【平台】C#开发规范

目录

[【平台】C#开发规范 1](#_Toc12026680)

[命名规范 1](#_Toc12026681)

[命名法 1](#_Toc12026682)

[总体原则 1](#_Toc12026683)

[起名原则 1](#_Toc12026684)

[大小写规则 2](#_Toc12026685)

[编码规范 2](#_Toc12026686)

[注释 2](#_Toc12026687)

[缩写 2](#_Toc12026688)

[空行 3](#_Toc12026689)

[换行 3](#_Toc12026690)

[声明 3](#_Toc12026691)

[枚举 3](#_Toc12026692)

[设计规范 4](#_Toc12026693)

[SRP 单一职责原则（Single responsibility principle） 4](#_Toc12026694)

[OCP 开放封闭原则（Open-Closed Principle） 4](#_Toc12026695)

[ISP 接口隔离原则（Interface Segregation Principle） 4](#_Toc12026696)

[DIP 依赖倒置原则（Dependency Inversion Principle） 4](#_Toc12026697)

[LSP 里氏替换原则（Liskov Substitution Principle） 5](#_Toc12026698)

[LOD迪米特法则（Law of Demeter) 5](#_Toc12026699)

[DRY原则（Don't Repeat Yourself） 5](#_Toc12026700)

命名规范

命名法

* Pascal   
  识符的首字母和后面连接的每个单词的首字母都大写。例如： BackColor。
* Camel   
  标识符的首字母小写，而每个后面连接的单词的首字母都大写。例如：backColor。

总体原则

* 名称应该说明它“是什么”而不是“如何”
* 要使用表现力强的名称   
  这是为了帮助人们阅读，尽量使人不分析上下文就能知道其含义。

起名原则

* 类：名词或名词词组   
  有时也可以用形容词词组。
* 字段、属性：名词、名词词组
* 方法、事件：动词或动词词组
* 布尔变量：应该包含 Is   
  如：IsFound。
* 集合的变量名使用复数形式 (List后缀也属于附属形式)  
  如：students、studentList。
* 相同意义的变量应该使用相同的变量名   
  如：appId和applicationId应该统一成相同的变量名；userId和uid应该统一成userId。

大小写规则

* Interface：IUpperCamelCase
* Type parameters：TUpperCamelCase
* 参数、本地变量、本地常量：lowerCamelCase
* private字段：\_lowerCamelCase
* 其他（命名空间、类、属性、方法）：UpperCamelCase

编码规范

注释

* 代码和注释必须同时修改   
  避免注释滞后，形成“错误的注释”。
* 避免避免杂乱、多余的或不适当的注释   
  如：如一长串的星号、搞笑段子。
* 减少注释。   
  使用准确的标识符名称，和清晰的代码逻辑。从而达到“代码即注释”。

缩写

* 慎用缩写
* 不要使用拼音缩写
* 不要创造未被普遍接受、意义不明确的缩写   
  比如将roleId缩写为rid。

空行

* 不能出现连续多个空行
* 以下情况下使用空行：
  + 方法与方法之间；
  + 属性与属性之间；
  + 不同的逻辑块之间；
  + 变量声明与语句之间。
* 列宽   
  代码列宽控制在 110 字符左右。

换行

当表达式超出或即将超出规定的列宽，遵循以下规则进行换行：

* 在逗号后换行
* 在操作符前换行

声明

* 一行只作一个声明   
  不推荐： int x, y;
* 声明时就对其做初始化。
* 避免不同层次间定义同名变量   
  因为作用域有优先级，所以不同层次间的同名变量并不会报错，但这会影响我们阅读代码。
* 不要使用 public 或 protected的实例字段，而是使用属性访问器

枚举

* 枚举值一般不要从0开始（注意：默认是从0开始）
* 特殊情况下，0本身就是一个有意义的值，那么也可以使用0

设计规范

SRP 单一职责原则（Single responsibility principle）

单一原则最易理解却是最难实现的一条规则，随着系统演进，很多时候，忘记了类设计的初衷。

OCP 开放封闭原则（Open-Closed Principle）

* 对扩展是开放
* 对更改是封闭

极端情况下，交付的代码都是不可修改的，同一业务域内的需求变化，通过模块或类的   
扩展来实现。

ISP 接口隔离原则（Interface Segregation Principle）

客户端不应该依赖它不需要的接口， 一个类对另一个类的依赖应该建立在最小的接口上；   
举例：ICache接口定义了GetString、SetString、GetHash、SetHash 4个方法，有一个RedisCache类实现了这个接口，后来又增加了一个MemcacheCache，但它没有hash类型数据所以没必要实现GetHash和SetHash方法，这时候不能强制让其继承ICache接口。

DIP 依赖倒置原则（Dependency Inversion Principle）

高层模块不应该依赖低层模块，两者都应该依赖其抽象；   
抽象不应该依赖细节；   
细节应该依赖抽象。

LSP 里氏替换原则（Liskov Substitution Principle）

任何使用基类的地方，都可以透明的使用其子类，这主要是指继承+透明(安全，不会出现行为不一致)。   
只有子类能够完全替代它们的基类时，使用基类的函数才能被安全的使用。

LOD迪米特法则（Law of Demeter)

又叫作最少知识原则（Least Knowledge Principle 简写LKP）。

迪米特法则的注意事项：

* 在类的划分上，应该创建有弱耦合的类；
* 在类的结构设计上，每一个类都应当尽量降低成员的访问权限；
* 在类的设计上，只要有可能，一个类应当设计成不变类；
* 在对其他类的引用上，一个对象对其它对象的引用应当降到最低；
* 尽量降低类的访问权限；
* 谨慎使用序列化功能；
* 不要暴露类成员，而应该提供相应的访问器(属性)。

DRY原则（Don't Repeat Yourself）

不要图省事而做简单的复制粘贴，应该将能够公共的部分抽象出来，封装成工具类或者用“abstraction”类来抽象公有的东西，降低代码的耦合性，这样不仅提高代码的灵活性、健壮性以及可读性，也方便后期的维护或者修改。

（完）