

VB.NET スライド教材

基礎から実践的な開発パターンまで



2025年8月14日

目次

- 1 ⓘ はじめに
- 2 </> 基本構文
- 3 📁 オブジェクト指向プログラミング
- 4 🏠 .NET Framework統合
- 5 ⚠ エラー処理とデバッグ
- 6 📄 データアクセス
- 7 📁 Windowsフォーム開発
- 8 ⚙ 実践的な開発パターン
- 9 ? よくある質問（FAQ）
- 10 📄 参考リンク・資料

このVB.NET学習資料では、基礎から実践的な開発パターンまで段階的に学習を進められるよう構成されています。

進捗状況: 2/12

はじめに

VB.NETは.NETプラットフォーム上で動作する強力なプログラミング言語です。読みやすい構文と豊富な機能により、初心者から上級者まで幅広い開発者に適しています。





VB.NETの特徴

- ✓ 英語に近い自然で読みやすい構文
- ✓ 大文字小文字を区別しない識別子
- ✓ Optionalパラメータによる柔軟な関数定義
- ✓ With文による効率的なオブジェクト操作

学習目標

VB.NETの基礎から実践的な開発技術まで体系的に学習し、実際のアプリケーション開発に活用できるスキルを身につけます。

適用領域

-  Windowsアプリケーション開発
-  Webアプリケーション開発
-  API開発・サービス作成
-  データベース連携システム

.NETエコシステムでの位置づけ

VB.NETは.NET Frameworkの一部として、C#と同じ機能を提供しながら、より読みやすく理解しやすい構文を特徴としています。

- C#と完全な相互運用性
- 同じ.NET CLR上で実行
- 共通の基本クラスライブラリを使用

基本構文

変数宣言とデータ型

D Dim文による変数宣言

Dim name As String = "太郎"

T 基本データ型

Integer, String, Boolean, Date, Decimal

制御構造

I 条件分岐

If-Then-Else, Select Case文

F 繰り返し処理

For-Next, While, Do-Loop文

VB.NET構文の特徴

英語に近い自然な記述で、コードが読みやすく理解しやすいのが特徴です。

プロシージャとファンクション

→ **Sub** : 値を返さない処理

← **Function** : 値を返す処理

⚙ **Optional** : 省略可能なパラメータ

↔ **ByRef/ByVal** : 参照渡し・値渡し

名前空間の活用

📁 **Namespace** : 名前空間の定義

⬇ **Imports** : 名前空間のインポート

👤 **階層構造** : System.IO.Fileなど

💡 コーディングのポイント

- Option Strict Onで型安全性を確保
- PascalCaseで命名規則を統一
- 適切なコメントで可読性向上

オブジェクト指向プログラミング

クラスとオブジェクトの基本

- C Class定義**
Public Class Person - End Class
- N コンストラクタ**
Public Sub New() - オブジェクト初期化

プロパティとメソッド

- P Property**
Get/Set アクセサーでデータ操作
- M Method**
Public Function/Sub で機能実装

カプセル化の重要性

Private フィールドとPublic プロパティを使い分けてデータを保護します。

継承とインターフェース

-  **Inherits** : 基底クラスから継承
-  **Implements** : インターフェース実装
-  **Overrides** : メソッドのオーバーライド
-  **Overloads** : メソッドのオーバーロード

アクセス修飾子

 **Public/Friend**

外部からアクセス可能

 **Private/Protected**

内部・継承先のみ

★ ポリモーフィズム

同じインターフェースで異なる実装を提供し、柔軟で拡張性の高いコードを実現

.NET Framework統合

.NET CLRとの関係

- C 共通言語ランタイム (CLR)**
VB.NETコードをILコードにコンパイル
- J JITコンパイル**
実行時にネイティブコードに変換

基本クラスライブラリ (BCL)

- S System名前空間**
基本型、コレクション、I/O機能
- L LINQ統合**
統合クエリ言語でデータ操作

ガベージコレクション

自動メモリ管理により、メモリリークを防ぎ安全なアプリケーションを構築できます。

相互運用性

-  **C#との連携**：同じアセンブリで共存可能
-  **COM連携**：既存COMコンポーネント利用
-  **P/Invoke**：Win32 API呼び出し
-  **Web Services**：SOAPやRESTサービス

.NETエコシステム



パフォーマンスの利点

- ネイティブコード並みの高速実行
- メモリ効率的な動作
- 豊富な最適化機能

エラー処理とデバッグ

Try-Catch-Finally構文

- T Try ブロック**
例外が発生する可能性があるコード
- C Catch ブロック**
特定の例外タイプをキャッチ
- F Finally ブロック**
必ず実行されるクリーンアップ処理

例外の種類

- A ArgumentException**
不正な引数が渡された場合
- N NullReferenceException**
Nothingオブジェクトにアクセス

▲ エラー処理のベストプラクティス

具体的な例外タイプをキャッチし、適切なエラーメッセージを提供しましょう。

デバッグツールの活用

- 1 ブレークポイント**
コードの実行を一時停止して状態確認
- 2 ウォッチウィンドウ**
変数の値をリアルタイムで監視
- 3 ステップ実行**
一行ずつコードを実行して動作確認

ログ記録とトレース

-  **Debug.WriteLine**：デバッグ情報出力
-  **Trace.Write**：実行トレース記録
-  **EventLog**：システムイベントログ
-  **Console.WriteLine**：コンソール出力

データアクセス

ADO.NETの基本

- C Connection**
データベースへの接続管理
- M Command**
SQLコマンドの実行
- R DataReader**
高速な前方専用データ読み取り
- S DataSet/DataTable**
メモリ内データ表現とキャッシュ

LINQ活用



統合クエリ言語

LINQ to Objects, LINQ to SQL, LINQ to Entitiesで統一されたクエリ構文を使用できます。

Entity Frameworkの活用

- ORM機能**：オブジェクト-リレーショナルマッピング
- Code First**：コードからDBスキーマ生成
- Database First**：既存DBからモデル生成
- Migration**：データベーススキーマの更新管理

データベース接続パターン

Connected

DataReader使用
常時接続状態

Disconnected

DataSet使用
切断型アーキテクチャ

パフォーマンス最適化

- 接続プーリングの活用
- パラメータ化クエリの使用
- 適切なインデックス設計
- 遅延読み込み (Lazy Loading)

Windowsフォーム開発

フォームデザイナーの使用

- D** **ドラッグ&ドロップ**
ツールボックスからコントロールを配置
- P** **プロパティウィンドウ**
コントロールの属性を設定

イベント駆動プログラミング

- E** **イベントハンドラー**
ユーザー操作に対する処理
- H** **Handles句**
イベントとメソッドの関連付け

主要コントロール



RAD開発の利点

Rapid Application Development - ビジュアル設計により高速なプロトタイプ作成が可能です。

MDIアプリケーション

-  **親フォーム**：MDI Container設定
-  **子フォーム**：MdiParentプロパティ設定
-  **メニューマージ**：統合メニューシステム
-  **ウィンドウ管理**：子ウィンドウの操作

レイアウト管理

-  **Anchor**：コントロールの固定位置
-  **Dock**：コンテナ内での配置
-  **TableLayoutPanel**：表形式レイアウト
-  **FlowLayoutPanel**：流動的配置

👍 ベストプラクティス

- ユーザビリティを重視した画面設計
- データバインディングの活用
- 適切な例外処理の実装
- リソースの適切な管理と解放

実践的な開発パターン

MVVMパターンの実装

- M Model**
データとビジネスロジック
- V View**
ユーザーインターフェース
- V ViewModel**
ViewとModelの仲介役

非同期プログラミング

- A Async/Await**
非同期処理の簡潔な記述
- T Task<T>**
非同期操作の結果を表現

パフォーマンス最適化

非同期処理を活用してUIの応答性を保ち、ユーザー体験を向上させます。

ユニットテストの作成

- 1 MSTest Framework**
Visual Studio統合テストフレームワーク
- 2 テストメソッド作成**
TestMethod属性でテストを定義
- 3 Assert文による検証**
期待値と実際の値を比較検証

設計パターンの活用

-  **Factory Pattern**：オブジェクト生成の抽象化
-  **Singleton Pattern**：単一インスタンス管理
-  **Observer Pattern**：イベント通知システム

生産性向上のヒント

- ・依存性注入（DI）の活用
- ・設定ファイルによる外部化
- ・ログ記録の戦略的実装
- ・継続的インテグレーション導入

よくある質問 (FAQ)

VB.NETとVB6の違い

- F .NET Framework基盤**
完全なオブジェクト指向・マネージドコード
- E 例外処理**
Try-Catch構文による構造化エラー処理

C#との選択基準

</> VB.NET

読みやすい英語風構文
初学者に優しい

C C#

C系言語風構文
コンパクトな記述

💡 学習曲線について

VB.NETは自然言語に近い構文で学習しやすく、.NETの強力な機能をすべて利用できます。

移行パスと学習順序

- ▶ **1. 基本構文習得**: 変数、制御構造、関数
- 📦 **2. オブジェクト指向**: クラス設計と継承
- 📄 **3. データアクセス**: ADO.NET、LINQ
- 🖥️ **4. UI開発**: Windowsフォーム

トラブルシューティング


- ⚠️ **Option Strict On**: 型変換エラーの回避
- 🔌 **メモリリーク**: Using文でリソース管理
- 🔍 **デバッグ情報**: 適切なログ出力設定
- 🛡️ **セキュリティ**: 入力値検証の徹底

📚 継続学習のポイント

- 公式ドキュメントの定期的な確認
- オープンソースプロジェクトへの参加
- 技術コミュニティでの情報交換
- 実践的なプロジェクト経験積み重ね

参考リンク・資料


公式ドキュメント

 Microsoft VB.NET ドキュメント

 .NET API リファレンス

 Microsoft Learn

コミュニティフォーラム


 Stack Overflow

 VB.NET Forums


学習を継続するために


実際のプロジェクトに取り組み、コミュニティで質問や情報交換を積極的に行いましょう。

次のステップへの案内

 ASP.NET Core MVC：Webアプリケーション開発

 C#プログラミング：C系言語への展開

 Entity Framework：高度なデータアクセス

 Blazor：モダンWebUI開発

関連技術スタック



🏆 学習完了おめでとうございます！

VB.NETの基礎から実践的な開発パターンまで学習しました。今後は実際のプロジェクトで経験を積み重ね、さらなるスキル向上を目指しましょう。