволов и различных правил их комбинирования очень много, поэтому регулярные выражения можно даже назвать отдельным языком программирования.

Регулярные выражения предоставляют различные возможности, некоторые из них:

* заменять в строке все одинаковые слова другим словом, или удалять такие слова;
* выделять из строки необходимую часть. Например, из любой ссылки

(http://mycsharp.ru/post/33/2013\_10\_19\_virtualnye\_metody\_v\_sisharp\_pereopredelenie\_metodov.html) выделять только доменную часть (mycsharp.ru);

* проверять соответствует ли строка заданному шаблону. Например, проверять, правильно ли введен email, телефон т.д.;
* проверять, содержит ли строка заданную подстроку;
* извлекать из строки все вхождения подстрок, соответствующие шаблону регулярного выражения. Например, получить все даты из строки.

Для того, чтобы работать с регулярными выражениями в С# необходимо подключить в начале программы пространство имен using System.Text.RegularExpressions; Работу с регулярными выражениями предоставляет класс Regex. Создание регулярного выражения имеет следующий вид:

*Regex myReg = new Regex([шаблон]);*

*Здесь [шаблон] – это строка содержащая символы и спецсимволы.*

*Приведем простой пример программы с использованием регулярных выражений:*

*static void Main(string[] args)*

*{*

*string data1 = "Петр, Андрей, Николай"; string data2 = "Петр, Андрей, Александр";*

*Regex myReg = new Regex("Николай"); // создание регулярного выражения*

*Console.WriteLine(myReg.IsMatch(data1)); // True*

*Console.WriteLine(myReg.IsMatch(data2)); // False*

*Console.ReadKey();*

*}*

Здесь в качестве шаблона выступает однозначная строка "Николай". Дальше был использован метод *IsMatch*, который проверят, содержит ли заданная строка (data1, data2) подстроку соответствующую шаблону.

Рассмотрим кратко методы класса *Regex* для работы с регулярными выражениями:

*IsMatch* – проверяет, содержит ли строка хотя бы одну подстроку

соответствующую шаблону регулярного выражения.

*Match* – возвращает первую подстроку, соответствующую шаблону, в виде объекта класса *Match.* *Класс Match* предоставляет различную

информацию о подстроке – длину, индекс, само значение и другое.

*Matches* – возвращает все подстроки соответствующие шаблону в виде коллекции типа *MatchCollection.* Каждый элемент этой коллекции типа *Match.*

*Replace –* возвращает строку, в которой заменены все подстроки,

соответствующие шаблону, новой строкой.

Регулярные выражения для парсинга сайтов следует использовать в случае, если нужно быстро получить небольшой объем известных и неизменяемых исходных данных. Если же исходные данные меняются, при использовании регулярных выражений потребуется переписать код программы. При больших объемах данных лучше использовать специальные языки запросов.

Существует много хороших библиотек, поддерживающих языки запросов к элементам для разных языков программирования, но для быстрой разработки одной из лучших по праву является *Html Agility Pack (HAP).* Это *NET* библиотека – одна из самых удобных, с полностью открытым исходным кодом.

2.3 Библиотека *HtmlAgilityPack*

Библиотека *HtmlAgilityPack* – это *NET* библиотека, которая позволяет

работать с HTML файлами "вне сети". Это парсер HTML, который поддерживает простой язык запросов к элементам XML (*XPath* или *XSLT*) и может строить доступную для чтения или записи объектную модель документа (DOM).

DOM [3] ( Document Object Model — объектная модель документа) – это программный интерфейс, не зависящий от платформы и языка, который позволяет программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять оформление, структуру и содержимое таких документов.

Любой документ известной структуры с помощью *DOM* может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект.

*XPath* (*XML Path Language*) — язык запросов к элементам XML документа. *XPath* является стандартом консорциума *W3C* и разработан для организации доступа к частям документа XML в файлах трансформации *XSLT* *.* Он призван реализовать навигацию по DOM в *XML*. В *XPath* используется компактный синтаксис, отличный от принятого в *XML.*

Базой языка *XPath* являются оси. Пример осей: *ancestor::* — возвращает множество предков; *ancestor-or-self::* — возвращает множество предков и текущий элемент; *attribute::* — возвращает множество атрибутов текущего элемента

(сокращенно - @); *descendant-or-self::* — возвращает полное множество потомков и

текущий элемент (сокращенно - //).

Библиотека HtmlAgilityPack позволяет также использовать регулярные выражения для задания шаблонов поиска.

Библиотеку HtmlAgilityPack можно скачать с официального сайта [2].

Для начала работы необходимо открыть Visual Studio, создать новый проект на C# и добавить проект HtmlAgilityPack (вкладка Project – Add Reference – Browse).

В библиотеке HtmlAgilityPack всего доступно двадцать основных

классов. Рассмотрим примеры классов:

*HtmlDocument* – создает полный HTML-документ;

*HtmlAtribute* – создает Html-атрибуты;

*HtmlAtributeCollection* – создает объединенный список и коллекцию атрибутов;

*HtmlNode* – создает узел Html;

*HtmlNodeCollection –* создает объединенный список и коллекцию узлов

*Html;*

*HtmlTexNode* – создает текстовые узлы *Html;*

*HtmlWeb* – вспомогательный класс для получения Html-документа из

Http.

Названия методов соответствуют интерфейсам DOM: *GetElementbyId(),* *CreateAttribute(),* C*reateElement()* и т.д.,

Для работы с библиотекой HtmlAgilityPack в начале проекта необходимо вставить следующий код:

using HtmlAgilityPack; using HtmlDocument = HtmlAgilityPack.HtmlDocument;

2.4 Объектная модель документа (DOM)

Объектная модель документа (Document Object Model – DOM) [3] регламентирует способ представления содержимого документа (в частности веб-страницы) в виде набора объектов и является стандартом, предложенным веб-консорциумом. Под содержимым понимается все, что может находиться на веб-странице: ссылки, рисунки, абзацы, текст и т. д.

Следует отметить, что DOM может применяться не только в вебстраницах, но и в любых других документах. В частности, она может использоваться с любыми словарями XML. Одним из таких словарей является HTML, а точнее, XHTML.

Документ в DOM представляется в виде древовидной структуры, которая является одной из наиболее употребительных структур в программировании. Это представление обеспечивает унифицированный способ навигации по документу.

Ниже и на рисунке 1 рассмотрен пример представления Html-документа в виде дерева:

*<html>*

*<head>*

*<title>Пример представления Html-документа в виде дерева</title>*

*</head>*

*<body>*

*<h1>Представление документа в виде дерева</h1>*

*<p>Абзац 1</p>*

*<p>Абзац 2</p>*

*</body>*

*</html>*

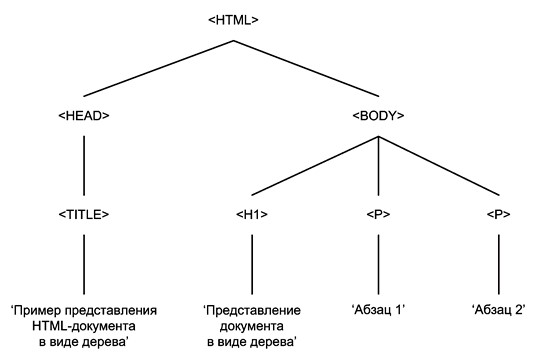


Рисунок 1 Представление HTML-документа в виде древовидной структуры

При использовании DOM для работы с текстовым файлом в формате XML исходный файл анализируется, разбивается на индивидуальные элементы, атрибуты, комментарии и т.д. Затем в памяти создается представление файла XML в виде дерева узлов, в котором каждый объект в документе рассматривается в виде узла. В качестве узлов могут выступать элементы, команды обработки, комментарии, атрибуты, и даже составляющий атрибуты обыкновенный текст.

После этого разработчик, используя дерево узлов, может обращаться к содержанию документа и, при необходимости, вносить в него изменения. Например, чтобы добавить новый элемент достаточно просто создать новый узел и прикрепить его в качестве потомка к нужному узлу.

Следует всегда выбирать модель DOM в качестве метода доступа к файлам XML. Так как этот метод имеет ряд преимуществ по сравнению с такими механизмами генерации документов XML, как запись непосредственно в поток:

1. Модель DOM гарантирует правильную грамматику и правильное оформление документов. DOM трансформирует текстовый файл в абстрактное представление дерева узлов. Это позволяет полностью избежать таких проблем, как незакрытые или неправильно вложенные теги. Работая с документом XML при помощи этого метода, разработчик должен беспокоиться не о текстовом выражении документа, а только о связях типа родитель-потомок и об относящейся к этому информации. DOM

предотвращает также создание в документе неправильных связей родительпотомок.

1. Модель DOM абстрагирует содержание от грамматики. Созданное моделью DOM дерево узлов - это логическое представление содержания файла XML. Дерево узлов показывает, какая информация представлена в файле и как ее фрагменты соотносятся друг с другом, вне непосредственной связи с грамматикой XML. Информация дерева узлов используется для создания страницы HTM или для обновления реляционной базы данных.

Разработчики при этом могут не вникать в специфику языка XML.

1. Модель DOM упрощает внутреннее манипулирование документом. Задача разработчика, использующего модель DOM для изменения внутренней структуры файла XML, становится значительно по сравнению с работой тех, применяет традиционные механизмы манипулирования файлами. DOM позволяет легко добавить элемент в середину документа. Кроме того, такие глобальные операции, как удаление из документа всех элементов с конкретным именем тега, могут быть выполнены с помощью пары команд, а не "метода грубой силы", который предполагает полное сканирование всего файла для удаления ненужных тегов.
2. Модель DOM напоминает структуры иерархических и реляционных баз данных. Способ, который использует DOM для представления связей между элементами данных, напоминает метод представления информации в современных иерархических и реляционных базах данных. С помощью этой модели упрощается процесс обмена данными между файлом XML и базой данных. Использование модели DOM для построения иерархической структуры документа позволяет легко передавать информацию между системами.

2.5 Многопоточные приложения

Парсинг будем осуществлять в отдельном потоке. Рассмотрим вопросы создания многопоточных приложений.

Многопоточность позволяет приложениям разделять задачи и работать над каждой независимо. Это позволяет максимально эффективно задействовать процессор и пользовательское время. Однако, следует отметить, не следует выбирать многопточность для всех приложений, так как при этом работа некоторых из них может даже замедлиться.

Поток является единицей обработки данных, а многозадачность — это одновременное исполнение нескольких потоков. Существует два вида многозадачности — вытесняющая (preemptive) и совместная (cooperative). Совместную многозадачность поддерживали самые ранние версии Microsoft Windows. Каждый поток при этом отвечал за возврат управления процессору, чтобы тот мог обрабатывать другие потоки.

Однако позже Microsoft Windows NT и затем Windows 95, Windows 98 и Windows 2000 стали поддерживать вытесняющую многозадачность, которую поддерживала OS/2. Процессор при этом отвечает за выдачу каждому потоку кванта времени (timeslice) – определенного количества времени, в течение которого поток может выполняться. Далее процессор переключается между разными потоками, выдавая каждому потоку его квант времени. Программист в этом случае может не заботится о том, как и когда возвращать управление, в результате чего могут работать и другие потоки.

Следует отметить, если вы работаете на однопроцессорной машине, даже в случае вытесняющей многозадачности, все равно в любой момент времени реально будет исполняться только один поток. Поскольку интервалы между переключениями процессора от процесса к процессу измеряются миллисекундами, возникает иллюзия многозадачности. Чтобы несколько потоков на самом деле работали одновременно, необходимо работать на многопроцессорной машине, разработав соответствующую программу.

Неотъемлемый атрибут потоков — переключение контекста (context switching). С помощью аппаратного таймера процессор определяет момент окончания кванта, выделенного для данного потока. Когда аппаратный таймер генерирует прерывание, процессор сохраняет в стеке содержимое всех регистров для данного потока. Затем процессор перемещает содержимое этих регистров в структуру данных CONTEXT. При необходимости переключения обратно на поток, выполнявшийся прежде, процессор выполняет обратную процедуру и восстанавливает содержимое регистров из структуры CONTEXT, ассоциированной с потоком. Весь этот процесс называется переключением контекста.

Рассмотрим создание многопоточного приложения с помощью языка C#.

Основной функционал для использования потоков в приложении сосредоточен в пространстве имен *System.Threading*. Создание потоков и управление ими осуществляется с помощью класса *System. Threading. Thread*.

Класс *Thread* определяет ряд методов и свойств, которые позволяют управлять потоком и получать информацию о нем. Основные свойства класса:

1. Статическое свойство *CurrentContext* позволяет получить контекст,

в котором выполняется потоки.

1. Статическое свойство *CurrentThread* возвращает ссылку на выполняемый поток.
2. Свойство *IsAlive* указывает, работает ли поток в текущий момент.
3. Свойство *IsBackground* указывает, является ли поток фоновым.
4. Свойство *Name* содержит имя потока.
5. Свойство *Priority* хранит приоритет потока – значение перечисления *ThreadPriority*.
6. Свойство *ThreadState* возвращает состояние потока - одно из значений перечисления *ThreadState*.

Некоторые методы класса *Thread*:

1. Статический метод *GetDomain* возвращает ссылку домен приложения.
2. Статический метод *GetDomainId* возвращает id домена приложения, в котором выполняется текущий поток.
3. Статический метод *Sleep* останавливает поток на определенное количество миллисекунд.
4. Метод *Abort* уведомляет среду *CLR* о том, что надо прекратить поток, однако прекращение работы потока происходит не сразу, а только тогда, когда это становится возможно. Для проверки завершенности потока следует опрашивать его свойство *ThreadState*.
5. Метод *Interrupt* прерывает поток на некоторое время.
6. Метод *Join* блокирует выполнение вызвавшего его потока до тех пор, пока не завершится поток, для которого был вызван данный метод.
7. Метод *Resume* возобновляет работу ранее приостановленного потока.
8. Метод *Start* запускает поток.
9. Метод *Suspend* приостанавливает поток.

При переключении процессора по окончании выделенного потоку кванта времени, процесс выбора следующего потока, предназначенного для исполнения, далеко не произволен. У каждого потока есть приоритет, указывающий процессору, как должно планироваться выполнение этого потока по отношению к другим потокам системы. Для потоков, создаваемых в период выполнения, уровень приоритета по умолчанию равен Normal. Потоки, созданные не в период выполнения, сохраняют свой исходный приоритет. Для просмотра и установки этого значения служит свойство *Thread*.Priority. Установщик свойства *Thread*.Priority принимает аргумент типа *Thread*. *ThreadPriority*. Приоритеты потоков располагаются в перечислении *ThreadPriority*: Lowest, *BelowNormal*, *Normal*, *AboveNormal*, Highest. По умолчанию потоку задается значение *Normal*. Однако мы можем изменить приоритет в процессе работы программы. Например, повысить важность потока, установив приоритет *Highest*. Среда CLR будет считывать и анализировать значения приоритета и на их основании выделять данному потоку то или иное количество времени.

Статусы потока содержатся в перечислении *ThreadState*:

*Aborted*: поток остановлен, но пока еще окончательно не завершен.

*AbortRequested*: для потока вызван метод Abort, но остановка потока еще не произошла.

*Background*: поток выполняется в фоновом режиме.

*Running*: поток запущен и работает (не приостановлен).

*Stopped*: поток завершен.

*StopRequested*: поток получил запрос на остановку.

*Suspended*: поток приостановлен.

*SuspendRequested*: поток получил запрос на приостановку.

*Unstarted*: поток еще не был запущен.

*WaitSleepJoin*: поток заблокирован в результате действия методов *Sleep* или *Join*.

В процессе работы потока его статус многократно может измениться под действием методов. Так, в самом начале еще до применения метода Start его статус имеет значение *Unstarted*. Запустив поток, мы изменим его статус на *Running*. Вызвав метод *Sleep*, статус изменится на *WaitSleepJoin*. А применяя метод Abort, мы тем самым переведем поток в состояние *AbortRequested*, а затем *Aborted*, после чего поток окончательно завершится.

1. Сериализация (Serialization) в C#

3.1 Сериализация и десериализация

Сериализация – это процесс преобразования структур данных или состояний объекта в формат, который может храниться (например, в файл или в буфер памяти, или в переданные по ссылке сети связи).

Десериализация - обратный процесс - восстановление объекта из потока байт или XML узлов.

Сериализация обычно используется для передачи объектов по сети или за границы приложения, для сохранения объектов внутри файлов или базы данных, а также для глубокого копирования объектов.

Классы для сериализации располагаются в двух пространствах имен:

System.Runtime.Serialization и System.Xml.Serialization.

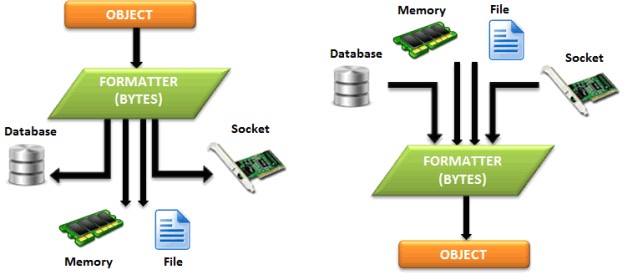


Рисунок 2 - Сериализация и десериализация.

Процесс сериализации осуществляется форматорами. В процессе сериализации данные, которые преобразованы форматорами в нужный формат записываются в поток. Если идет процесс десериализации, то данные, которые читаются из потока с помощью форматоров пишутся на объект.

Сериализованный объект, записанный на поток, не только сохраняет информацию об объекте, а также содержит его тип и версию.

С помощью сериализации разработчики смогут:

1. Отправлять объекты к удаленному приложению.
2. Отправлять объекты с одного домена к другому.
3. Отправить объекты через брандмауэр как XML или переносить конкретные данные об объектах между приложениями.

Для сериализации объекта в начале необходимо задать объекту соответствующий тип, так чтобы можно было применить к нему процесс сериализации. Затем, преобразовав его в другой формат с помощью форматорам, запишем его на стрим, получим сериализованный объект.

Пространство имен *System.Runtime.Serialization* содержит классы, которые можно использовать для сериализации и десериализации объектов.

При сериализации все сериализуемые поля объекта класса и их значения (т. е. не помеченные как *NonSerialized*) сохраняются в файл. Удобно, когда заранее известно, какие данные понадобиться хранить в файле (и какие соответственно поля должен иметь класс), и когда возможны дальнейшие доработки (добавить какое-то поле или удалить). Для этого используется класс [*SerializableAttribute****.***](http://msdn.microsoft.com/en-us/library/system.serializableattribute.aspx) Если попытаться сериализовать помеченный объект этого класса, то получим ошибку [*SerializationException****.***](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.serializationexception.aspx)

Если понадобится сериализовать в классе [*SerializableAttribute*](http://msdn.microsoft.com/en-us/library/system.serializableattribute.aspx)то нужно его пометить [*NonSerializedAttribute****.***](http://msdn.microsoft.com/en-us/library/system.nonserializedattribute.aspx)А если сериализуемый объект имеет типы указателей, класс handle или при сериализации эти данные будут непригодными в других средах так же используем пометку

NonSerializedAttribute. Когда все типы сконфигурированы для участия в

схеме сериализации .NET с применением необходимых атрибутов, следующий шаг состоит в выборе формата (двоичного, SOAP или XML) для сохранения состояния объектов. Перечисленные возможности представлены следующими классами:

* BinaryFormatter
* SoapFormatter
* XmlSerializer
* СustomSerialization (protobuf.net, NewtonSoft.Json)

При создании новых классов для сериализации обращается внимание:

* Созданный класс будет ли передавать application domains;
* Созданный класс будет ли использован c remoting или WCF;
* В классе объект будет ли предоставляться с постоянного источника.
* Если объект абстрактный то, будут ли сериализоваться классы образовывающейся в этом же классе.

На что следует обратить внимание при выборе сериализатора, на содержание объектов и способ сериализации. Если работать с большими объемами, то XML-сериализация не будет верным выбором. Потому что, преобразовав данные в XML, объем увеличивается, а скорость отправления снижается. В такой ситуации, лучшие способы, BinaryFormatter или SOAP.

* 1. Бинарная сериализация (Binary Serialization)

*Бинарная сериализация*  защищает надежность типа системы ( *type system fidelity*). Т.е в процессе сериализации сохраняются типы данных и таким образом обеспечивается надежность типа системы. Безопасность типа системы (*type system fidelity*) обеспечивает использование одних и тех же системных типов в процессе сериализации и десериализации. С помощью *type system fidelity* очень удобно передавать сериализованные объекты между клиентом (сlient) и сервером (server),. XML serialization поддерживает *type system fidelity*.

В процессе бинарной сериализации, преобразовав данные (класс объекта, private, public свойства объекта и в каком ассамблеи находится объект) на байт стрим (byte sream) в стрим (stream) пишется. А в десериализации делается все аналогично, только в обратном порядке.

Бинарный сериализатор прост в применении, хорошо автоматизирован и поддерживается повсеместно в .NET. Бинарный сериализация используется инфраструктурой Remoting, в т.ч. при взаимодействии между двумя доменами приложений в одном и том же процессе. Бинарный сериализатор работает быстрее чем сериализатор контрактов данных, но дает худшую переносимость, поскольку тесно связывает внутреннюю структуру типа с форматом сериализованных данных. Он также не может быть использован для получения XML.

* 1. Простая сериализация (Basic serialization)

Вывод сериализатора оформляется с помощью подключаемого форматера. Форматер приводит форму финального представления в соответствие с конкретной средой или контекстом сериализации. Доступно два форматера: форматер XML и двоичный форматер. XML форматер используется в контексет чтения/записи XML, тектовых файлов и потоков, обмена сообщениями SOAP. Двоичный форматер используется в контексте произвольного потока байт. Двоичный вывод по размерам обычно меньше чем XML, иногда значительно.

Форматор (Formatter)-это класс который реализует процесс сериализации и десериализации. Для форматора сериализации ( formatter serialization) разработан интерфейс(*interface*)

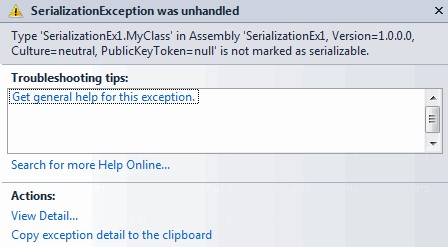
*System.Runtime.Serialization.IFormatter****.***

Разработчик должен принимать решение можно ли сериализовать объект.

Разработчику следует обратить внимание на перечисленные пункты:

* Типы по умолчанию не сериализуются. Самый простой способ для сериализации; помечать его *[Serializable].*
* Отметка *[Serializable]*используется только с referans tips,value tips,enumerators и delegates.
* Объект образован из одного типа если нуждается в сериализации то нуждается и отметке *[Serializable].*
* Если класс отмечен с *[Serializable]*то все основные классы нужно

отмечать с *[Serializable].*



# Рисунок 3 - Процесс сериализации

* Так же если тип (type) отмечен с *[Serializable]*но указанный класс не возможно сериализовать, то получим ту же самую ошибку.
* Типы Enumeration и delegate всегда можно сериализовать, по этому

не нужно их пометить с *[Serializable].*

Пример 1

*[Serializable()]*

*public class MyClass*

*{ private string sPrivate;*

*public string sPublic;*

*public MyClass(string arg1, string arg2)*

*{*

*sPrivate = arg1;*

*sPublic = arg2;*

*}*

*}*

*private void btnBasicBinarySerialization\_Click(object sender, EventArgs e)*

*{*

*SerializeObject();*

*DeSerializeObject();*

*}*

*private void SerializeObject() {*

*// Создается instance сериализованного типа.*

*MyClass ob = new MyClass("Private veri", "Public veri");*

*// Создается стрим в котором будет хранится данные объекта. //Это стрим(Stream) может и быть создан из абстрактного класса //System.IO.Stream (MemoryStream, FileStream, NetworkStream).*

*// Используя FileStream сериализуем.*

*Stream stream = new FileStream("SerializedFile.dat", FileMode.Create,*

*FileAccess.Write, FileShare.Write);*

*// Для сериализации объекта создав, BinaryFormatter,*

*// реализуется сериализация*

*System.Runtime.Serialization.IFormatter formatter = new*

*System.Runtime.Serialization.Formatters.Binary.BinaryFormatter(); formatter.Serialize(stream, ob);*

*// закрываем стрим(stream)*

*stream.Close();*

*}*

Пример 2 *private void DeSerializeObject() {*

*// Читается с файла и пишется в стрим(stream).*

*Stream stream = new FileStream("SerializedFile.dat", FileMode.Open, FileAccess.Read, FileShare.Read);*

*// BinaryFormatter создается.*

*// Не надо забывать что в сериализации и десериализации с одной и тот же //форматором(formatter) работаем. System.Runtime.Serialization.IFormatter formatter = new*

*System.Runtime.Serialization.Formatters.Binary.BinaryFormatter(); // Объект десериализируется и получает начальное состояние.*

*// вызывается конструктор(constructor)*

*MyClass ob = (MyClass)formatter.Deserialize(stream); MessageBox.Show(ob.sPublic);*

*//стрим(stream) закрывается.*

*stream.Close();*

3.4 Форматоры (Formatters)

Форматоры передают ссылку на метаданные и определяют как объекта сериализовать. Метод *Serialize* использует функцию reflection, и узнает все об объектах. Если сериализуемый объект передает ссылку на другой объект, то так же для этого объекта определяется путь сериализования. Форматоры имеют умный алгоритм. На одной стриме(stream) смогут сериализовать больше одного объекта. Если оба объекта имеют точно такие же свойства (оба объекта передают ссылку в тот же адрес) то, форматор(formatter) это понимает и только сериализует один объект. В процессе сериализации форматор пишет полный пространство имен

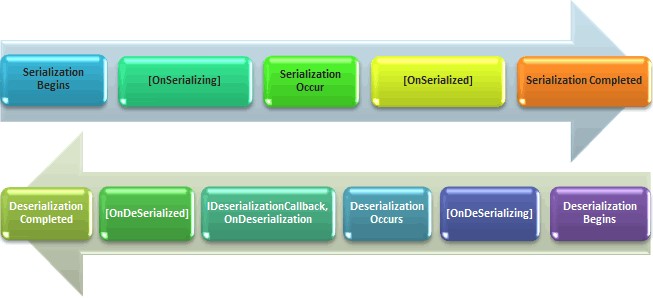
(MyNamespace.MySubNamespace.MyClass и т.п.) и в какой ассамблеи находится тип (MyAssembly.dll и т.п.). По умолчанию BinaryFormatter и SoapFormatter создает выходные данные ассамблеи (name, version, public key, and culture). Если нужно узнать только в какой ассамблеи находится объект то, меняем *AssemblyFormat*на *FormatterAssemblyStyle.Simple.*

*(FormatterAssemblyStyle.Full*выбран по умолчанию)

3.5 События бинарных форматоров и сериализация

События сериализации (events) появились с .Net Framework 2.0. Т.е.

когда реализуем класс, можем добавить события Serializing, Serialized, Deserializing, Deserialized, используя пометки.



# Рисунок 4 - События сериализации

События *OnSerializing, OnSerialized, OnDeserializing* и *OnDeserialized* работают только для бинарной сериализации. *С Soap* сериалией работают только *serialization events.*

Пример для события сериализации и десериализации:

*[Serializable]*

*public class MyClass*

*{*

*[OnSerializing]*

*void OnSerializing(StreamingContext context)*

*{ }*

*[OnSerialized]*

*void OnSerialized(StreamingContext context)*

*{ }*

*[OnDeserializing]*

*void OnDeserializing(StreamingContext context)*

*{ }*

*[OnDeserialized] void OnDeserialized(StreamingContext context)*

*{ }}*

1. Разработка проекта на C#

4.1 Анализ сайтов

Для разработки программного агента рассмотрим сайты сети Интернет, содержащие в открытом доступе большое количество публикаций по различным областям знаний. Нас будут интересовать только публикации по дисциплинам математика и компьютерные науки.

Для поиска необходимых статей нужно осуществить парсинг данных сайтов. Парсинг сайтов состоит из двух частей:

1. Найти ссылки на интересующие нас страницы.
2. Выбрать из этих страниц необходимую информацию.

Для работы создаем объект *hweb* класса *HtmlWeb* для получения *Html*документа из *Http:*

HtmlAgilityPack.HtmlWeb hweb = new HtmlAgilityPack.HtmlWeb(); Для выборки необходимой информации будем использовать язык XPath. Необходимо составить правило для выборки ссылок на нужные нам страницы. Например, это элементы div с атрибутом class равным b-photo:

1. Для начала: //
2. Теперь надо указать, что выбираются элементы: //div
3. Но нужно конкретизировать, какие элементы мы выбираем: //div[]
4. Нужны элементы div с атрибутом class равным b-photo:

//div[@class=' b-title ']

Здесь использовались следующие оси языка *XPath*: *attribute*:: — Возвращает множество атрибутов текущего элемента (сокращенно - @).

*descendant-or-self::* — Возвращает полное множество потомков и

текущий элемент (сокращенно - //).

Так же язык *XPath* имеет функции для работы с множествами:

[] — дополнительные условия выборки;

/ — определяет уровень дерева.

Создаем объект gelHDoc класса HtmlAgilityPack.HtmlDocument и коллекцию узлов (HtmlNodeCollection) и будем получать необходимые нам данные со страниц сайта:

HtmlAgilityPack.HtmlNodeCollection gelenCollection = gelHDoc.DocumentNode.SelectNodes(@"//span[@class='heading-text']");

У объекта *gelHDoc* есть свойство *DocumentNode* (указывает на верхний узел документа). У него же в свою очередь есть методы *SelectNodes* и *SelectSingleNode*. Первый выбирает коллекцию элементов, а второй - только

один. Нам нужен первый метод.

Метод *SelectNodes* может вернуть null, если не будет найдено элементов.

Поэтому необходимо сделать проверку if (gelenCollection != null).

Далее в цикле обрабатываем каждый элемент коллекции:

foreach (HtmlAgilityPack.HtmlNode gelenItem in gelenCollection)

Полный код метода будет выглядеть следующим образом: public void ilanLinkIslem(HtmlAgilityPack.HtmlDocument

gelHDoc)

{

// Получаем список ссылок на страницы с недвижимостью

HtmlAgilityPack.HtmlNodeCollection gelenCollection = gelHDoc.DocumentNode.SelectNodes(@"//span[@class='heading-text']") if (gelenCollection != null)

{

// Обрабатываем каждую страницу (парсим из нее выбранные данные) foreach (HtmlAgilityPack.HtmlNode gelenItem in

gelenCollection) {

if (gelenItem.InnerHtml.ToString() != null ||!string.IsNullOrWhiteSpace(gelenItem.InnerHtml.ToString())||!string.

IsNullOrEmpty(gelenItem.InnerHtml.ToString())|| gelenItem.InnerHtml.ToString()!="")

{ try

{

|  |  |  |
| --- | --- | --- |
| string  gelenItem.InnerHtml.ToString(); | gelenKaynak | = |
| gelenKaynak |  | = |

gelenKaynak.Remove(gelenKaynak.IndexOf("title"), gelenKaynak.Length - gelenKaynak.IndexOf("title")); gelenKaynak = gelenKaynak.Remove(0,

gelenKaynak.IndexOf("href")); gelenKaynak = gelenKaynak.Replace("href=",

"").Replace("\"", "");

// Загружаем страничку сайта

Uri uriSon = new

Uri("http://research.google.com/" + gelenKaynak);

ilanUrlleri.Add(uriSon);

}

catch (ArgumentOutOfRangeException)

{

continue;

}

}

Для метода *SelectNodes* используем свойство *InnerHtml*. Оно вернет чистый текст без тегов. Если нужно получить html-код найденной таблицы, необходимо использовать свойство *OuterHtml*.

Парсинг сайта будет идти в отдельном потоке. Для поддержания многопотоности необходимо подключить пространство имен:

*using* *System*.*Threading*

Для ожидания времени обработки данных используем метод Sleep: *Thread*.*Sleep*(600);

4.2 Пример работы программы

Разработанное программное приложение позволяет осуществлять поиск научных статей на русском и английском языках по дисциплинам математика и компьютерные науки, которые представлены в открытом доступе.

Для примера поиска научный статей на русском языке был рассмотрен сайт [http://cyberleninka.ru](http://cyberleninka.ru/)  научной библиотеки открытого доступа

«Киберленинка». Данная библиотека имеет более 1 миллиона научных статей в различных областях знаний. Библиотека предоставляет бесплатный, быстрый, постоянный, полнотекстовый доступ к научным и учебным материалам в режиме реального времени. Материалы предоставляются по лицензии Creative Commons Attribution (CC-BY), позволяющей неограниченно использовать произведения при условии указания авторства.

Для поиска статей на английском языке использовался сайт [http://research.google.com](http://research.google.com/) сайт издательства Google, которое публикует сотни научных работ каждый год.

Разработанное программное приложение имеет удобный для пользователей интерфейс.

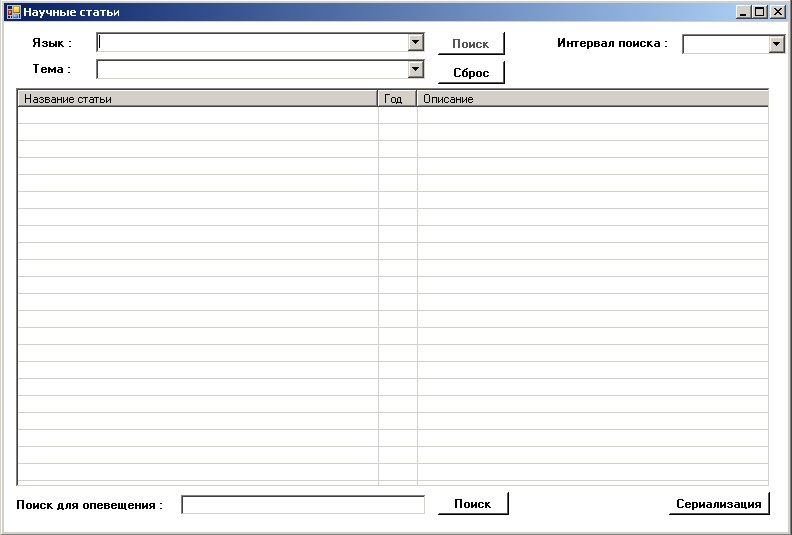
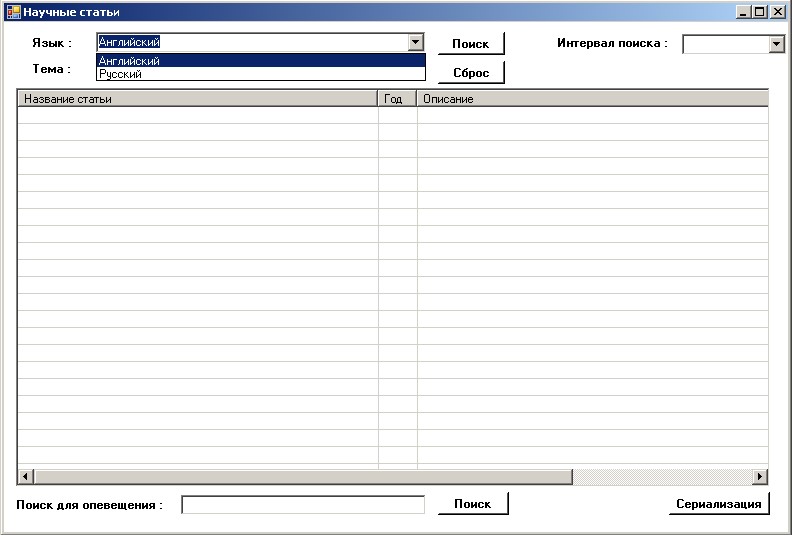


Рисунок 5 - Интерфейс приложения.

Для начала работы можно выбрать язык поиска научных статей:

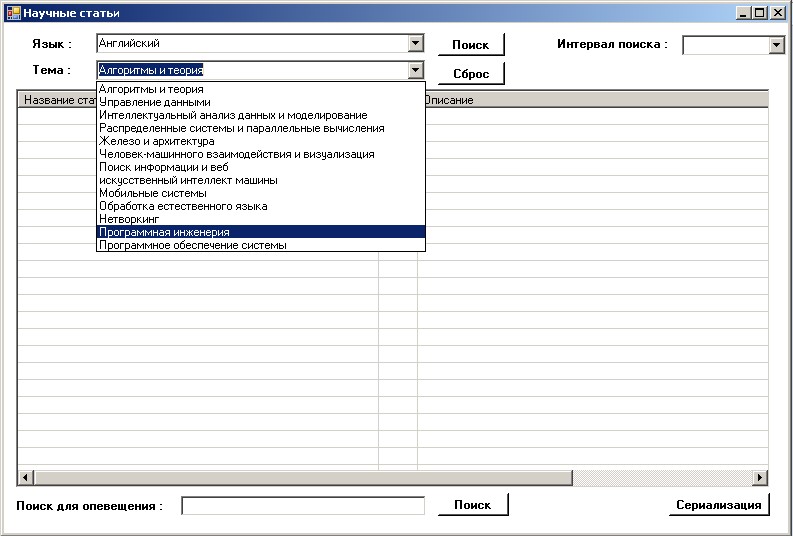
русский или английский.



# Рисунок 6 - Выбор языка

Затем выбирается тема научных публикаций. Программное приложение проанализирует сайты на наличие нужных статей, и результаты поиска выведет на экран. Мы можем задать интервал времени, по прошествии которого, программа самостоятельно, без повторного запуска пользователем будет осуществлять новый поиск.

Для сохранения найденных статей в таблицу базы данных в разные моменты времени применяется сериализация.



# Рисунок 7 - Выбор темы

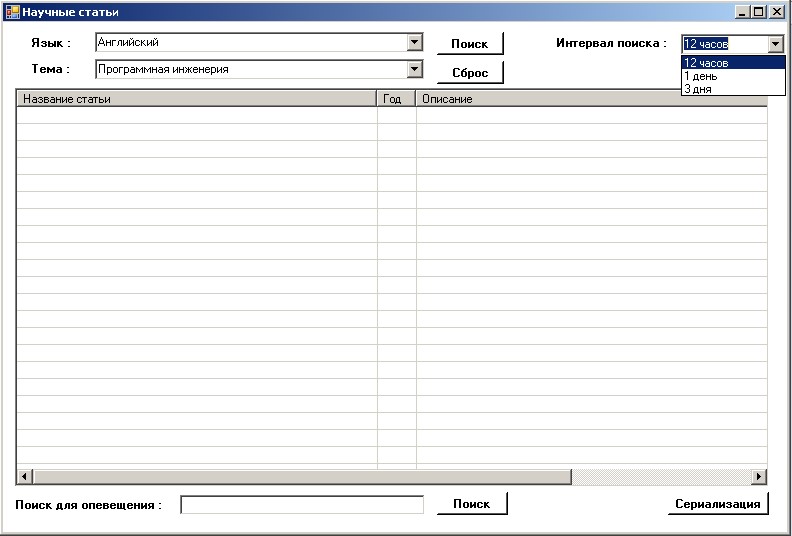


Рисунок 8 - Выбор интервала времени для поиска

После того как пользователь выберет, язык на котором ищет статьи и тематику, при нажатии кнопки «Поиск», получит перечень всех научных статей по дисциплинам математика и компьютерные науки, размещенных на анализируемых сайтах. Если через некоторое время на сайтах появятся новые статьи, программа автоматически добавит их в таблицу и выведет сообщение «Появилась новая статья».

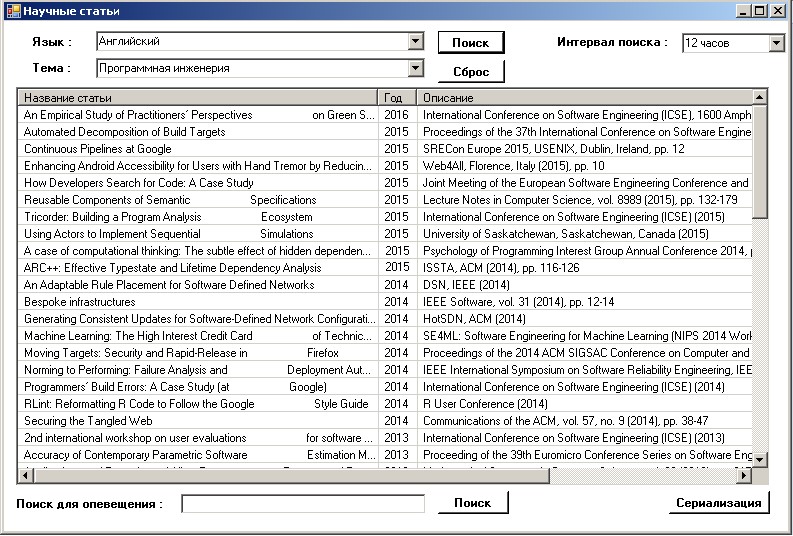
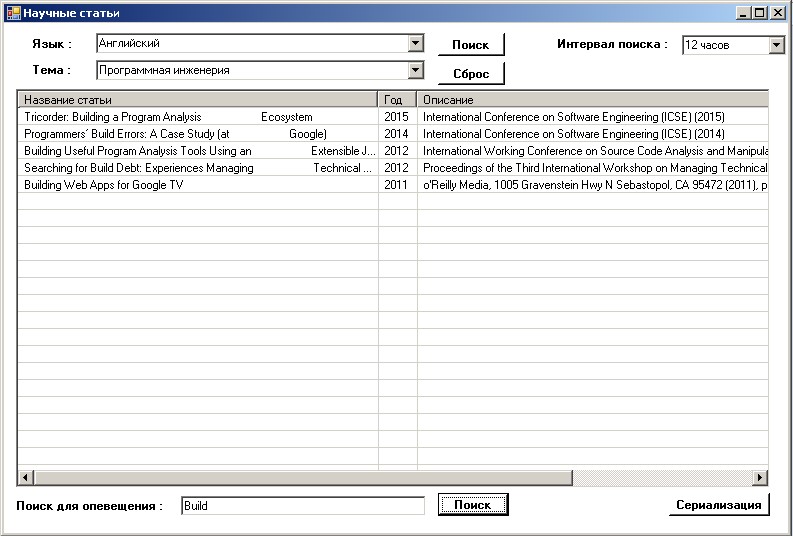


Рисунок 9 - Результаты работы программы

Можно выбрать нужную статью и открыть его. Мы также можем провести поиск по выбранным статьям, задав необходимые критерии поиска. Задав, например, «Building», получим, список статей, содержащих «классификация» (рисунок 8). И если появится статья которая нас интересует, то получим оповещение в нижем левом углу. (рисунок 11)



# Рисунок 10 - Поиск для оповещения

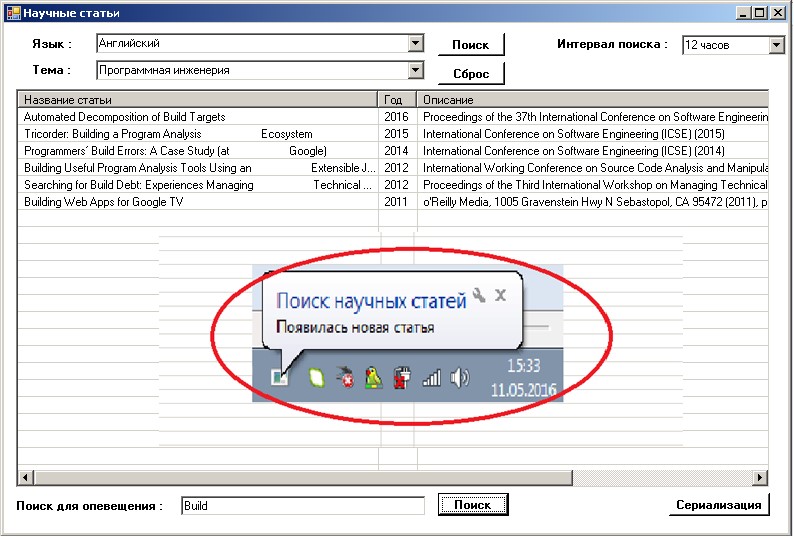


Рисунок 11 - Оповещения

# ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы был получен готовый к применению программный продукт. Были достигнуты поставленные на этапе разработки цели.

Результат работы – разработанный автономный агент в форме программного приложения, который проводит анализ популярных сайтов, содержащих в открытом доступе научно-исследовательские статьи на русском и английском языках по дисциплинам математики и компьютерные науки. Программное приложение производит поиск статей по заданным характеристикам и сохраняет их в базу данных. При появлении новой публикации программный агент добавляет ее в список сохраненных публикаций.

Программа позволяет осуществлять быстрый поиск информации в сети интернет и ее анализ: выбор только той информации, которая удовлетворяет заданным критериям.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Т. А. Гаврилова, В. Ф. Хорошевский, Базы знаний интеллектуальных систем. — СПб: Питер, 2000. — 384 с.
2. Библиотека HtmlAgilityPack [Электронный ресурс] URL: <http://htmlagilitypack.codeplex.com/>
3. Языки информационного обмена. Курс лекций. [Электронный ресурс] URL: <http://www.intuit.ru/studies/courses/1176/186/info>
4. С#. Уроки программирования с нуля. [Электронный ресурс] URL: <http://mycsharp.ru/>
5. Акимов С.В. Технологии Internet / Intranet в почтовой связи: учебное пособие / СПбГУТ. СПб, 2005
6. DOM – объектная модель документа. [Электронный ресурс] URL: <http://structuralist.narod.ru/it/internet/dom.htm>
7. Document Object Model Core. [Электронный ресурс] URL: <http://www.w3.org/TR/DOM-Level-2-Core/core.html>
8. XPath (XML Path Language) — язык запросов к элементам XML-

документа. [Электронный ресурс] URL: <https://ru.wikipedia.org/wiki/XPath>

1. С# Tutorial (Simple easy learning by tutorialspoint.com)
2. Her yönüyle C# (Sefer ALGAN) (турецкая литература)
3. Parsing Techniques-A Practical Guide / Dick Grune and Cerial J.H.Jacobs

Originally published by Ellis Horwood, Chichester, England, 1990; ISBN 0 13 651431 6

1. Progressive Parsing Book/Allen Hayden Weld
2. Easily Parse HTML Documents in C# [Электронный ресурс]

URL[:http://olussier.net/2010/03/30/easily-parse-html-documents-in-csharp/](http://olussier.net/2010/03/30/easily-parse-html-documents-in-csharp/)

1. Html Agility Pac[k](https://htmlagilitypack.codeplex.com/)  [Электронный ресурс]

URL:<https://htmlagilitypack.codeplex.com/>

ПРИЛОЖЕНИЕ А

Код программы

using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Linq; using System.Text; using System.Windows.Forms; using System.Threading; using System.Net; using System.Collections; using System.Text.RegularExpressions; using System.Windows; using System.Xml; using HtmlAgilityPack; using System.IO; using System.Xml.XPath; using System.Web; using System.Net.Mail; using System.IO.Compression; using System.Runtime.InteropServices; using System.Data.Sql; using System.Data.SqlClient;

using AvitoIlanProje1.Properties;

namespace Avito

{

public partial class MainForm : Form

{

public class DataSource

{

public string Name { get; set; } public string Value { get; set; }

}

private class Article

{

/// <summary>

/// Заголовок /// </summary> public String Title { set; get; } /// <summary>

/// Год

/// </summary>

public Int32 Year { set; get; }

/// <summary>

/// Описание /// </summary>

public String Description { set; get; }

/// <summary>

/// Ссылка /// </summary>

public Uri URL { set; get; }

}

private List<Article> parsedURLs = new List<Article>(); private List<Article> allParsedURLs = new List<Article>();

private String Link = String.Empty;

/// <summary>

/// Соединение с базой данных

/// </summary>

private SqlConnection connection = new

SqlConnection(Settings.Default.recordsConnectionString);

/// <summary>

/// Защите соединения с базой

/// </summary>

private Mutex mutex = new Mutex();

/// <summary>

/// Конструктор класса по умолчанию

/// </summary>

///

List<DataSource> list;

public MainForm()

{

list=new List<DataSource>();

InitializeComponent();

// открываем соединение с базой try

{

// connection.Open();

}

catch (System.Exception ex)

{

MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Составление списка тем

/// </summary>

/// <param name="document">Документ, откуда парсить ссылки</param> private void parseThemesCyberlinka(HtmlAgilityPack.HtmlDocument document)

{

HtmlNodeCollection themes =

document.DocumentNode.SelectNodes(@"//span[@class='heading-text']"); HtmlNodeCollection years =

document.DocumentNode.SelectNodes(@"//span[@class='num']"); HtmlNodeCollection texts =

document.DocumentNode.SelectNodes(@"//div[@class='text-inner']");

Int32 index = 0;

if (themes != null)

{

foreach (HtmlNode theme in themes)

{

if (theme.InnerHtml.ToString() != null ||

!String.IsNullOrWhiteSpace(theme.InnerHtml.ToString()) ||

!String.IsNullOrEmpty(theme.InnerHtml.ToString()) || theme.InnerHtml.ToString() != "")

{ try

{

// ссылка

String source = theme.InnerHtml.ToString();

source = theme.FirstChild.GetAttributeValue(@"href", ""); Uri uriSon = new Uri(@"http://cyberleninka.ru" + source);

// заголовок

String title = theme.FirstChild.InnerHtml.ToString();

// описание

String desription = texts[index].InnerText.Trim();

// год

Int32 year =

Int32.Parse(years[index].InnerText.Replace(@"Год", String.Empty));

// добавление

parsedURLs.Add(new Article

{

Title = title,

Year = year,

Description = desription,

URL = uriSon

});

}

catch (ArgumentOutOfRangeException)

{

continue;

} finally

{

++index;

}

}

}

}

}

/// <summary>

/// Составление списка тем

/// </summary>

/// <param name="document">Документ, откуда парсить ссылки</param> private void parseThemesGoogle(HtmlAgilityPack.HtmlDocument document)

{

HtmlNodeCollection themes =

document.DocumentNode.SelectNodes(@"//p[@class='pub-title']"); HtmlNodeCollection links =

document.DocumentNode.SelectNodes(@"//a[@class='pdf-icon tooltip']");

Int32 index = 0;

if (themes != null)

{

foreach (HtmlNode theme in themes)

{

if (theme.InnerHtml.ToString() != null ||

!String.IsNullOrWhiteSpace(theme.InnerHtml.ToString()) ||

!String.IsNullOrEmpty(theme.InnerHtml.ToString()) || theme.InnerHtml.ToString() != "")

{ try

{

// ссылка

String source = links[index].GetAttributeValue(@"href",

String.Empty);

if (!source.StartsWith(@"http"))

source =

@"http://static.googleusercontent.com/media/research.google.com/en//" + source;

Uri uriSon = new Uri(source);

// заголовок

String title = theme.InnerText.Trim();

// описание

String desription =

theme.NextSibling.NextSibling.NextSibling.NextSibling.InnerText.Trim();

// год

Int32 year = 0;

if (theme.InnerHtml.Contains(@"<a"))

{

String link =

theme.FirstChild.NextSibling.GetAttributeValue(@"href", String.Empty);

HtmlWeb web = new HtmlWeb();

web.UserAgent = "Mozilla/5.0 (Windows NT 6.2; WOW64; rv:19.0) Gecko/20100101 Firefox/19.0";

HtmlAgilityPack.HtmlDocument hDoc = web.Load(@"http://research.google.com" + link);

//maia-col-4 String text =

hDoc.DocumentNode.SelectNodes(@"//div[@class='sidebar']")[0].InnerHtml;

Int32 start = text.IndexOf(@"Publication Year"); start = text.IndexOf(@"<p>", start); text = text.Substring(start + 3);

Int32 end = text.IndexOf(@"</p>");

text = text.Substring(0, end).Trim(); year = Int32.Parse(text);

}

// добавление

parsedURLs.Add(new Article

{

Title = title,

Year = year,

Description = desription,

URL = uriSon

});

}

catch (ArgumentOutOfRangeException)

{

continue;

} finally

{

++index;

}

}

}

}

}

/// <summary>

/// Начать парсинг /// </summary>

private void searchbutton1\_Click(object sender, EventArgs e)

{

new Thread(

delegate ()

{

//<a class="photo-wrapper" HtmlWeb hWeb = new HtmlWeb();

hWeb.UserAgent = "Mozilla/5.0 (Windows NT 6.2; WOW64; rv:19.0) Gecko/20100101 Firefox/19.0";

HtmlAgilityPack.HtmlDocument hDoc = hWeb.Load(Link);

// скачать страницу и распарсить

if (Link.StartsWith(@"http://cyberleninka.ru")) parseThemesCyberlinka(hDoc);

else if (Link.StartsWith(@"http://research.google.com")) parseThemesGoogle(hDoc); else {

MessageBox.Show(@"Ошибка. Неизвестный сайт"); return;

}

foreach (Article a in parsedURLs)

{

{

ListViewItem item = new ListViewItem();

// название темы из краткой страницы

item.Text = a.Title;

// описание из полной страницы item.SubItems.Add(a.Year.ToString());

// получение ссылки на картинку из краткого описания

item.SubItems.Add(a.Description);

// сама ссылка на тему

item.SubItems.Add(a.URL.AbsoluteUri);

// добавление информации в базу

//if (AddToDatabase(item.Text, item.SubItems[1].Text,

// item.SubItems[2].Text,

// item.SubItems[3].Text,

// item.SubItems[4].Text)) //{

// добавление в GUI

Invoke((MethodInvoker)delegate()

{

listView1.Items.Add(item);

});

notifyIcon1.Icon = SystemIcons.Application; notifyIcon1.BalloonTipTitle = "Поиск научных статей"; notifyIcon1.BalloonTipText = "Появилась новая статья"; notifyIcon1.ShowBalloonTip(1000);

//}

}

}

allParsedURLs.AddRange(parsedURLs); parsedURLs.Clear();

if (!processTimer.Enabled)

processTimer.Enabled = true;

}).Start();

}

/// <summary>

/// Добавление в базу с проверкой

/// </summary>

public bool AddToDatabase(String name, String description,

String image, String link, String cost)

{

bool exists = false;

mutex.WaitOne();

using (SqlCommand command = new SqlCommand())

{

command.CommandText = @"select count(link) from records where link = @link";

command.Connection = connection;

command.Parameters.Add(@"@link", SqlDbType.NVarChar).Value = link;

exists = Convert.ToBoolean(command.ExecuteScalar());

if (!exists)

{

command.CommandText = @"insert into records ([link], [name],

[description], [cost], [image])

values (@link, @name, @desc, @cost, @image)"; command.Parameters.Add(@"@name", SqlDbType.NVarChar).Value

= name;

command.Parameters.Add(@"@desc", SqlDbType.NVarChar).Value

= description;

command.Parameters.Add(@"@year", SqlDbType.NVarChar).Value = image;

try

{

command.ExecuteNonQuery();

}

catch (Exception ex)

{

}

}

}

mutex.ReleaseMutex();

return !exists;

}

/// <summary> /// Поиск по шаблону /// </summary>

private void button2\_Click(object sender, EventArgs e)

{

string searchedQuery = textBox2.Text;

// очищение списка listView1.Items.Clear();

foreach (Article a in allParsedURLs)

{

if (a.Title.Contains(searchedQuery))

{

ListViewItem item = new ListViewItem();

// название темы из краткой страницы

item.Text = a.Title;

// описание из полной страницы item.SubItems.Add(a.Year.ToString());

// получение ссылки на картинку из краткого описания

item.SubItems.Add(a.Description);

// сама ссылка на тему

item.SubItems.Add(a.URL.AbsoluteUri);

listView1.Items.Add(item);

}

}

//mutex.WaitOne();

//using (SqlCommand command = new SqlCommand())

//{

// command.Connection = connection;

// command.CommandText = String.Format(@"select \* from records where name like '%{0}%'", searchedQuery);

// SqlDataReader reader = command.ExecuteReader();

// while (reader.Read())

// {

// String name = reader[@"name"].ToString();

// String description = reader[@"description"].ToString();

// String cost = reader[@"cost"].ToString();

// String image = reader[@"image"].ToString();

// String link = reader[@"link"].ToString();

// ListViewItem item = new ListViewItem(name);

// item.SubItems.Add(description);

// item.SubItems.Add(image);

// item.SubItems.Add(link);

// item.SubItems.Add(cost);

// listView1.Items.Add(item);

// }

// reader.Dispose();

//}

//mutex.ReleaseMutex();

}

/// <summary>

/// Открыть в браузере /// </summary>

private void button3\_Click(object sender, EventArgs e)

{

if (listView1.SelectedItems.Count > 0)

{

String linkToOpen = listView1.SelectedItems[0].SubItems[3].Text; System.Diagnostics.Process.Start(linkToOpen);

}

}

/// <summary>

/// Заново поискать новые ссылки

/// </summary>

private void timer1\_Tick(object sender, EventArgs e)

{

searchbutton1\_Click(sender, e);

// serializationbutton(sender, e);

}

/// <summary>

/// Закрываем соединение с базой

/// </summary>

private void MainForm\_FormClosing(object sender, FormClosingEventArgs e)

{ try

{

mutex.WaitOne(); connection.Close();

}

catch (System.Exception ex)

{ } finally

{

mutex.ReleaseMutex();

}

}

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

searchbutton1.Enabled = comboBox1.SelectedIndex != -1;

if (comboBox1.SelectedIndex == 0)

{

list = new List<DataSource>();

list.Add(new DataSource() { Name = "Алгоритмы и теория", Value

= "http://research.google.com/pubs/AlgorithmsandTheory.html" });

list.Add(new DataSource() { Name = "Управление данными", Value

= "http://research.google.com/pubs/DataManagement.html" });

list.Add(new DataSource() { Name = "Интеллектуальный анализ данных и моделирование", Value =

"http://research.google.com/pubs/DataMiningandModeling.html" }); list.Add(new DataSource() { Name = "Распределенные системы и параллельные вычисления", Value =

"http://research.google.com/pubs/DistributedSystemsandParallelComputing.html"

});

list.Add(new DataSource() { Name = "Железо и архитектура", Value

= "http://research.google.com/pubs/HardwareandArchitecture.html" }); list.Add(new DataSource() { Name = "Человек-машинного взаимодействия и визуализация", Value = "http://research.google.com/pubs/HumanComputerInteractionandVisualization.html" });

list.Add(new DataSource() { Name = "Поиск информации и веб", Value = "http://research.google.com/pubs/InformationRetrievalandtheWeb.html"

});

list.Add(new DataSource() { Name = "искусственный интеллект машины", Value = "http://research.google.com/pubs/MachineIntelligence.html"

});

list.Add(new DataSource() { Name = "Мобильные системы", Value

= "http://research.google.com/pubs/MobileSystems.html" });

list.Add(new DataSource() { Name = "Обработка естественного языка", Value =

"http://research.google.com/pubs/NaturalLanguageProcessing.html" }); list.Add(new DataSource() { Name = "Нетворкинг", Value =

"http://research.google.com/pubs/Networking.html" });

list.Add(new DataSource() { Name = "Программная инженерия", Value = "http://research.google.com/pubs/SoftwareEngineering.html" }); list.Add(new DataSource() { Name = "Программное обеспечение системы", Value = "http://research.google.com/pubs/SoftwareSystems.html" }); comboBox2.DataSource = list; comboBox2.DisplayMember = "Name";

comboBox2.ValueMember = "Value";

}

else if (comboBox1.SelectedIndex == 1)

{

list = new List<DataSource>();

list.Add(new DataSource() { Name = "Математика", Value =

"http://cyberleninka.ru/article/c/matematika" });

list.Add(new DataSource() { Name = "Физика", Value =

"http://cyberleninka.ru/article/c/fizika" });

list.Add(new DataSource() { Name = "Кибернетика", Value =

"http://cyberleninka.ru/article/c/kibernetika" });

list.Add(new DataSource() { Name = "Информатика", Value =

"http://cyberleninka.ru/article/c/informatika" });

list.Add(new DataSource() { Name = "Электроника-Радиотехника",

Value = "http://cyberleninka.ru/article/c/elektronika-radiotehnika" }); comboBox2.DataSource = list; comboBox2.DisplayMember = "Name"; comboBox2.ValueMember = "Value";

// Link = @"http://cyberleninka.ru/article/c/matematika";

}

}

private void listView1\_SelectedIndexChanged(object sender, EventArgs e)

{

}

private void notifyIcon1\_MouseDoubleClick(object sender, MouseEventArgs e)

{

}

private void comboBox2\_SelectedIndexChanged\_1(object sender, EventArgs e)

{

Link =@comboBox2.SelectedValue.ToString();

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void button3\_Click\_1(object sender, EventArgs e)

{

System.Diagnostics.Process.Start(Application.ExecutablePath); // to start new instance of application

this.Close(); //to turn off current app

}

private void comboBox3\_SelectedIndexChanged(object sender, EventArgs e)

{

comboBox3.DisplayMember = "Text";

comboBox3.ValueMember = "Value";

var items = new[] { new { Text = "12 часов", Value = "reportA" }, new { Text = "1 день", Value = "reportB" }, new { Text = "3 дня", Value = "reportC" }

};

comboBox3.DataSource = items;

}

}}