Ссылки

С1#:

[**Основы С**](#ОсновыС)**#**

**...........**[**File**](#File)

**...........**[**Коллекции**](#Коллекции)

**...........**[**Свойства**](#Свойства)

[**ADO.NET**](#ADONET)

[LINQ](#LINQ)

[**System Programming**](#SystemProgramming)

[WCF](#WCF)

[**WPF**](#WPF)

WEB:

[HTML](#HTML)

[CSS](#CSS)

[JavaScript](#JavaScript)

.....[jQuery](#jQuery)

CrossPlatform:

[**Pattern**](#Pattern)

[Cross](#Cross)

[Math](#Math)

[**GIT**](#GIT)

[**XP Programming**](#ExtremeProgramming)

[**Тестирование Программного Обеспечения**](#ТестированиеПрограмногоОбеспечения)

[**UML**](#UML)

Unity:

[**Unity**](#Unity)

Sites:

Тесты :

indiabix.com

Собеседование:

http://www.isd.dp.ua/ru/career-ru/recruitment-procedure-ru/questions-example-ru/the-list-of-potential-questions-that-may-be-asked-during-the-interview-for-c-sharp-developer-position-ru.html

* Основы С#

struct- это тип значений, который обычно используется для формирования связных значений в простейшую группу (int , double, etc).

Использовать стоит только при создании какой ни будь математической формулы(к примеру поиск координат, площадь и проче), какого-то аналога Int или Double.

this - внутрення ссылка на себя. При создании любого класса в this сохраняется ссылка, на выделенную память.

OOП

6 парадигм ООП:

1. Инкапсуляция

2. Наследование

3. Полиморфизм

4. Абстракция

5. Посылка сообщений

6. Повторное использование

Инкапсуляция

Объединение данных и методов по работе с ЭТИМИ данными внутри ОДНОГО ОБОБЩЕННОГО класса. Основное отличие от структуры - это модификаторы доступа (в С# их 5!!!).

public - доступный из вне всем! Не рекомендуется!

private - не доступен из вне, доступен только в объявленном классе, доступ возможен через свойства. Рекомендуется!!!

protected - не доступен из вне, но доступен Наследникам класса!

internal - доступен только внутри this сборки!

protected internal -

Защищенный внутренний уровень доступности означает "защищенный OR внутренний", а не "защищенный AND внутренний". Другими словами, доступ к защищенному внутреннему члену может осуществляться из любого класса в одной сборке, в том числе из производных классов. Чтобы ограничить доступность только производными классами в одной и той же сборке, сам класс необходимо объявить внутренним, а его члены – защищенными.

Полиморфизм

http://habrahabr.ru/post/37576/

Позволяет использовать ссылку базового класса, поддерживающего реализацию, для определения (переопределения) объекта (метода)!!!

virtual - уведомляет что метод может быть переопределен

new - указывает что метод НЕ поддерживает полиморфизм.

override - указывает что метод поддерживает полиморфизм.

Без ключевого слова virtual метод так же может быть перегружен! Но без поддержки полиморфизма:

namespace ConsoleApplication2

{

class Program

{

static void Main(string[] args)

{

DerivedClass b = new DerivedClass();

Console.WriteLine(b.GetData());

}

}

class BaseClass

{

int N {get; set;}

public BaseClass(int n)

{

this.N = n;

}

public int GetData ()

{

return N;

}

}

class DerivedClass : BaseClass

{

public DerivedClass() : base(5) { }

public int GetData()

{

return 123;

}

}

}

Выведется 123

Но, если будет работа через ссылку на базовый класс. Компилятор будет думать что обращение идет к базовому классу, хотя реально там лежит класс наследника:

static void Main(string[] args)

{

BaseClass b = new DerivedClass();

Console.WriteLine(b.GetData());

}

Увидим 5

Для поддержки полиморфизма в данном случаем нужно использовать virtual в методе базового класса и override в классе наследнике.

class BaseClass

{

int N {get; set;}

public BaseClass(int n)

{

this.N = n;

}

public virtual int GetData ()

{

return N;

}

}

class DerivedClass : BaseClass

{

public DerivedClass() : base(5) { }

public override int GetData()

{

return 123;

}

}

Для обрыва цепочки полиморфизма вместо override нужно использовать new. Без override и new компилятор автоматически подставит new.

Наследование

позволяет создавать новое описание (новый класс) , включающий данные и методы базовому классу.

sealed - запечатанный класс, т.е. запрещает наследование

Перед инициализацией класса наследника, должен быть проинициализирован базовый класс. Если в базовом классе конструктор по умолчанию отсутствует, нужно использовать ключевое слово base при вызове конструктора наследника

namespace ConsoleApplication2

{

class Program

{

static void Main(string[] args)

{

DerivedClass b = new DerivedClass();

Console.WriteLine(b.GetData());

}

}

class BaseClass

{

int N;

public BaseClass(int n)

{

this.N = n;

}

public int GetData ()

{

return N;

}

}

class DerivedClass : BaseClass

{

public DerivedClass() : base(5) { }

}

}

Интерфейсы(Контракты)

набор некоторых свойств и методов, которые мы хотим чтобы были реализованы в классе , реализующий данный интерфейс.

Класс реализующий интерфейс ОБЯЗАН реализовывать методы и свойства интерфейса.

Ссылкой на интерфейс можно работать так же как и с ссылкой на базовый класс!

namespace ConsoleApplication2

{

class Program

{

static void Main(string[] args)

{

INameInterface b = new DerivedClass();

Console.WriteLine(((DerivedClass)b).GetData());

}

}

class BaseClass

{

Int32 N { get; set; }

public BaseClass(int n)

{

this.N = n;

}

public virtual int GetData ()

{

return N;

}

}

class DerivedClass : BaseClass, INameInterface, ICloneable

{

public DerivedClass() : base(5) { }

public override Int32 GetData()

{

return 123;

}

public Boolean num (int n )

{

if (n >= 0)

return true;

else

return false;

}

public object Clone()

{

throw new NotImplementedException();

}

}

interface INameInterface

{

Boolean num(Int32 n);

}

}

ПРОТОТИП ИНТЕРФЕЙСА:

Интерфейс реализует только прототипы методов!

1. В интерфейсе по умолчанию реализуются методы с public, т.к. они ДОЛЖНЫ быть реализованы.

2.В интерфейсе методы реализуются БЕЗ ДАННЫХ

3. Интерфейс это намеренье , никаких реализаций! Тело метода пустое.

Если реализовано два интерфейса с одинаково именованными методами, в классе нужно ЯВНО реализовать метод!

class DerivedClass : BaseClass, INameInterface, INameDiffInterface

{

public DerivedClass() : base(5) { }

public override Int32 GetData()

{

return 123;

}

public Boolean INameInterface.num (int n )

{

if (n >= 0)

return true;

else

return false;

}

public object Clone()

{

throw new NotImplementedException();

}

}

interface INameInterface

{

Boolean num(Int32 n);

}

interface INameDiffInterface

{

Boolean num(Int32 n);

}

Абстрактные классы

Не могут быть созданы экземпляры этого класса, хоть и абстрактный класс по сути, являться полноценным классом!

По сути абстрактный класс создается для наследования.

В абстрактном классе могут быть созданы абстрактные методы без реализации, которые ДОЛЖНЫ быть переопределены у наследника(с ключевым словом OVERRIDE).

namespace ConsoleApplication2

{

class Program

{

static void Main(string[] args)

{

INameInterface b = new DerivedClass();

Console.WriteLine(((DerivedClass)b).GetData());

}

}

abstract class BaseClass

{

Int32 N { get; set; }

public BaseClass(int n)

{

this.N = n;

}

public virtual int GetData ()

{

return N;

}

public abstract void AbstractMethod();

}

class DerivedClass : BaseClass, INameInterface

{

public DerivedClass() : base(5) { }

public override Int32 GetData()

{

return 123;

}

public Boolean num (int n )

{

if (n >= 0)

return true;

else

return false;

}

public object Clone()

{

throw new NotImplementedException();

}

public override void AbstractMethod()

{

}

}

interface INameInterface

{

Boolean num(Int32 n);

}

Свойства

prop и нажав Tab - автоматически сгенерирует свойство по умолчанию:

public int MyProperty { get; set; }

По сути своей, public методы позволяющие принимать и ОБРАБАТЫВАТЬ входные/выходные данные.

С помощью свойств можно инициализировать объекты вот таким образом:

namespace Testing

{

class Program

{

static void Main(string[] args)

{

FIO man = new FIO() { FirstName = "Dmytro", LastName = "Zepsen", Age = 26 };

FIO woman = new FIO() { FirstName = "Olya", LastName = "Kojevnikova", Age = 24 };

}

}

public class FIO

{

public string FirstName { get; set; }

public string LastName { get; set; }

public int Age { get; set; }

}

}

Атрибуты

по сути класс, наследуемый от System.Attribute которым можно ДОПОЛНИТЕЛЬНО описать информацию возможности класса, метода или сборки.

Сам по себе ничего не несет, просто расширяет таблицу метаданных. Которые мы хотим включить.

Значение атрибуты может быть только константа! Что означает что она не может изменяться во время выполнения!

Атрибуты:

Динамический поиск атрибутов:

System.Reflection.MemberInfo typeInfo;

typeInfo = typeof(Form1);

object[] att = typeInfo.GetCustomAttributes(false);

Индексаторы

Индексаторы оп сути, позволяют перегрузить [ ]. Только в этом случае нужно перегружать как get так и set свойства.

namespace Indexator

{

class Program

{

static void Main(string[] args)

{

Coords c = new Coords();

Console.WriteLine( c['h', 2]);

}

}

class Coords

{

public int this[Char x, Int32 y]

{

get

{

return "abcdefgh".IndexOf(x);

}

set

{

}

}

}

}

Индексатором может выступать любой тип(индексы можно даже придумывать самому )!

Делегаты

- ссылочный тип данных, который может хранить ссылку на метод с идентичной сигнатурой и вызывать его по ссылке.

Обобщенные делегаты

Action <T1,T2....Tn> - обобщенный делегат который ссылается на методы без выходного параметра (void).

Func<T1,T2..Tn, T out> - обобщенный делегат который ссылается на метод, с выходным параметром. Где T out - тип выходного параметра!

Работа с файлами

http://vbbook.ru/visual-c.net/rabota-s-faiylami-na-c/

Сериализация

Коллекции

foreach

в foreach нельзя добавлять / удалять элементы, пока он сформирован и в нем идет работа.

Использовать mutex для ожидания пока foreach закончит прохождение, во имя избежания ошибки.

Ключевые слова С#

yield - метод-итератор. При использование yield, метод, оператор или свойство становится итератором.

Обозначив yield return в коллекции, возвращает поочередно все элементы

yield break - прекращает возврат.

Пример: Возведение числа в степень!

Тип возвращаемого значения метода-итератора — [IEnumerable](https://msdn.microsoft.com/ru-ru/library/system.collections.ienumerable.aspx) (интерфейс перечислимого объекта), который является типом интерфейса итератора. Метод-итератор, будучи вызванным, возвращает перечислимый объект, содержащий степени числа.

namespace Testing

{

class Program

{

static void Main(string[] args)

{

foreach (var item in SQRT(2,10))

{

Console.WriteLine(item);

}

}

public static System.Collections.Generic.IEnumerable<int> SQRT(int n, int s)

{

int res = 1;

for (int i = 1; i < s; i++)

{

res = n \* res;

if (res > 100)

{

yield break; *// прекращает итерацию*

}

yield return res;

}

}

}

}

Пример показывающий использования yield для применения итерации get. Вызывает каждый экземпляр класса, до yield break!

namespace Testing

{

class Program

{

static void Main(string[] args)

{

Groups gr = new Groups();

foreach (var item in gr.NextStudent)

{

StringBuilder build = new StringBuilder(item.LastName + " " + item.FirstName + " " + item.Age);

Console.WriteLine(build);

}

}

}

public class Groups

{

public System.Collections.Generic.IEnumerable<Student> NextStudent

{

get

{

yield return new Student() { FirstName = "Dmytro", LastName = "Zepsen", Age = 26 };

yield return new Student() { FirstName = "Olya", LastName = "Kojevnikova", Age = 24 };

yield break;

yield return new Student() { FirstName = "1123", LastName = "123123", Age = 21234 };

}

}

}

public class Student

{

public string FirstName { get; set; }

public string LastName { get; set; }

public int Age { get; set; }

}

}

Параметры методов

ref - при передаче аргумента, аргумент передается по ССЫЛКЕ, а не по значению. Эффект от такой передачи в том, что изменения в передаваемом элементе непосредственно влияют на аргумент который передается!!!

Не следует путать понятие передачи по ссылке с понятием ссылочных типов. Эти два понятия не совпадают. Параметр метода может быть изменен с помощью ref независимо от того, принадлежит ли он к типу значения или ссылочному типу. При передаче по ссылке упаковка-преобразование типа значения не производится.

При использовании ref необходимо явно указать ключевое слово ref , как в определение так и при вызове метода!

Ключевые слова операторов

is - булевский оператор проверяющий принадлежит ли аргумент к заданному типу.

object a = new Game();

if (a is Game)

{

//true

}

as -

Плюшки

for (int i = 1,j = 1; j < BordSize; i++, j++)

* Windows

Comunication Foundation

(Cлужбы)

Служба WCF состоит из:

1. Класс службы – класс написанный на любом языке .NET и имеющий один или несколько методов.
2. Набор оконечных точек (endpoint)– взаимодействие службы и клиента
3. Процесс, в котором данная функция хостится. Можно использовать процесс другого приложения, процессы сервера и внедряться в саму операц. систему.

Класс службы определяет те методы, которые служба будет выполнять по запросу клиенту. Такие методы так же называются операциями. Для того чтобы указать что данный класс является службой, перед объявлением класса необходимо указать атрибут [ServiceContract] .

Перед теми методами, которые должны быть доступны клиентами, необходимо указать [OperationContact]

Набор методов, открытых клиенту, называется Контрактом Службы.

Вариант для связывания клиента

1. Копировать с сервера, интерфейс! И подключится через адрес. (Не расширяемо, но простой способ)
2. Создать .mex point (Корректный способ)

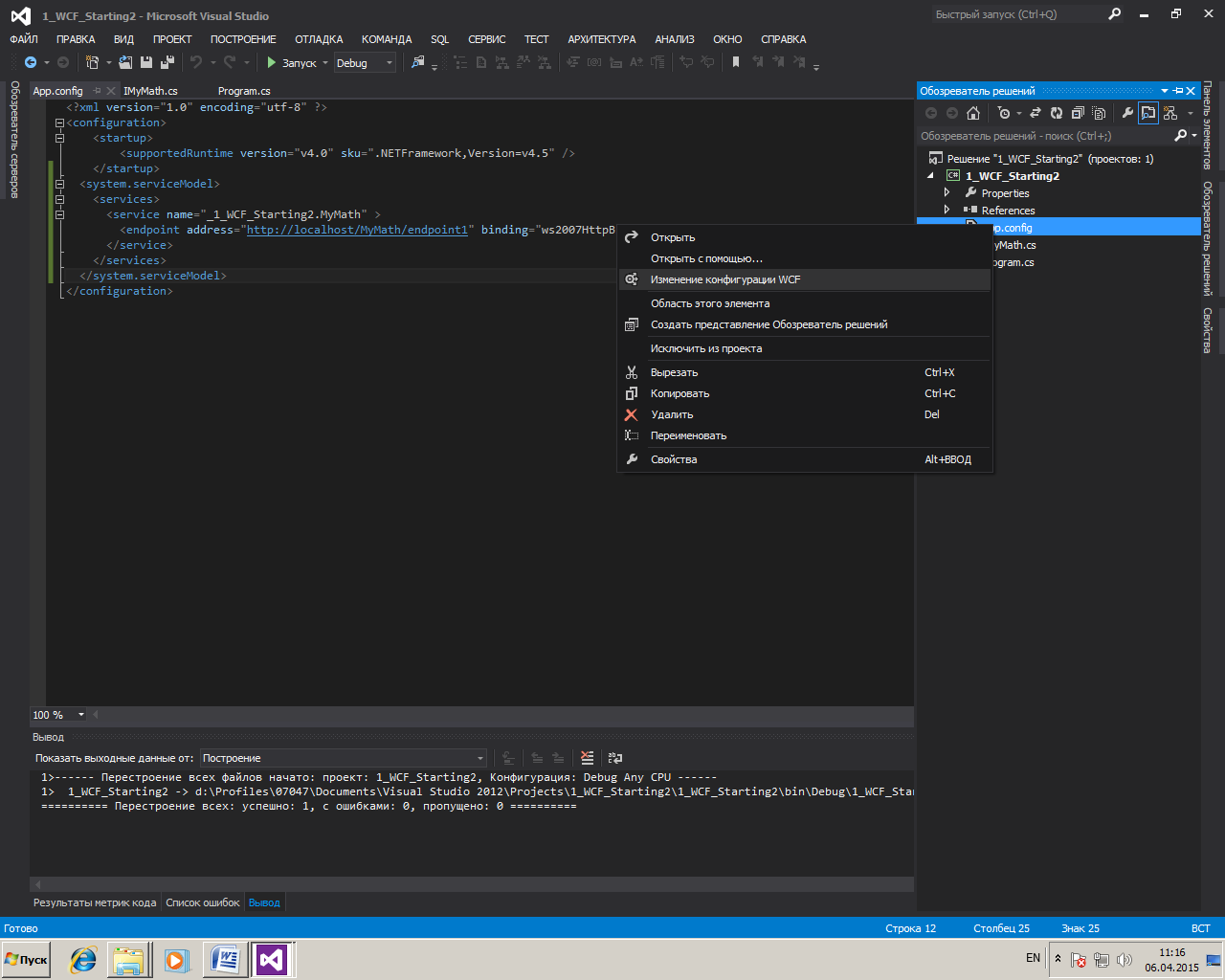
Служба может создать специальную конечную точку MEX POINT

Которая необходима для того, чтобы служба отправляла описание своего контракта при запросе.

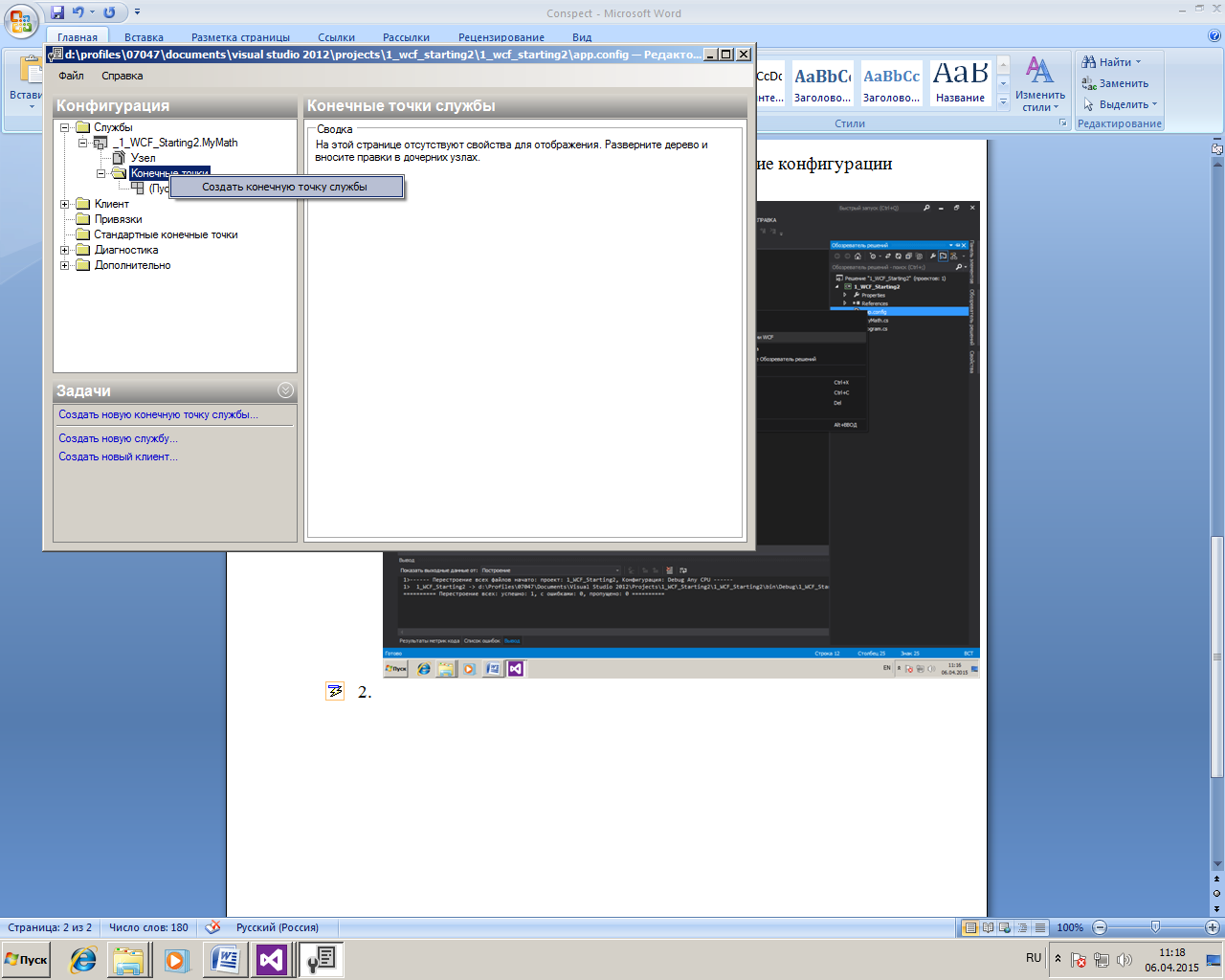
Клиенту необходимо знать только адрес MEX Point из этой информации будет создан класс, который является слепком(копией) Контракта.

Добавление MEX POINT

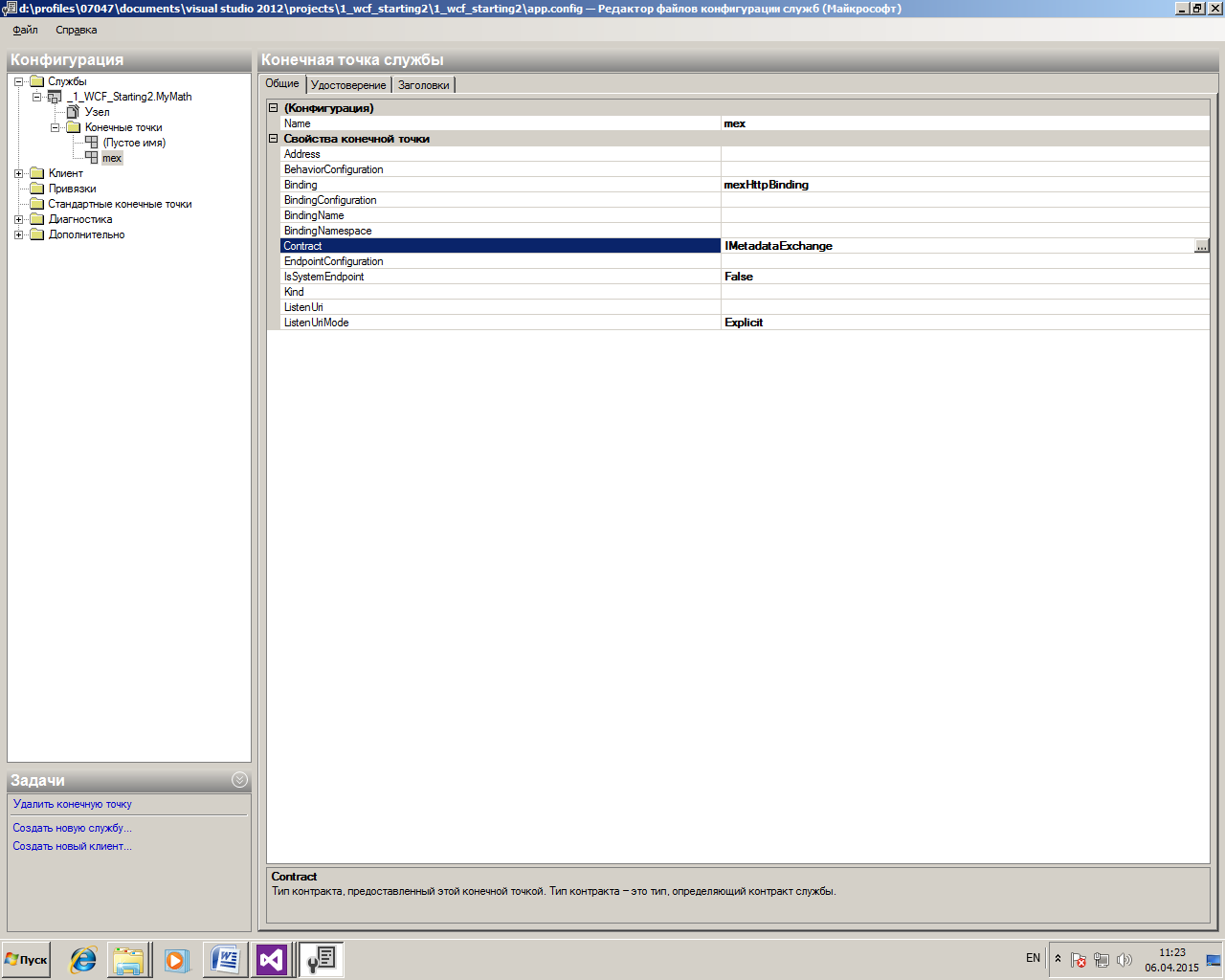
1. На сервере AppConfig/ правой кнопкой / изменение конфигурации WCF



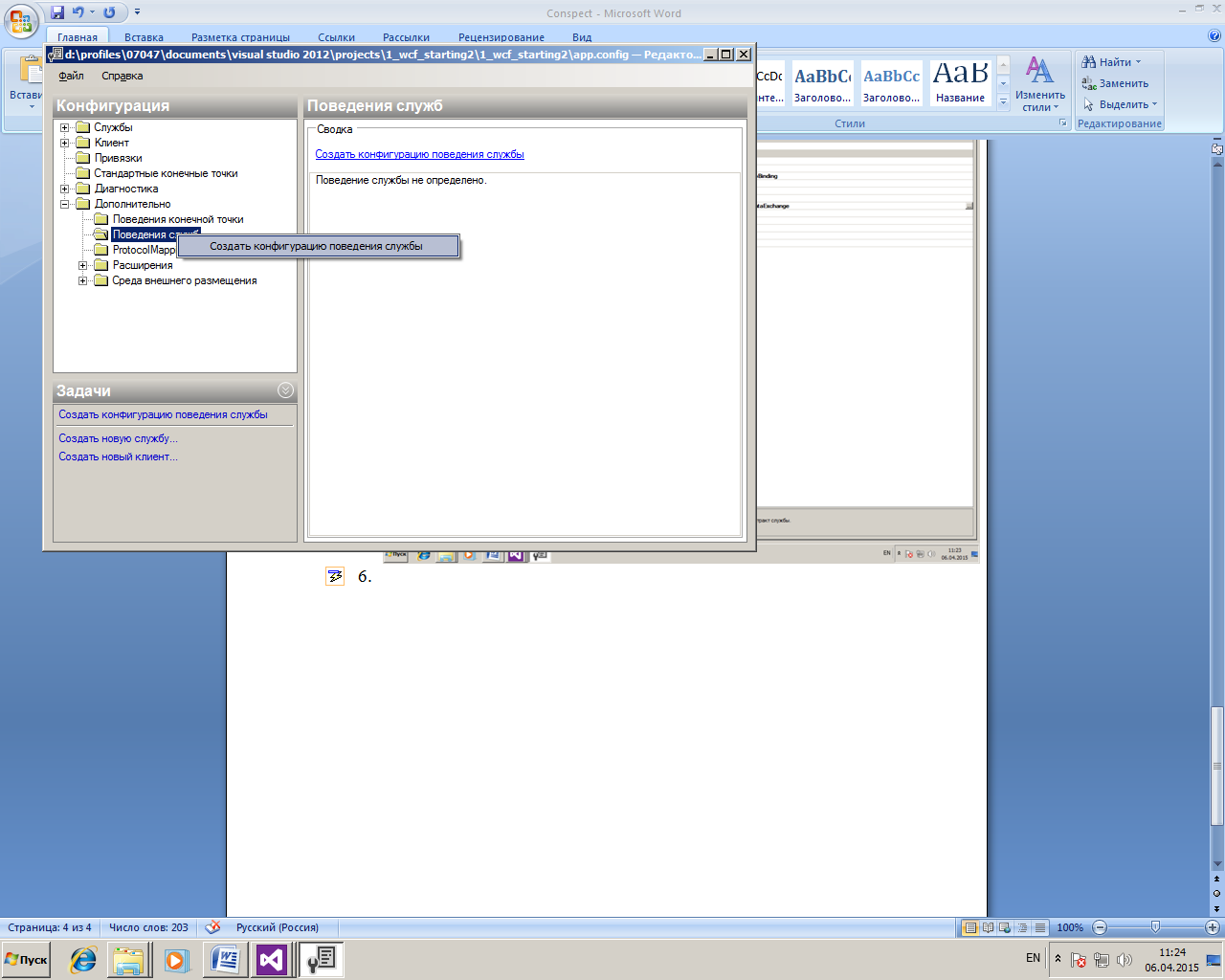
1. Создаем в конечных точках новую конечную точку

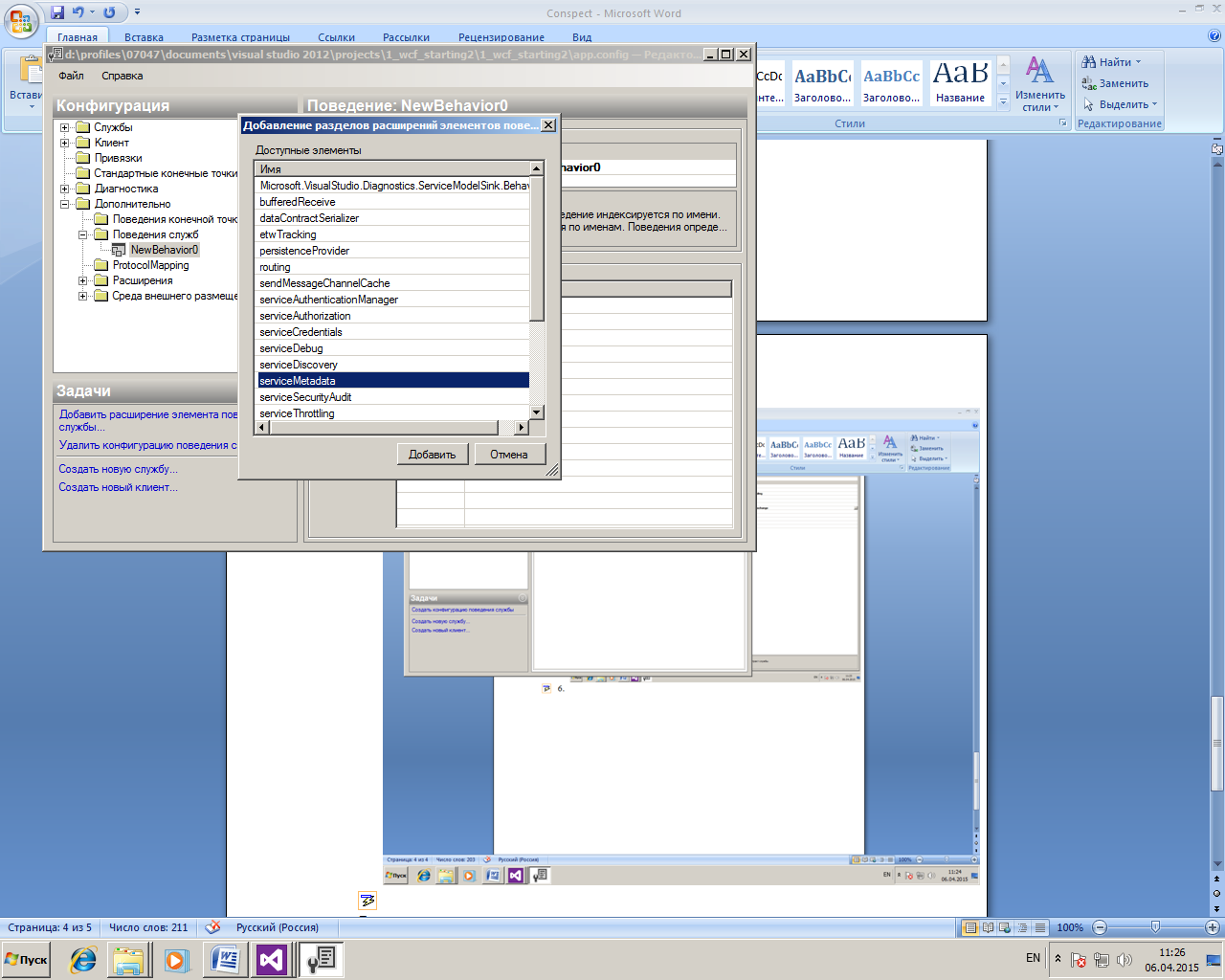
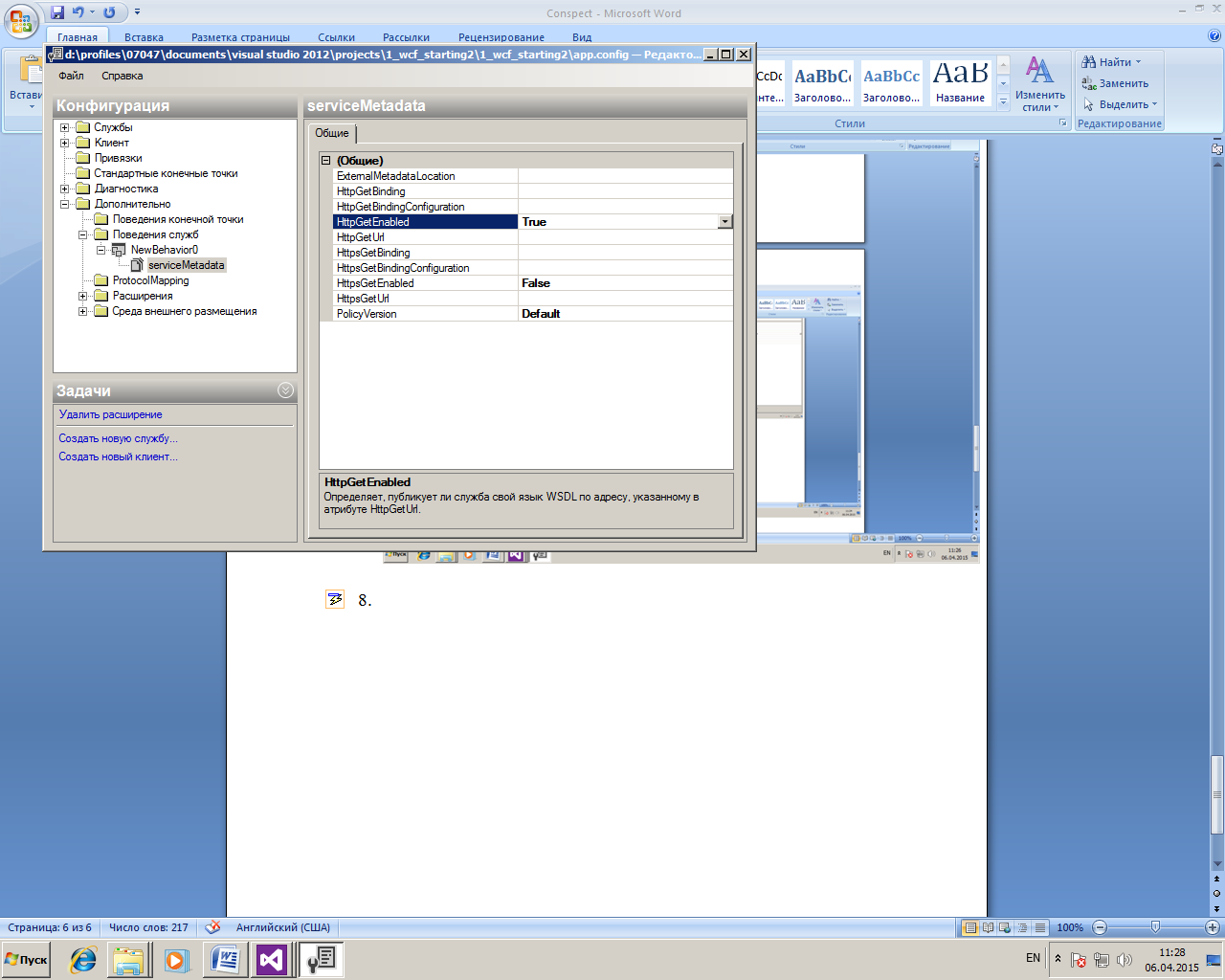


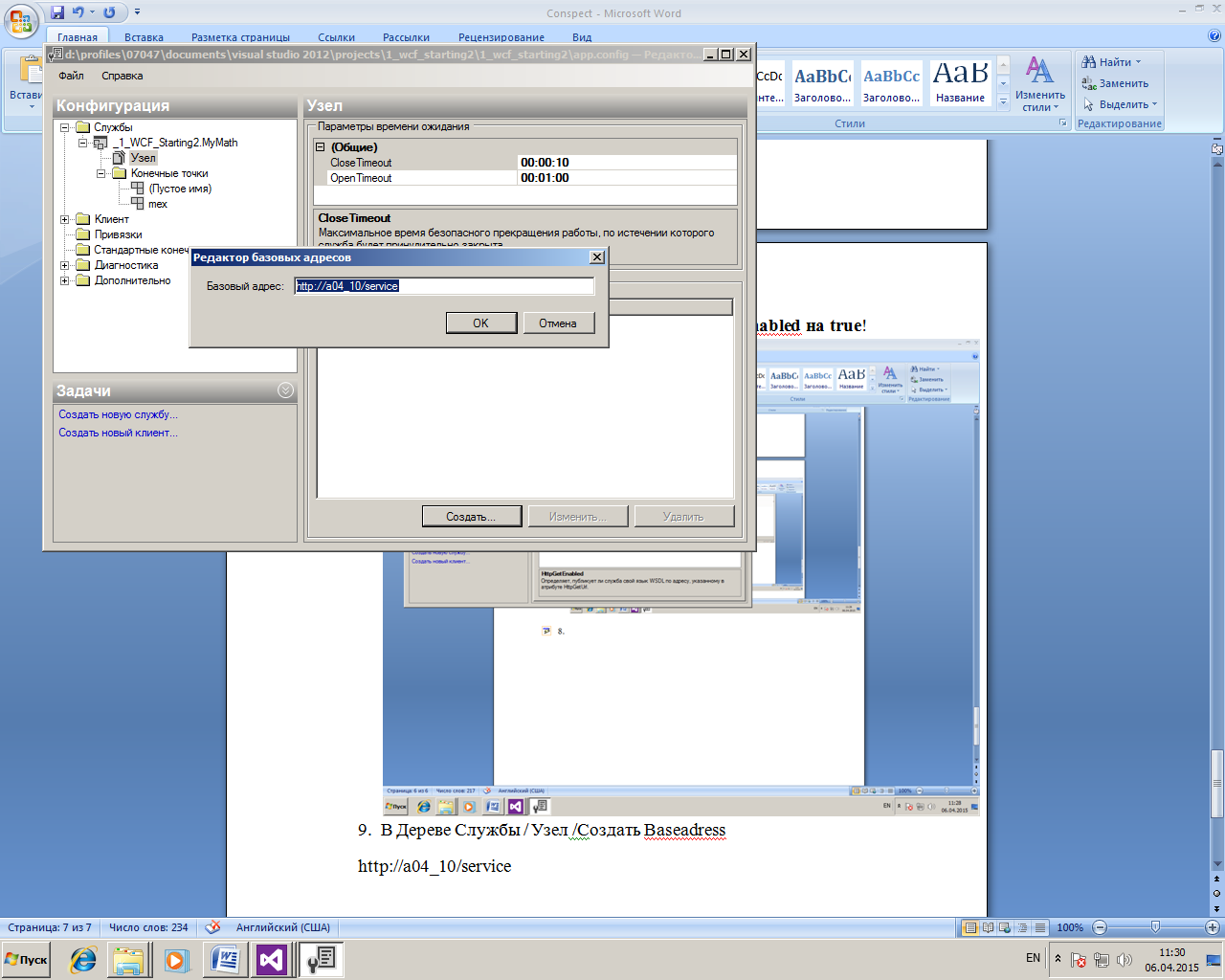
1. Указываем имя mex – в нижнем регистре!!!!!!!
2. Binding – выбираем mexHttpbinding
3. Contract – прописываем IMetadataExchange



1. Конфигурация/Дополнительно/Поведение службы – правая кнопка Мыши/Создать конфигурацию поведения службы

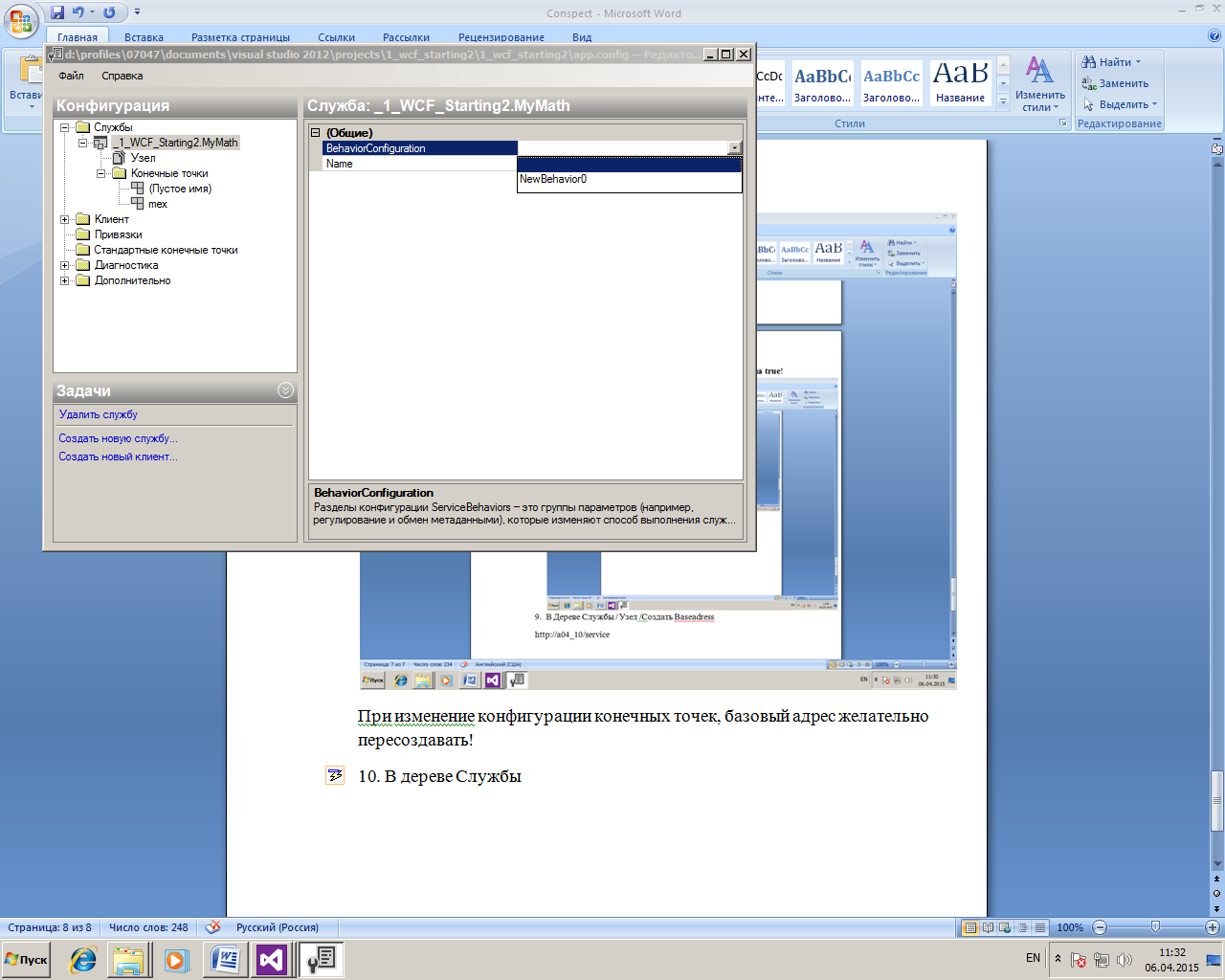


1. Добавить / Добавить service metadata
2. В serviceMetaData(в дереве) изменить HttpGetEnabled на true!
3. В Дереве Службы / Узел /Создать Baseadress (по умолчанию)



При изменение конфигурации конечных точек, базовый адрес желательно пересоздавать!

1. В дереве Службы / По своей службе / BehaviorConfiguration добавить newbahavior0



Сохранить.

BINDING

Существуют следующие варианты привязок (BINDiNG)

1. basicHttpBinding

простая привязка служб, не обеспечивающая безопасность сообщения

1. wsHttpBinding(ws - web servise)

Обеспечивает безопасность сообщений, а именно кодирует переписку клиента – сервера. Так же позволяет указать , важна ли очередность отправки сообщений.

1. wsDualHttpBinding

wsHttpBinding + дуплексный режим

1. netTcpBinding

Моежт использоваться только в пределах .NET FrameWork и передает данные с помощью tcp протокола. Так же поддерживает дуплексный контракт.

1. netPeerTcpBinding

Торрент трекеры

Перечисление свойство и у класса Binding

(enum)SecurityMode.

.None - нет защиты.

.Transport - защита которая поддерживается при условии , что ее поддерживает транспортный протокол.(tcp , httpAs)

.Message – защита которая обеспечивается протоколом SOAP.

.TransportWithMessageCredential – транспортный протокол обеспечивает защиту сообщений а SOAP протокол механизм аутотентификации клиента.

SecurityMode можно задать либо при создании привязки к конструктору:

WSHttpBinding binding = new WSHttpBinding(SecurityMode.Transport, true);

True – включает его;

Либо когда объект уже создан:

\_binding.Security.Mode = SecurityMode.Transport;

Либо в App.config:

<bindings>

<ws2007HttpBinding>

<binding name="WS2007HttpBinding\_IWords">

<security mode="Transport"></security>

</binding>

</ws2007HttpBinding>

</bindings>

Варианты поведения служб:

1. [ServiceBehavior(InstanceContextMode=InstanceContextMode.PerCall)] – означает, что для каждого нового (отдельного) запроса клиента, создается отдельный экземпляр службы.
2. П.С. baseHttpbinding – только с PerCall
3. [ServiceBehavior(InstanceContextMode=InstanceContextMode.PerSession)] – для каждого клиента создается свой экземпляр службы , служба работает с каждым клиентом индивидуально. (10 клиентов – 10 служб)
4. [ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)] – служба одна на всех, независимо от кол-ва клиентов и кол-ва запросов.

5. ServiceBehavior(ConcurrencyMode=ConcurrencyMode.Single)] – доступ только в одном потоке

6. [ServiceBehavior(ConcurrencyMode=ConcurrencyMode.Multiple)] – доступ из нескольких потоков.

7. [ServiceBehavior(ConcurrencyMode=ConcurrencyMode.Reentrant)] – разрешает доступ к службам из дополнительного потока, если это контракт клиента. Новая операция может быть начата, если поступил запрос по контракту обратного вызова.

Дуплексные конракты

- позволяет службе вызывать методы клиента, без ведома клиента!

Не все привязки разрешают использовать дуплексные контракты.

Позволяют: DualHttpBinding, или связки по TCP протоколу!

Дуплексный контакт описывается как на стороне сервера, так и на клиенте!

* System

Programming

(Потоки)

**Process** – это специальная конструкция операционной системы, предназначенная для доступа к ресурсам, необходимых для запущенного приложения, одновременно с этим обеспечивает защиту ресурсов от других приложений.

В С# процесс реализован в виде класса Process. Находится в библиотеке System.Diagnostics.

В процессе формируется сборка, класс Assembly. Которое представляет собой скомпилированное, запущенное приложение в ОЗУ.

Каждая сборка состоит из модулей – отдельный файл загруженный в виде компонентов приложения (exe или dll).

Внутри процесса, с помощью сборки (которая представляет собой сам код приложения, ресурсы, картинки и т.д.), создается первичный (главный) поток.

Поток, будучи создан, забирает в свой стек код приложения из сборки и несет его к процессору на выполнение.

Приоритеты процесса (базовые приоритеты):

1. Idle (фоновый) – имеет самый низкий приоритет
2. Normal (по умолчанию)
3. High (высокий)
4. RealTime – самый высокий, приоритет реального времени.

Приоритеты потока могут быть на два уровня выше или ниже от их базового приоритета (Приоритета процесса)

Расположены в классе **ThreadPriority**.

* .Lowest
* .BellowNormal
* .Normal
* .AboveNormal
* .Highest

С каждым потоком связан метод, как только потоковый метод завершается, завершается и сам поток. Если необходимо чтобы потоковый метод работал постоянно, необходимо код этого метода оборачивают в вечный цикл.

System.Threading

**СПОСОБЫ СОЗДАНИЯ ПОТОКОВ**

Есть несколько способов создания потоков.

1. Явный способ:

Thread t = new Thread( new ThreadStart(nameMethod));

t.Start();

t.IsBackGround = true;

Делегат типа ThreadStart принимает адреса методов только void и без параметров.

IsBackGround – сво-во указывающее, что при завершение главного потока, данный дочерний поток так же должен завершится.

nameMethod – могут быть как статическими так и объектными методами!

2. способ:

Thread t = new Thread ( new ParametriesedThreadStart ( nameMethod ));

t.IsBackGround = true;

t.Start(object);

Делегат этого вида, позволяет использовать потоковый метод с одним параметром типа object. Этот параметр передается в поток с помощью метода Start.

Параметр должен быть только object и только 1 !!!!!!!!!!!

3. Не явный способ:

Thread t = new Thread (nameMethod);

t.IsBackGround = true;

t.Start();

Создается анонимный делегат, который связывается с потоковым методом (nameMethod) и сразу запускается.

**Синхронность потоков**

При работе с потоками важно помнить о потоковой безопасности.

Потоковая безопасность – это гарантия того, что приложение будет работать одинаково корректно в независимости от кол-ва потоков и того что они делают.

Использование общих переменных для нескольких потоков не является потоко-безопасной. Один из вариантов решения проблемы структура LOCK.

for (int i = 0; i < 1000000; i++)

{

lock (locker)

{

num++;

}

}

В круглых скобках структуры lock указывается имя переменной ссылочного типа object и обязательно private!

private object locker = new object();

**Invoke**

Графические элементы управления не являются потоково-безопасными! И доступ к ним из дочерних потоков может привести к ошибке. С графическими элементами необходимо работать только из того потока, в котором они были созданы! Для этого можно использовать метод INVOKE , которому нужно передать адрес метода, который требуется вызвать

Invoke выполняет метод независимо от того в каком потоке он был вызван

В WPF потоках любое изменение делать в Dispatcher.Invoke!!!!

**SEMAPHORE**

- это специальный объект, задача которого контролировать кол-во одновременно запущенных потоков и приостанавливать те потоки, которые попадают сверх заданного лимита. Когда один из работающих потоков завершается, семафор разблокирует один из ожидающих потоков.

Семафором необходимо управлять вручную.

Методом semaphorename.WAITONE();

- указывается семафору, что какой-то поток пришел к ресурсу, и кол-во активных на данный момент потоков, увеличено на единицу!

Метод semaname.Release() указывает семафору что поток завершил свою работу, и кол-во активных потоков уменьшается на единицу.

Аналог Семафора:

**EventWaitHandle**

У которого два наследника

AutoResetEvent and ManualResetEvent

Эти объекты реализуют работу по принципу Турникета (триггера), которые могут быть в двух состояниях - открыт и закрыт!

Когда EventWaitHandle в закрытом состоянии он блокирует доступ к ресурсам, приостанавливает поток. Как только состоянии меняется на открытое, поток разблокируется.

Управление осуществляется методом WaitOne.

WaitOne() – > close;

Set() –> open;

**Assembly**

Логическая единица, содержащая скомпилированный код, библиотеки(dll), ресурсы в ОЗУ!

Технологии .Net позволяют в пределах одного процесса запускать несколько сборок! Эти сборки являются относительно изолированными друг от друга (т.е.при возникновении ошибки в одной сборке, не привидет к возникновению ошибке в другой сборке!). Но при это они могут обмениватся информацией.

Загруженная сборка в терминологии .Net называется Domain.

**АСИНХРОННОСТЬ**

Часто при разработке приложения требуется вызывать долго выполняющиеся методы (к примеру, чтение из файлов). При этом вызывающий поток блокируется до завершения операции. Выход из этой операции – вызов этих методов АСИНХРОННО!

Для этого используются интерфейс IAsync.Rezult. Ссылочка этого типа будет хранить результат работы метода.

Методы, реагирующие на завершение асинхронных методов ДОЛЖНЫ БЫТЬ void и принимать ОДИН парамет в виде IAsync.Rezult .

Дочерний поток, который выполнял какую-то задачу, после завершения работы формирует объект IAsync.Rezult и в сво-во .IAsyncState заносит информацию, необходимую для работы обработчика.

**ThreadPool**

- Класс пулов

Методы:

* GetMaxThreads – позволяет получить кол-во потоков, которые могут быть выполнены одновременно.
* SetMaxThreads – устанавливает кол-во потоков, которые будут выполнятся одновременно, когда один выполнится, его место автоматически займет новый, по принципу очереди и семафора.
* GetMinTreads – получает миним. кол-во потоков создавамых пулом потока, по мере поступления запроса.
* SetMinThreds –
* QueueUserWorkItem(delegate,[object]) – устанавливает пток в очередь

**Класс Task**

- класс, инкапсулирующий в себя асинхронную операцию! Этот класс позволяет любой метод сделать асинхронно. Так же класс позволяет управлять этим методом, запускать его или приостанавливать, а также остановить полностью.

Task task = new Task(delegate);

Если создавать потоковые методы через класс Task, то удобно использовать следующий механизм.

Вызов метода через ключевое слово - await Name();

Тип возвращаемого значение метода становится Task!!!!!! для Void методов, Task<TRezult> для методов с взвращаемым значением.( Task<int or string or…var>)

**ASYNC / AWAIT**

Async - помечается метод, который указывает компилятору, что внутри себя(него), он может хранить асинхронные операции!

await - помечается асинхронная функция или метод - которая вызывается в параллельном потоке.

Что вызывать в параллельном потоке свой метод нужно создать метод типа Task и пометить его как async!

Пример:

namespace TestingWPF

{

public partial class MainWindow : Window

{

public Random r;

public MainWindow()

{

InitializeComponent();

r = new Random();

}

private async void Button\_Click(object sender, RoutedEventArgs e)

{

if (qwe.Text == "")

return;

asd.Text = qwe.Text;

asd.FontSize = await MyAsyncMethod();

}

public async Task<Int32> MyAsyncMethod()

{

int a = r.Next(8, 72);

return a;

}

}

}

* ADO.NET

Введение

**ADO.NET** - набор DLL который позволяет работать с любой базой данных, с любым сервером.

С сервером баз данных можно работать в 2-х режимах:

1.Присоединенный режим – приложение присоединяется к базе и остается в таком режиме, пока не вызовется метод завершения соединения или пока не завершится сама программа. Этот способ оптимален с точки зрения работы, но нагрузка на сервер возрастает. При большом количестве подключенных клиентов, сервер может просто лечь.

2. Отсоединенный режим – приложение подключается к серверу только на момент выполнения запроса. По окончание которого отключается. Более медленный режим, так как тратится время на аутотентефикацию, подключение и отключение. Но при этом этот способ не загружает сервер.

Строка подключения – это особого рода запись, уникальная для каждого сервера и позволяющая приложения подключится к серверу баз данных.

**Строка подключения для MS SQL**

Для WindowsAutotenfications:

In homePC = DataSource “NamePC-SQL”

String connect = @ “Data Source = 10.3.0.5 (Home/Sqlexpress);Initial Catalog = Shop2211;Integrated Security = SSPI; ”

Для SQL:

String connect = @”Data source=10.3.0.5;Initial Catalog=Shop2211;User ID=students\_SP2211;Password=sp2211”

Для работы с базами данных подключаем следующие библиотеки:

using System.Data.Common;

using System.Data.SqlClient;

Список классов:

DbConnection – реализует подключения к серверу в одном из режимов (SqlConnection, OracleConnection, OleDbConnection, OdbcConnection)

DbCommand – абстрактный класс, который позволяет отправлять запрос на сервер и если необходимо, получить ответ. (SqlCommand, OracleCommand, OleDbCommand, OdbcCommand)

DbDataReader – абстрактный класс, работа с этим классом начинается тогда , когда пришел ответ от сервера в виде таблицы и нужно построчно считать информацию.

(OracleDataReader, SqlDataReader, OleDbDataReader, OdbcDataReader)

DbDataAdapter – абстрактный класс, реализует работу в отсоединенном режиме сервера.

(SqlDataAdapter, OracleDataAdapter, OleDbDataAdapter,OdbcDataAdapter)

3 способа подключения:

//1st вариант формирования параметров.

string sql = @"Select \* From OnShelfs Where id =" + textBox.Text;

this.command = con.CreateCommand(); //привязка к кону

/\*2-вариант

this.command = new SqlCommand();

this.command.Connection = con;

\*/

this.command.CommandText = sql;

3-вариант через передачу в функц. с параметрами!

Для OLE

Data source ”Путь к файлу”

”d:\Profiles\07047\Desktop\InternetShop.mdb”

Provider –:

Microsoft.Jet.OLEDB.4.0

**Остоединенный режим работы**

- осуществляется классом DBDataAdapter у которого есть наследники, (SqlDataAdapter,OraclDataAdapter,OleDBDataAdapter,ODbSDataAdapter)

Эти классы при инициализации требуют строку команды и подключения

* LINQ

LINQ to Object

Методы LINQ

**Select** - делает выборку

**Where** – возвращает все элементы удовл-е условию

**Average**() – возвращает среднее значение кол-ии

**Min**() **Max**() – аналогично Average

.**Take**(int i) принимает интовое значения и возвращает указанное кол-во коллекции(начиная с первого)

**TakeWhile** (func <class bool> predicat) – возвращает все элементы массива удовлетвор. определенному условию

**Skip**(int n) принимает значения пропуская указанное кол-во элементов

**SkipWhile**(func<class bool>pr) - выбирает все элементы пропуская ТЕ значения(с начала) пока не найдет элемент НЕ удовлетворяющие значения

**Reverse**() -

**New**

1. В обозревателе серверов, в пункте подключение данных создали новое подключение к удаленной БД.

2. В проект добавляем новый элемент класс LINQtoSql

3. В дизайнер объектов перетаскиваем нужную таблицу(Уже должна быть созданна).

4. В коде создаем объект управления БД (Context), у него есть метод GetTable<назв.таблицы> которому нужно указать название таблицы и он вернет эту таблицу из базы данных в виде объекта table!

5.Объект table это таблица из БД, где каждый столбец превращается в св-во.

* WPF

создание Grid в коде:

(пример с шахмат)

private void CreateField()

{

for (int i = 0; i < 8; i++)

{

ColumnDefinition col = new ColumnDefinition();

col.Width = new GridLength(50);

\_fieldCheckMate.ColumnDefinitions.Add(col);

}

for (int i = 0; i < 8; i++)

{

RowDefinition row = new RowDefinition();

row.Height = new GridLength(50);

\_fieldCheckMate.RowDefinitions.Add(row);

}

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

if ((i + j) % 2 == 0)

{

Border blackRow = new Border();

blackRow.Background = new SolidColorBrush(Colors.Transparent);

Grid.SetColumn(blackRow, i);

Grid.SetRow(blackRow, j);

\_Grid.Children.Add(blackRow);

}

else

{

Border whiteRow = new Border();

whiteRow.Background = new SolidColorBrush(Colors.WhiteSmoke);

Grid.SetColumn(whiteRow, i);

Grid.SetRow(whiteRow, j);

\_Grid.Children.Add(whiteRow);

}

}

}

добавление в коде ТАЙМЕРА

TimeSpan (временной промежуток) - задает, указывает временные промежутки и их представление.

public partial class MainWindow : Window

{

Random r = new Random();

DispatcherTimer \_timerEnemy = new DispatcherTimer();

DispatcherTimer \_timerTarget = new DispatcherTimer();

Boolean \_humanCaptured = false;

public MainWindow()

{

InitializeComponent();

\_timerEnemy.Tick += \_timerEnemy\_Tick;

\_timerEnemy.Interval = TimeSpan.FromSeconds(2);

\_timerTarget.Tick += \_timerTarget\_Tick;

\_timerEnemy.Interval = TimeSpan.FromSeconds(.1);

}

void \_timerTarget\_Tick(object sender, EventArgs e)

{

}

void \_timerEnemy\_Tick(object sender, EventArgs e)

{

}

}

Комманды:

SizeToContent=" "

- устанавливает размеры основного окна в соответствии с размерами контента.

**IsHitTestVisible** и «видимость» элементов

Обычно на любом элементе страницы можно «щелкнуть» указателем

мыши — при условии, что он отвечает определенным критериям. Он

должен быть видимым (это зависит от свойства Visibility), должен

иметь отличные от null свойства Background или Fill (но обладать

свойством Transparent), должен быть доступен (через свойство

IsEnabled) и обладать отличными от нуля свойствами height и

width. При соблюдении этих условий свойство IsHitTestVisible

возвращает значение True, и элемент отвечает на события указателя

или мыши.

Это свойство особенно полезно, если элемент нужно сделать «неви-

димым» для мыши. Присвойте IsHitTestVisible значение False,

и щелчки мыши будут проникать сквозь элемент. Реагировать на

событие будет элемент, расположенный под ним.

**Visibility** - Определяет состояние отображения элемента. С помощью данного инструмента можно контролировать прорисовку, эмулируя переходы по страницам, картам, окнам и т.п.

К примеру можно создать 2 Grid, в одной и той же ячейке, с одними и теми же размера. Указав одному из них Hidden и по нажатию , к примеру по клавише, изменять их Visibility!

\_fieldCheckMate.Visibility = System.Windows.Visibility.Hidden;

\_fieldStartMenu.Visibility = System.Windows.Visibility.Visible;

\_fieldStartMenu.Visibility = System.Windows.Visibility.Collapsed;

Collapsed - не резервирует для объекта место(сбрасывает настройки размещения и размеров).

Hidden - скрывает, но резервирует для объекта место! Проще сказать объект становится невидимым но с теми же размерами и в том же месте!

**Dispatcher.Invoke -** создает потоко-безопасный способ изменения элемента.

В WPF доступ к этому объекту может получить только поток, создавший объект [DispatcherObject](https://msdn.microsoft.com/ru-ru/library/system.windows.threading.dispatcherobject(v=vs.110).aspx). Например, фоновый поток, отделившийся от основного потока пользовательского интерфейса, не может использоваться для обновления содержимого элемента управления [Button](https://msdn.microsoft.com/ru-ru/library/system.windows.controls.button(v=vs.110).aspx), созданного в потоке пользовательского интерфейса. Чтобы фоновый поток получил доступ к свойству Content элемента управления [Button](https://msdn.microsoft.com/ru-ru/library/system.windows.controls.button(v=vs.110).aspx), он должен делегировать работу объекту [Dispatcher](https://msdn.microsoft.com/ru-ru/library/system.windows.threading.dispatcher(v=vs.110).aspx), связанному с потоком пользовательского интерфейса. Это достигается с помощью Invoke или [BeginInvoke](https://msdn.microsoft.com/ru-ru/library/system.windows.threading.dispatcher.begininvoke(v=vs.110).aspx). Перегрузка Invoke является синхронной, а [BeginInvoke](https://msdn.microsoft.com/ru-ru/library/system.windows.threading.dispatcher.begininvoke(v=vs.110).aspx) — асинхронной. Данная операция добавлена в очередь событий объекта [Dispatcher](https://msdn.microsoft.com/ru-ru/library/system.windows.threading.dispatcher(v=vs.110).aspx) с заданным [DispatcherPriority](https://msdn.microsoft.com/ru-ru/library/system.windows.threading.dispatcherpriority(v=vs.110).aspx).

Invoke является синхронной операцией; поэтому управление не возвращается вызывающему объекту до тех пор, пока функция обратного вызова не завершит свою работу.

Пример: Создать метод который прорисовывает взрыв, состоящий из 5 картинок-спрайтов!

namespace exploision

{

///<summary>

/// *Interaction logic for MainWindow.xaml*

///</summary>

public partial class MainWindow : Window

{

Image img = new Image();

public MainWindow()

{

InitializeComponent();

List<Point> points = new List<Point>();

points.Add(new Point(2, 1));

points.Add(new Point(0, 1));

points.Add(new Point(1, 2));

points.Add(new Point(1, 0));

points.Add(new Point(1, 1));

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

img = new Image();

img.Source = new BitmapImage(new Uri(@"../../Image/empty.jpg", UriKind.RelativeOrAbsolute));

img.Stretch = Stretch.Fill;

Grid.SetRow(img, i);

Grid.SetColumn(img, j);

MapGrid.Children.Add(img);

}

}

;

for (int i = 0; i < points.Count; i++)

{

Incaps a = new Incaps(points , i);

Thread t1 = new Thread(ExpolisionAnimation);

t1.IsBackground = true;

t1.Start(a);

}

}

private void ExpolisionAnimation(object obj)

{

List<Point> points1 = (obj as Incaps).list;

int i = (obj as Incaps).K;

while (true)

{

for (int k = 1; k < 6; k++)

{

Dispatcher.Invoke(

new Action(delegate {

(MapGrid.Children[3 \* (int)points1[i].X + (int)points1[i].Y + 1] as Image).Source = new BitmapImage(new Uri(@"../../Image/Explosion" + k.ToString() + ".png", UriKind.RelativeOrAbsolute));

}));

Thread.Sleep(200);

}

}

}

public class Incaps

{

public List<Point> list;

public int K;

public Incaps(List<Point> l, int k)

{

this.list = l;

this.K = k;

}

}

}

}

Thread благодаря Invoke не приостанавливает при этом работу программы.

Потоку нужно подсовывать метод без параметров или же метод с единственным принимающим значением типа object.

Здесь использую класс для инкапсуляции передаваемых значений object!

Формула для получения индекса элемента Childrens у Grid (изначально создается 2-х мерный массив, но потом Grid использует что-то типа списка элементов UIElementCollection)

(MapGrid.Children[3 \* (int)points1[i].X + (int)points1[i].Y + 1] as Image)

Здесь:

3 - количество ячеек в строке, нужно записывать при создании или запоминать.

Х - координата ячейки, количество полностью заполненных строк. (Если считать не с нуля, нужно отнимать единицу!)

Y - координата ячейки. (Если считать С нуля добавляем единицу!)

**Три способа хранения ресурсов:**

1. Создание папки в обозреватели решений, добавляем новый элемент с выставлением в его сво-ваx Copy Always(Всегда копировать).

pictureBox1.Image = Image.FromFile(@"Img\images.jpg");

Этот способ обязуется вместе с .exe переносить и папку с ресурсами.

1. Embedded resource

По проекту правый клик – свойства! Пункт Ресурсу(сверху по стрелочке) и добавляем существующий файл(или создаем новый)

pictureBox1.Image = .Properties.Resources.Image1;

Если в ресурсах не изображение, то лучше его вытаскивать в байтовый массив!

Byte[] b = xxx!

1. По проекту правым кликом добавить сущес-й элемент.

Выбираем файл, затем в св-вах файла – св-во «действие при построении» выставляем «Embedded resource

(внедренный ресурс)»

//Получаем ссылку на текущую сборку приложения

Assembly asm = Assembly.GetExecutingAssembly();

//Получаем имя сборки

AssemblyName name = asm.GetName();

pictureBox1.Image = new Bitmap( asm.GetManifestResourceStream(name.Name + ".2.jpg") );

* Тестирование

программного

обеспечения

Тестовые методы всегда public и без параметров!

Атрибуты

[Test]

– уведомляет, что этот метод может быть протестирован.

[ExpectedException(typeof(NullReferenceException))] - Атрибут для проверок исключений.

NUnit

[TestFixture]

public class TestMyClass

{

[Test]

public void TestAdd1()

{

MyClass mc = new MyClass();

int expected = 5;

int actual = mc.add(3, 2);

Assert.AreEqual(expected, actual);

}

}

Assert делается в отдельных методах, потом собираются

в один ОБЩИЙ метод!

* PATTERNS

Singelton

Используется для гарантирования существования объекта в одном (в частных случаях N-го кол-ва) экземпляре!

Пример:

namespace Testing

{

class Program

{

static void Main(string[] args)

{

Singelton a = Singelton.Instance();

Singelton b = Singelton.Instance();

Console.WriteLine(ReferenceEquals(a,b));

*//Выводит тру!*

}

}

class Singelton

{

private static Singelton singl;

private Singelton()

{

}

public static Singelton Instance()

{

if (singl != null)

singl = new Singelton();

return singl;

}

}

}

* HTML

- язык разметки гипертекста(ссылок)!!!!

http://htmlbook.ru/

Sublime:

http://docs.emmet.io/cheat-sheet/

HotKey:

WebFont ctrl + alt +F

ColorPicker ctrl + shift + C

Теги и атрибуты:

Каждый тег может иметь одновременно все свои атрибуты, но только в одном экземпляре!

Позиционирование:

<float> - как бы приподнимает элемент и под него залазит следующий!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Семантические теги:

- блочные элементы.

<section>

<aside> рекламный блок

<articl> - завершенная статья, публикация.

<header>

<figure> - фигуры или рисунки.

<figcaption>

<nav> - навигационные ссылки.

<footer>

Карта ссылок:

Circle

Rect

Poly

Form:

Свойства:

* type
* name
* value
* disabled

Event

* onfocus – событие фокуса
* onblur – потеря фокуса ввода

Методы

* focus() – получение фокуса
* blur() – потеря фокуса

Добавление форм:

Address?data&data&ndata - (без пробелов)

<form action="" method="get">

<p>

Input name: <input type="text" id="name" size="20" />

</p>

<p>

Input pass: <input type="password" id="pass" size="20" />

</p>

<p>

<input type="radio" name="r" size="20" value="Female" /> F

<input type="radio" name="r" size="20" value="'Male" /> M

</p>

<p>

Languages:

<br/><input type="checkbox" name="ua" value="ua" size="20" /> UA

<br/><input type="checkbox" name="ru" value="ru" size="20" /> RU

<br/><input type="checkbox" name="en" value="en" size="20" /> EN

</p>

</form>

Добавление счетчика посещения:

Google Analitycs

Yandex Metrik

24log видимый

Работа с тегом <iframe>:

Отображение контента с другой страничке или ресурса в контейнере!

<!**DOCTYPE** html>

<**html** lang="en">

<**head**>

<**meta** charset="UTF-8">

<**title**>Document</**title**>

<**style**>

\*

{

box-sizing: border-box;

}

.d1

{

width: 50%;

height: 750px;

border: 1px inset black;

float: left;

}

#idFr

{

height: 90%;

width: 90%;

padding: 10px;

margin: 50px;

background-color: lightpink;

}

</**style**>

</**head**>

<**body**>

<**div** class="d1" style="background-color:lightblue;">

<**ul**>

<**a** href="#" class="menu" onclick="m1()">menu1</**a**>

<**a** href="#" class="menu" onclick="m2()">menu1</**a**>

<**a** href="#" class="menu" onclick="m3()">menu1</**a**>

<**a** href="http://wikipedia.com" class="menu" onclick="" target="fr">menu1</**a**>

</**ul**>

</**div**>

<**div** class="d1" style="background-color:lightgreen;">

<**iframe** src="" name="fr" id="idFr" width="400" height="400" frameborder="1">

</**iframe**>

</**div**>

<**script** type="text/javascript">

function m1()

{

idFr.src="pages/1.html";

}

function m2()

{

idFr.src="pages/2.html";

}

function m3()

{

idFr.src="images/picture1.jpg";

}

</**script**>

</**body**>

</**html**>

* CSS

Селекторы

\* -для всех тегов файла;

A - все теги A;

.А - ВСЕ теги с классом А

#A -тег с id A;

A > B - все теги B входящие в тег A

A + B -все теги B идущие СРАЗУ(следующим) после(за) тега A

A~B - A and B имеют общего родителя ( являются sibling) и B стоит после A!

A[B] -все теги A c атрибутом B;

A[B^="C"] -теги A где атрибут B начинается с C;

A[B$="C"] - теги A где атрибут B заканчивается с C;

A[B\*="C"] - теги A где в атрибут B входит C;

Стилевые правила:

box-shadow: -10px 10px 5px lightgreen; - добавление теней!

/\* Шаблон селектора \*/

[class\* ="col-"] !!!УКАЗЫВАТЬ БЕЗ ПРОБЕЛОВ

{ float: left; padding: 15px;}

backgroun-image - можно указывать НЕСКОЛЬКО фонов для одного и того же блока! Указываются через запятую! Все параметры для на них так же указываются в последующих правилах через запятую!

border-radius - через / можно указывать элипсовое скругление

10px/20px для одного угла!

box-shadow - x, y, размытие, растяжение, цвет, расположение ( inset внутри)

display: flexbox - grid для отображения блоков, указывается с(box-oriental : horizontal(vertical)!

Мета теги:

<!**DOCTYPE** html>

<**html** lang="en" xmlns="http://www.w3.org/1999/xhtml">

<**head**>

<**meta** charset="utf-8" />

<!-- Обязатьельные 2 тега для раскрутки сайта -->

<**meta** name="description" content="content" />

<**meta** name="keywords" content=""/>

<!--Авторство и защита прав-->

<**meta** name="author" content="Zepsen"/>

<**meta** name="copyright" content="hfasdjfhaj"/>

<!-- -->

<**meta** name="revisit-after" content="20 days" />

<**title**></**title**>

</**head**>

<**body** style="background-color:aqua">

<!--Обязательно наличие одного заголовка h1

с ключевыми словами выделенными семантическими тегами!

<strong>-->

</**body**>

</**html**>

Responsive: (отзывчивого )

1. box-sizing: border-box; - точно устанавливает размеры с заданными!
2. <meta charset="UTF-8" name="viewport" content="width=device-width, initial-scale=1.0"/>

(НЕ ДЛЯ apple)

1. @media

* Java Script

learn.javascript.ru

JS – интерпретированный язык. Выполнение происходит построчно!

Подключения скрипта рекомендуется подключать перед закрытым тегом body!

Не называть функции системными именами

* Click;

Область видимости {} задает только function!

Лексическое окружение – область видимости родителя!

Type:

* null;
* undefined;
* number ;
  + NaN – not a number;
  + Infinity – бесконечность;
* boolean;
* string;

Object:

* браузерные;
* встроенные(массивы);
* свои классы (var)

Function

Event:

Каждое событие в JS имеет тип ВСПЛЫВАНИЯ!

<!**DOCTYPE** html>

<**html** lang="en" xmlns="http://www.w3.org/1999/xhtml">

<**head**>

<**meta** charset="utf-8" />

<**title**></**title**>

<**style**>

div

{

margin: 25px;

border: 1px groove black;

background-color: white;

}

#d1

{

width: 200px;

height: 200px;

}

#d2 {

width: 150px;

height: 150px;

}

#d3{

width: 100px;

height: 100px;

}

</**style**>

<**script** type="text/javascript">

function func1(elem) {

elem.style.backgroundColor = "pink";

}

</**script**>

</**head**>

<**body**>

<**div** id="d1" onclick="func1(this)">

<**div** id="d2" onclick="func1(this)">

<**div** id="d3" onclick="func1(this)">

</**div**>

</**div**>

</**div**>

</**body**>

</**html**>

Что бы запретить в скрипте нужно добавить cancelBubble

<**script** type="text/javascript">

function func1(elem) {

elem.style.backgroundColor = "pink";

event = event || window.event;

event.cancelBubble = true;

}

</**script**>

returnValue Запрещает элементу выполнять событие по умолчанию.

<!**DOCTYPE** html>

<**html** lang="en" xmlns="http://www.w3.org/1999/xhtml">

<**head**>

<**meta** charset="utf-8" />

<**title**></**title**>

<**style**>

<**script** type="text/javascript">

function func2() {

document.body.style.backgroundColor = "lightgreen";

event = event || window.event;

event.returnValue = false;

}

</**script**>

</**head**>

<**body**>

<**a** href="http://www.google.com" target="\_blank" onclick="func2(this)">

Google

</**a**>

</**body**>

</**html**>

Валидация данных:

GET – в адресной строке

POST – не имеет ограничения на передачу

Добраться к форме

* document. forms[index];
* document. forms[‘nameform’];
* documents.forms.nameform;
* documents.name;
* var f = formname.elementname;

Window

Методы:

.open() .open(“http://”, “reference”, … )

.close

.moveTo (x ,y ) –передвинуть В точки x ,y

.moveBy (x,y) – передвинуть НА x,y

.resizeTo(x,y) – сделать окно размером x,y

.resizeBy (x,y) –уменьшить / увеличить на x,y

Свойства объекта window:

.status

.opener

.closed (bool) – закрытое ли окно

.navigation

1. If in UserAgent есть слово Opera – this is Opera!
2. If in UserAgent -> MSIE – this is IE;
3. If in UserAgent -> Firefox! or Gecko and No Webkit! -> this is Mozilla;

.location

http://www.abc.ua:88/index.html?id=10&name=vasya

.port

.host - name+port

.pathname – все до ?

.search

.protocol

Свойства:

Replace(…) – перейти

Reload(…) -

.history

history.back()

history.forward()

.screen

Width, height

availWidth, availHeight

colorDepth – глубина цвета!

JS OOП

<**script** type="text/javascript">

/\*класс\*/

function fish(n,w,p)

{

this.name = n;

this.weight = w;

this.price = p;

/\*set get\*/

this.setPrice = function(np)

{

this.price = np;

}

this.getPrice = function()

{

return this.price;

}

/\*анонимный метод класса\*/

this.show = function()

{

document.write("Name: " + this.name + "<br/>" + "Weight: " + this.weight + "<br/>" + "Price: " + this.price + "<br/>" );

}

}

/\*var f1 = new fish();

f1.name = "Gold";

f1.weight = 3;

f1.price = 100.23;\*/

var f2 = new fish("Guppi", 5, 23.65)

f2.show();

</**script**>

jQuery

(запрос)

$ == jQuery;

Подключение в head:

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">

</script>

Более новую версию смотреть на сайте

Селекторы:

* (\*) - все
* ("tagName") - все теги заданного формата (div p )
* ('tagname1, tagname2, tagname3 ') - через запятую можно указывать несколько тегов;
* ('tagNameParent tagNameChild') - через пробелы указываются вложенные теги.
* ('А +В') - все елементы В непосредственно стоящие после А, на том же уровне вложенности.
* ('А > В') - все В являющиеся потомками А. На первом уровне вложенности
* ("А has(B)") - все элементы А имеющие в себе элементы В!
* (А.classВ) - все элементы А с именем классом В!
* (А#idВ) - все элементы А с именем id В!
* (A~B) - все элементы B следующие за элементом А (не обязательно непосредственно за) на том же уровне вложенности(имеют одного родителя)
* ("А[B]") - все элементы А с атрибутом В.
* ("А[B=ХХХ]") -все элементы А с атрибутом В равному ХХХ.
* ("А[B^=ХХХ]") - все элементы А с атрибутом В заканчивающемуся на ХХХ.
* ("А[B$=ХХХ]")- все элементы А с атрибутом В начинающемуся с ХХХ.
* ("А[B\*=ХХХ]") - все элементы А с атрибутом В содержащие ХХХ.

Методы:

почти все методы без параметров возвращают значение !!!!!!

* .split("n") - разбивает элементы по заданному символу.
* .join('<div></div>') - оборачивает элементы в заданный элемент
* .hover(over, out ) - отслеживание попадание мышки на объект

function over при захвате, function out - при выходе из элемента. Можно передавать и одну функцию захвата или утраты!

* .querySelector("А") -получает ПЕРВЫЙ элемент соответствующий тегу А
* .querySelectorAll ("А") -получает ВСЕ элементы соответствующие тегу А
* .on("event (БЕЗ ON!)", funcName) - добавляет текущему элементы событие.

DOM:

Выборка:

* .eq(i) - выбирает элемент в массиве элементов по заданному индексу.
* .not(n) - возвращает все элементы КРОМЕ (n) заданного.
* .is(n) - проверяет текущий набор элементов на соответствие указанному выражению N. Возвращает true если хотя бы одно условие выполнится!
* .filter(selector) - используется для получения соответствующего элемента
* .filter(func) - в качестве параметра функция получат порядковый номер элемента. Если функция возвращает true, элемент проходит! $(this) обращение к текущему элементу
* .slice(index1, index2) - выбирает подмножество из совпавших. От index1 до index2. Если передать один индекс, выбрать ОТ index1 и ДО САМОГО конца. Если передать отрицательное значение - то это означает элемент с конца. -3 это 3-ий элемент с конца!

Поиск:

* .add(n) - добавляет к текущему, уже существующему, элементу набора - N.
* .find(n) - отыскать дочерний элемент N, у указанного элемента
* .contents(n) - отыскать дочерние элементы n, включая Текстовые узлы (nodeType = 3) и узлы комментарии!

Parent and Child

* .children(n) - отыскать всех прямых потомков, совпавших с выборкой(N)!
* .parent() - возвращает ссылку на ближайшего родителя!
* .parents(n) - выбирает все родительские элемента для данного элемента. Если вызвать без селектора! То будут возвращены все элементы, вплоть до body!
* .closest(selector) - получает БЛИЖАЙШИЙ заданный набор (selector) родительского элемента.
* .offsetParent() - ближайший предок, который ПОЗИЦИОНИРОВАН любым из свойств (absolute, fixed, relative)

Siblings

* .siblings() - все соседние(родственные) элементы
* .next() - получает набор следующий родственный(соседний) элемент! .next(sel) непосредственно следующий после sel .
* .prev() - получает набор предшествующих () родственный(соседний) элемент. .prev(s) непосредственно до s
* .nextAll() - получение ВСЕХ соседних элементов после текущего. .nextAll(p) = ~ но только элементы р!
* .prevAll() - получение ВСЕХ соседних элементов до текущего. .nextAll(p) = ~ непосредственно до элементов р!
* .nextUntill(E) - получение всех элементов После текущего, ДО заданного элемента Е!
* .prevUntill(Е) - получение всех элементов До текущего, ОТ заданного элемента Е!
* .andSelf() - объединяет текущий набор элементов, с предыдущем в рамках одной цепочки! И возвращает новый набор!
* .end() - отменяет последнее деструктивное действие. Возвращая элементы к их предшествующему состоянию.

Такие функции и методы как (add, andSelf, children, map, find , next , prev , slice, append, insert ...)

Insertion:

* .html() - получает( разметку) html - содержимое первого элемента содержимого в наборе соответствующего заданному. Или устанавливает разметку для каждого совпавшего.
* .html(val) - получает разметку совпавшего или задает каждому соответствующему элемент val
* .text() - получить содержимое тестового узла.
* .text('text') - внести текстовое содержимое
* .append() - добавляет контент внутрь совпавшего элемента, сразу за указанным элементом
* .appendTo() - добавляет элемент набора во внутрь другого элемента, в конец
* .prepend() - добавляет контент внутрь совпавшего элемента, сразу за ПЕРЕД элементом
* .prependTo() - добавляет элемент набора во внутрь другого элемента, ПЕРЕД существующем элементом
* .wrap() - вставляет каждый элемент в указанную конструкцию HTML разметки
* .empty() - удаляет все содержимое(текст, контент) внутри совпавшего элемента
* .remove() - удаляет все элементы из DOM (и обработчики событий) в совпавшем элементе

Class:

* .addClass("class1") - добавляет класс к элементу. НЕ ИЗМЕНЯЕТ!

.addClass(cl1 cl2 cl3) -добавление нескольких классов

.addClass(function(index, value)) - задает список классов которые возвращает функция. Функция должна вернуть строку,, где классы через пробел!. index - позиция элемента в наборе.

* . removeClass() - удаляет текущий класс из элемента

.removeClass(cl1 cl2 cl3) - удаляет нескольких классов

.removeClass(function(index, value)) - задает список классов которые возвращает функция для удаления. Функция должна вернуть строку, где классы через пробел!. index - позиция элемента в наборе.

* .toggleClass() - удаляет или задает текущий класс из элемента по принципу переключателя. Если есть удалить, нет - задает!

.toggleClass(cl1 cl2 cl3) - тоже что и выше!

.toggleClass(cl1 cl2 ... ,switch) - только удаляет или только задает класс(ы) в зависимости от параметра switch. true - только добавление, false - delete!

.toggleClass(function(index, value), switch) - задает список классов которая возвращает функция(стринг через пробелы!), вызывается для каждого элемента. index - позиция элемента в наборе. switch тот же что и выше.

* .hasClass('cl') - возвращает true , если такой класс существует.

Attr and Prop:

* .attr - получает или задает текущий атрибут.

('a') - получить

('a', 'b') - задать атрибуту А значение b

({ a:''b", c:"d") - задать атрибуту А и С значение B and D

при помощи JSON объекта

('a', func()) - задать атрибуту А значение из функции

* .prop - задать или получить свойство объекта

('р') - получить свойство

('р', 'b') - задать свойство Р значение b

({ р1:''b", р2:"d") - задать свойство Р1 и Р2 значение B and D

при помощи JSON объекта

('р', func()) - задать свойство Р значение из функции

* .removeProp('p') - удалить свойство объекта

Рекомендуется удалять только искусственные свойства. Не нативные (родные) т.к. это может привести к ошибкам или не корректной работе. К примеру при удаление cheked будет просто изменен в false!

CSS:

* .css ('стиль', "параметр стиля") - задает стили!

({стиль: "параметр стиля", стиль2: "параметр стиля2" }) - задает множество стилей через JSON!

* .height/width () - получить высоту/ширину БЕЗ учета внутренних отступов и границ!
* .innerHeight/innerWidth () - получить высоту/ширину С учетом внутренних отступов (padding)!
* .outerHeight/ outerWidth () - получить высоту/ширину C учетом внутренних отступов и границ! если передать (true), то еще и с учетом внешних
* .height/width (value) - задать высоту/ширину для всех выбранных элементов!
* .height/width (function(index, value)) - задать высоту/ширину функцией.
* . offset () - получить координаты текущей позиции относительно начала страницы
* . offset (value) - задать координаты текущей позиции относительно начала страницы

value должно быть объектом с двумя свойствами {top:nT, left:nL}

* . offset (function(index, value)) - задать координаты текущей позиции функцией относительно начала страницы

При изменение координат с помощью offset, все выбранные элементы, у которых не задан тип позиционирования, автоматически присваивается как relative!

Используя offset, получаем только координаты ПЕРВОГО элемента из выбранного!

* . offsetParent () - возвращает ближайшего родителя (аж до начала DOM), у которого задано позиционирование.
* .position () - получить координаты выбранного элемента относительно БЛИЖАЙШЕГО родителя, У КОТОРОГО ЗАДАНО ПОЗИЦИОНИРОВАНИЕ

Элементы с позиционированием играют важную роль в позиционирование всех элементов внутри них!!!

* append() - добавляет контент внутрь каждого элемента набора, добавленный контент следует за последним.

val - позволяет получать и изменять значения элементов форм.

() - возвращает значение элемента value

(valueN) - присвоить элементу значения valueN

(function(index, newVal)) - присвоить значения, возвращаемое пользовательской функцией. index - позиция элемента в наборе.

Фильтры:

Базовые фильтры:

По всему документу:

* $('E : even ') - все четные элементы Е.
* $('E : odd ') - все нечетные элементы Е
* $('E : first ') - первый элемент Е
* $('E : last ') - последний элемент Е

Для дочерних элементы

* $('E D: first-child ') - первый дочерний элемент D in E
* $('E D: last-child') - последний дочерний элемент D in E
* $('E D: only-child ') - выбирает Е с ЕДИНТСВЕННЫМ дочерним элемент D в нем.

Nth-Child:

* $('.E D: nth-child(index) ') - по нумерации в DOM
* $('#E D: nth -child(even) ') - четные
* $('E D: nth -child(odd) ') - не четные
* $('E D: nth -child(aN + b) ') - подставляет формулу an+ b, где a,b - числовые значения подставляющиеся программистом. n - счетчик, начинающийся с нуля! Пример(3n+1)

EQ:

* $('.E : eq(index) ') - выбор элемента по индексу
* $('#E : gt(index) ') - все элементы, индекс которого большего указаного (great than)
* $('E : lt(index) ') - все элементы, индекс которых меньше заданного (less than)

Not:

* $('.E : not(eq(index)) ') - выбор все элемента кроме указанного

Header:

* $(':header') - выбор всех заголовков

Animated:

* $(':animated') - выбор всех элементов, в которых происходит анимация!

Content фильтры:

Empty:

* $('.E : empty ') - выбор всеx пустых элементов

Has:

* $('.E :has(D) ') - выбор все элементов содержащих в себе D

Contains:

* $('E : contains('text inner') ') - выбор элемента который содержит в себе слово 'text inner'!!!!

Доп. фильтры jQ:

**~=:**

* $("E[AttrName~=value]") - выбор всех элементов Е с атрибутом AttrName содержащий в себе VALUE!

РАЗДЕЛЕННЫЙ ПРОБЕЛОМ!?!?!?

**!=:**

* $("E[AttrName!=value]") все элементы Е с аттрибутом , НЕ содержащий value!

Фильтры форм:

**:**button**:**

* $(":button") - выбор всех кнопок на странице

**:**checkbox**:**

* $(":checkbox") - выбор всех checkbox's на странице

**:**checked**:**

* $("input:checked") - выбор только выбранных checkbox на странице

**:**disabled**:**

* $("input:disabled") - выбор всех НЕ активных элементов на странице

**:**enabled**:**

* $(":enabled") - выбор всех активных элементов на странице

**:**file**:**

* $(":button") - выбор всех элементов с типом файл на странице

Cookie/Session

Session

- хранится на сервере. Временной промежуток хранения задается на определенное значение! По истечению которого файл подчиститься.

Cookie

- текстовый файл (ключ - значение), хранятся на клиентском компьютере. Единственный файл с которым может работать JavaScript!!!

Для куки задается определенный момент (в будущем) до которого будет хранится файл.

Добавление в куки в JS

<**script** type="text/javascript">

function mclick()

{

var n = fun1.n1.value;

var p = fun1.pass1.value;

var e = fun1.email.value;

var obj = '{"name": "' + n + '", "password":"' + p + '","email":"' + e + '"}';

var obj1 = JSON.parse(obj);

writeCoockie(obj1.name, obj1, 1);

}

function writeCoockie(name, value, days)

{

//

var exp;

if (typeof (days) != "undefined")

{

var d = new Date();

d.setTime(d.getTime() + days \* 1000 \* 60 \* 60 \* 24);

exp = "expires = " + d.toUTCString();

}

else

{

exp = "";

}

var str = name + "=" + value + ";" + exp + "; path=/";

alert(str);

document.cookie = str; /\* path/ - открыть всем \*/

}

</**script**>

* CROSS

RGB (red, green blue)

Yellow = red + green;

Magenta = red + blue;

Cyan = green + blue;

CMYK (cyan, magenta, yellow, blacK)

HSL

Бесконечный цикл , плавного изменения цвета по 360 градусам hsl!

var color = 0;

setInterval(function(){

$(document.body).css({backgroundColor:"hsl(" + ++color % 360 + ", 100%, 50%)"});

}, 100);

* Math

Полет шарика :

ball.style.left = r \* Math.cos(a) + bx + "px";

ball.style.top = r \* Math.sin(a) + by + "px";

r – radius-vector!

Бесконечный цикл с помощью %

var i = 0;

var images = ["images/1.jpg","images/2.jpg","images/3.jpg"]

$(document).ready(

function()

{

setInterval(

function()

{

setTimeout(

function()

{

$('#img1').attr("src",

images[++i % images.length]); *//БЕСКОНЕЧНЫЙ ЦИКЛ С ПОМОЩЬЮ % !!!!!!!!!!!!!!! ЗАПОМНИТЬ!*

$('#img1').attr( "class", "img1");

$('#img1').fadeTo(2000,1);

}, 1000);

$('#img1').fadeTo(900,0);

}, 7000);

});

* GIT

<https://github.com/>

https://try.github.io/levels/1/challenges/4

GIT - Локальная система контроля версий.

Перед каждым вызовом команды указываем **git !!!!**

~/ корневая папка

git init - выполняется один раз для создания репозитория.

git status – текущее состоянии

track – файлы за которыми следят

untrack – файлы за которыми не следят

git add name.\* – следить за файлом, если выполняется в первый раз. Все последующие разы команда add должна вызываться перед commit для отслеживания файлов.

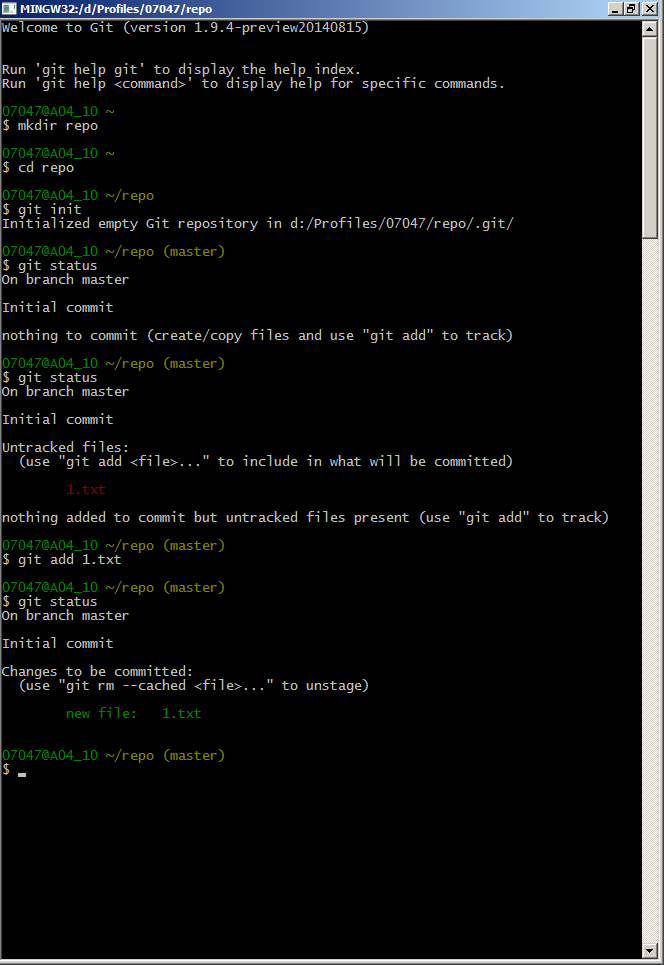
git log – выводит все commit на экран.

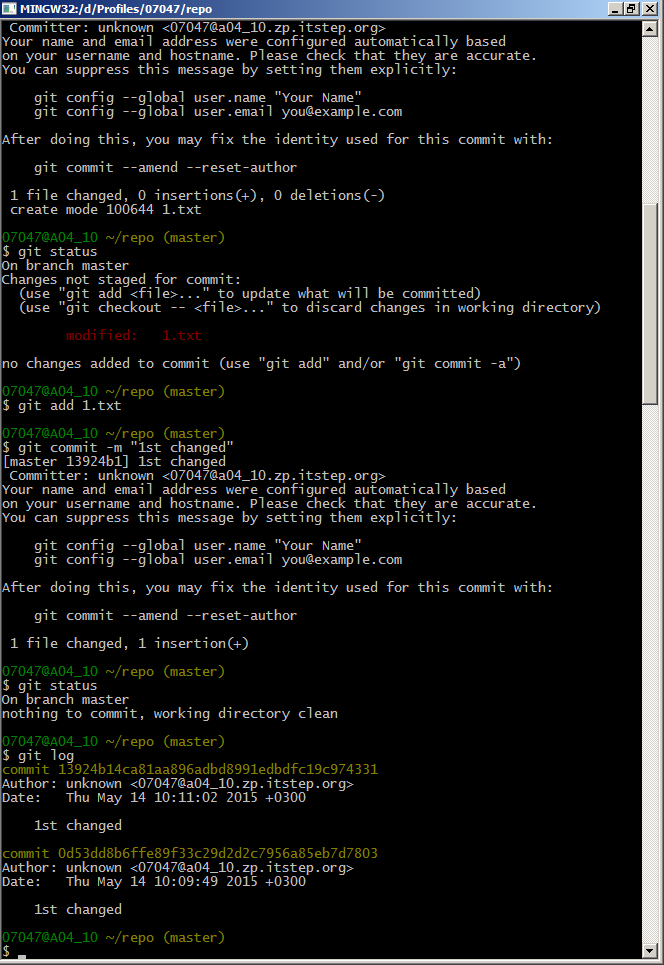
commit - это односвязный список, имеет свое имя (40рандомных символов) моментальный снимок, создание текущей копии. Beckup.

git add \*.\* git commit –m “\*\*\*\*” == git commit –a –m “\*\*\*\*\*”

Q - quit

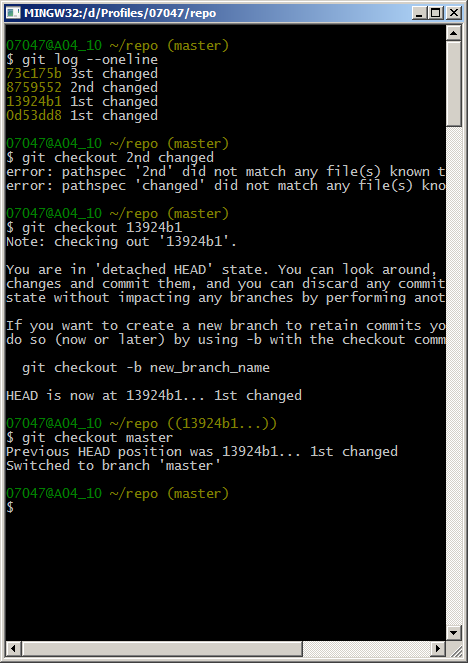
git diff – выводит изменения которые еще не были commit





git checkout Key – перейти к конкретному коммиту.

git checkout master – возвращает файл к конечному файлу



git HEAD @{1} – backup на 1 commit назад

git log --oneline – выводлит все коммиты

Ветви

Test1

Master

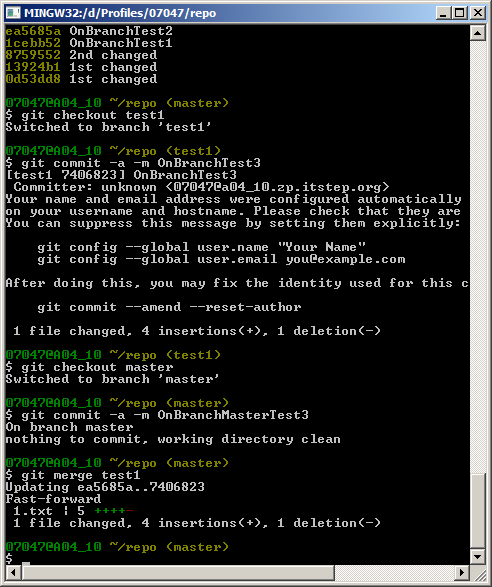
Git branch – показывает все ветви

Git branch NAME HEAD^ - создают новую ветвь с именем NAME от основного коммита.

Слияние всегда происходит в ТУ ветку в которой находишься!!

Git merge test1 – cливает в текущую ветку , ветку тест1

Перед сливанием необходимо сделать коммит



Git branch –D test1 – удаляет ветку test1

GitHUB

Синхронизация%

Git config –global USER.name “”

Git config –global USER.email“”

Git remote – выводит удаленные репозитории

Git

Как пройти через прокси

Git config --global http.proxy inet:123456 @10.3.0.3:3128

Пример:

1.Создать локальный репозиторий

2. Ввести команды git config –-global user.name & user.email

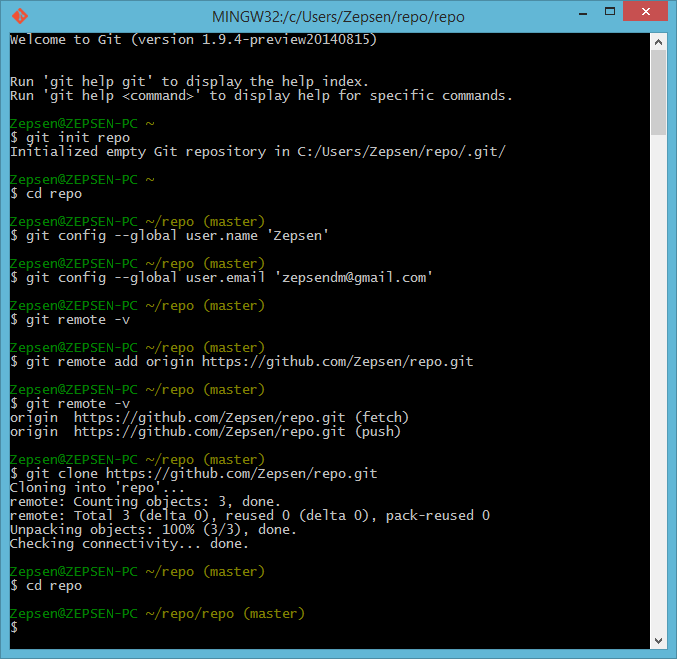
3. Ввести git remote –v и убедится что никаких репозиториев нет

4. набрать команду

Git remote add origin <https://github.com/Zepsen/repo.git>

5. git clone <https://github.com/Zepsen/repo.git>

6. cd name



git push – загрузить все на удаленный репозиторий

git pull – выгрузить из удаленного репозитория

* UML

<https://www.draw.io>

* XP

Programming

**Основные принципы**

1. **Позднее принятие решения.**

Система должна разбиваться на меньшие подсистемы. А подсистемы на еще меньшие подсистемы.

1. **Кодирование в глубину**.
2. **Истории пользователя.**
3. **План итерации.**
4. **Представитель заказчика.**
5. **Единые правила для всех.**

* UNITY

Основные понятия:

**Rigidbody** - позволяет присвоить объекту физические параметры.

Таким образом предмет падает и т.п.

**MainCamera** -

**Совет: по умолчанию игровой объект привязан к положению родителя. Это приводит к интересному побочному эффекту при использовании объекта камеры: если камера является дочерним объектом, она автоматически будет отслеживать положение родителя. Если же она является корневым объектом сцены или находится внутри пустого игрового объекта, она всегда показывает один и тот же вид. Однако если вы поместите камеру в движущийся игровой объект, она будет следовать за его передвижениями в пределах сцены. В данном случае нам нужна фиксированная камера, поэтому мы помещаем ее в пустой объект**