# 学习dcc.core之前需要知道的C#基础知识（1）

一.类自动属性

1 public class Person

2 {

3 //自动属性

4 public string Name { get; set; }

5

6 private int \_age;

7

8 public int age {

9 get { return \_age; }

10 set { \_age = value; }

11 }

12 }

Name为类的自动属性，其实跟下面age的定义方法是一样的，只不过是微软自动帮你生成了字段。

二.隐式类型

1 static void Var()

2 {

3 var person = new Person();

4

5 var p = person;

6 }

1.var类型会被编译器根据初始值的类型自动推断出具体的类型。

2. var类型不能做参数。

3.无法将null赋值给var类型。

4.语句中只声明一次变量，声明后不能更改类型。

三、匿名类

1 static void W02AnmClass()

2 {

3 var d = new { name = "张三", age = 13 };

4 var d2 = new { name = 12, age = "13" };

5 var d3 = new { name = "张三", age = "13", gender="男"};

6 }

1.匿名类本质上是编译器生成的一个泛型类。

2.当匿名类的参数个数一样的时候是共用一个泛型类。

3.当匿名类的参数个数不一样的时候是各自生成不同的泛型类。

4.匿名类初始化的时候一定要赋值。

四、默认值和命名参数

1 static void W03DefaultValue()

2 {

3 W0301();

4 W0301("李四");

5 W0301(age: 20);

6 }

7 static void W0301(string name="张三",int age=18)

8 {

9 Person person = new Person();

10 person.Name = name;

11 person.Age = age;

12 }

1.当一个方法里面的参数有默认值的时候，可以直接调用方法不用传值，编译器会自动调用参数的值。

2.当方法有默认值时，如果调用的时候只传入第一个参数，那么编译器会把第二个参数设置为默认值。

3.如果需要给某一个参数赋值，书写的格式是 参数名：值。

五、对象/集合初始化器

1 static void W0401Object()

2 {

3 Person person = new Person() {

4 Name="张三",

5 Age=23

6 };

7 }

8 static void W0402Object()

9 {

10 var personlist = new List<Person>() {

11 new Person() {Name="张三",Age=22 },

12 new Person() {Name="李四",Age=12 }

13 };

14 }

15 static void W0403Object()

16 {

17 Person[] arrperson = new Person[] {

18 new Person() {Name="张三",Age=22 }

19 };

20 }

1.对象/集合初始化器，可以在new一个对象或者集合的时候，直接在后面加{}里面对属性直接赋值或者直接new一个对象给集合。

六、匿名方法（匿名函数、匿名委托）

1 static bool Process(int p)

2 {

3 return p > 2;

4 }

5

6 static void W0401()

7 {

8 List<int> list = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

9 //直接传入一个方法名称也可以不过这个方法名称的参数以及返回值类型要跟

10 //public delegate bool Predicate<in T>(T obj);一样

11 var nlist =list.FindAll(Process);

12 foreach (var item in nlist)

13 {

14 Console.WriteLine(item);

15 }

16

17 Console.WriteLine("=================");

18 //匿名方法的写法：delegate（编写匿名方法的参数，参数有程序员自己定义）{函数的方法体代码}

19 var nlist2 = list.FindAll(delegate (int p) { return p > 3; });

20 foreach (var item in nlist2)

21 {

22 Console.WriteLine(item);

23 }

24 }

七、lambda表达式的推断以及书写方法

1 List<Person> persons = new List<Person>() {

2 new Person { Name="张三",Age=12},

3 new Person { Name="李四",Age=32},

4 new Person { Name="王五",Age=44}

5 };

6 //使用匿名方法返回符合条件的数据。

7 var nlistperson= persons.FindAll(delegate (Person person) { return person.Age > 22 && person.Name.Contains("李"); });

8 //lambda表达式的推断使用goto语句来替换匿名函数 =>

9 var nlistperson2 = persons.FindAll((Person person)=> { return person.Age > 22 && person.Name.Contains("李"); });

10

11 /\*

12 1、lambda表达式的推断使用goto语句来替换匿名函数 => 继续简化

13 2、当只有一个参数的时候可以省略（）参数的括号，当返回值只有一句语句的时候可以省略return以及{}花括号

14 3、当有多个参数的时候需要加上（），当有多条语句的时候需要添加{}每个语句后面加分号；最后返回的语句要加return

15 \*/

16 var nlistperson3 = persons.FindAll(person => person.Age > 22 && person.Name.Contains("李"));

17

18 var nlistperson4 = persons.FindAll(person => {

19 person.Age++;

20 return person.Age > 22 && person.Name.Contains("李");

21 });

22

23 foreach (var item in nlistperson4)

24 {

25 Console.WriteLine("名称：{0};年龄：{1}",item.Name,item.Age);

26 }

八、扩展方法

1 /// <summary>

2 /// 1.扩展方法必须是一个静态方法

3 /// 2.静态方法必须放在静态类中

4 /// 3.扩展方法第一个参数必须是this开头，并且指定扩展方法是哪个类型上的

5 /// 4.扩展方法只能有指定的类型才能点出来

6 /// 5.扩展方法的this后面的参数不属于 方法的参数

7 /// 6.如果扩展方法跟实例方法同名，先调用实例方法

8 /// 7.子类可以调用父类的扩展方法

9 /// 8.接口上的扩展方法可以被实现类的对象直接使用

10 /// 9.扩展方法的本质，最终还是被编译器编译成了 静态类.静态方法（）

11 /// </summary>

12 static void W0601()

13 {

14 DateTime dt = DateTime.Now;

15 string fmtstr= dt.FmtDate();

16 }

1 public static class ExtHelper

2 {

3 public static string FmtDate(this DateTime dt)

4 {

5 return dt.ToString("YYYY-MM-DD HH:MM:ss");

6 }

7 public static string FmtDate(this DateTime dt,string str)

8 {

9 return dt.ToString("YYYY-MM-DD HH:MM:ss"+str);

10 }

11 }

九、系统内置的委托

1 #region 系统内置委托

2 #region Action委托 接收参数 无法回值

3 static void W0701Action()

4 {

5 List<int> list = new List<int>() { 1, 2, 3, 4, 5, 6 };

6

7 list.ForEach(i => Console.WriteLine(i));

8 }

9 #endregion

10

11 #region Predicate 泛型委托：接收一个参数，返回bool类型

12 static void W0702Predicate()

13 {

14 var list = new List<Person>(){

15 new Person { Name = "张三", Age = 12 },

16 new Person { Name = "李四", Age = 32 },

17 new Person { Name = "王五", Age = 44 }

18 };

19

20 var nlist = list.FindAll(c => c.Age > 22);

21 //获取当前list集合中，名字为张三的对象,如果有多个只返回第一个

22

23 var person = list.Find(c => c.Name == "张三");

24 }

25 #endregion

26

27 #region Func:接收参数，返回参数，但是不固定

28 static void W0703Func()

29 {

30 var list = new List<Person>(){

31 new Person { Name = "张三", Age = 12 },

32 new Person { Name = "李四", Age = 32 },

33 new Person { Name = "王五", Age = 44 }

34 };

35

36 var nlist = list.Where(c => c.Age > 1);

37 var nlist1 = nlist.ToList();

38 nlist1.ForEach(c => Console.WriteLine(c.Name.ToString()));

39 }

40 #endregion

41

42 #region System.Comparison<T> 返回一个整数，接收两个同类型的参数

43 static void W0704Comparison()

44 {

45 var list = new List<Person>(){

46 new Person { Name = "张三", Age = 12 },

47 new Person { Name = "李四", Age = 32 },

48 new Person { Name = "王五", Age = 44 }

49 };

50

51 list.Sort((p, n) =>p.Age - n.Age);

52 list.ForEach(c=>Console.WriteLine(c.Name.ToString()+":"+c.Age.ToString()));

53 }

54

55 #endregion

56 #endregion

十、SQO方法 -标准查询运算符

创建两个类

1 public class Person

2 {

3 public int TypeID { get; set; }

4

5 public int ID { get; set; }

6 //自动属性

7 public string Name { get; set; }

8

9 private int \_age;

10

11 public int Age {

12 get { return \_age; }

13 set { \_age = value; }

14 }

15

16 public override string ToString()

17 {

18 return "ID=" + this.ID + "名称=" + this.Name + ",年龄=" + this.Age + "所属类别=" + this.TypeID;

19 }

20 }

21

22 public class PsersonType

23 {

24 public int TypeID { get; set; }

25 public string TName { get; set; }

26 }

SQO常用方法的使用 查询、排序、关联、分组等

1 #region SQO方法 -标准查询运算符

2 #region 1 where() firstordefult() LastOrDefault方法进行查找操作 (链式编程)

3 static void W0801()

4 {

5 var list = new List<Person>(){

6 new Person { Name = "张三", Age = 12 },

7 new Person { Name = "李四", Age = 32 },

8 new Person { Name = "王五", Age = 44 }

9 };

10

11 //利用where查找出list中年龄大于22或者名称叫王五

12 //链式编程

13 list.Where(c => c.Age > 5 || c.Name == "王五").ToList().ForEach(c => Console.WriteLine(c.ToString()));

14

15 //firstordefult()的演示

16 var person= list.FirstOrDefault(c => c.Age > 5 || c.Name == "王五");

17 Console.WriteLine(person.ToString());

18

19 //获取集合中满足条件的最后一个元素

20 var person2 = list.LastOrDefault(c => c.Age > 5 || c.Name == "王五");

21 Console.WriteLine(person.ToString());

22

23 }

24 #endregion

25

26 #region 2 对集合进行排序(正序和倒序)

27 static void W0802()

28 {

29 var list = new List<Person>(){

30 new Person {ID=1, Name = "张三", Age = 12,TypeID=1 },

31 new Person {ID=2,Name = "李四", Age = 32 ,TypeID=2},

32 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=1}

33 };

34 //正序

35 list.OrderBy(c => c.Age).ToList().ForEach(c => Console.WriteLine(c.ToString()));

36 Console.WriteLine("---------");

37 //倒序

38 list.OrderByDescending(c => c.Age).ToList().ForEach(c => Console.WriteLine(c.ToString()));

39 }

40 #endregion

41

42 #region 3 根据多个字段的组合排序

43 static void W0803()

44 {

45 var list = new List<Person>(){

46 new Person {ID=1, Name = "张三", Age = 12,TypeID=1 },

47 new Person {ID=2,Name = "李四", Age = 32 ,TypeID=2},

48 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=1}

49 };

50 //多个条件进行正序 先按照年龄然后按照ID正序

51 list.OrderBy(c => c.Age).ThenBy(c => c.ID).ToList().ForEach(c => Console.WriteLine(c.ToString()));

52

53 //多条件倒序

54 list.OrderByDescending(c => c.Age).ThenByDescending(c => c.ID).ToList().ForEach(c => Console.WriteLine(c.ToString()));

55 }

56 #endregion

57

58 #region 4 投影方法(Select) ( 重点 )

59 static void W0804()

60 {

61 var list = new List<Person>(){

62 new Person {ID=1, Name = "张三", Age = 12,TypeID=1 },

63 new Person {ID=2,Name = "李四", Age = 32 ,TypeID=2},

64 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=1}

65 };

66 //取出list集合里面的部分字段

67 List<string> nlist = list.Select(c => c.Name).ToList();

68

69 //重新创建一个新的list集合只包含name和age

70 list.Select(c => new { N = c.Name, A = c.Age }).ToList().ForEach(c => Console.WriteLine(c.ToString()));

71 }

72 #endregion

73

74 #region 5 分页方法 Skip(skipNum).Take(页容量) Skip():表示跳过集合中的前面多少行

75 static void W0805()

76 {

77 int rowcount = 0;

78 W0805ByPage(1, 2, out rowcount).ForEach(c => Console.WriteLine(c.ToString()));

79 }

80 static List<Person> W0805ByPage(int pageindex, int pagesize, out int rowcount)

81 {

82 var list = new List<Person>(){

83 new Person {ID=1, Name = "张三", Age = 12,TypeID=1 },

84 new Person {ID=2,Name = "李四", Age = 32 ,TypeID=2},

85 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=1}

86 };

87 // 给rowcount赋予满足条件的数据总行数

88 rowcount = list.Count();

89 //经过第几页跳过跳过多少条

90 int skipCount = (pageindex - 1) \* pagesize;

91 return list.Where(c => c.Age > 1).Skip(skipCount).Take(pagesize).ToList();

92 }

93 #endregion

94

95 #region 6 连表查询Join

96 static void W0806()

97 {

98 var list = new List<Person>(){

99 new Person {ID=1, Name = "张三", Age = 12,TypeID=1 },

100 new Person {ID=2,Name = "李四", Age = 32 ,TypeID=2},

101 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=1},

102 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=3}

103 };

104

105 var typelist = new List<PsersonType>()

106 {

107 new PsersonType(){TypeID=1,TName="老师"},

108 new PsersonType(){TypeID=3,TName="学生"},

109 new PsersonType(){ TypeID=2, TName="保安"}

110 };

111

112 //利用SQO方法Join() 来实现上述业务逻辑

113 var nlist = list.Join(typelist, p => p.TypeID, c => c.TypeID, (p, c) =>new { p.Name,p.Age,c.TName }).ToList();

114

115 nlist.ForEach(c => Console.WriteLine(c.ToString()));

116 }

117 #endregion

118

119 #region 7 分组 GroupBy()

120 static void W0807()

121 {

122 var list = new List<Person>(){

123 new Person {ID=1, Name = "张三", Age = 12,TypeID=1 },

124 new Person {ID=2,Name = "李四", Age = 32 ,TypeID=2},

125 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=1},

126 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=3}

127 };

128

129 //利用SQO方法进行分组操作

130 list.GroupBy(c => c.TypeID).ToList().ForEach(c => {

131 Console.WriteLine("----分组号{0}" + c.Key,c.ToList().Count());

132 c.ToList().ForEach(g => Console.WriteLine(g.ToString()));

133 });

134 }

135 #endregion

136 #endregion

十一、linq的常用方法

1 #region Linq使用方法

2 #region 1 Linq实现排序

3 static void w0901()

4 {

5 var list = new List<Person>(){

6 new Person {ID=1, Name = "张三", Age = 12,TypeID=1 },

7 new Person {ID=2,Name = "李四", Age = 32 ,TypeID=2},

8 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=1},

9 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=3}

10 };

11 //需求：根据list集合中的age倒序排列后取出pig对象中的Name和Age

12 //ascending :正序 descending:倒序

13 var orderbyList = (from c in list

14 orderby c.Age descending

15 select new { c.Name, c.Age }).ToList();

16 orderbyList.ForEach(c => Console.WriteLine(c.ToString()));

17 }

18 #endregion

19

20 #region 2 LinqJoin实现集合的链接查询

21 static void W0902()

22 {

23 var list = new List<Person>(){

24 new Person {ID=1, Name = "张三", Age = 12,TypeID=1 },

25 new Person {ID=2,Name = "李四", Age = 32 ,TypeID=2},

26 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=1},

27 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=3}

28 };

29

30 var typelist = new List<PsersonType>()

31 {

32 new PsersonType(){TypeID=1,TName="老师"},

33 new PsersonType(){TypeID=3,TName="学生"},

34 new PsersonType(){ TypeID=2, TName="保安"}

35 };

36

37 var nlist = (from c in list

38 join t in typelist

39 on c.TypeID equals t.TypeID

40 select new { c.Name, c.ID, t.TName }).ToString();

41

42 }

43 #endregion

44

45 #region Linq分组

46 static void W0903()

47 {

48 var list = new List<Person>(){

49 new Person {ID=1, Name = "张三", Age = 12,TypeID=1 },

50 new Person {ID=2,Name = "李四", Age = 32 ,TypeID=2},

51 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=1},

52 new Person {ID=3,Name = "王五", Age = 44 ,TypeID=3}

53 };

54

55 var groupbylist = (from c in list

56 group c by c.TypeID).ToList();

57

58 groupbylist.ForEach(c =>

59 {

60 Console.WriteLine("-----分组号=" + c.Key);

61 c.ToList().ForEach(g => Console.WriteLine(g.ToString()));

62 });

63

64 }

65 #endregion

66 #endregion