

C# 快速参考

类型、LINQ、async/await、集合、OOP 要点

基础

Hello World

```
Console.WriteLine("Hello, World!"); // top-level (C# 10+)
// Classic: class Program { static void Main() { ... } }
```

构建与运行

```
dotnet new console -n MyApp # create project
dotnet run # compile and run
dotnet build # compile only
```

变量与常量

```
int x = 42;
var name = "Alice"; // type inference
const double Pi = 3.14159;
readonly int maxRetries = 3; // set once, in ctor
```

类型

值类型

int	32 位有符号整数
long	64 位有符号整数
float	32 位浮点 (后缀 f)
double	64 位浮点
decimal	128 位高精度 (后缀 m)
bool	true / false
char	16 位 Unicode 字符

引用类型

string	不可变 UTF-16 文本
object	所有类型的基类
dynamic	跳过编译期类型检查
int[]	整数数组
List<T>	泛型列表 (System.Collections.Generic)

可空类型与元组

```
int? age = null; // nullable value type
string? name = null; // nullable reference (C# 8+)
var point = (X: 1, Y: 2); // named tuple
Console.WriteLine(point.X);
```

字符串特性

```
string name = "World";
string msg = $"Hello, {name}!"; // interpolation
string path = @"C:\Users\file.txt"; // verbatim
string raw = ""raw "string" here""; // raw (C# 11+)
```

控制流

If / Else

```
if (x > 0) Console.WriteLine("positive");
else if (x == 0) Console.WriteLine("zero");
else Console.WriteLine("negative");
```

Switch 与模式匹配

```
string label = x switch {
    > 0 => "positive", 0 => "zero", _ => "negative"
};
if (obj is string s && s.Length > 0) { } // pattern match
```

循环

```
for (int i = 0; i < 10; i++) { }
foreach (var item in collection) { }
while (condition) { }
do { } while (condition);
```

类

类定义

```
public class Person {
    public string Name { get; set; }
    public int Age { get; init; } // init-only (C# 9+)
    public Person(string name, int age) { Name = name; Age = age; }
}
```

Records (C# 9+)

```
public record Point(double X, double Y);
var p1 = new Point(1, 2);
var p2 = p1 with { X = 3 }; // non-destructive copy
// auto: Equals, GetHashCode, ToString, deconstruct
```

继承

```
public abstract class Shape { public abstract double Area(); }
public class Circle(double r) : Shape {
    public override double Area() => Math.PI * r * r;
}
```

访问修饰符

public	任意位置可访问
private	仅同一类 (成员默认)
protected	同一类及派生类
internal	同一程序集 (类默认)
protected internal	同一程序集或派生类

接口

接口定义

```
public interface IShape {
    double Area();
    double Perimeter() => 0; // default impl (C# 8+)
}
public class Rect(double w, double h) : IShape { public double
Area() => w * h; }
```

常用接口

IEnumerable<T>	支持迭代 (foreach、LINQ)
IDisposable	确定性清理 (using 语句)
IComparable<T>	排序的自然顺序
IEquatable<T>	值相等比较
ICloneable	对象克隆

LINQ

方法语法

```
var result = numbers
    .Where(n => n > 3)
    .OrderBy(n => n)
    .Select(n => n * 2)
    .ToList();
```

查询语法

```
var result = from n in numbers
              where n > 3
              orderby n
              select n * 2;
```

常用 LINQ 方法

.Where(pred)	过滤元素
.Select(func)	投影 / 转换元素
.OrderBy(key)	升序排序
.GroupBy(key)	按键分组
.First() / .FirstOrDefault()	第一个元素 (或默认值)
.Any(pred)	任意元素匹配则为 true
.Count()	元素数量
.Sum() / .Average()	数值聚合
.Distinct()	去重
.SelectMany(func)	展开嵌套集合

Async/Await

异步方法

```
public async Task<string> FetchAsync(string url) {
    using var client = new HttpClient();
    return await client.GetStringAsync(url);
}
```

Task 组合

```
var results = await Task.WhenAll(task1, task2, task3);
var first = await Task.WhenAny(task1, task2);
```

异步模式

Task	异步无返回值
Task<T>	异步返回 T 类型结果
ValueTask<T>	同步快速路径的轻量 Task
await foreach	异步迭代 IAsyncEnumerable<T>
CancellationToken	异步操作的协作取消

集合

常用集合

List<T>	动态数组, 索引访问快
Dictionary<K, V>	哈希表, O(1) 按键查找
HashSet<T>	唯一元素, O(1) 查找
Queue<T>	FIFO 队列
Stack<T>	LIFO 栈
LinkedList<T>	双向链表
SortedDictionary<K, V>	按键排序 (树结构)

Dictionary 用法

```
var dict = new Dictionary<string, int> {
    ["Alice"] = 90, ["Bob"] = 85
};
dict.TryGetValue("Alice", out int score);
foreach (var (key, val) in dict) { }
```

不可变集合

```
using System.Collections.Immutable;
var list = ImmutableList.Create(1, 2, 3);
var newList = list.Add(4); // returns new list
```

属性

属性语法

```
public string Name { get; set; }
public int Age { get; private set; }
public string Email { get; init; } // init-only
public string Display => $"{Name} ({Age})"; // computed
```

索引器

```
public double this[int row, int col] {  
    get => data[row, col];  
    set => data[row, col] = value;  
}
```

属性模式

<code>{ get; set; }</code>	读写自动属性
<code>{ get; }</code>	只读 (仅构造函数可设置)
<code>{ get; init; }</code>	初始化后只读 (C# 9+)
<code>{ get; private set; }</code>	公开可读, 私有可写
<code>=> expression</code>	表达式体 (计算属性)

异常

Try / Catch / Finally

```
try { int result = int.Parse(input); }  
catch (FormatException ex) { Console.Error.WriteLine(ex.Message); }  
catch (Exception ex) when (ex is not OutOfMemoryException) { }  
finally { /* always executes */ }
```

Using 语句

```
using var file = File.OpenRead("data.txt");  
// file.Dispose() called automatically at scope end  
// equivalent to try/finally with Dispose()
```

常见异常

ArgumentNullException	传入了 null 参数
ArgumentOutOfRangeException	参数超出有效范围
InvalidOperationException	当前状态下操作无效
NullReferenceException	解引用了 null 对象
KeyNotFoundException	字典中未找到键
NotImplementedException	方法尚未实现

自定义异常

```
public class AppException : Exception {  
    public int Code { get; }  
    public AppException(string msg, int code)  
        : base(msg) { Code = code; }  
}
```