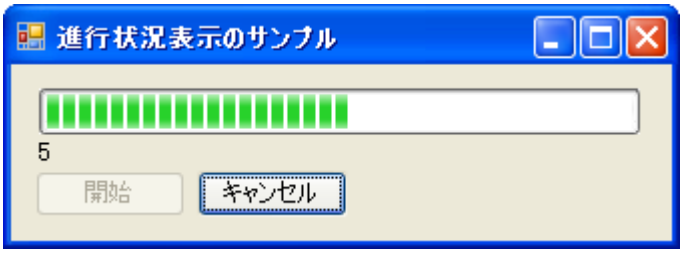
時間のかかる処理の進行状況を表示する

大きなファイルを読み込んだり、大量のファイルをコピーする時のように、時間のかかる処理を行うとき、何の表示もないとユーザーは「アプリケーションがフリーズしたのでは」と不安になってしまうかもしれません。そのようなときは、処理の進行状況をメッセージやプログレスバーで表示することが有効です。ここでは、時間のかかる処理の進行状況を表示する方法と、ユーザーが途中でキャンセルできるようにする方法を説明します。

ここでは単純な例として、WindowsフォームにLabelコントロール(Label1)とProgressBarコントロール(ProgressBar1)とButtonコントロール(Button1)を貼り付け、Button1をクリックすると1秒おきにLabel1とProgressBar1の内容が変化する（1から10までカウントアップする）アプリケーションを作成します。



うまくいかない例

まずは、何も考えないで、Label.TextプロパティとProgressBar.Valueプロパティを変更し、それ以外何もしないでどのように表示されるか確かめてみましょう。

VB.NET コードを隠す コードを選択

```
'Button1のClickイベントハンドラ
Private Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
    'コントロールを初期化する
    ProgressBar1.Minimum = 0
    ProgressBar1.Maximum = 10
    ProgressBar1.Value = 0
    Label1.Text = "0"

    '時間のかかる処理を開始する
    Dim i As Integer
    For i = 1 To 10
        '1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000)

        'ProgressBar1の値を変更する
        ProgressBar1.Value = i
        'Label1のテキストを変更する
        Label1.Text = i.ToString()
    Next

    '結果を報告する
    Label1.Text = "完了しました。"
End Sub
```

C# コードを隠す コードを選択

```
//Button1のクリックイベントハンドラ
private void Button1_Click(object sender, System.EventArgs e)
{
    //コントロールを初期化する
    ProgressBar1.Minimum = 0;
    ProgressBar1.Maximum = 10;
    ProgressBar1.Value = 0;
    Label1.Text = "0";

    //時間のかかる処理を開始する
    for (int i = 1; i <= 10; i++)
    {
        //1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000);

        //ProgressBar1の値を変更する
        ProgressBar1.Value = i;
        //Label1のテキストを変更する
        Label1.Text = i.ToString();
    }
```

```
    //結果を報告する
    Label1.Text = "完了しました。";
}
```

上記のコードを実行すると、ProgressBarは1秒おきにバーが伸びていきますが、Labelに表示される文字列は変化することなく、Button1_Clickイベントハンドラを抜けてはじめて"完了しました。"と表示されます。これは、ProgressBarコントロールがそのValueプロパティが変更されるとコントロールを再描画するのに対して、Labelコントロールは Textプロパティを変更してもそうしないためです。ほとんどのコントロールはLabelコントロールと同様の挙動となるでしょう。

コントロールを再描画する方法

ProgressBarコントロールのようにLabelコントロールでも1秒おきに内容を変更して表示するためには、Label.Textプロパティを設定後に、Updateメソッドを呼び出してコントロールを再描画するようにします。

VB.NET コードを隠す コードを選択

```
'Button1のClickイベントハンドラ
Private Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
    'コントロールを初期化する
    ProgressBar1.Minimum = 0
    ProgressBar1.Maximum = 10
    ProgressBar1.Value = 0
    Label1.Text = "0"
    'Label1を再描画する
    Label1.Update()

    '時間のかかる処理を開始する
    Dim i As Integer
    For i = 1 To 10
        '1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000)

        'ProgressBar1の値を変更する
        ProgressBar1.Value = i
        'Label1のテキストを変更する
        Label1.Text = i.ToString()

        'Label1を再描画する
        Label1.Update()
        '（フォーム全体を再描画するには、次のようにする）
        'Me.Update()
    Next

    '結果を報告する
    Label1.Text = "完了しました。"
End Sub
```

C# コードを隠す コードを選択

```
//Button1のクリックイベントハンドラ
private void Button1_Click(object sender, System.EventArgs e)
{
    //コントロールを初期化する
    ProgressBar1.Minimum = 0;
    ProgressBar1.Maximum = 10;
    ProgressBar1.Value = 0;
    Label1.Text = "0";
    //Label1を再描画する
    Label1.Update();

    //時間のかかる処理を開始する
    for (int i = 1; i <= 10; i++)
    {
        //1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000);

        //ProgressBar1の値を変更する
        ProgressBar1.Value = i;
        //Label1のテキストを変更する
        Label1.Text = i.ToString();

        //Label1を再描画する
        Label1.Update();
        //（フォーム全体を再描画するには、次のようにする）
        //this.Update();
    }

    //結果を報告する
    Label1.Text = "完了しました。";
}
```

進行状況を表示するだけであれば、このようにUpdateメソッドを使うのが最も簡単で、安全な方法です。しかしこの方法では処理の間、フォームやコントロールを一切操作することが出来なくなります。このような状況が好ましくなければ、マルチスレッド化や、Application.DoEventsメソッドを使う方法になります。

.NET Framework 2.0以降で、BackgroundWorkerコンポーネントを使用する方法

.NET Framework 2.0以降であれば、BackgroundWorkerコンポーネントがまさにおあつらえ向きです。BackgroundWorkerコンポーネントを使えば、本来は非常に難しいマルチスレッド化を簡単に行うことができます。

BackgroundWorkerコンポーネントを使用するためには、まずフォームデザイナで、BackgroundWorkerコンポーネントをフォームに配置してください。BackgroundWorkerコンポーネントはツールボックスの「コンポーネント」グループにあります。ここでは、配置されたBackgroundWorkerコンポーネントの名前を「BackgroundWorker1」としたとします。

次に、BackgroundWorker1のDoWork、ProgressChanged、RunWorkerCompletedイベントのイベントハンドラを作成して、以下のようなコードを記述します。

VB.NET コードを隠す コードを選択

```
'Button1のClickイベントハンドラ
Private Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
    '処理が行われているときは、何もしない
    If BackgroundWorker1.IsBusy Then
        Return
    End If

    'Button1を無効にする
    Button1.Enabled = False

    'コントロールを初期化する
    ProgressBar1.Minimum = 0
    ProgressBar1.Maximum = 10
    ProgressBar1.Value = 0
    Label1.Text = "0"

    'BackgroundWorkerのProgressChangedイベントが発生するようにする
    BackgroundWorker1.WorkerReportsProgress = True
    'DoWorkで取得できるパラメータ(10)を指定して、処理を開始する
    'パラメータが必要なければ省略できる
    BackgroundWorker1.RunWorkerAsync(10)
End Sub

'BackgroundWorker1のDoWorkイベントハンドラ
'ここで時間のかかる処理を行う
Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, _
        ByVal e As DoWorkEventArgs) _
        Handles BackgroundWorker1.DoWork
    Dim bgWorker As BackgroundWorker = DirectCast(sender, BackgroundWorker)

    'パラメータを取得する
    Dim maxLoops As Integer = CInt(e.Argument)

    '時間のかかる処理を開始する
    Dim i As Integer
    For i = 1 To maxLoops
        '1秒間待機する（時間のかかる処理があるものとする）
```

```
        System.Threading.Thread.Sleep(1000)

        'ProgressChangedイベントハンドラを呼び出し、
        'コントロールの表示を変更する
        bgWorker.ReportProgress(i)
    Next

    'ProgressChangedで取得できる結果を設定する
    '結果が必要なければ省略できる
    e.Result = maxLoops
End Sub

'BackgroundWorker1のProgressChangedイベントハンドラ
'コントロールの操作は必ずここで行い、DoWorkでは絶対にしない
Private Sub BackgroundWorker1_ProgressChanged(ByVal sender As Object, _
        ByVal e As ProgressChangedEventArgs) _
        Handles BackgroundWorker1.ProgressChanged
    'ProgressBar1の値を変更する
    ProgressBar1.Value = e.ProgressPercentage
    'Label1のテキストを変更する
    Label1.Text = e.ProgressPercentage.ToString()
End Sub

'BackgroundWorker1のRunWorkerCompletedイベントハンドラ
'処理が終わったときに呼び出される
Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, _
        ByVal e As RunWorkerCompletedEventArgs) _
        Handles BackgroundWorker1.RunWorkerCompleted
    If Not e.Error Is Nothing Then
        'エラーが発生したとき
        Label1.Text = "エラー:" & e.Error.Message
    Else
        '正常に終了したとき
        '結果を取得する
        Dim result As Integer = CInt(e.Result)
        Label1.Text = result.ToString() & "回で完了しました。"
    End If

    'Button1を有効に戻す
    Button1.Enabled = True
End Sub
```

C# コードを隠す　コードを選択

```
//フォームのLoadイベントハンドラ
private void Form1_Load(object sender, System.EventArgs e)
{
    //イベントハンドラをイベントに関連付ける
    //フォームデザイナを使って関連付けを行った場合は、不要
    BackgroundWorker1.DoWork +=
        new DoWorkEventHandler(BackgroundWorker1_DoWork);
    BackgroundWorker1.ProgressChanged +=
        new ProgressChangedEventHandler(BackgroundWorker1_ProgressChanged);
    BackgroundWorker1.RunWorkerCompleted +=
```

```
        new RunWorkerCompletedEventHandler(BackgroundWorker1_RunWorkerCompleted);
}

//Button1のClickイベントハンドラ
private void Button1_Click(object sender, System.EventArgs e)
{
    //処理が行われているときは、何もしない
    if (BackgroundWorker1.IsBusy)
        return;

    //Button1を無効にする
    Button1.Enabled = false;

    //コントロールを初期化する
    ProgressBar1.Minimum = 0;
    ProgressBar1.Maximum = 10;
    ProgressBar1.Value = 0;
    Label1.Text = "0";

    //BackgroundWorkerのProgressChangedイベントが発生するようにする
    BackgroundWorker1.WorkerReportsProgress = true;
    //DoWorkで取得できるパラメータ(10)を指定して、処理を開始する
    //パラメータが必要なければ省略できる
    BackgroundWorker1.RunWorkerAsync(10);
}

//BackgroundWorker1のDoWorkイベントハンドラ
//ここで時間のかかる処理を行う
private void BackgroundWorker1_DoWork(
    object sender, DoWorkEventArgs e)
{
    BackgroundWorker bgWorker = (BackgroundWorker)sender;

    //パラメータを取得する
    int maxLoops = (int)e.Argument;

    //時間のかかる処理を開始する
    for (int i = 1; i <= maxLoops; i++)
    {
        //1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000);

        //ProgressChangedイベントハンドラを呼び出し、
        //コントロールの表示を変更する
        bgWorker.ReportProgress(i);
    }

    //ProgressChangedで取得できる結果を設定する
    //結果が必要なければ省略できる
    e.Result = maxLoops;
}

//BackgroundWorker1のProgressChangedイベントハンドラ
//コントロールの操作は必ずここで行い、DoWorkでは絶対にしない
private void BackgroundWorker1_ProgressChanged(
```

```
    object sender, ProgressChangedEventArgs e)
{
    //ProgressBar1の値を変更する
    ProgressBar1.Value = e.ProgressPercentage;
    //Label1のテキストを変更する
    Label1.Text = e.ProgressPercentage.ToString();
}

//BackgroundWorker1のRunWorkerCompletedイベントハンドラ
//処理が終わったときに呼び出される
private void BackgroundWorker1_RunWorkerCompleted(
    object sender, RunWorkerCompletedEventArgs e)
{
    if (e.Error != null)
    {
        //エラーが発生したとき
        Label1.Text = "エラー:" + e.Error.Message;
    }
    else
    {
        //正常に終了したとき
        //結果を取得する
        int result = (int)e.Result;
        Label1.Text = result.ToString() + "回で完了しました。";
    }

    //Button1を有効に戻す
    Button1.Enabled = true;
}
```

上記のコードの説明をします。

時間のかかる処理は、DoWorkイベントハンドラに記述します。この時気を付けなければならないのは、DoWorkイベントハンドラは別スレッドで実行されるということです。よって、DoWorkイベントハンドラ内からは、それ以外のところで作成したオブジェクトには、スレッドセーフでない限り、アクセスすることができません。例えば、DoWorkイベントハンドラからはコントロール（フォームを含む）を操作することができません。上記の例で言えば、BackgroundWorker1_DoWork内でProgressBar1.ValueやLabel1.Textの値を変更してはいけません。値を変更してしまうと、例外System.InvalidOperationExceptionが発生します。

コントロールの操作は、必ずProgressChangedイベントハンドラ内で行ないます。ProgressChangedイベントハンドラではコントロールの操作だけを行ない、それ以外のことはなるべく行わないようにします。それ以外の無駄な処理は、パフォーマンスを大きく低下させます。

DoWorkイベントハンドラでReportProgressメソッドを呼び出すと、ProgressChangedイベントが発生します。ProgressChangedイベントは、WorkerReportsProgressプロパティがTrueでないと発生しません。WorkerReportsProgressプロパティをTrueにしないでReportProgressメソッドを呼び出すと、例外System.InvalidOperationExceptionが発生します。

ReportProgressメソッドを呼び出す時、処理の進行状況を整数のパラメータで渡します。通常この値はパーセンテージですが、そうである必要はありません。この値をProgressChangedイベントハンドラで受け取るには、ProgressChangedEventArgs.ProgressPercentageプロパティを使います。整数以外の値を渡したい場合は、ReportProgressメソッドに2つ目のパラメータ（Object型）を指定することができます。2番目のパラメータに指定された値をProgressChangedイベントハンドラで取得するには、ProgressChangedEventArgs.UserStateプロパティを使います。

DoWorkイベントハンドラの処理が完了すると、RunWorkerCompletedイベントが発生します。上の例ではRunWorkerCompletedイベントハンドラを用意し、結果を表示していますが、必要なければ省略できます。

RunWorkerCompletedイベントハンドラでは、まずRunWorkerCompletedEventArgs.Errorプロパティnにより、エラーが発生しなかったかを調べます。エラーが発生していない時だけ、結果を取得します。

この方法（あるいはこれ以下に紹介するすべての方法）では、時間のかかる処理（上記の例では、BackgroundWorker1_DoWorkメソッドで行っている処理）はバックグラウンドで行っているため、その間に別のイベントが発生する可能性があります。よって、発生しては困るイベントを発生しないようにするか、発生しても大丈夫なようにしておく必要があります。例えば上の例では、一度Button1がクリックされて処理が開始されたら、その最中にもう一度Button1がクリックされないように、Button1のEnabledプロパティをFalseにしています。上記の例ではさらに、Button1_Clickのはじめに処理が実行中か調べ、実行中ならば何もしないようにしています。

その他、フォームが閉じられる可能性があるなどということにも気をつけてください。フォームを閉じられなくする方法を「条件によりフォームが閉じられないようにする」で紹介していますので、処理の実行中はフォームを閉じられなくすると良いでしょう。

ユーザーがキャンセルできるようにする

BackgroundWorkerコンポーネントのDoWorkイベントハンドラで実行中の処理を中止できるようにするには、BackgroundWorkerのCancellationPendingプロパティをTrueにする必要があります。そして、中止するときに、BackgroundWorker.CancelAsyncメソッドを呼び出します。ただし実際に処理を中止させるコードは、自分で記述しなければなりません。

CancelAsyncメソッドが呼び出されると、DoWorkイベントハンドラでBackgroundWorker.CancellationPendingプロパティがTrueになります。この値を監視し、True

になったところで、DoWorkEventArgs.Cancelプロパティにtrueを設定して、DoWorkイベントハンドラから抜けます。

このようにして中止されると、RunWorkerCompletedイベントハンドラではRunWorkerCompletedEventArgs.CancelledプロパティがTrueになります。なお中止された場合は、結果を取得することはできません。取得しようとすると、Application.Runメソッドで例外System.Reflection.TargetInvocationExceptionが発生します。

上記のコードを書き換えて、ユーザーがキャンセルできるようにした例を示します。この例ではフォームにButtonコントロールをもう一つ追加し（Button2）、これをクリックすると処理をキャンセルできるようにしています。

VB.NET コードを隠す コードを選択

```
'Button1のClickイベントハンドラ
Private Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
    '処理が行われているときは、何もしない
    If BackgroundWorker1.IsBusy Then
        Return
    End If

    'Button1を無効にする
    Button1.Enabled = False
    'Button2を有効にする
    Button2.Enabled = True

    'コントロールを初期化する
    ProgressBar1.Minimum = 0
    ProgressBar1.Maximum = 10
    ProgressBar1.Value = 0
    Label1.Text = "0"

    'BackgroundWorkerのProgressChangedイベントが発生するようにする
    BackgroundWorker1.WorkerReportsProgress = True
    'キャンセルできるようにする
    BackgroundWorker1.WorkerSupportsCancellation = True
    'DoWorkで取得できるパラメータ(10)を指定して、処理を開始する
    'パラメータが必要なければ省略できる
    BackgroundWorker1.RunWorkerAsync(10)
End Sub

'Button2のClickイベントハンドラ
Private Sub Button2_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button2.Click
    'Button2を無効にする
    Button2.Enabled = False

    'キャンセルする
    BackgroundWorker1.CancelAsync()
End Sub
```

```
'BackgroundWorker1のDoWorkイベントハンドラ
'ここで時間のかかる処理を行う
Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, _
        ByVal e As DoWorkEventArgs) _
        Handles BackgroundWorker1.DoWork
    Dim bgWorker As BackgroundWorker = DirectCast(sender, BackgroundWorker)

    'パラメータを取得する
    Dim maxLoops As Integer = CInt(e.Argument)

    '時間のかかる処理を開始する
    Dim i As Integer
    For i = 1 To maxLoops
        'キャンセルされたか調べる
        If bgWorker.CancellationPending Then
            'キャンセルされたとき
            e.Cancel = True
            Return
        End If

        '1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000)

        'ProgressChangedイベントハンドラを呼び出し、
        'コントロールの表示を変更する
        bgWorker.ReportProgress(i)
    Next

    'ProgressChangedで取得できる結果を設定する
    '結果が必要なければ省略できる
    e.Result = maxLoops
End Sub

'BackgroundWorker1のProgressChangedイベントハンドラ
'コントロールの操作は必ずここで行い、DoWorkでは絶対にしない
Private Sub BackgroundWorker1_ProgressChanged(ByVal sender As Object, _
        ByVal e As ProgressChangedEventArgs) _
        Handles BackgroundWorker1.ProgressChanged
    'ProgressBar1の値を変更する
    ProgressBar1.Value = e.ProgressPercentage
    'Label1のテキストを変更する
    Label1.Text = e.ProgressPercentage.ToString()
End Sub

'BackgroundWorker1のRunWorkerCompletedイベントハンドラ
'処理が終わったときに呼び出される
Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, _
        ByVal e As RunWorkerCompletedEventArgs) _
        Handles BackgroundWorker1.RunWorkerCompleted
    If Not e.Error Is Nothing Then
        'エラーが発生したとき
        Label1.Text = "エラー:" & e.Error.Message
    ElseIf e.Cancelled Then
        'キャンセルされたとき
        Label1.Text = "キャンセルされました。"
```

```
    Else
        '正常に終了したとき
        '結果を取得する
        Dim result As Integer = CInt(e.Result)
        Label1.Text = result.ToString() & "回で完了しました。"
    End If

    'Button1を有効に戻す
    Button1.Enabled = True
    'Button2を無効に戻す
    Button2.Enabled = False
End Sub
```

C# コードを隠す コードを選択

```
//フォームのLoadイベントハンドラ
private void Form1_Load(object sender, System.EventArgs e)
{
    //イベントハンドラをイベントに関連付ける
    //フォームデザイナを使って関連付けを行った場合は、不要
    BackgroundWorker1.DoWork +=
        new DoWorkEventHandler(BackgroundWorker1_DoWork);
    BackgroundWorker1.ProgressChanged +=
        new ProgressChangedEventHandler(BackgroundWorker1_ProgressChanged);
    BackgroundWorker1.RunWorkerCompleted +=
        new RunWorkerCompletedEventHandler(BackgroundWorker1_RunWorkerCompleted);
}

//Button1のClickイベントハンドラ
private void Button1_Click(object sender, System.EventArgs e)
{
    //処理が行われているときは、何もしない
    if (BackgroundWorker1.IsBusy)
        return;

    //Button1を無効にする
    Button1.Enabled = false;
    //Button2を有効にする
    Button2.Enabled = true;

    //コントロールを初期化する
    ProgressBar1.Minimum = 0;
    ProgressBar1.Maximum = 10;
    ProgressBar1.Value = 0;
    Label1.Text = "0";

    //BackgroundWorkerのProgressChangedイベントが発生するようにする
    BackgroundWorker1.WorkerReportsProgress = true;
    //キャンセルできるようにする
    BackgroundWorker1.WorkerSupportsCancellation = true;
    //DoWorkで取得できるパラメータ(10)を指定して、処理を開始する
    //パラメータが必要なければ省略できる
    BackgroundWorker1.RunWorkerAsync(10);
}
```

```
//Button2のClickイベントハンドラ
private void Button2_Click(object sender, System.EventArgs e)
{
    //Button2を無効にする
    Button2.Enabled = false;

    //キャンセルする
    BackgroundWorker1.CancelAsync();
}

//BackgroundWorker1のDoWorkイベントハンドラ
//ここで時間のかかる処理を行う
private void BackgroundWorker1_DoWork(
    object sender, DoWorkEventArgs e)
{
    BackgroundWorker bgWorker = (BackgroundWorker)sender;

    //パラメータを取得する
    int maxLoops = (int)e.Argument;

    //時間のかかる処理を開始する
    for (int i = 1; i <= maxLoops; i++)
    {
        //キャンセルされたか調べる
        if (bgWorker.CancellationPending)
        {
            //キャンセルされたとき
            e.Cancel = true;
            return;
        }

        //1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000);

        //ProgressChangedイベントハンドラを呼び出し、
        //コントロールの表示を変更する
        bgWorker.ReportProgress(i);
    }

    //ProgressChangedで取得できる結果を設定する
    //結果が必要なければ省略できる
    e.Result = maxLoops;
}

//BackgroundWorker1のProgressChangedイベントハンドラ
//コントロールの操作は必ずここで行い、DoWorkでは絶対にしない
private void BackgroundWorker1_ProgressChanged(
    object sender, ProgressChangedEventArgs e)
{
    //ProgressBar1の値を変更する
    ProgressBar1.Value = e.ProgressPercentage;
    //Label1のテキストを変更する
    Label1.Text = e.ProgressPercentage.ToString();
}
```

```
//BackgroundWorker1のRunWorkerCompletedイベントハンドラ
//処理が終わったときに呼び出される
private void BackgroundWorker1_RunWorkerCompleted(
    object sender, RunWorkerCompletedEventArgs e)
{
    if (e.Error != null)
    {
        //エラーが発生したとき
        Label1.Text = "エラー:" + e.Error.Message;
    }
    else if (e.Cancelled)
    {
        //キャンセルされたとき
        Label1.Text = "キャンセルされました。";
    }
    else
    {
        //正常に終了したとき
        //結果を取得する
        int result = (int)e.Result;
        Label1.Text = result.ToString() + "回で完了しました。";
    }

    //Button1を有効に戻す
    Button1.Enabled = true;
    //Button2を無効に戻す
    Button2.Enabled = false;
}
```

Threadクラスを使用したスレッド化による方法

BackgroundWorkerコンポーネントを使用できない.NET Framework 1.1以前でも、もちろんマルチスレッド化は可能です。マルチスレッドプログラミングは非常に難しいですが、ここではサンプルのみを示します。マルチスレッドプログラミングについては、こちらで詳しく説明しています。このサンプルでは、Threadクラスを使用して、時間のかかる処理（CountUpメソッド）をメインとは別のスレッドで行っています。

VB.NET　コードを隠す　コードを選択

```
'コントロールの値を変更するためのデリゲート
Private Delegate Sub SetProgressValueDelegate(ByVal num As Integer)
```

```
'バックグラウンド処理が終わった時にコントロールの値を変更するためのデリゲート
Private Delegate Sub ThreadCompletedDelegate()

'別処理をするためのスレッド
Private workerThread As System.Threading.Thread

'Button1のClickイベントハンドラ
Private Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
    'Button1を無効にする
    Button1.Enabled = False

    'コントロールを初期化する
    ProgressBar1.Minimum = 0
    ProgressBar1.Maximum = 10
    ProgressBar1.Value = 0
    Label1.Text = "0"

    'CountUpメソッドを別スレッドで実行する
    workerThread = New System.Threading.Thread( _
        New System.Threading.ThreadStart(AddressOf CountUp))
    workerThread.IsBackground = True
    workerThread.Start()
End Sub

'進行状況を表示する処理
Private Sub CountUp()
    'デリゲートの作成
    Dim progressDlg As New SetProgressValueDelegate(AddressOf SetProgressValue)
    Dim completeDlg As New ThreadCompletedDelegate(AddressOf ThreadCompleted)

    '時間のかかる処理を開始する
    Dim i As Integer
    For i = 1 To 10
        '1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000)

        'コントロールの表示を変更する
        Me.Invoke(progressDlg, New Object() {i})
    Next

    '完了したときにコントロールの値を変更する
    Me.Invoke(completeDlg)
End Sub

'コントロールの値を変更する
Private Sub SetProgressValue(ByVal num As Integer)
    'ProgressBar1の値を変更する
    ProgressBar1.Value = num
    'Label1のテキストを変更する
    Label1.Text = num.ToString()
End Sub

'処理が完了した時にコントロールの値を変更する
Private Sub ThreadCompleted()
```

```
    Label1.Text = "完了しました。"
    Button1.Enabled = True
End Sub
```

C# コードを隠す コードを選択

```
//コントロールの値を変更するためのデリゲート
private delegate void SetProgressValueDelegate(int num);
//バックグラウンド処理が終わった時にコントロールの値を変更するためのデリゲート
private delegate void ThreadCompletedDelegate();

//別処理をするためのスレッド
private System.Threading.Thread workerThread;

//Button1のClickイベントハンドラ
private void Button1_Click(object sender, System.EventArgs e)
{
    //Button1を無効にする
    Button1.Enabled = false;

    //コントロールを初期化する
    ProgressBar1.Minimum = 0;
    ProgressBar1.Maximum = 10;
    ProgressBar1.Value = 0;
    Label1.Text = "0";

    //CountUpメソッドを別スレッドで実行する
    workerThread = new System.Threading.Thread(
        new System.Threading.ThreadStart(CountUp));
    workerThread.IsBackground = true;
    workerThread.Start();
}

//進行状況を表示する処理
private void CountUp()
{
    //デリゲートの作成
    SetProgressValueDelegate progressDlg =
        new SetProgressValueDelegate(SetProgressValue);
    ThreadCompletedDelegate completeDlg =
        new ThreadCompletedDelegate(ThreadCompleted);

    //時間のかかる処理を開始する
    for (int i = 1; i <= 10; i++)
    {
        //1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000);

        //コントロールの表示を変更する
        this.Invoke(progressDlg, new object[] { i });
    }

    //完了したときにコントロールの値を変更する
    this.Invoke(completeDlg);
```

```
}

//コントロールの値を変更する
private void SetProgressValue(int num)
{
    //ProgressBar1の値を変更する
    ProgressBar1.Value = num;
    //Label1のテキストを変更する
    Label1.Text = num.ToString();
}

//処理が完了した時にコントロールの値を変更する
private void ThreadCompleted()
{
    Label1.Text = "完了しました。";
    Button1.Enabled = true;
}
```

ユーザーがキャンセルできるようにする

ユーザーが処理を途中で中止できるようにするには、キャンセルボタンが押された時に、処理中のスレッドのThread.Abortメソッドを呼び出してスレッドを中止させればよいという考えもあるでしょう。しかしAbortメソッドを使って別のスレッドを終了させることは、どこで中断されるか予測できなく、それだけ危険です。よって、Abortメソッドはなるべく使わないようにして、フラッグを使って適当なタイミングでループを終了させる方法をお勧めします。

以下に、このような方法により、上記のコードにキャンセルボタン(Button2)の処理を付けたコードを示します。

VB.NET コードを隠す コードを選択

```
'コントロールの値を変更するためのデリゲート
Private Delegate Sub SetProgressValueDelegate(ByVal num As Integer)
'バックグラウンド処理が終わった時にコントロールの値を変更するためのデリゲート
Private Delegate Sub ThreadCompletedDelegate()
'処理がキャンセルされた時にコントロールの値を変更するためのデリゲート
Private Delegate Sub ThreadCanceledDelegate()

'キャンセルボタンがクリックされたかを示すフラッグ
Private _canceled As Boolean = False
Private ReadOnly canceledSyncObject As Object = New Object
Private Property canceled() As Boolean
```

```
    Get
        SyncLock (canceledSyncObject)
            Return Me._canceled
        End SyncLock
    End Get
    Set(ByVal Value As Boolean)
        SyncLock (canceledSyncObject)
            Me._canceled = Value
        End SyncLock
    End Set
End Property

'別処理をするためのスレッド
Private workerThread As System.Threading.Thread

'Button1のClickイベントハンドラ
Private Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
    'Button1を無効にする
    Button1.Enabled = False
    'Button2を有効にする
    Button2.Enabled = True
    canceled = False

    'コントロールを初期化する
    ProgressBar1.Minimum = 0
    ProgressBar1.Maximum = 10
    ProgressBar1.Value = 0
    Label1.Text = "0"

    'CountUpメソッドを別スレッドで実行する
    workerThread = New System.Threading.Thread( _
        New System.Threading.ThreadStart(AddressOf CountUp))
    workerThread.IsBackground = True
    workerThread.Start()
End Sub

'Button2のClickイベントハンドラ
Private Sub Button2_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button2.Click
    'Button2を無効にする
    Button2.Enabled = False

    'キャンセルのフラッグを立てる
    canceled = True
End Sub

'進行状況を表示する処理
Private Sub CountUp()
    'デリゲートの作成
    Dim progressDlg As New SetProgressValueDelegate(AddressOf SetProgressValue)
    Dim completeDlg As New ThreadCompletedDelegate(AddressOf ThreadCompleted)
    Dim canceledDlg As New ThreadCanceledDelegate(AddressOf ThreadCanceled)

    '時間のかかる処理を開始する
```

```
    Dim i As Integer
    For i = 1 To 10
        'キャンセルボタンがクリックされたか調べる
        If canceled Then
            'キャンセルされたときにコントロールの値を変更する
            Me.Invoke(canceledDlg)
            '処理を終了させる
            Return
        End If

        '1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000)

        'コントロールの表示を変更する
        Me.Invoke(progressDlg, New Object() {i})
    Next

    '完了したときにコントロールの値を変更する
    Me.Invoke(completeDlg)
End Sub

'コントロールの値を変更する
Private Sub SetProgressValue(ByVal num As Integer)
    'ProgressBar1の値を変更する
    ProgressBar1.Value = num
    'Label1のテキストを変更する
    Label1.Text = num.ToString()
End Sub

'処理が完了した時にコントロールの値を変更する
Private Sub ThreadCompleted()
    Label1.Text = "完了しました。"
    Button1.Enabled = True
    Button2.Enabled = False
End Sub

'処理がキャンセルされた時にコントロールの値を変更する
Private Sub ThreadCanceled()
    Label1.Text = "キャンセルされました。"
    Button1.Enabled = True
    Button2.Enabled = False
End Sub
```

C# コードを隠す　コードを選択

```
//コントロールの値を変更するためのデリゲート
private delegate void SetProgressValueDelegate(int num);
//バックグラウンド処理が終わった時にコントロールの値を変更するためのデリゲート
private delegate void ThreadCompletedDelegate();
//処理がキャンセルされた時にコントロールの値を変更するためのデリゲート
private delegate void ThreadCanceledDelegate();

//キャンセルボタンがクリックされたかを示すフラッグ
private volatile bool canceled = false;
```

```
//別処理をするためのスレッド
private System.Threading.Thread workerThread;

//Button1のClickイベントハンドラ
private void Button1_Click(object sender, System.EventArgs e)
{
    //Button1を無効にする
    Button1.Enabled = false;
    //Button2を有効にする
    Button2.Enabled = true;
    canceled = false;

    //コントロールを初期化する
    ProgressBar1.Minimum = 0;
    ProgressBar1.Maximum = 10;
    ProgressBar1.Value = 0;
    Label1.Text = "0";

    //CountUpメソッドを別スレッドで実行する
    workerThread = new System.Threading.Thread(
        new System.Threading.ThreadStart(CountUp));
    workerThread.IsBackground = true;
    workerThread.Start();
}

//Button2のClickイベントハンドラ
private void Button2_Click(object sender, System.EventArgs e)
{
    //Button2を無効にする
    Button2.Enabled = false;

    //キャンセルのフラッグを立てる
    canceled = true;
}

//進行状況を表示する処理
private void CountUp()
{
    //デリゲートの作成
    SetProgressValueDelegate progressDlg =
        new SetProgressValueDelegate(SetProgressValue);
    ThreadCompletedDelegate completeDlg =
        new ThreadCompletedDelegate(ThreadCompleted);
    ThreadCanceledDelegate canceledDlg =
        new ThreadCanceledDelegate(ThreadCanceled);

    //時間のかかる処理を開始する
    for (int i = 1; i <= 10; i++)
    {
        //キャンセルボタンがクリックされたか調べる
        if (canceled)
        {
            //キャンセルされたときにコントロールの値を変更する
            this.Invoke(canceledDlg);
            //処理を終了させる
```

```
            return;
        }

        //1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000);

        //コントロールの表示を変更する
        this.Invoke(progressDlg, new object[] { i });
    }

    //完了したときにコントロールの値を変更する
    this.Invoke(completeDlg);
}

//コントロールの値を変更する
private void SetProgressValue(int num)
{
    //ProgressBar1の値を変更する
    ProgressBar1.Value = num;
    //Label1のテキストを変更する
    Label1.Text = num.ToString();
}

//処理が完了した時にコントロールの値を変更する
private void ThreadCompleted()
{
    Label1.Text = "完了しました。";
    Button1.Enabled = true;
    Button2.Enabled = false;
}

//処理がキャンセルされた時にコントロールの値を変更する
private void ThreadCanceled()
{
    Label1.Text = "キャンセルされました。";
    Button1.Enabled = true;
    Button2.Enabled = false;
}
```

Application.DoEventsを使用する方法

DoEventsを使用する方法は、VB6以前ではよく使われていた方法です。ただ、.NET Framework ではマルチスレッドプログラミングをサポートしていますので、DoEventsを使わずにマルチスレ

ッドにする方がよりお勧めできます。ここでは一応、DoEventsを使った方法も紹介しておきます。

Application.DoEventsメソッドを呼び出すと、キュー内で待機中のイベントを処理することができるため、次のようにループ内にDoEventsを入れることにより、フォームやコントロールが再描画されるのはもちろんのこと、フォームの移動やクリックなど、ユーザーによる操作も可能になります。

VB.NET コードを隠す コードを選択

```
'Button1のClickイベントハンドラ
Private Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
    'Button1を無効にする
    Button1.Enabled = False

    'コントロールを初期化する
    ProgressBar1.Minimum = 0
    ProgressBar1.Maximum = 10
    ProgressBar1.Value = 0
    Label1.Text = "0"

    '時間のかかる処理を開始する
    Dim i As Integer
    For i = 1 To 10
        '待機中のイベントを処理する
        Application.DoEvents()

        '1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000)

        'ProgressBar1の値を変更する
        ProgressBar1.Value = i
        'Label1のテキストを変更する
        Label1.Text = i.ToString()
    Next

    'Button1を有効に戻す
    Button1.Enabled = True
End Sub
```

C# コードを隠す コードを選択

```
//Button1のクリックイベントハンドラ
private void Button1_Click(object sender, System.EventArgs e)
{
    //Button1を無効にする
    Button1.Enabled = false;

    //コントロールを初期化する
    ProgressBar1.Minimum = 0;
    ProgressBar1.Maximum = 10;
    ProgressBar1.Value = 0;
```

```
    Label1.Text = "0";

    //時間のかかる処理を開始する
    for (int i = 1; i <= 10; i++)
    {
        //待機中のイベントを処理する
        Application.DoEvents();

        //1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000);

        //ProgressBar1の値を変更する
        ProgressBar1.Value = i;
        //Label1のテキストを変更する
        Label1.Text = i.ToString();
    }

    //Button1を有効に戻す
    Button1.Enabled = true;
}
```

スレッド化と比べ、DoEventsメソッドには数々の欠点があります。イベントが処理されるのはDoEventsメソッドが呼び出された時だけですので、例えば下で紹介するように、キャンセルボタンを設置したとき、ユーザーがキャンセルボタンをクリックしてもすぐには反応せず、しばらくたってから押されたようになります。さらに、DoEventsメソッドが呼び出されイベントが処理される時、その処理がすべて終わるまでDoEventsメソッド以降の処理はブロックされます。例えば上記の例の場合、フォームをマウスで移動している間ループ処理は中断され、移動をやめてからようやくカウントアップが再開されます。

ユーザーがキャンセルできるようにする

新たなボタン（Button2）を設置し、このボタンが押されたときに処理をキャンセルできるようにするコードを以下に示します。キャンセルボタンがクリックされた時にフラグを立て、ループ内でフラグをチェックし、フラグが立っていれば処理を中断しているだけです。

VB.NET　コードを隠す　コードを選択

```
'キャンセルボタンがクリックされたかを示すフラッグ
Private canceled As Boolean = False

'Button1のClickイベントハンドラ
Private Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button1.Click

    'Button1を無効にする
    Button1.Enabled = False
    'Button2を有効にする
    Button2.Enabled = True
    canceled = False

    'コントロールを初期化する
```

```
    ProgressBar1.Minimum = 0
    ProgressBar1.Maximum = 10
    ProgressBar1.Value = 0
    Label1.Text = "0"

    '時間のかかる処理を開始する
    For i As Integer = 1 To 10
        '待機中のイベントを処理する
        Application.DoEvents()

        'キャンセルボタンがクリックされたか調べる
        If canceled Then
            'キャンセルされた時
            MessageBox.Show(Me, "ユーザーにより中止されました。")
            'ループを抜ける
            Exit For
        End If

        '1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000)

        'ProgressBar1の値を変更する
        ProgressBar1.Value = i
        'Label1のテキストを変更する
        Label1.Text = i.ToString()
    Next

    'Button1を有効に戻す
    Button1.Enabled = True
    'Button2を無効に戻す
    Button2.Enabled = False
End Sub

'Button2のClickイベントハンドラ
Private Sub Button2_Click(ByVal sender As Object, _
        ByVal e As EventArgs) Handles Button2.Click
    'Button2を無効にする
    Button2.Enabled = False

    'キャンセルのフラッグを立てる
    canceled = True
End Sub
```

C# コードを隠す コードを選択

```
//キャンセルボタンがクリックされたかを示すフラッグ
private bool canceled = false;

//Button1のClickイベントハンドラ
private void Button1_Click(object sender, System.EventArgs e)
{
    //Button1を無効にする
    Button1.Enabled = false;
    //Button2を有効にする
```

```
    Button2.Enabled = true;
    canceled = false;

    //コントロールを初期化する
    ProgressBar1.Minimum = 0;
    ProgressBar1.Maximum = 10;
    ProgressBar1.Value = 0;
    Label1.Text = "0";

    //時間のかかる処理を開始する
    for (int i = 1; i <= 10; i++)
    {
        //待機中のイベントを処理する
        Application.DoEvents();

        //キャンセルボタンがクリックされたか調べる
        if (canceled)
        {
            //キャンセルされた時
            MessageBox.Show(this, "ユーザーにより中止されました。");
            //ループを抜ける
            break;
        }

        //1秒間待機する（時間のかかる処理があるものとする）
        System.Threading.Thread.Sleep(1000);

        //ProgressBar1の値を変更する
        ProgressBar1.Value = i;
        //Label1のテキストを変更する
        Label1.Text = i.ToString();
    }

    //Button1を有効に戻す
    Button1.Enabled = true;
    //Button2を無効に戻す
    Button2.Enabled = false;
}

//Button2のClickイベントハンドラ
private void Button2_Click(object sender, EventArgs e)
{
    //Button2を無効にする
    Button2.Enabled = false;

    //キャンセルのフラッグを立てる
    canceled = true;
}
```

関連：

- 進行状況ダイアログを表示する
- BackgroundWorkerクラスを使用して進行状況ダイアログを作成する

履歴：

- 2010/10/24 「時間のかかる処理をユーザーが停止できるようにする」の内容をこちらに移す。「BackgroundWorkerコンポーネントを使用する方法」を追加。コードを多少変更（「スレッド化による方法」で、ボタンの状態を変更するメソッドの代わりに、処理が終わったときに呼び出されるメソッドを追加したなど）。
- 2010/12/13 BackgroundWorkerの説明を補充。
- 2011/2/16 VB.NETのコメントの誤記を修正。
- 2011/11/17 フォームを閉じられなくする方法へのリンクを追記。
- 2013/12/16 「WorkerReportsProgressプロパティ」と書くべきところが「CancellationPendingプロパティ」となっていたのを修正。

（この記事は、「.NETプログラミング研究 第44号」で紹介したものを基にしています。）

注意：この記事では、基本的な事柄の説明が省略されているかもしれません。初心者の方は、特に以下の点にご注意ください。

- **イベントハンドラの意味が分からない、C#のコードをそのまま書いても動かないという方は、こちらをご覧ください。**
- **.NET Tipsをご利用いただく際は、注意事項をお守りください。**

この記事へのコメント

評価の理由 [VB2017使ってます] *2019年3月3日 18:06:56*

評価：良い
とても分かりやすくて良かったです！ぼくは、VB2017を使ってプログレスバーを動かす、ということをしているのですが、すごく参考になりました！

評価の理由 [素人] *2016年3月8日 20:13:26*

評価：良い
すばらしい・・・

評価の理由 [匿名] *2014年4月15日 13:41:38*

評価：良い
非同期処理に困っていたため、大変助かりました。

通常のコメント [管理人] *2013年12月11日 04:17:12*

> "BackgroundWorkerのCancellationPendingプロパティをTrueにする必要があります。"
> とありますが、BackgroundWorker.CancellationPendingはReadOnlyではないでしょうか?

「WorkerReportsProgress」の間違いでした。ご指摘いただき、ありがとうございました。次の機会に修正させていただきます。

通常のコメント [匿名] *2013年12月9日 16:50:05*

"BackgroundWorkerのCancellationPendingプロパティをTrueにする必要があります。"

とありますが、BackgroundWorker.CancellationPendingはReadOnlyではないでしょうか?

残りのコメントをすべて見る（残り15件）

この記事に関するコメントを投稿するには、下のボタンをクリックしてください。投稿フォームへ移動します。通常のご質問、ご意見等は掲示板へご投稿ください。

コメントを投稿する

VB.NETのコードを非表示

C#のコードを非表示

その他

掲示板
Wiki
DOBON.NETへの要望
管理人に連絡
トップページに戻る