

大澤 文孝  
OSAWA, Fumitaka

# エンドユーザーは 帳票システムに何を求めているか カスタマイズ機能を視野に入れる

## Technology Tools

- ☐ Visual Basic .NET
- ☐ Visual C# .NET
- ☐ SQL Server 2000
- ☐ Oracle 9i
- ☐ Access 2002
- ☐ ASP.NET
- ☐ Internet Information Services
- ☐ Other:

## Level



## Samples

## はじめに

近年のエンドユーザーは、ワープロや表計算ソフトに使い慣れているため、業務アプリケーションの帳票機能にも、それと同等の機能を望むことが少なくありません。

その一方で、開発者にとっての帳票は、アプリケーションの中核ではなく、さして重要な部分ではないと考える節があります。

実際、業務アプリケーションはグラフィックソフトではありませんから、重要なのは、ロジック通りに動作するかどうかであり、美しく最終的な印刷物を作るのは主目的ではありません。

そのため開発者は、帳票機能の実装に、あまり時間や手間をかけたくないというのが本音でしょう。

もちろん、十分な時間がありさえすれば、エンドユーザーの要求通りの帳票を作ることができます。しかしそれは、開発期間、そして、開発コストが増すということを意味し、それがエンドユーザー（そして発注者）のために

なるとは限りません。

そのため帳票の実装は、「いかに手間をかけずにエンドユーザーの要求に応じていくのか」がポイントになると言えるでしょう。

そこで本稿では、エンドユーザーが帳票にどのような機能を望んでおり、どのようなアプローチで解決していけばよいのかを考えていきます。

## 帳票の難しさ

帳票とは、印刷面の指定された場所に特定のデータを埋め込んで印刷する、いわゆる「差し込み印刷機能」です。

原理的に言えば、集計したデータなどを特定の位置に印刷するロジックをアプリケーションに組み込めばよいことになります。

もっとも原始的な実装としては、APIを使ってプリンタの印刷面を得て、指定された位置にデータを文字として描画すればよいということになります。

具体的に.NET Frameworkで言えば、PrintDocumentオブジェクトを作り、

その描画ルーチン内で、DrawStringメソッドを使って、文字列を描画することになるでしょう。

このように述べると帳票は簡単に思えますが、実際には、用紙サイズに収まるようにするため、いくつかの例外的な処理が必要になります。

## 可変文字長の問題

差し込まれるデータの長さが、いつも一定長以下ならば、あらかじめ印刷面に埋め込まれる位置や大きさを固定できます。

しかし実際には、データの長さがまちまちで、データの内容によって、定めた大きさに入りきらないことがあります。

そこで、入りきらないときには、次のような実装が必要になります(図1)。

### ①無視してトリミング

入りきらない部分を無視して、印刷しないようにします。実装は簡単ですが、エンドユーザーは、これを許さないでしょう。

### ②無視してそのまま描画

はみ出た部分もそのまま描画します。これも実装は簡単ですが、他の場所に文字がかかったり、最悪、用紙サイズを超えたりすることになるので、やはりエンドユーザーは、許してくれないでしょう。

### ③縮小して押し詰める

文字の大きさを小さくして強引に押し詰めます。実装にあたっては、どの程度文字を小さくすれば収まるのかを計算するロジックが必要になるので少し複雑になります。とくにあまりに文字数が多すぎると、目に見えないほど文字が小さくなってしまう恐れがありますから、その配慮が必要です。縮小する場合には、文字の大きさを小さくする方法だけでなく、横幅だけを縮めて長体をかける方法もとれます。

### ④折り返す

入らない場合には、折り返して2行以上にして描画します。

行が増えるので、今度は、垂直方向に、はみ出してしまう恐れもあります。そこで、③の方法と組み合わせて、折り返しつつ文字を縮小して納める実装も考えられます。また日本語では、句読点などが行頭に来ることは許されないという「禁則処理」の問題もあり、どこで折り返すのかを決めるのが難しい側面もあります。

多くの場合、望ましい実装は、③か④ですが、どのような用紙に印刷するのか、また、どの程度はみ出るのかによって異なってきます。

はみ出してしまうのは、物理的に仕方ないので、「いかに破綻していないように見せるのか」が帳票実装の腕の見せ所になります。

## 入力できる文字長は帳票サイズから決定するのが望ましい

このように、文字がはみ出すことで考慮すると、帳票の実装が複雑になってきます。

そこで、すでにデータの時点で、「このデータは何文字以上には、なり得ない」と決めうちできるならば、そうすべきです。

たとえば、「住所」は、「住所」という1項目のデータとして入力するのではなく、「都道府県」「市区町村」「番地」「ビル名」などに分け、それぞれ決まった文字しか入らないようにしておいたほうが、帳票の実装が簡単になります。

同様に、「摘要」などのメモ的要素をもつ欄も、あまりに長い文字長の入力を許すと、帳票印刷のときに、はみ出す心配が出てきます。

発注者は、このあたりのことを知るよしもないので、

開発者：「この欄は、どのぐらいの長さが必要ですか？」

発注者：「今は、XX文字も入れば十分ですが、余裕をもって長いほうが良いですね」

といったやりとりで、あまりに余裕を持たせると、帳票印刷で、はみ出し処理を考慮しなければなくなる可能性があります。

もちろん、今後の拡張を考えれば、

図1：幅に入りきらない場合の処理

#### ①無視してトリミング

東京都新宿区舟町5 株式会社翔

#### ②無視してそのまま描画

東京都新宿区舟町5 株式会社翔泳社 出版局

#### ③縮小して押し詰める

(a) フォントの大きさを小さくする

東京都新宿区舟町5 株式会社翔泳社 出版局

(b) 長体をかける(幅だけ縮める)

東京都新宿区舟町5 株式会社翔泳社 出版局

#### ④折り返す

東京都新宿区舟町5 株式会社翔泳社 出版局

ある程度の余裕をもつことは重要です。しかし、余裕をもちすぎることもまた考え物です。

もしすでにどのような帳票に打ち出すのかが決まっているのであれば、帳票を見て、それに入りきる文字長を、入力可能な文字長として採用するのが良いでしょう。

## 表形式の帳票

帳票のうち、1枚でひとつのデータを出力する単票形式のものは、実装が比較的容易です。

それに対して、表形式(レポート形式)で出力する場合には、さらなる考慮が必要です。

表形式では、次の点が問題になります。

### ①改ページ

データの量が多いと、出力は多ページに渡ります。すなわち、改ページ処理が必要です。

行を描画していった、収まらなくなったならば、改ページするというのが基本的な処理の流れです。しかし、図1の④のケースのように、幅に収まりきらないときに折り返す場合には、折り返した部分がページをまたいでしまうことを避け、改ページ位置を調整しなければならないこともあります。

### ②用紙サイズと列の幅

表形式の出力では、エンドユーザーは、任意の用紙サイズを使いたがる傾向があります。

たとえば、普段はA4サイズで出力し

図2: 表形式の印刷

○用紙幅が足りないときには、適時折り返す

郵便番号	住所	顧客名	部署	電話番号
160-0006	東京都新宿区舟町 5	株式会社 翔泳社	出版局 ドットネットマガジン編集部	03-5362-3855

◎用紙幅が十分なときには、折り返さずに印刷する

郵便番号	住所	顧客名	部署	電話番号
160-0006	東京都新宿区舟町 5	株式会社 翔泳社	出版局 ドットネットマガジン編集部	03-5362-3855

×用紙サイズによらず、列幅を決め打ちして、余白を残すのは、エンドユーザーは望まない

郵便番号	住所	顧客名	部署	電話番号
160-0006	東京都新宿区舟町 5	株式会社 翔泳社	出版局 ドットネットマガジン編集部	03-5362-3855

△余白を残すのをやめて、単純に拡大出力するというのも、エンドユーザーの望む結果ではない

郵便番号	住所	顧客名	部署	電話番号
160-0006	東京都新宿区舟町 5	株式会社 翔泳社	出版局 ドットネットマガジン編集部	03-5362-3855

ているけれども、横に長いデータを含んでいるときには、B4サイズとかA3サイズといった大きめの用紙で印刷したいという要望です。

エンドユーザーは、当然、用紙幅を目一杯使って印刷しようと考えています。そのためプログラム側では、エンドユーザーが選択した用紙幅に合わせて、列の幅を調整する必要があります(図2)。

表形式の帳票では、エンドユーザーは、「特定の用紙に印刷したい」ということではなく、「用紙幅一杯に納めた表を印刷したい」ということを望みます。

ですから、エンドユーザーに列幅、そして場合によっては、列高さも設定できるカスタマイズの余地を与えるべきです。

表形式の帳票において、プログラム

側で描画位置や列の幅を決めうちしてしまうと、エンドユーザーにとって、使いにくい帳票になってしまいます。

## 実装を簡単にする コンポーネントの活用

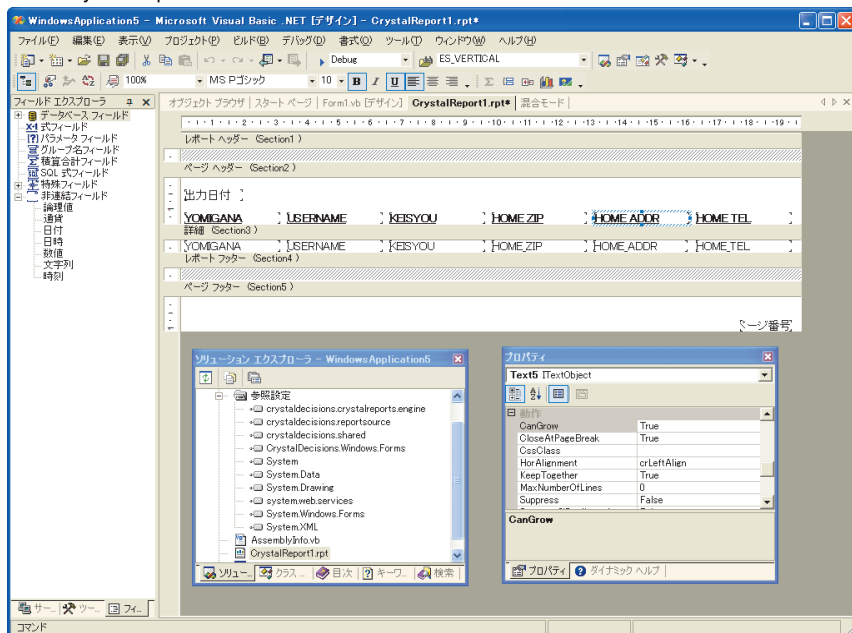
APIを使って帳票を実装するのは手間がかかるため、多くの場合、帳票コンポーネントを使うことになるでしょう。

たとえば、Visual Studio .NETには、帳票コンポーネントとして、Crystal Reportが付属しています。

帳票コンポーネントの多くは、印刷位置をテンプレートとして定めて開発します。

すなわち、あらかじめデータが差し込まれる位置をテンプレートとして定義しておき、実行時に、プログラムから差し込まれるデータをバインドして、印刷する方法をとります(図3)。

図3: Crystal Reportでのレポートの設計例



フィールドエクスプローラからドラッグして、どの項目をどこに差し込んでいくかを定義していく

## 帳票コンポーネントのメリット

帳票コンポーネントのメリットは、印刷位置をGUI画面できめ細かく設定することができるという点です。帳票コンポーネントの種類によっては、画像やグラフ、さらにはバーコードを挿入できるものもあります。

また、印刷プレビュー機能を実装しているものも多く、エンドユーザーが必要とするユーザーインターフェイス部分まで任せられるものもあります。

たとえば、Crystal Reportでは、CrystalReportViewerコントロールを使うことで、印刷と印刷プレビュー処理を任せることができます。

さらに、他の形式へのエクスポートをサポートしている帳票コンポーネントもあります。Crystal Reportでは、PDF形式やExcel形式、Word形式、RTF形式へのエクスポートができます。

## テンプレート方式では、開発時に位置やサイズが固定される

しかし帳票コンポーネントは、万能ではありません。

あらかじめ定型用紙や定型フォーマットがあり、そのなかに印刷してゆくアプリケーションであれば、多くの場合、うまくいくでしょう。

しかし図2に示したように、用紙サイズに応じて列の幅を変化させる必要があると、実現が難しくなります。

なぜなら、Crystal Reportのように、あらかじめ開発時にテンプレートを作り、そこにデータを差し込んで出力していく方式では、出力する位置や幅が開発時に定まってしまうためです。

そのため、表形式の帳票を作りたい場合には、あらかじめテンプレートを定めて埋め込むタイプの帳票コンポーネントではなく、列幅の変更が容易な表形式の処理を得意とする帳票コンポ

ーネントを採用する必要があります。

## 帳票をWordやExcelで作る

帳票の用紙や印刷位置は、実際に運用しはじめたあとに変更を要求される場面もあります。

帳票の定型用紙のフォーマットが運用中に変わってしまったという大きな変更だけでなく、「この部分は大きなフォントで」とか「単線ではなく二重線で」となどというエンドユーザーのわがままともとれる、細かい見栄えの要望も出てくるのがつねです。

そのような要望を逐一採り入れ、開発者が帳票のテンプレートを修正するというやり方もないわけではありませんが、それだと、手間がかかりますし、要望通りの帳票を作るには、さまざまな試行錯誤が必要になるでしょう。

そこで、柔軟な帳票を作れるようにするために、帳票の書式をエンドユーザーがカスタマイズできるようにすることが望まれます。

しかしそのためには、「どのデータをどの位置に印刷するのか」をエンドユーザーがコントロールできる実装にする必要があります。下手をすると、グラフィックソフト並みの機能が必要になってしまいます。

そこで考えたいのが、帳票の元となるデータをWordやExcelのデータとして吐き出し、エンドユーザーが、WordやExcelを使って、自分の使いやすいように帳票を整えられるようにする方法です。

## データをエクスポートする

アプリケーションから、WordやExcelのデータとして書き出す場合に、まず考えられるのが、COM (OLE) インターフェイスを使って、クライアントにインストールされているWordやExcelを制御する方法です。

この方法は、悪くはありませんが、次のような問題があります。

### ①クライアントにインストールされている

#### WordやExcelのバージョンの問題

クライアントにインストールされているWordやExcelのバージョンはまちまちである可能性があります。そのため、クライアントの環境によっては、正しく動作しないというトラブルが発生しやすくなります。

また、COMインターフェイスを参照設定して、事前バインディング (アーリーバインディング) で操作すると、COMインターフェイスはバージョンに依存しますから、うまくいきません。

すべてのバージョンに対応するためには、参照設定せず、事後バインディング (レイトバインディング) とし、Object型で操作することになります。

### ②処理速度が遅い

①とも絡みますが、COM経由でのWordやExcelの操作は、処理速度が遅いのが難点です。とくにObject型で操作することになれば、さらに遅くなります。

このような理由から、アプリケーションからWordやExcelを制御するのでは

なく、アプリケーションからWordやExcelが読み取り可能なデータを書き出し、エンドユーザーに、それをWordやExcelで開いてもらうほうがよいでしょう。

具体的には、アプリケーションからCSV形式でファイルとして書き出して、Excelで読み込んでもらうとか、Word形式やExcel形式のファイルを書き出せるコンポーネントを使って、ネイティブなファイルを作るといった方法をとると良いでしょう。

## Visual Studio Tools for OfficeやVBAを使う

エンドユーザーが、Word 2003やExcel 2003を利用しているのであれば、Visual Studio Tools for Officeを使うのも、良い方法です。

サーバー側でデータを保持しているのであれば、帳票データを吐き出すWebサービスのメソッドを用意することで、Visual Studio Tools for Officeで作り込んだアプリケーションから呼び出して、WordドキュメントやExcelワークブックにデータを取り込むことができます。

このような方法をとれば、エンドユーザーは、WordドキュメントやExcelワークブックのレイアウトや書式を変更することによって、帳票を自由にカスタマイズできるようになります。

もちろん、Visual Studio Tools for Officeでなく、VBAを使って実装しても良いでしょう。

## WordやExcelを使った帳票のメリット

WordやExcelを使って帳票出力する

ようにすれば、開発者は帳票の元データを用意するだけでよく、レイアウトや印刷を一切考えなくて済むので、帳票実装の手間を大幅に削減できます。

そしてエンドユーザーにとっては、帳票のレイアウトを自由に変更できるばかりか、そのまま印刷する以外にも、他の用途に帳票データを流用できるというメリットもあります。

WordやExcelに任せてしまうという方法は、一見すると開発者が手を抜いたとも思われがちです。

しかし、エンドユーザーにとっても、アプリケーション独自の低機能な帳票印刷だけが実装されるよりもずっとメリットが多く、双方が幸せになれる方法のひとつだと言えるでしょう。

## PDFの活用

近年は帳票の作成に、PDFを利用する例も増えています。そこで、最後にPDFを帳票に使うときのメリットと注意点について述べておきます。

### 開発者にとってのメリット

開発者にとってのPDFのメリットは、プリンタを直接制御する必要がないという点です。そのため、サーバー側で帳票を生成する場面にも向いています。

PDFを作るには、PDFを書き出す機能をもつコンポーネントを使えばよく、実装も比較的簡単です。

PDFに対応するコンポーネントは数多く、フリーのPDF作成コンポーネントもいくつか存在します。また、Crystal



## 帳票にAccessを使う

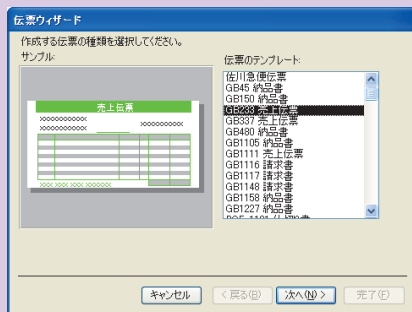
WordやExcelではなく、帳票処理にAccessを使うという方法もあります。

たとえば、データの保存にSQL Serverを利用しているのであれば、期限や集計条件などを引数に渡すと、その帳票データを返すようなストアドプロシージャを作っておけば、それをAccessで取り込み、レポートとして印刷できます。

Accessで帳票を作るメリットは、「宛名ラベルウィザード」「はがきウィザード」「伝票ウィザード」(図A)といった各種ウィザードを使うことによって、さまざまな用紙を使った帳票を簡単に作れるという点です。

ラベルや伝票、葉書や封筒に印刷したいという場面では、Accessのレポート機能を使うと、簡単に実装できます。

図A：伝票ウィザード



ReportでもPDFの書き出しができます。

## エンドユーザーにとってのメリット

PDFは、エンドユーザーにとっても、メリットがあります。

PDFは、ファイルとして扱えますから、メールに添付してやりとりすることが容易です。

そしてPDF内のテキストは検索することもできるため、サーバーに長期保存しておきたい場合にも向きます。

また、(Adobe Readerではなく) Adobe Acrobatをもっていれば、不要な部分を取り除いたり、コメントを付けたり、マーキングしたりするなどの加工もできます。

PDFは、主にWebアプリケーションにおける帳票作成に使われますが、Windowsアプリケーションであっても、PDF書き出し機能があると、エンドユーザーにとって、利便性が高くなるでしょう。

## PDFを使う注意点

PDFは便利な反面、いくつか注意しなければならないこともあります。

### ①フォントの埋め込み

PDFを作成するときには、フォントを埋め込むかどうかを指定できます。

フォントを埋め込むとPDFファイルサイズが大きくなるので、帳票として利用するのであれば、フォントは埋め込まないのが一般的です。しかし他のOSで開くことを想定する場合には、フォントを埋め込んでおかないと、異なるフォントに置き換わるばかりか、妙に間延びしたり、詰まったりして表示されることがあります。

そのためレイアウトの崩れを心配するならば、フォントは埋め込んでおいたほうがよいかもしれません。

### ②暗号化と編集禁止

PDFファイルは、(Adobe Readerで

はなく) Adobe AcrobatやAdobe Illustratorなどのアプリケーションで開いて編集できます。そのためPDFの偽造を防止するのであれば、暗号化しなければなりません。

また、PDFファイルを生成するときには、「印刷禁止」や「編集禁止」の属性を付けることもできます。ただし、「印刷禁止」や「編集禁止」は、紳士協定的な単純なフラグにすぎません。

PDFの仕様書を見るとわかりますが、印刷禁止や編集禁止の項目は、それぞれ1ビットのフラグとして実装されており、暗号化されていない場合には、原理的には解除できます。

よって変更させたくない場合には、暗号化を施す必要があります。

## PDFの生成能力はコンポーネントに大きく依存する

PDFの生成能力は、現状では、利用するコンポーネントによって大きく違うので注意が必要です。

たとえば、フリーのPDF作成コンポーネントで、元々英語版であったものは、日本語の対応が十分でないものも多くあります。

日本語に対応していても、フォントの埋め込みがうまくできないものや、縦書き出力できないとか、縦書き出力するとカギ括弧や長音記号などが回転されずに出力されるという問題をもつものもあります。

また暗号化できるのかとか、印刷禁止や編集禁止設定ができるか否かという点も、コンポーネントに依存します。

そしてPDFに埋め込んだテキストが正しく検索可能かどうかコンポーネ

ントに依存します。コンポーネントによっては、文字列をブツ切りにして1文字ずつ描画したり、折り返された部分で文字列を分離して描画したりするものもあり、切られた部分で文字列の検索ができないこともあります。

そのため、アプリケーションでPDFをサポートするのであれば、どのようなPDF作成コンポーネントを使うべきか、よく検証すべきです。

## まとめ

今は多種多様なアプリケーションがある時代です。

そのため、エンドユーザーが求める帳票システムとは、「業務アプリケーション単体で帳票印刷できるようにするもの」とは限らないということを念頭に置くべきです。

もちろん帳票が毎日数百枚、数千枚の単位で出力される場面では、業務アプリケーションだけで帳票処理が完結し、エンドユーザーの手間をかけさせないものが望まれるでしょう。

しかしそうではなく、見積書や請求書を出したいといった場面では、単純に印刷するのではなく、Excelなどの形式で出力し、「あとからコメントを書き込んで顧客に渡す」「メールで送付する」「別の請求書と合算して新たな請求

書を作る」など、エンドユーザーがあとから手を加えられる帳票が望まれることもあります。

現在でも、メールを中心に帳票データがファイルとしてやりとりされる機会は増えていますが、今後、電子署名の活用によって、印刷物としての帳票ではなく、ファイルとしての帳票が要求される場面は増えていくでしょう。

そう考えると、今後の帳票システムは、「アプリケーション内で印刷できればそれで良い」というものではなく、エンドユーザーが望むデータを、柔軟な形式で、ファイルとして書き出せるものが求められると言えるでしょう。

## 標準ASP.NETプログラミング ① ~Webアプリケーション構築編~ 標準ASP.NETプログラミング ② ~XML Webサービス構築編~

SCOTT MITCHELL 他 著 西谷亮 監訳

定価①4,410円(本体4,200円+税5%) ②3,990円(本体3,800円+税5%) 各 B5変型判 ①640ページ ②528ページ ①ISBN4-7981-0125-7 ②ISBN4-7981-0126-5

本書は、中～上級者レベルの開発者を対象に、ASP.NETを使ったさまざまなソリューションの構築について、「Webアプリケーション構築編」と「XML Webサービス構築編」の2巻構成で説明します。実行画面やサンプルコードを豊富に紹介しているので、経験の浅い開発者でもASP.NETの奥深い技術をしっかりと学ぶことができます。

## 標準VB.NETプログラミング ~.NET環境への移行と開発の基礎~

DAN APPLEMAN 著 グレープシティ株式会社 監訳

定価4,410円(本体4,200円+税5%) B5変型判 640ページ ISBN4-7981-0216-4

本書は、VB6.0の中～上級ユーザーを対象に、VB6.0からVB.NETへの移行をテーマにしています。VBを知り尽くしているダニエル・アップルマンが、これまでVisual Basicを使い続けてきた開発者がつまづくことのないよう解説します。また、豊富なサンプルコードを掲載しているので、迷うことなく.NET対応プログラムを開発することができます。

## 実践C++/C#.NETプログラミング Visual Studio .NETによるアプリケーション構築

RICHARD GRIMES 著 ハラバン・メディアテック 宇野俊夫監訳

定価6,279円(本体5,980円+税5%) B5変型判 744ページ ISBN4-7981-0348-9

本書は、公式解説書やその他の解説書を卒業したC++/C#プログラマーが、Visual Studio .NETを使って、より実践的、実用的なプログラミングテクニックを身に付けるための書籍です。1～4章では.NET Framework、5～7章ではVisual Studio .NETでアプリケーション開発に利用できるツール類、8～9章ではアプリケーションの開発とデバッグについて取り上げ、著者が独自にIL(中間言語)を逆アセンブルしてハック(解析)した結果をもとにして、様々なテクニックやノウハウが紹介されています。

## 標準C#.NETプログラミング ① ~C#言語構文編~ 標準C#.NETプログラミング ② ~.NETアプリケーション開発編~

ANDREW TROELSEN 著 矢沢久雄 監訳

定価①3,360円(本体3,200円+税5%) ②3,360円(本体3,200円+税5%) 各 B5変型判 ①448ページ ②608ページ ①ISBN4-7981-0111-7 ②ISBN4-7981-0112-5

本書は、中～上級者レベルの開発者を対象に、C#言語の詳細および.NET対応プログラムの開発技法を解説した書籍です。「C#言語を知りたい、.NETも知りたい」という開発者にお勧めです。

株式会社翔泳社 東京都新宿区舟町5 〒160-0006  
出版局営業部: TEL.03-5362-3810 FAX.03-5362-3817 巻末の振替用紙で直接ご購入いただけます。

プロフェッショナルSEの知的探求心を満足させる日本初のITセレクトショップ  
<http://www.seshop.com/>

SEshop.com

SE  
SHOEISHA