```
        aPoint.X = x * 2
        aPoint.Y = y * 2
        Return aPoint   '構造体を返す
    End Function
    Private Sub button1_Click(sender As Object, e As EventArgs) Handles
button1.Click
        Dim pt As StructPoint

        'メソッドを呼び出して、構造体を戻り値として受け取る
        pt = twicePoint(100, 200)
        MessageBox.Show("xの位置 : " & pt.X & vbLf & "yの位置 : " & pt.Y, "
戻り値の構造体の値 ")
    End Sub
```

4-10 非同期

Tips 186

Level ●

対応 EXP PRO

タスクを作成する

ここがポイントです！

バックグラウンドで動作する処理（Taskクラス、Startメソッド）

ユーザーからのアクションを処理するときに、**同期処理**と**非同期処理**があります。

例えば、同期処理ではボタンをクリックした後にファイルの読み書きや印刷などをしている間に画面の操作ができなくなります。

しかし、非同期処理を使うと、バックグラウンドで操作をしている間、ユーザーは画面の操作ができるようになります。

非同期処理の場合には、バックグラウンドの処理を行っている途中や、終了した後のタイミングを考える必要がありますが、画面操作がよりスムースになるためアプリケーション作成によく使われます。

Visual Basicでは非同期処理を行うための仕組みがすでに備わっています。バックグラウンド処理には**Taskクラス**を利用し、非同期処理を制御するために**Async/Awaitキーワード**を使います。

```
Dim 変数 = New Task( ラムダ式 );
```

あるいは、次のようにも記述できます。

```
Dim 変数 = New Task( AddressOf 処理関数 );
```

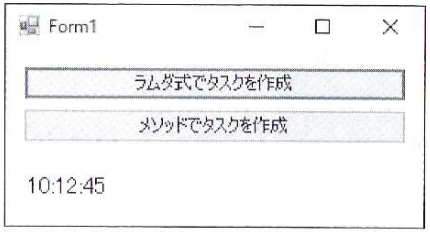
Taskクラスのインスタンスを生成するときに、引数のないメソッドを渡します。この処理メソッドは**ラムダ式**やクラスのメソッドとして渡すことができます。

ラムダ式を利用すると、インスタンスを生成しているメソッド内の内部変数をラムダ式内で使うことができます。メソッド処理関数を別に用意した場合は、変数の独立性がよくなります。

リスト1では、ラムダ式でバックグラウンド処理を記述しています。

リスト2では、バックグラウンド処理を別メソッドとして記述しています。

▼実行結果



リスト1 タスクをラムダ式で生成する（ファイル名：pg186.sln、Form1.vb）

```
Private _task As Task
Private Sub button1_Click(sender As Object, e As EventArgs) Handles
button1.Click
    ' ラムダ式を使う場合
    _task = New Task(Sub()
                         For i As Integer = 0 To 59
                             Me.Invoke(New Action(Sub()
                                 label1.Text = DateTime.Now.
ToString("HH:MM:ss")
                                                 End Sub))
                             System.Threading.Thread.Sleep(1000)
                         Next

                     End Sub)
    _task.Start()
End Sub
```

リスト2 タスクをメソッドで生成する（ファイル名：pg186.sln、Form1.vb）

```
Private _task As Task
Private Sub button2_Click(sender As Object, e As EventArgs) Handles
button2.Click
    ' メソッドを使う場合
    _task = New Task(AddressOf OnWork)
    _task.Start()
End Sub
' バックグラウンドプロセス
Private Sub OnWork()
    For i As Integer = 0 To 59
        Me.Invoke(New Action(Sub()
            label1.Text = DateTime.Now.ToString("HH:MM:ss")
                                    End Sub))
        System.Threading.Thread.Sleep(1000)
    Next
End Sub
```

Tips 187

タスクを作成して実行する

Level ●
対応 EXP PRO

ここがポイントです！

バックグラウンドで動作する処理（Taskクラス、Factoryプロパティ、StartNewメソッド）

タスクを生成すると同時に処理を開始するためには、TaskクラスのFactoryプロパティにある**StartNewメソッド**を使います。

StartNewメソッドは、非同期処理が行われるため、**Awaitキーワード**で処理待ちをすることができます。

StartNewメソッドに処理関数は、Taskクラスのコンストラクターと同じようにラムダ式や引数を持たないメソッドを渡すことができます。

```
Task.Factory.StartNew( ラムダ式 )
```

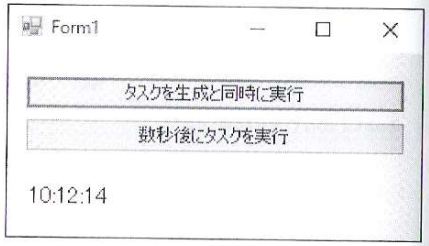
あるいは、次のようにも記述できます。

```
Task.Factory.StartNew( AddressOf
処理関数 )
```

リスト1では、タスクを生成すると同時に処理関数を動かしています。

リスト2では、Taskオブジェクトを生成した後、5秒後にタスクを実行しています。

▼実行結果



リスト1 タスクを実行する（ファイル名：pg187.sln、Form1.vb）

```
' タスク生成と同時に実行
Private Async Sub button1_Click(sender As Object, e As EventArgs)
Handles button1.Click
    Await Task.Factory.StartNew(
        Sub()
            For i As Integer = 0 To 59
                Me.Invoke(New Action(
                    Sub()
                        label1.Text = DateTime.Now.
ToString("HH:MM:ss")
                    End Sub))
                System.Threading.Thread.Sleep(1000)
            Next
        End Sub)
End Sub
```


リスト2 タスクを数秒後に実行する（ファイル名：pg187.sln、Form1.vb）

```
' 数秒後に実行
Private Async Sub button2_Click(sender As Object, e As EventArgs)
Handles button2.Click
    Dim task1 As New Task(
        Sub()
            For i As Integer = 0 To 59
                Me.Invoke(New Action(
                    Sub()
                        label1.Text = DateTime.Now.
ToString("HH:MM:ss")
                    End Sub))
                System.Threading.Thread.Sleep(1000)
            Next
        End Sub)
    Await Task.Delay(5000)
    task1.Start()
End Sub
```

Tips 188

戻り値を持つタスクを作成する

Level ● ●
対応 EXP PRO

ここがポイントです！

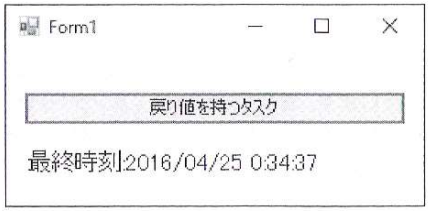
処理終了時に戻り値を設定（Taskクラス、Returnキーワード、Awaitキーワード）

バックグラウンドの処理を行った後に、元のメソッドに戻り値を返すためには**Returnキーワード**を使います。

StartNewメソッドを使ってタスクを起動した場合には、非同期処理を待つための**Awaitキーワード**が使えます。このときの戻り値をタスクの処理関数内で渡すことができます。

リスト1では、10秒間経過した後に最終時刻をReturnキーワード元のメソッドに戻しています。

▼実行結果



リスト1 タスクから戻り値を取得する（ファイル名：pg188.sln、Form1.vb）

```
Private Async Sub button1_Click(sender As Object, e As EventArgs)
Handles button1.Click
```

```
    Dim ret = Await Task.Factory.StartNew(
        Function()
            For i As Integer = 0 To 9
                Me.Invoke(New Action(
                    Sub()
                        label1.Text = String.Format("{0}秒経過", i)
                    End Sub))
                System.Threading.Thread.Sleep(1000)
            Next
            Return DateTime.Now
        End Function)
    label1.Text = String.Format("最終時刻:{0}", ret)
End Sub
```

Tips 189

タスクの完了を待つ

Level ●●

対応 EXP PRO

ここがポイントです！

処理終了時に戻り値を設定（Taskクラス、Runメソッド、Awaitキーワード）

アプリケーション内で非同期のタスクを実行する場合に、メソッド内で**処理待ちをする方法**と**処理待ちを行わない方法**が使えます。

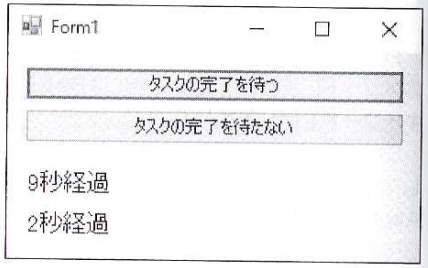
処理待ちをしたいときには、**Awaitキーワード**を使います。Awaitキーワードを使うと、バックグラウンド処理を順序よく記述できます。

複数のタスクを同時に実行させたい場合は、Awaitキーワードを付けずに実行させます。

リスト1では、2つのタスクを順序のまま実行します。最初のonTask1メソッド処理が終わった後にonTask2メソッドが実行されます。

リスト2では、2つのタスクが同時に実行されます。

▼実行結果



リスト1 タスクの完了を待つ場合（ファイル名：pg189.sln、Form1.vb）

```
Private Sub onTask1()
    For i As Integer = 0 To 9
        Me.Invoke(New Action(
            Sub()
                label1.Text = String.Format("{0}秒経過", i)
            End Sub))
        System.Threading.Thread.Sleep(1000)
    Next
End Sub
Private Sub onTask2()
    For i As Integer = 0 To 9
        Me.Invoke(New Action(
            Sub()
                label2.Text = String.Format("{0}秒経過", i)
            End Sub))
        System.Threading.Thread.Sleep(1000)
    Next
End Sub

' タスクの完了を待つ
Private Async Sub button1_Click(sender As Object, e As EventArgs)
Handles button1.Click
    Await Task.Run(Sub() onTask1())
    Await Task.Run(Sub() onTask2())
End Sub
```

リスト2 タスクの完了を待たない場合（ファイル名：pg189.sln、Form1.vb）

```
' タスクの完了を待たない
Private Sub button2_Click(sender As Object, e As EventArgs) Handles
button2.Click
    Task.Run(Sub() onTask1())
    Task.Run(Sub() onTask2())
End Sub
```

Tips 190

複数のタスクの実行を待つ

Level ●●
対応 EXP PRO

ここがポイントです！

処理終了時に戻り値を設定（Taskクラス、WaitAllメソッド）

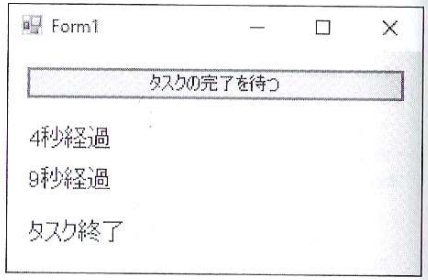
非同期に実行される複数のタスクの終了を待つためには、**TaskクラスのWaitAllメソッド**を使います。

Awaitキーワードを使うとタスクが順序実行されますが、WaitAllメソッドではタスクを同時に動作させた後に、それぞれのタスクが終了するまで待つことができます。

リスト1では、2つのタスクを同時に実行させています。

5秒間動作するタスクと、10秒間動作するタスクの2つをWaitAllメソッドを利用して処理待ちを行います。

▼実行結果



リスト1 複数タスクの完了を待つ（ファイル名：pg190.sln、Form1.vb）

```
Private Sub onTask1()
    For i As Integer = 0 To 4
        Me.Invoke(New Action(
                  Sub()
                      label1.Text = String.Format("{0}秒経過", i)

                  End Sub))
        System.Threading.Thread.Sleep(1000)
    Next
End Sub
Private Sub onTask2()
    For i As Integer = 0 To 9
        Me.Invoke(New Action(
                  Sub()
                      label2.Text = String.Format("{0}秒経過", i)

                  End Sub))
        System.Threading.Thread.Sleep(1000)
    Next
End Sub
Private Sub button1_Click(sender As Object, e As EventArgs) Handles bu
tton1.Click
    Task.Factory.StartNew(
        Sub()
            Me.Invoke(New Action(
                      Sub()
                          label3.Text = "タスク開始"
                      End Sub))
            Dim task1 = Task.Factory.StartNew(Sub() onTask1())
            Dim task2 = Task.Factory.StartNew(Sub() onTask2())
            ' 複数のタスクの完了を待つ
            Task.WaitAll(task1, task2)
            Me.Invoke(New Action(
                      Sub()
                          label3.Text = "タスク終了"

                      End Sub))
        End Sub)
End Sub
```


Tips 191

非同期メソッドを呼び出す

Level ●
対応 EXP PRO

ここがポイントです！

非同期処理を実行（Asyncキーワード、Awaitキーワード）

Taskメソッドで作成したタスクの処理待ちを行うためには、**Awaitキーワード**を使うと便利です。

タスク処理の終了待ちをして戻り値を取得することができます。

Awaitキーワードを使う場合には、呼び出しメソッドに**Asyncキーワード**を付けます。タスクを呼び出されている間でも、ユーザーは画面の操作ができます。またタスクを実行している間も、ほかのボタン操作やテキスト入力などを続けることができます。

リスト1では、非同期処理でonWorkメソッドを呼び出しています。処理した結果の合計値を処理終了時にダイアログで表示します。

▼実行結果

リスト1 非同期メソッドを呼び出す（ファイル名：pg191.sln、Form1.vb）

```
' 非同期処理
Private Function onWork() As Task(Of Integer)
```

```
    Dim task = New Task(Of Integer)(
        Function()
            ' 合計値を1秒ごとに計算する
            Dim sum As Integer = 0
            For i As Integer = 1 To 10
                Me.Invoke(New Action(
                            Sub()
                             label1.Text = String.Format("{0}秒経過", i)
                            End Sub))
                sum += i
                System.Threading.Thread.Sleep(1000)
            Next
            Return sum
        End Function)
    task.Start()
    Return task
End Function

' 非同期処理を呼び出す
Private Async Sub button1_Click(sender As Object, e As EventArgs)
Handles button1.Click
    Dim sum As Integer = Await onWork()
    MessageBox.Show(String.Format("合計値 {0}", sum))
End Sub
```

さらにワンポイント

タスクが実行されている間も画面の操作が可能なため、ユーザーから再び同じボタンをクリックされることがあります。これを回避するためにはタスクオブジェクトやフラグを使って、再入不可にします。

Tips 192

Level ●●

対応 EXP PRO

タスクの終了時に実行を継続する

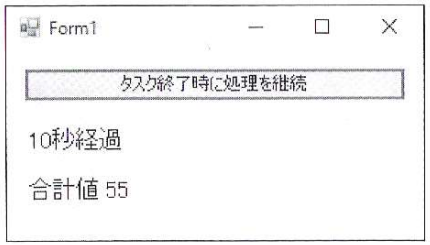
ここがポイントです！

タスク終了時に続けて処理をする（Taskクラス、ContinueWithメソッド）

順序よくタスクを実行するためにはAwaitキーワードを使いますが、1つのタスクの直後だけに処理をつなげたい場合は、ContinueWithメソッドを使うと便利です。

リスト1では、合計値を処理するタスクを実行した直後に、計算した合計値を表示する処理を追加しています。

▼実行結果



リスト1 タスク終了時の処理を行う（ファイル名：pg192.sln、Form1.vb）

```
' 非同期処理
Private Function onWork() As Task(Of Integer)
    Dim task = New Task(Of Integer)(
        Function()
            ' 合計値を1秒ごとに計算する
            Dim sum As Integer = 0
            For i As Integer = 1 To 10
                Me.Invoke(New Action(
                            Sub()
                             label1.Text = String.Format("{0}秒経過", i)
                            End Sub))
                sum += i
                System.Threading.Thread.Sleep(1000)
            Next
            Return sum

        End Function)
    task.Start()
    task.ContinueWith(
        Sub(t)
            Dim res As Integer = t.Result
            Me.Invoke(New Action(
                        Sub()
                            label2.Text = String.Format("合計値 {0}", res)
                        End Sub))
        End Sub)
```

```
    Return task
End Function

' 非同期処理を呼び出す
Private Async Sub button1_Click(sender As Object, e As EventArgs)
Handles button1.Click
    Await onWork()
End Sub
```


Tips 193

Level ●●

対応 EXP PRO

スレッドを切り替えてUIを変更する

ここがポイントです！

スレッド間でメソッドを利用する（Invokeメソッド）

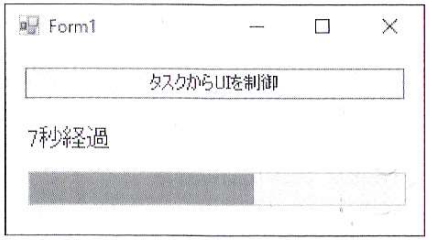
バックグラウンド処理を行うTaskクラスは、画面のユーザーインターフェイス操作するスレッドとは異なるスレッドになります。このためユーザーインターフェイスのコントロールを直接操作することはできません。

スレッド間でプロパティやメソッドを操作する場合は、**Invokeメソッド**を使います。

Invokeメソッドに引数なしのメソッドやラムダ式を記述することで、画面のコントロールのプロパティを操作できます。

リスト1では、動作しているタスクの中から経過時間をラベルに表示させています。

▼実行結果



リスト1 タスク内からUIコントロールを変更する（ファイル名：pg193.sln、Form1.vb）

```
 Private Async Sub button1_Click(sender As Object, e As EventArgs)
Handles button1.Click
    progressBar1.Minimum = 0
    progressBar1.Maximum = 10
    Await Task.Factory.StartNew(
        Sub()
            For i As Integer = 1 To 10
                Me.Invoke(New Action(
                          Sub()
                           label1.Text = String.Format("{0}秒経過", i)
                          End Sub))
                Me.Invoke(New Action(
                          Sub()
                              progressBar1.Value = i
                          End Sub))
                System.Threading.Thread.Sleep(1000)
            Next
        End Sub)
    MessageBox.Show("10秒経過しました")
End Sub
```

Tips 194

一定時間停止する

Level ●●
対応 EXP PRO

ここがポイントです！ 待ち時間を設定する（Threadクラス、Sleepメソッド、Taskクラス、Delayメソッド）

タスクを実行しているときに、数秒間処理を待ちたいときには、**Threadクラス**の**Sleepメソッド**あるいはTaskクラスの**Delayメソッド**を使います。

どちらも、ミリ秒単位で時間指定をすることができます。

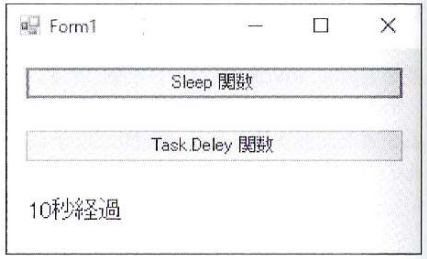
ThreadクラスのSleepメソッドでは、そこで処理が止まります。

TaskクラスのDelayメソッドでは、Awaitキーワードを利用して画面操作などの処理を続行できます。

リスト1では、Sleepメソッドで時間待ちを行います。

リスト2では、Delayメソッドで待ち時間を設定しています。

▼実行結果



リスト1 Sleepメソッドで待ち時間を設定する（ファイル名：pg194.sln、Form1.vb）

```
Private Async Sub button1_Click(sender As Object, e As EventArgs)
Handles button1.Click
    Await Task.Factory.StartNew(
        Sub()
            For i As Integer = 1 To 10
                Me.Invoke(New Action(
                    Sub()
                        label1.Text = String.Format("{0}秒経過", i)
                    End Sub))
                System.Threading.Thread.Sleep(1000)
            Next
        End Sub)
    label1.Text = "10秒経過"
End Sub
```

リスト2 Delayメソッドで待ち時間を設定する（ファイル名：pg194.sln、Form1.vb）

```
Private Async Sub button2_Click(sender As Object, e As EventArgs)
Handles button2.Click
    Await Task.Factory.StartNew(
        Async Function()
            For i As Integer = 1 To 10
                Me.Invoke(New Action(
                    Sub()
                        label1.Text = String.Format("{0}秒経過", i)
                    End Sub))
                Await Task.Delay(1000)
            Next
        End Function)
    label1.Text = "10秒経過"
End Sub
```


Tips 195

イベントが発生するまで停止する

Level ●●
対応 EXP PRO

ここがポイントです！ 他スレッドからのイベントを待つ（ManualResetEventクラス、WaitOneメソッド、Setメソッド）

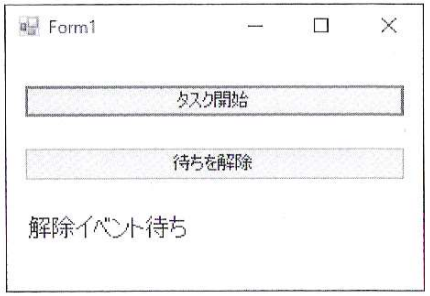
ほかのスレッドからイベントが発生するまで待つためには、**ManualResetEventクラス**の**WaitOneメソッド**を使います。

ミューテックスのように、非同期処理を行っている各スレッドの同期を取るために使えます。

イベントを解除するためには**Setメソッド**を呼び出します。

リスト1では、ボタンをクリックしてManualResetEventオブジェクトを作成してイベント待ちを行います。もう1つのボタンをクリックするとイベントが解除されて、処理が再開されます。

▼実行結果



リスト1 イベント待ちを解除する（ファイル名：pg195.sln、Form1.vb）

```
Private mre As System.Threading.ManualResetEvent
Private Async Sub button1_Click(sender As Object, e As EventArgs)
Handles button1.Click
    mre = New System.Threading.ManualResetEvent(False)
    Await Task.Factory.StartNew(
        Sub()
            For i As Integer = 1 To 20
                If i = 10 Then
                    ' 10秒後にイベント待ちになる
                    Me.Invoke(New Action(
                        Sub()
                            label1.Text = "解除イベント待ち"
```

```
                                End Sub))
                    mre.Reset()
                    mre.WaitOne()
                End If
                Me.Invoke(New Action(
                            Sub()
                             label1.Text = String.Format("{0}秒経過", i)
                            End Sub))
                System.Threading.Thread.Sleep(1000)
            Next
        End Sub)
    label1.Text = "タスク終了"
End Sub

Private Sub button2_Click(sender As Object, e As EventArgs) Handles
button2.Click
    mre.Set()
End Sub
```


Tips 196

Level ●●

対応 EXP PRO

タスクの実行をキャンセルする

ここがポイントです！

他スレッドからのイベントを待つ（CancellationTokenSourceクラス、Tokenプロパティ、Cancelメソッド）

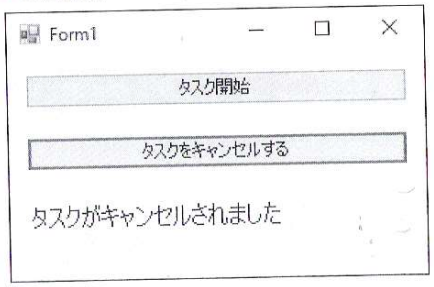
実行中のタスクをキャンセルするためにはCancellationTokenSourceクラスを使います。

Taskクラスでオブジェクトを生成するときに引数に、CancellationTokenSourceクラスのTokenプロパティを渡します。

タスクの実行時にCancelメソッドを呼び出すことにより、実行中のタスクが停止します。

リスト1では、ボタンをクリックすると10秒間タスクを実行します。もう1つのボタンをクリックすると実行中のタスクがキャンセルされます。キャンセルされたかどうかは、タスクの戻り値に設定しています。

▼実行結果



リスト1 実行タスクをキャンセルする（ファイル名：pg196.sln、Form1.vb）

```
Private cts = New CancellationTokenSource()
Private Async Sub button1_Click(sender As Object, e As EventArgs)
Handles button1.Click
    Dim res = Await Task.Factory.StartNew(Of Boolean)(
        Function()
            For i As Integer = 1 To 10
                If cts.Token.IsCancellationRequested Then
                    Return False
                End If
                Me.Invoke(New Action(
                            Sub()
                             label1.Text = String.Format("{0}秒経過", i)
                            End Sub))
                System.Threading.Thread.Sleep(1000)
            Next
            Return True
        End Function, cts.Token)
    If res Then
        label1.Text = "タスク正常終了"
    Else
        label1.Text = "タスクがキャンセルされました"
    End If
```

```
End Sub

Private Sub button2_Click(sender As Object, e As EventArgs) Handles
button2.Click
    cts.Cancel()
End Sub
```

Column コードエディター内で使用できる主なショートカットキー

コード入力中や編集時に使える主なショートカットキーには、次のようなものがあります。

■コードエディター内での検索と置換

機能	ショートカットキー
クイック検索	Ctrl + F キー
クイック検索の次の結果	Enter キー
クイック検索の前の結果	Shift + Enter キー
クイック検索でドロップダウンを展開	Alt + Down キー
検索を消去	Esc キー
クイック置換	Ctrl + H キー
クイック置換で次を置換	Alt + R キー
クイック置換ですべて置換	Alt + A キー

■コードエディター内での操作

機能	ショートカットキー
IntelliSense候補提示モード	Ctrl + Alt + Space キー
IntelliSenseの強制表示	Ctrl + J キー
クイックヒントの表示	Ctrl + K、I キー
移動	Ctrl + , キー
定義へ移動	F12 キー
エディターのズーム（拡大）	Ctrl + Shift + > キー
エディターのズーム（縮小）	Ctrl + Shift + < キー
ブロック選択	Alt キーを押したままマウスをドラッグ、 Shift + Alt + ↑、↓、←、→ キー
行を上下に移動（上へ移動）	Alt + Up キー
行を上下に移動（下へ移動）	Alt + Down キー
定義をここに表示	Alt + F12 キー
[定義をここに表示]ウィンドウを閉じる	Esc キー

第5章
197~212

文字列操作の極意