

機能一覧

<https://ufcpp.net/study/csharp/> より
メンドイのでVer2.0以前は省略

| # | サポートバージョン | 分類 | 使う機会 | 機能 | 実装例 |
|---|-----------|-------|------|---------------|--|
| 1 | 3.0 | 変数宣言 | ヘビロテ | 暗黙的型付け var | <code>var n = 1; var x = 1.0; var s = "test";</code> |
| 2 | 3.0 | メソッド | たまに | 拡張メソッド | <code>static class StringExtensions { public static string ToggleCase(this string s) 中身省略 } string s = "This Is a Test String."; string s1 = StringExtensions.ToggleCase(s); // 通常の呼び出し方。 string s1 = s.ToggleCase(); // 拡張メソッド呼び出し。</code> |
| 3 | 3.0 | 式 | たまに | ラムダ式 | <code>Func<int, bool> p = n => n > 0;</code> |
| 4 | 3.0 | 変数初期化 | ヘビロテ | 初期化子 | <code>Point p = new Point{ X = 0, Y = 1 }; List<int> list = new List<int> {1, 2, 3};</code> |
| 5 | 3.0 | 型 | たまに | 匿名型 | <code>var x = new { FamilyName = "糸色", FirstName="望"};</code> |
| 6 | 3.0 | 型 | ヘビロテ | 暗黙的型付け配列 | <code>int[] array = new[] {1, 2, 3, 4};</code> |
| 7 | 3.0 | 式 | たまに | LINQ | <code>var 学籍番号前半名 = from p in 学生名簿 where p.学生番号 <= 15 orderby p.学生番号 select p.名;</code> |
| 8 | 3.0 | プロパティ | ヘビロテ | 自動プロパティ | <code>public string Name { get; set; }</code> |

| | | | | | |
|----|-----|--------|------|----------------------|--|
| 9 | 3.0 | メソッド | 絶対ない | パースシャルメソッド | メンドイので省略（使う機会はないはず） |
| 10 | 4.0 | 型 | たまに | 動的型付け変数 dynamic | static dynamic GetX(dynamic obj) { return obj.X; } |
| 11 | 4.0 | 引数 | ヘビロテ | オプション引数・名前付き引数 | static int Sum(int x = 0, int y = 0, int z = 0) { return x + y + z; } |
| 12 | 4.0 | ジェネリック | 多分ない | ジェネリックの共変性 | public interface IEnumerable< out T > { ... } |
| 13 | 4.0 | ジェネリック | 多分ない | ジェネリックの反変性 | public delegate void Action< in T > (T arg); |
| 14 | 4.0 | アンマネージ | 多分ない | COM 相互運用時の特別処理 | メンドイので省略 |
| 15 | 5.0 | 非同期 | たまに | 非同期処理 async/await | メンドイので省略 |
| 16 | 5.0 | 属性 | 多分ない | Caller Info 属性 | public static class Trace { public static void WriteLine(string message, [CallerFilePath] string file = "", [CallerLineNumber] int line = 0, [CallerMemberName] string member = "") { var s = string.Format("{0}:{1} - {2}: {3}", file, line, member, message); Console.WriteLine(s); } } |
| 17 | 5.0 | 文 | 不明 | foreach の仕様変更 | メンドイので省略 |
| 18 | 6.0 | プロパティ | ヘビロテ | 自動プロパティの拡張 初期化子 | class Point { public int X { get; set; } = 10; public int Y { get; set; } = 20; } |

| | | | | | |
|----|-----|-------|------|---------------------------------|--|
| 19 | 6.0 | プロパティ | ヘビロテ | 自動プロパティの拡張 getter のみの自動プロパティ | <pre> class Point { // ↓ set; を消すだけ public int X { get; } = 10; public int Y { get; } = 20; } </pre> |
| 20 | 6.0 | メソッド | ヘビロテ | expression-bodied な関数メンバー | <pre> public class Point { public int X { get; set; } public int Y { get; set; } public Point(int x = 0, int y = 0) { X = x; Y = y; } public int InnerProduct(Point p) => X * p.X + Y * p.Y; public static Point operator -(Point p) => new Point(-p.X, -p.Y); } public class Polygon { private Point[] _vertexes; public int Count => _vertexes.Length; public Point this[int i] => _vertexes[i]; } </pre> |
| 21 | 6.0 | 演算子 | ヘビロテ | null 条件演算子 | <pre> public static int? X(Sample s) => s?.Name?.Length; static char? X(string s, int i) => s?[i]; </pre> |
| 22 | 6.0 | 文字列 | ヘビロテ | 文字列挿入 | <pre> var formatted = \$"({x}, {y})"; </pre> |
| 23 | 6.0 | 演算子 | ヘビロテ | nameof 演算子 | <pre> Console.WriteLine(nameof(MyClass)); </pre> |

| | | | | | |
|----|-----|-------|------|------------------------------|---|
| 24 | 6.0 | 宣言 | ヘビロテ | using static | <pre>using System; using static System.Math; class Program { static void Main() { var pi = 2 * Asin(1); Console.WriteLine(PI == pi); } }</pre> |
| 25 | 6.0 | 変数初期化 | ヘビロテ | インデックス初期化子 | <pre>var dic = new Dictionary<string, int> { ["one"] = 1, ["two"] = 2, };</pre> |
| 26 | 6.0 | 例外 | たまに | 例外フィルター | <pre>catch (ArgumentException e) when (e.ParamName == "x")</pre> |
| 27 | 6.0 | 例外 | たまに | catch/finally 句内での await 演算子 | <pre>public static async Task XAsync() { try { await SomeAsyncMethod(); } catch (InvalidOperationException e) { using (var s = new StreamWriter("error.txt")) await s.WriteAsync(e.ToString()); } finally { using (var s = new StreamWriter("trace.txt")) await s.WriteAsync("XAsync done."); } }</pre> |
| 28 | 6.0 | 変数初期化 | たまに | 拡張メソッドでコレクション初期化子 | メンドイので省略 |

| | | | | | |
|----|-----|---------|------|------------------------------------|--|
| 29 | 6.0 | コンパイル | 多分ない | ユーザー定義コード解析に対する #pragma warning | メンドイので省略 |
| 30 | 6.0 | 構造体 | たまに | 構造体のプロパティ初期化 | <pre> struct Point { public int X { get; private set; } public Point(int x) { // C# 5.0まではエラーに。 X = x; } } </pre> |
| 31 | 6.0 | コンストラクタ | たまに | コンストラクターの循環参照 | <pre> class C { public C(int x) : this() { } public C() : this(0) { } // C# 6ではコンパイル エラーに } </pre> |
| 32 | 6.0 | 文 | たまに | 「確実な初期化」の判定改善 | <pre> static void Main() { int x; if (false && x == 3) // C# 5.0まではエラーに { x = x + 1; // ここはC# 5.0までもOK } } </pre> |
| 33 | 6.0 | 型 | ヘビロテ | 列挙型の基底型 | <pre> enum X : System.Int32 // C# 5.0まではエラーに { A, B, C, } </pre> |
| 34 | 6.0 | 変数 | 多分ない | 変数の「意味不変」ルール | メンドイので省略 |
| 35 | 6.0 | メソッド | 多分ない | オーバーロード解決の改善 | メンドイので省略 |
| 36 | 6.0 | コンパイル | 不明 | 内部的な最適化 | メンドイので省略 |

| | | | | | |
|----|-----|----|------|--------|---|
| 37 | 7.0 | 型 | ヘビロテ | タプル | <pre>// タプルを使って2つの戻り値を返す static (int count, int sum) Tally(IEnumerable<int> items) { var count = 0; var sum = 0; foreach (var x in items) { sum += x; count++; } return (count, sum); } static void Main() { var data = new[] { 1, 2, 3, 4, 5 }; var t = Tally(data); Console.WriteLine(\$"{t.sum}/{t.count}"); }</pre> |
| 38 | 7.0 | 型 | ヘビロテ | 分解 | <pre>var (count, sum) = Tally(data);</pre> |
| 39 | 7.0 | 引数 | ヘビロテ | 出力変数宣言 | <pre>public void GetCoordinate(out int x, out int y) { x = X; y = Y; } p.GetCoordinate(out var x, out var y);</pre> |

| | | | | | |
|----|-----|----|------|----------------|--|
| 40 | 7.0 | 文 | 多分ない | 型スイッチ | <pre> case int n when n > 0: Console.WriteLine("正の数の時にここに来る " + n); // ただし、上から順に判定するので、7 の時には来なくなる break; case int n: Console.WriteLine("整数の時にここに来る" + n); // 同上、0 以下の時にしか来ない break; </pre> |
| 41 | 7.0 | 変数 | 多分ない | 式の中での変数宣言 | <pre> if (obj is string s) </pre> <p>メンドイので省略</p> |
| 42 | 7.0 | 変数 | 多分ない | 値の破棄 | メンドイので省略 |
| 43 | 7.0 | 変数 | ヘビロテ | 参照戻り値と参照ローカル変数 | <pre> using System; class Program { static void Main() { var x = 10; var y = 20; // x, y のうち、大きい方の参照を返す。この例の場合 y を参照。 ref var m = ref Max(ref x, ref y); // 参照の書き換えなので、その先の y が書き換わる。 m = 0; Console.WriteLine(\$"{x}, {y}"); // 10, 0 } static ref int Max(ref int x, ref int y) { if (x < y) return ref y; else return ref x; } } </pre> |

| | | | | | |
|----|-----|-------|------|--------------------------|---|
| 44 | 7.0 | メソッド | たまたま | ローカル関数 | <pre>static void Main() { // Main 関数の中で、ローカル関数 f を定義 int f(int n) => n >= 1 ? n * f(n - 1) : 1; Console.WriteLine(f(10)); }</pre> |
| 45 | 7.0 | 非同期 | たまたま | 非同期メソッドの戻り値に任意の型を使用するように | <pre>using System; using System.Threading.Tasks; class Program { static async ValueTask<int> XAsync(Random r) { if (r.NextDouble() < 0.99) { // 99% ここを通る。 // この場合、await が1度もなく、非同期処理にならない。 // 非同期処理じゃないのに Task<int> のインスタンスが作られるのはもったいない return 1; } // こちらは本当に非同期処理なので、Task<int> が必要。 await Task.Delay(100); return 0; } }</pre> |
| 46 | 7.0 | 変数初期化 | たまたま | 数値リテラルの改善 | <pre>byte bitMask = 0b1100_0000;</pre> |

| | | | | | |
|----|-----|-----|------|-------------|---|
| 47 | 7.0 | 例外 | たまに | throw 式 | <pre>// 式形式メンバーの中(=> の直後) static void A() => throw new NotImplementedException(); static string B(object obj) { // null 合体演算子(??)の後ろ var s = obj as string ?? throw new ArgumentException(nameof(obj)); // 条件演算子(?:)の条件以外の部分 return s.Length == 0 ? "empty" : s.Length < 5 ? "short" : throw new InvalidOperationException("too long"); }</pre> |
| 48 | 7.0 | 式 | たまに | 式形式のメンバーの拡充 | <pre>class Counter { static int x; // コンストラクター Counter() => x++; // デストラクター ~Counter() => x--; }</pre> |
| 49 | 7.1 | 非同期 | 多分ない | 非同期Main | <pre>static Task<int> Main() static Task<int> Main(string[] args) static Task Main() static Task Main(string[] args)</pre> |
| 50 | 7.1 | 式 | 多分ない | default 式 | <pre>static async Task DefaultExpression(CancellationToken c = default) { while (c != default && !c.IsCancellationRequested) { await Task.Delay(1000); Console.WriteLine("."); } }</pre> |

| | | | | | |
|----|-----|--------|------|-------------------------------|---|
| 51 | 7.1 | 型 | ヘビロテ | タプル要素名の推論 | <pre> var x = 1; var y = 2; var t = (x, y); </pre> |
| 52 | 7.1 | ジェネリック | たまに | ジェネリック型に対するパターン マッチング (型スイッチ) | <pre> static void M<T>(T x) { switch (x) { case int i: break; case string s: break; } } class Base { } class Derived1 : Base { } class Derived2 : Base { } class Derived3 : Base { } // こういう、型制約付きのやつですら 7.0 ではダメだった static void N<T>(T x) where T : Base { switch (x) { case Derived1 d: break; case Derived2 d: break; case Derived3 d: break; } } </pre> |

| | | | | | |
|----|-----|-------|------|-------------------|---|
| 53 | 7.2 | 変数初期化 | たまた | 先頭区切り文字 | <pre>// C# 7.0 から書ける var b1 = 0b1111_0000; var x1 = 0x0001_F408; // C# 7.2 から書ける // b, x の直後に _ 入れてもOKに var b2 = 0b_1111_0000; var x2 = 0x_0001_F408;</pre> |
| 54 | 7.2 | 引数 | たまた | 非末尾名前付き引数 | <pre>// C# 7.2 // 末尾以外でも名前を書けるように Sum(x: 1, 2, 3);</pre> |
| 55 | 7.2 | クラス | たまた | private protected | メンドイので省略 |
| 56 | 7.2 | 演算子 | たまた | 条件演算子での ref 利用 | <code>x > y ? ref x : ref y</code> |
| 57 | 7.2 | 変数宣言 | ヘビロテ | ref readonly | <pre>public static Quaternion operator *(in Quaternion a, in Quaternion b) static ref readonly int Max(in int x, in int y) { ref readonly var t = ref x; ref readonly var u = ref y; if (t < u) return ref u; else return ref t; }</pre> |
| 58 | 7.2 | 引数 | たまた | 演算子のin引数 | <pre>public static Complex operator +(in Complex a, in Complex b) => new Complex(a.X + b.X, a.Y + b.Y);</pre> |
| 59 | 7.2 | メソッド | ヘビロテ | 参照渡し of 拡張メソッド | <pre>// 構造体の書き換えを拡張メソッドでやりたい場合に ref 引数が見える public static void Conjugate(ref this Quaternion q) // コピーを避けたい場合に in 引数が見える public static Quaternion Rotate(in this Quaternion p, in Quaternion q)</pre> |

| | | | | | |
|----|-----|--------|------|-----------------|--|
| 60 | 7.2 | 構造体 | ヘビロテ | readonly struct | <pre>// 構造体自体に readonly を付ける readonly struct Point { // フィールドには readonly が必須 public readonly int X; public readonly int Y; public Point(int x, int y) => (X, Y) = (x, y); // readonly を付けない場合と違って、以下のような this 書き換えも不可 //public void Set(int x, int y) => this = new Point(x, y); }</pre> |
| 61 | 7.2 | アンマネージ | 多分ない | 安全な stackalloc | <pre>// Span<byte> で受け取ることで、new (配列)を stackalloc (スタック確保)に変更できる Span<byte> buffer = stackalloc byte[BufferSize];</pre> |
| 62 | 7.2 | 構造体 | たまに | ref 構造体 | <pre>// ref 構造体を持てるのは ref 構造体だけ ref struct RefStruct { private Span<int> _span; //OK }</pre> |
| 63 | 7.3 | 演算子 | たまに | タプルの ==, != 比較 | <pre>void M((int a, (int x, int y) b) t) { // このタプル == 比較は、 Console.WriteLine(t == (1, (2, 3))); // こんな感じで、メンバーごとの == を && で繋いだものに展開される。 Console.WriteLine(t.a == 1 && t.b.x == 2 && t.b.y == 3); }</pre> |

| | | | | | |
|----|-----|--------|------|--|---|
| 64 | 7.3 | 変数初期化 | たまたま | ref 再代入 | <pre> int x = 1; int y = 2; // x を参照。 ref var r = ref x; // このとき、r に対する代入は x に反映される。 r = 10; // x が 10 になる。 // これが ref 再代入。 // r が y を参照するようになる。 r = ref y; // 今度は、r に対する代入が y に反映される。 r = 20; // y が 20 になる。 </pre> |
| 65 | 7.3 | 変数宣言 | 多分ない | 式中での変数宣言(使える場所の拡充) | <pre> var q = from s in new[] { "a", "abc", "112", "132", "451", null } where s is string x && x.Length > 1 where int.TryParse(s, out var x) && (x % 3) == 0 select s; </pre> |
| 66 | 7.3 | ジェネリック | 多分ない | ジェネリック型引数に対する Enum、Delegate、unmanaged 制約 | メンドイので省略 |
| 67 | 7.3 | メソッド | 多分ない | オーバーロード解決の改善 | メンドイので省略 |
| 68 | 7.3 | 変数初期化 | 多分ない | stackalloc 初期化子 | <pre> // 初期化子。{ } を使って初期値を与えられる。 Span<int> x1 = stackalloc int[3] { 0xEF, 0xBB, 0xBF }; // 初期化子があるとき、サイズは省略可能。 Span<int> x2 = stackalloc int[] { 0xEF, 0xBB, 0xBF }; // 初期化子から推論できるときは型名も省略可能。 Span<int> x3 = stackalloc[] { 0xEF, 0xBB, 0xBF }; </pre> |
| 69 | 7.3 | アンマネージ | 多分ない | ユーザー定義型の fixed ステートメント利用 | メンドイので省略 |

| | | | | | |
|----|-----|--------|------|----------------------------------|---|
| 70 | 7.3 | プロパティ | たまに | 自動プロパティのバック フィールドに対する field 属性指定 | <pre>using System; class XAttribute : Attribute { } class Sample { [field:X] // 自動実装で生成されるフィールドに対する属性の指定 public int AutoProperty { get; }</pre> |
| 71 | 7.3 | アンマネージ | 多分ない | 固定長バッファの読み書きで、fixed ステートメント不要に | メンドイので省略 |
| 72 | 8.0 | メソッド | 多分ない | 再帰パターン | <pre>public class Point { public int X { get; set; } public int Y { get; set; } public Point(int x = 0, int y = 0) => (X, Y) = (x, y); public void Deconstruct(out int x, out int y) => (x, y) = (X, Y); } class Program { static int M(object obj) => obj switch { 0 => 1, int i => 2, Point (1, _) => 4, // new! 位置パターン。 Point { X: 2, Y: var y } => y, // new! プロパティ パターン。 _ => 0 }; };</pre> |
| 73 | 8.0 | 式 | 多分ない | switch 式 | <pre>public int Compare(int? x, int? y) => (x, y) switch { (int i, int j) => i.CompareTo(j), ({ }, null) => 1, (null, { }) => -1, (null, null) => 0 };</pre> |

| | | | | | |
|----|-----|--------|------|------------------|---|
| 74 | 8.0 | 演算子 | 多分ない | null 合体代入 (??=) | <pre>static void M(string s = null) { s ??= "default string"; Console.WriteLine(s); }</pre> |
| 75 | 8.0 | 文字列 | たまに | @\$ | メンドイので省略 |
| 76 | 8.0 | アンマネージ | 多分ない | アンマネージなジェネリック構造体 | <pre>using System.Collections.Generic; class Program { unsafe static void Main() { var kv = new KeyValuePair<int, int>(1, 2); KeyValuePair<int, int>* pkv = &kv; } }</pre> |