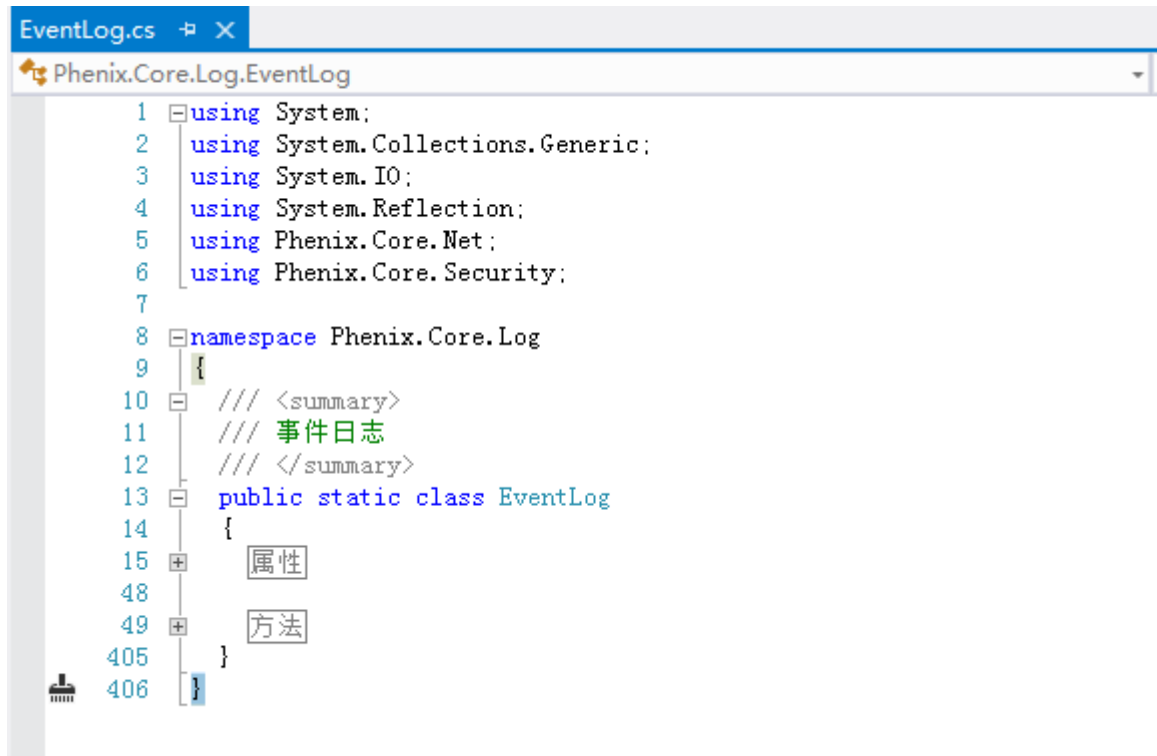


一个好的程序代码一定要是清晰易懂的，程序只写一次，但以后会有无数次的阅读。因此，编写出好读的程序代码是很重要的，本文会介绍编写代码的一些方法。

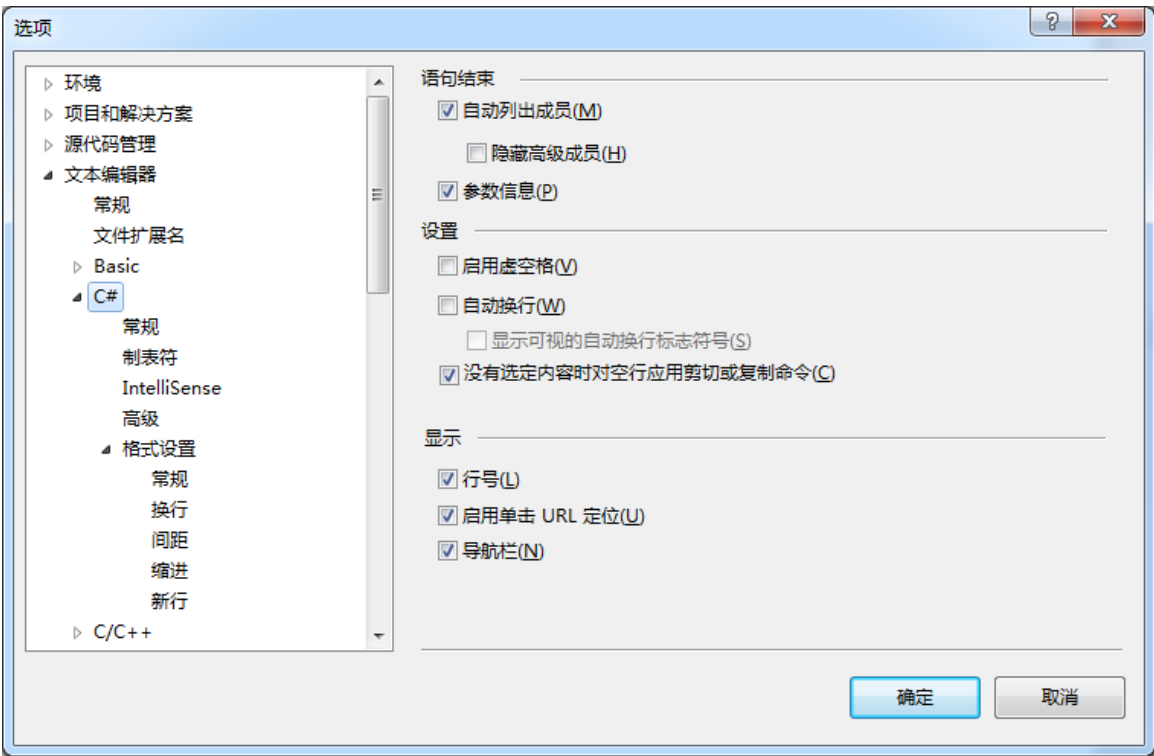
1 排版格式一致化、编码习惯一贯性

你及你的合作伙伴，要在一个项目（解决方案）里自始至终保持一贯的排版格式。请养成在编写完一个类时，将光标移到类的末尾，重输一次最末一个大括号（触发 IDE 的自动排版），再提交到配置管理工具的习惯：



```
EventLog.cs
Phenix.Core.Log.EventLog

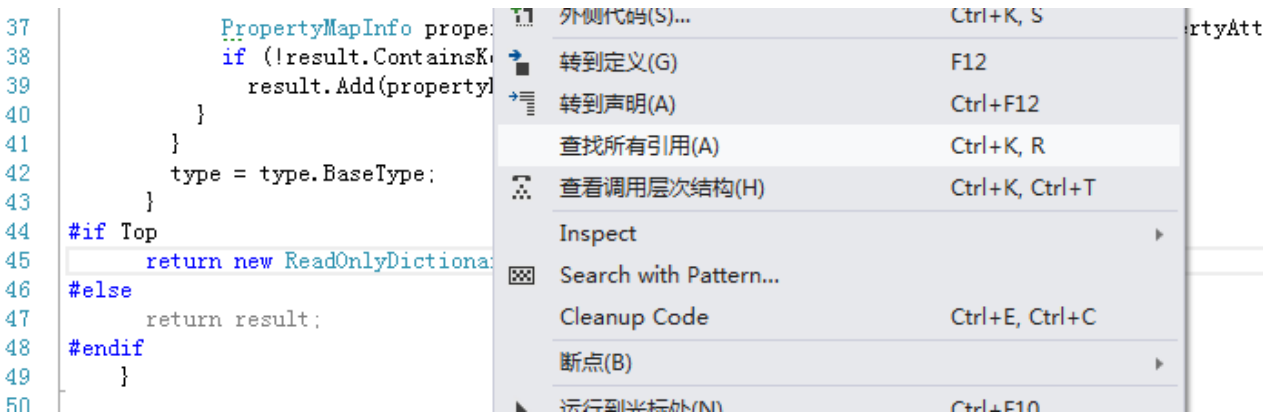
1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using System.Reflection;
5 using Phenix.Core.Net;
6 using Phenix.Core.Security;
7
8 namespace Phenix.Core.Log
9 {
10     /// <summary>
11     /// 事件日志
12     /// </summary>
13     public static class EventLog
14     {
15         // 属性
16
17         // 方法
18
19         // ...
20
21         // ...
22
23         // ...
24
25         // ...
26
27         // ...
28
29         // ...
30
31         // ...
32
33         // ...
34
35         // ...
36
37         // ...
38
39         // ...
40
41         // ...
42
43         // ...
44
45         // ...
46
47         // ...
48
49         // ...
50
51         // ...
52
53         // ...
54
55         // ...
56
57         // ...
58
59         // ...
60
61         // ...
62
63         // ...
64
65         // ...
66
67         // ...
68
69         // ...
70
71         // ...
72
73         // ...
74
75         // ...
76
77         // ...
78
79         // ...
80
81         // ...
82
83         // ...
84
85         // ...
86
87         // ...
88
89         // ...
90
91         // ...
92
93         // ...
94
95         // ...
96
97         // ...
98
99         // ...
100
101         // ...
102
103         // ...
104
105         // ...
106
107         // ...
108
109         // ...
110
111         // ...
112
113         // ...
114
115         // ...
116
117         // ...
118
119         // ...
120
121         // ...
122
123         // ...
124
125         // ...
126
127         // ...
128
129         // ...
130
131         // ...
132
133         // ...
134
135         // ...
136
137         // ...
138
139         // ...
140
141         // ...
142
143         // ...
144
145         // ...
146
147         // ...
148
149         // ...
150
151         // ...
152
153         // ...
154
155         // ...
156
157         // ...
158
159         // ...
160
161         // ...
162
163         // ...
164
165         // ...
166
167         // ...
168
169         // ...
170
171         // ...
172
173         // ...
174
175         // ...
176
177         // ...
178
179         // ...
180
181         // ...
182
183         // ...
184
185         // ...
186
187         // ...
188
189         // ...
190
191         // ...
192
193         // ...
194
195         // ...
196
197         // ...
198
199         // ...
200
201         // ...
202
203         // ...
204
205         // ...
206
207         // ...
208
209         // ...
210
211         // ...
212
213         // ...
214
215         // ...
216
217         // ...
218
219         // ...
220
221         // ...
222
223         // ...
224
225         // ...
226
227         // ...
228
229         // ...
230
231         // ...
232
233         // ...
234
235         // ...
236
237         // ...
238
239         // ...
240
241         // ...
242
243         // ...
244
245         // ...
246
247         // ...
248
249         // ...
250
251         // ...
252
253         // ...
254
255         // ...
256
257         // ...
258
259         // ...
260
261         // ...
262
263         // ...
264
265         // ...
266
267         // ...
268
269         // ...
270
271         // ...
272
273         // ...
274
275         // ...
276
277         // ...
278
279         // ...
280
281         // ...
282
283         // ...
284
285         // ...
286
287         // ...
288
289         // ...
290
291         // ...
292
293         // ...
294
295         // ...
296
297         // ...
298
299         // ...
300
301         // ...
302
303         // ...
304
305         // ...
306
307         // ...
308
309         // ...
310
311         // ...
312
313         // ...
314
315         // ...
316
317         // ...
318
319         // ...
320
321         // ...
322
323         // ...
324
325         // ...
326
327         // ...
328
329         // ...
330
331         // ...
332
333         // ...
334
335         // ...
336
337         // ...
338
339         // ...
340
341         // ...
342
343         // ...
344
345         // ...
346
347         // ...
348
349         // ...
350
351         // ...
352
353         // ...
354
355         // ...
356
357         // ...
358
359         // ...
360
361         // ...
362
363         // ...
364
365         // ...
366
367         // ...
368
369         // ...
370
371         // ...
372
373         // ...
374
375         // ...
376
377         // ...
378
379         // ...
380
381         // ...
382
383         // ...
384
385         // ...
386
387         // ...
388
389         // ...
390
391         // ...
392
393         // ...
394
395         // ...
396
397         // ...
398
399         // ...
400
401         // ...
402
403         // ...
404
405     }
406 }
```



一般情况下，大家采用 IDE 的默认设置就可以了（这对于大的团队尤其适用）。

这除了让你的代码让人看得赏心悦目外，主要是为维护工作带来一定的便利。精准的缩进间距和恰到好处、清晰的注解，都可以充分说明你的编码意图，完全可以代替设计文档（不必花费时间画流程图，除非你在编码前需要借此厘清思路）；有规则的代码行就像你的指纹一样，也有利于用 IDE 的“查找和替换”工具来定位，因为有时候“查找所有引用”工具太局限而不大好用：

```
return new ReadOnlyDictionary<string, PropertyMapInfo>(result);
```



以上案例，如果直接查找 `ReadOnlyDictionary` 的引用，检索到的信息太多、太杂：

```

lapInfo.cs - (68, 14) : return new ReadOnlyDictionary<string, CriteriaFieldMapInfo>(result);
nfo.cs - (47, 14) : return new ReadOnlyDictionary<string, MethodMapInfo>(result);
Info.cs - (45, 14) : return new ReadOnlyDictionary<string, PropertyMapInfo>(result);
ry.cs - (1121, 14) : return new ReadOnlyDictionary<string, TableColumnInfo>(result);
ry.cs - (1644, 14) : return new ReadOnlyDictionary<string, BusinessCodeFormat>(result);
ry.cs - (183, 14) : return new ReadOnlyDictionary<long, DepartmentInfo>(result);
ry.cs - (218, 14) : return new ReadOnlyDictionary<long, PositionInfo>(result);
ry.cs - (285, 18) : return new ReadOnlyDictionary<string, TableFilterInfo>(result);
ry.cs - (321, 14) : return new ReadOnlyDictionary<string, RoleInfo>(result);
ry.cs - (355, 14) : return new ReadOnlyDictionary<string, SectionInfo>(result);
ry.cs - (631, 50) : configValues != null ? new ReadOnlyDictionary<string, string>(configValues) : null,
ry.cs - (682, 100) : new ReadOnlyDictionary<string, AssemblyClassPropertyInfo>(classPropertyInfos), new ReadOnlyDictionary<
ry.cs - (682, 21) : new ReadOnlyDictionary<string, AssemblyClassPropertyInfo>(classPropertyInfos), new ReadOnlyDictionary<
ry.cs - (886, 46) : configValues != null ? new ReadOnlyDictionary<string, string>(configValues) : null,
ry.cs - (937, 17) : new ReadOnlyDictionary<string, AssemblyClassPropertyInfo>(classPropertyInfos), new ReadOnlyDictionary<
ry.cs - (937, 96) : new ReadOnlyDictionary<string, AssemblyClassPropertyInfo>(classPropertyInfos), new ReadOnlyDictionary<
cs - (128, 14) : return new ReadOnlyDictionary<string, RoleInfo>(result);
cs - (157, 14) : return new ReadOnlyDictionary<string, RoleInfo>(result);
cs - (186, 14) : return new ReadOnlyDictionary<string, SectionInfo>(result);
cs - (225, 14) : return new ReadOnlyDictionary<string, Identity>(result);

```

此时，我们借助“文本检索”的话：



就能精准筛选到有相同特征的代码行：

```

<string, “, 子文件夹, 查找结果 1, 整个解决方案, “”
y\DataDictionary.cs(285):      return new ReadOnlyDictionary<string, TableFilterInfo>(result);
y\DataDictionary.cs(321):      return new ReadOnlyDictionary<string, RoleInfo>(result);
y\DataDictionary.cs(355):      return new ReadOnlyDictionary<string, SectionInfo>(result);
y\DataDictionary.cs(1121):     return new ReadOnlyDictionary<string, TableColumnInfo>(result);
y\DataDictionary.cs(1644):     return new ReadOnlyDictionary<string, BusinessCodeFormat>(result);
y\DataSecurity.cs(128):      return new ReadOnlyDictionary<string, RoleInfo>(result);
y\DataSecurity.cs(157):      return new ReadOnlyDictionary<string, RoleInfo>(result);
y\DataSecurity.cs(186):      return new ReadOnlyDictionary<string, SectionInfo>(result);
y\DataSecurity.cs(225):      return new ReadOnlyDictionary<string, IIdentity>(result);
y\Workflow.cs(54):      return new ReadOnlyDictionary<string, WorkflowInfo>(result);
a\Database.cs(440):      return new ReadOnlyDictionary<string, Table>(result);
assMemberHelper.cs(792):      return new ReadOnlyDictionary<string, IPropertyInfo>(value);
assMemberHelper.cs(914):      return new ReadOnlyDictionary<string, IMethodInfo>(value);
riteriaFieldMapInfo.cs(68):      return new ReadOnlyDictionary<string, CriteriaFieldMapInfo>(result);
thodMapInfo.cs(47):      return new ReadOnlyDictionary<string, MethodInfo>(result);
opertyMapInfo.cs(45):      return new ReadOnlyDictionary<string, PropertyMapInfo>(result);
ng\ClassMemberHelper.cs(319):      return new ReadOnlyDictionary<string, IPropertyInfo>(value);
ng\CriteriaFieldMapInfo.cs(47):      return new ReadOnlyDictionary<string, CriteriaFieldMapInfo>(result);
ng\PropertyMapInfo.cs(43):      return new ReadOnlyDictionary<string, PropertyMapInfo>(result);
文件: 1718

```

要能做到精准，不会有遗漏，所有代码的排版格式必须一致。否则得多费点神，比如用表达式来查询，或用各种可能的组合查个几遍：

```

return new ReadOnlyDictionary<string, PropertyMapInfo>(result); //new 后多出空格
return new ReadOnlyDictionary <string, PropertyMapInfo>(result); //<前多出空格
return new ReadOnlyDictionary< string, PropertyMapInfo>(result); //string 前多出空格

```

另外，是代码的编码习惯要一贯性。如果经常别出心裁、换着花样码字（比如变量命名规则、赋值语句写法等等）的话，就人为减少了代码的特征文本，很难通过“文本检索”查询到：

```

var result = new ReadOnlyDictionary<string, PropertyMapInfo>(tmp); //先赋值给变量
return result; //再返回变量值

```

注释也是一样，写法也要保持一贯性，比如应避免同一含义的内容在多处有不同的单词来表达：

```

118         }
119         catch (TypeLoadException)
120         {
121             //Ignore
122         }
123         catch (NotSupportedException)
124         {
125             //Ignore
126         }
127         catch (ArgumentException)
128         {
129             //Ignore
130         }
131         catch (Exception ex)
132         {

```

在解决方案里，只要有是吃掉 catch 的代码段，都用“//Ignore”打上注释。这样，一旦有需要重构这些代码的时候，检索“//Ignore”就能快速定位到它们。

请不要相信你的代码永远不可能被改写，这个时候如果你能快速定位到它们并完成任务，就是你编程能力的具体体现。

比如：

```

protected override void DoExecute(System.Data.Common.DbTransaction transaction)
{
    using (var cmd = Phenix.Core.Data.DbCommandHelper.CreateCommand(transaction))
    {
        cmd.CommandType = System.Data.CommandType.Text;
        cmd.CommandText = string.Format(@"update CTN_CONTAINER_PRESENT_INFO t
                                         set t.CPI_REMARK='{0}'
                                         where t.cpi_id = {1}", _control, _id);

        cmd.ExecuteNonQuery();
    }
}

```

以上代码段，其实隐含着一种编码模式，成为了这个编码者的一个习惯写法，黄底黑字部分是其代码特征。任何地方有遇到此类场景，他应该照搬这个模式，也就是大家喜闻乐见的“拷贝-粘贴”。替换掉的代码段，只是黄底黑字部分：

```

protected override void DoExecute(System.Data.Common.DbTransaction transaction)
{
    using (var cmd = Phenix.Core.Data.DbCommandHelper.CreateCommand(transaction))
    {
        cmd.CommandType = System.Data.CommandType.Text;

```

```

        cmd.CommandText = string.Format(@"update CTN_CONTAINER_PRESENT_INFO t
                                         set t.CPI_REMARK=' {0}'
                                         where t.cpi_id = {1}", _control, _id);

        cmd.ExecuteNonQuery();
    }
}

```

在没有现成的框架收纳这个模式的前提下，这种做法本身并没有什么问题。只是需要注意的是，请在项目中一以贯之地保持这种风格，千万不要改动特征文本，比如把 `cmd` 改成 `command`，把 `var` 改成 `DbCommand`，把 `Phenix.Core.Data.DbCommandHelper.CreateCommand` 缩写为 `CreateCommand`，甚至 1 行代码折成 2 行的事情，也都不要轻易改变，要改就全部一起改掉。因为，只有在你保持一贯的编码风格的前提下，代码里才会有丰富的特征文本，才能给到“文本检索”做关键字，批量重构到。

在此，顺便演示下对上述代码的重构。

因为 Phenix 已封装了这种编码模式，以上代码段可重构为：

```

protected override void DoExecute(System.Data.Common.DbTransaction transaction)
{
    Phenix.Core.Data.DbCommandHelper.ExecuteNonQuery(transaction,
        string.Format(@"update CTN_CONTAINER_PRESENT_INFO t
                        set t.CPI_REMARK=' {0}'
                        where t.cpi_id = {1}", _control, _id);
}

```

接下来还需进一步优化。

因为第一点，带参数的 SQL 语句，可以提高数据库 select 引擎的缓存命中率，也能避免出现安全漏洞，也能避免 SQL 可能拼入特殊字符（比如引号）而抛出异常（往往很难测试到），所以：

```

protected void DoExecute(System.Data.Common.DbTransaction transaction)
{
    Phenix.Core.Data.DbCommandHelper.ExecuteNonQuery(transaction,
        @"update CTN_CONTAINER_PRESENT_INFO t
        set t.CPI_REMARK=:CPI_REMARK
        where t.cpi_id=:cpi_id",
        ParamValue.Input("CPI_REMARK", _control), ParamValue.Input("cpi_id", _id));
}

```

注：示例中 SQL 参数的定义，虽然用了 Oracle 的 “:” 标识，但如果系统切换到 SQL Server 数据库的话，Phenix 会自动将它们切换为 “@”。反之亦然，所以能做到业务系统的数据库无关性。

第二点，如果使用 Phenix.Core.Data.EntityListBase<T, TEntity>类的话，可利用它的批量持久化函数 UpdateRecord()、DeleteRecord()，代码会更加简练，减少了因拼 SQL 带来的差错和硬码问题：

```
ContainerPresentInfoEasyList.UpdateRecord(transaction,  
    p => p.ID == _id, ContainerPresentInfoEasy.REMARKProperty.Set(_control));
```