

Visual COBOL チュートリアル

Apache Tomcat と JVM COBOL コンポーネントを利用した RESTful Web サービス開発

1 目的

Visual COBOL は、他言語・他システムとの連携手段として、「Enterprise Server」を利用したサービス連携だけではなく、COBOL プログラムをそのまま利用し、Java 技術と連携可能な JVM COBOL 機能を提供しています。本機能を利用することで、Java, Scala などの Java 言語で作成した RESTful Web サービス上で COBOL を利用するといった方法が可能です。

このドキュメントでは JVM COBOL 機能を利用して Java で実装する RESTful Web サービスと COBOL の連携方法について説明します。

2 前提条件

本チュートリアルは、下記環境を前提に作成されています。

OS	Windows 10
COBOL 環境製品	Visual COBOL 10.0 for Eclipse
Java アプリケーションサーバー	Apache Tomcat 10.1.28

Java 言語で作成する RESTful Web サービスは JAX-RS という Web サービスを構築するための Java API を使用しています。本ドキュメントでは、Tomcat 10.1 を Java アプリケーションサーバーとして使用して動作確認を行います。Apache Tomcat に依存せず標準的な API 仕様に基づいたサンプルアプリケーションとなります。このため、本ドキュメントで紹介するようなサービスを、他の Java アプリケーションサーバー上にて開発・利用することも可能です。

また、Eclipse の Tomcat プラグインに関する設定および動作などにつきましては、弊社製品外であり、インターネットなどの各種文献をご参照ください。

下記のリンクから事前にチュートリアル用のサンプルファイルをダウンロードして、任意のフォルダーに解凍しておいてください。

このサンプルプログラムは、COBOL で作成された簡単な書籍情報を管理するアプリケーションであり、索引ファイルを利用しています。以降の手順では、C:\jvmRESTfulWSTutorial を解凍先のフォルダーとして説明しています。

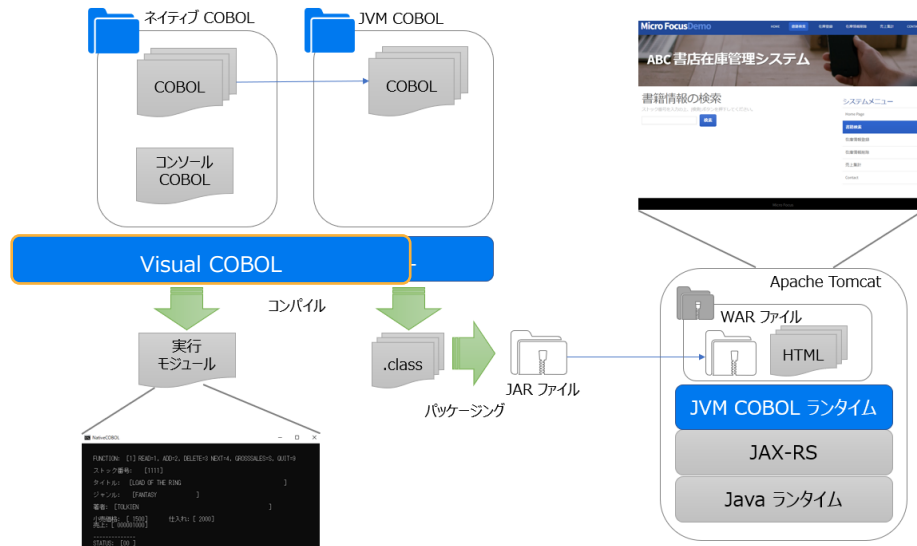
[チュートリアル用のサンプルファイルのダウンロード](#)

内容

- 1 目的
- 2 前提条件
- 3 チュートリアル手順の概要
 - 3.1 プロジェクトの作成と COBOL アプリケーションの確認
 - 3.1.1 プロジェクトの作成
 - 3.1.2 アプリケーションの確認
 - 3.2 JVM COBOL プロジェクトの作成
 - 3.3 Apache Tomcat 上で動作する RESTful Web サービスの作成
 - 3.3.1 Java Web アプリケーションプロジェクトの作成
 - 3.3.2 RESTful Web サービスの確認

3 チュートリアル手順の概要

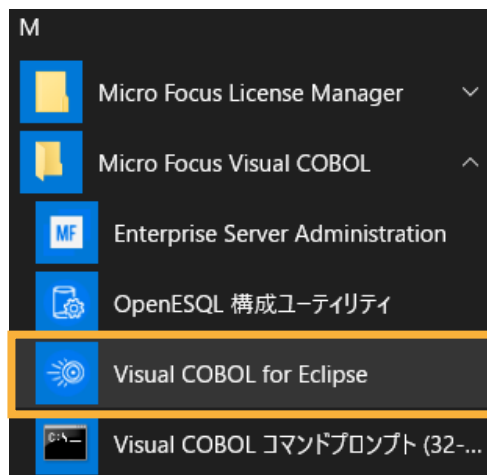
従来型の書籍情報を管理するコンソールアプリケーションを、JVM COBOL の機能を利用して COBOL プログラムそのまま流用し、Java クラスを生成します。そして、このクラスを利用して Java ベースの RESTful Web アプリケーションを構築します。最後に、Web アプリケーションを Apache Tomcat アプリケーションサーバーにデプロイし、実際の動作を確認します。



3.1 プロジェクトの作成と COBOL アプリケーションの確認

3.1.1 プロジェクトの作成

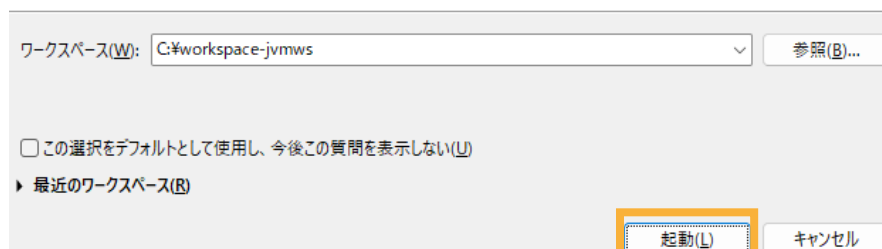
- 1) Visual COBOL for Eclipse を起動します。



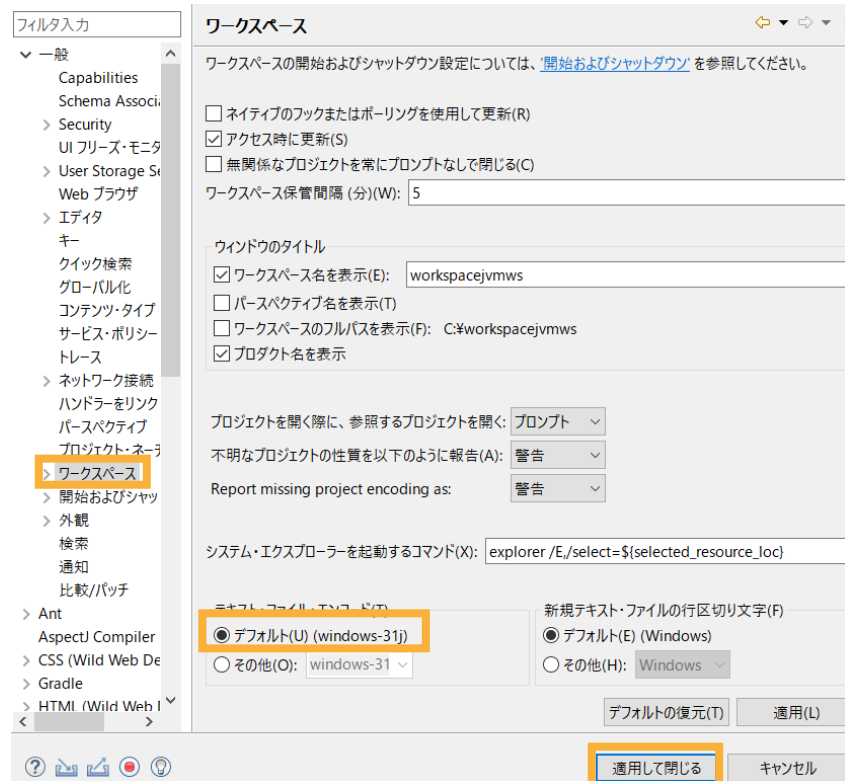
- 2) 任意のワークスペースを選択し、[起動(L)] をクリックします。

ディレクトリーをワークスペースとして選択

Eclipse は、ワークスペースディレクトリを使用して、環境設定と開発成果物を保存します。

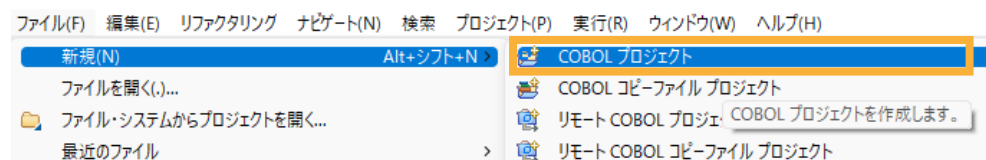


- 3) SJIS 資産を利用する場合、文字コードの指定を明確に行う必要があります。[Window(W)] > [設定(P)] より [一般] > [ワークスペース] とナビゲートし、テキストファイルエンコードを「デフォルト(windows-31j)」に変更し、[適用して閉じる] をクリックします。



Preference Recorder のダイアログが表示された場合は、[キャンセル] をクリックします。

- 4) [ファイル(F)] > [新規(N)] > [COBOL プロジェクト] を選択します。



- 5) 以下の入力を行い、[終了(F)] をクリックします。

プロジェクト名 : “NativeCOBOL”

プロジェクトテンプレート : 64 ビット

COBOL プロジェクト

ワークスペースまたは外部の場所にCOBOL プロジェクトを作成します。



プロジェクト名(P):

プロジェクトテンプレートを選択

☒ Micro Focus テンプレート [32 ビット]
 ☒ Micro Focus テンプレート [64 ビット]

[テンプレートの設定を構成...](#)

☐ テンプレートの参照

場所:

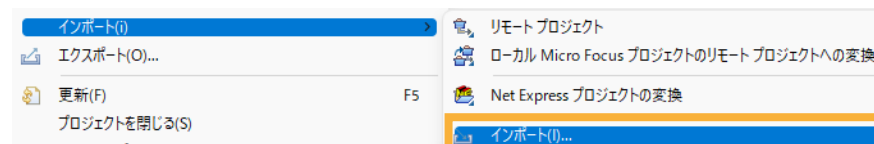
ファイルシステムを選択: default ▾

☒ デフォルト・ロケーションの使用(D)

ロケーション(L):

ファイル・システムを選択(Y): デフォルト ▾

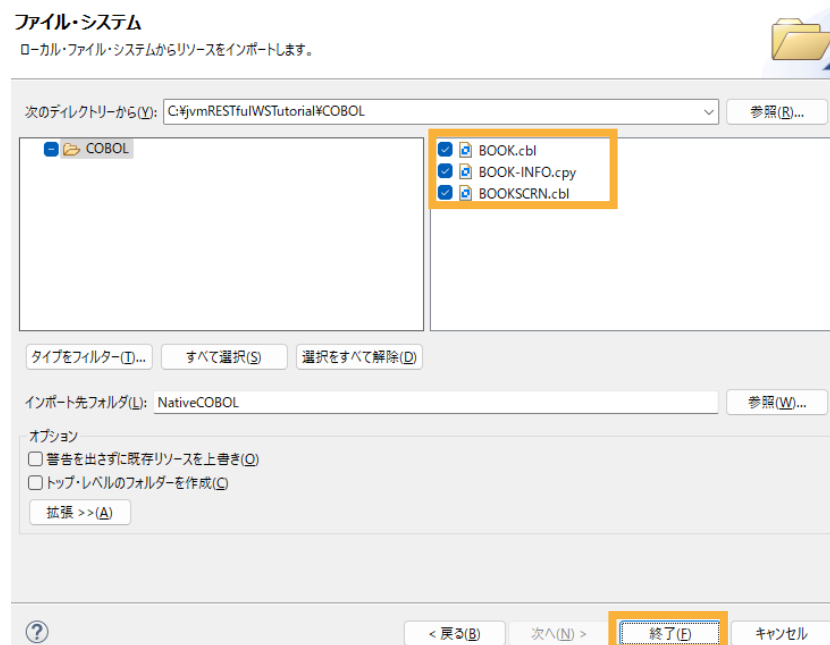
- 6) NativeCOBOL プロジェクトを選択し、マウスの右クリックにてコンテキストメニューを開き、[インポート(I)] > [インポート(I)] を選択します。



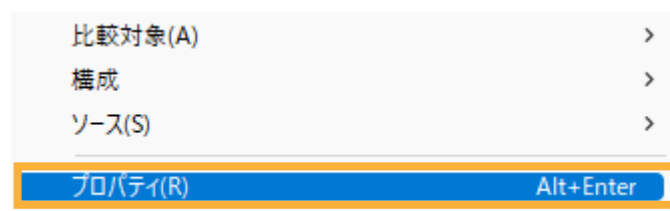
- 7) 一般フォルダー配下の [ファイル・システム] を選択して、[次へ(N)] をクリックします。



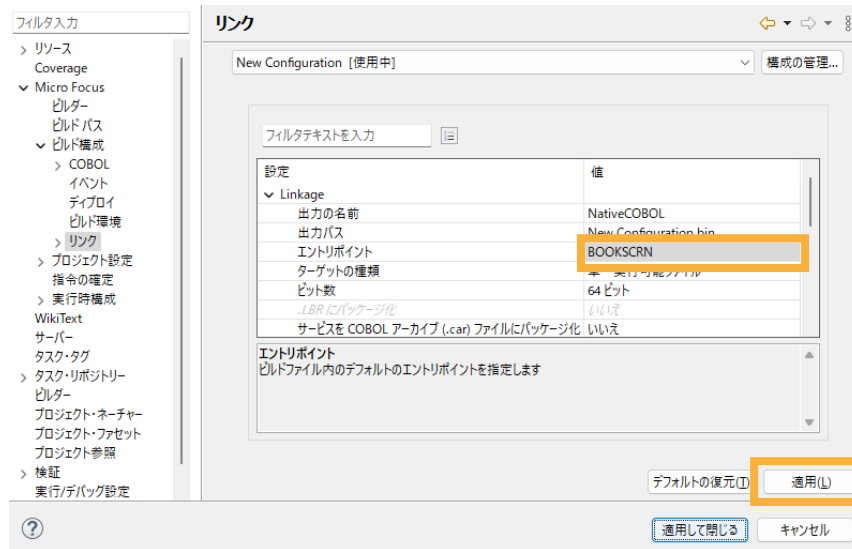
- 8) C:\jvmRESTfulWSTutorial¥COBOL 配下の 3 つのファイルを選択し、[終了(F)] をクリックします。



- 9) NativeCOBOL プロジェクトを選択し、マウスの右クリックにてコンテキストメニューを表示した上で、[プロパティ(R)] を選択します。



- 10) [Micro Focus] > [ビルド構成] > [リンク] を選択し、「エントリポイント」に “BOOKSCRN” を入力したうえで、
[適用(L)] をクリックします。



フィルタ入力

- > リソース
 - Coverage
 - Micro Focus
 - ビルダー
 - ビルドパス
 - ビルド構成
 - COBOL
 - イベント
 - ディプロイ
 - ビルド環境
 - リンク
 - プロジェクト設定
 - 指令の確定
 - 実行時構成
 - WikiText
 - サーバー
 - タスク・タグ
 - タスク・リポジトリ
 - ビルダー
 - プロジェクト・ネーチャー
 - プロジェクト・ファセット
 - プロジェクト参照
 - 検証
 - 実行/デバッグ設定

リンク

New Configuration [使用中]

フィルタテキストを入力

設定	値
Linkage	
出力の名前	NativeCOBOL
出力パス	New Configuration bin
エントリポイント	BOOKSCRN
ターゲットの種類	実行可能ファイル
ビット数	64 ビット
LLBR にパッケージ化	いいえ
サービスを COBOL アーカイブ (.car) ファイルにパッケージ化	いいえ

エントリポイント
ビルドファイル内のデフォルトのエントリポイントを指定します

デフォルトの復元(D) 適用(L)

適用して閉じる キャンセル

- 11) [Micro Focus] > [プロジェクト設定] > [COBOL] を選択し、以下の入力を行ったうえで、[適用して閉じる]
をクリックします。

[ソースエンコーディング]: “ANSI”

[追加指令]: “ASSIGN(EXTERNAL)”



設定	値
一般	
文字セット	ASCII
ソース エンコーディング	ANSI
COBOL 方言	Micro Focus
ソース フォーマット	固定
デバッグ用にコンパイル	はい
EXIT PROGRAM を GOBACK として処理	ANSI
詳細	いいえ
.GNT にコンパイル	いいえ
出力	
指令ファイルを生成する	いいえ
リストファイルを生成	いいえ
コードカバレッジを有効にする	false
プロファイラを有効にする	false
エラー/警告	
警告レベル	回復可能なエラーを含める(レベル E)
最大エラー数	100
追加指令	
追加指令	ASSIGN(EXTERNAL)

追加指令
コンパイラに渡す追加のCOBOLコンパイラ指令です

COBOL コンパイル設定:

CHARSET"ASCII" SOURCE-ENCODING"ANSI" DIALECT"MF" SOURCEFORMAT"fixed" NOLIST anim
EXITPROGRAM"ANSI" NOTESTCOVER NOPROFILE WARNING"1" MAX-ERROR"100" ASSIGN
(EXTERNAL)

デフォルトの復元(D) 適用(L)

適用して閉じる キャンセル

3.1.2 アプリケーションの確認

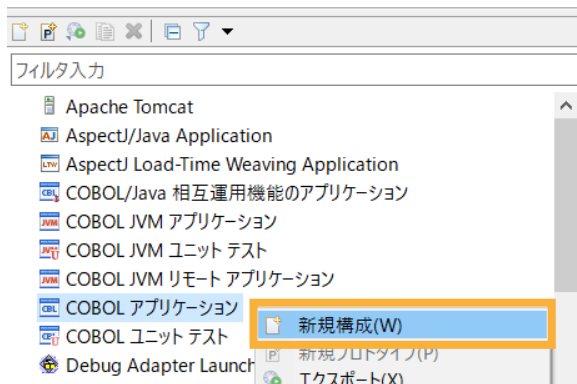
- 1) NativeCOBOL プロジェクトを選択し、[実行(R)] > [実行構成(N)] を選択します。



- 2) [COBOL アプリケーション] を選択し、マウスの右クリックにてコンテキストメニューを表示した上で、[新規構成(W)] をクリックします。

構成の作成、管理、および実行

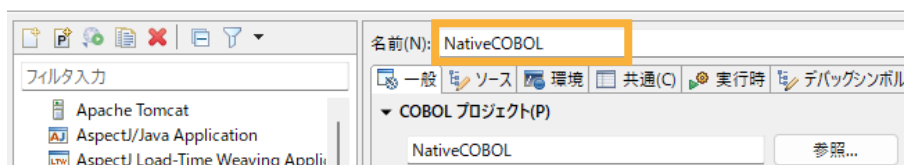
COBOL プログラムを実行します



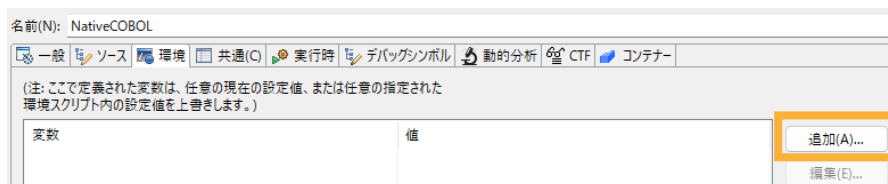
- 3) 「名前」に “NativeCOBOL” を入力します。

構成の作成、管理、および実行

COBOL プログラムを実行します



- 4) [環境] タブを選択し、[追加(A)] をクリックします。



- 5) 以下の入力を行い、[OK] をクリックします。

変数: “BOOKINFO”

値: “C:¥jvmRESTfulWSTutorial¥DAT¥BOOKINFO.DAT”

環境変数を追加または変更します



変数: BOOKINFO

値: C:\jvm\RESTfulWSTutorial\DAT\BOOKINFO.DAT

?

OK キャンセル

6) BOOKINFO 変数が追加されたことを確認して、[実行(R)] をクリックします。

名前(N): NativeCOBOL

一般 ソース 環境 共通(C) 実行時 デバッグシンボル 動的分析 CTF コンテナー

(注: ここで定義された変数は、任意の現在の設定値、または任意の指定された環境スクリプト内の設定値を上書きします。)

変数	値
BOOKINFO	C:\jvm\RESTfulWSTutorial\DAT\BOOKINFO.DAT

追加(A)...
編集(E)...
削除(R)

実行する環境スクリプト:
場所: 参照...

ファイルがプロジェクト内にある場合、絶対パスは相対パスになります。

パラメータ:

☐ 関連付けられたプロジェクトのビルド環境から値を継承

前回保管した状態に戻す(V) 適用(Y)

実行(R) 閉じる

以下のような画面が表示されます。

NativeCOBOL

```

FUNCTION:  [_] READ=1, ADD=2, DELETE=3 NEXT=4, GROSSSALES=S, QUIT=9
ストック番号:  [   ]
タイトル:  [                                     ]
ジャンル:  [                                     ]
著者:  [                                     ]
小売価格: [   0]   仕入れ: [   0]
売上: [ 000000000]
-----
STATUS:  [   ]
    
```

以下の入力を行った後、Enter キーを押すと、該当書籍情報が表示されます。

FUNCTION : "1"

ストック番号 : "1111"

NativeCOBOL

FUNCTION: [1] READ=1, ADD=2, DELETE=3 NEXT=4, GROSSSALES=S, QUIT=9

ストック番号: [1111]

タイトル: [LOAD OF THE RING]

ジャンル: [FANTASY]

著者: [TOLKIEN]

小売価格: [1500] 仕入れ: [2000]

売上: [000001000]

STATUS: [00]

現在、ストック番号：4444 は未登録のため、以下の入力を行い、Enter キーをクリックします。

FUNCTION: "2"

ストック番号： "4444"

タイトル： "鏡の国のアリス"

ジャンル： "ファンタジー"

著者： "ルイス・キャロル"

小売価格： "1200"

仕入れ： "3000"

売上： "4000"

NativeCOBOL

FUNCTION: [2] READ=1, ADD=2, DELETE=3 NEXT=4, GROSSSALES=S, QUIT=9

ストック番号: [4444]

タイトル: [鏡の国のアリス]

ジャンル: [ファンタジー]

著者: [ルイス・キャロル]

小売価格: [1200] 仕入れ: [3000]

売上: [000004000]

STATUS: [00]

実行後、以下の入力を行い、書籍情報が登録されていることを確認してください。

FUNCTION: "1"

ストック番号： "4444"

確認後、以下の入力を行い、Enter キーを押すことで登録した書籍情報が削除されます。

FUNCTION: "3"

ストック番号： "4444"

```
NativeCOBOL

FUNCTION: [2] READ=1, ADD=2, DELETE=3 NEXT=4, GROSSSALES=S, QUIT=9
ストック番号: [4444]
タイトル: [鏡の国のアリス]
ジャンル: [ファンタジー]
著者: [ルイス・キャロル]
小売価格: [1200] 仕入れ: [3000]
売上: [000004000]

-----
STATUS: [00]
```

実行後、READ 機能でストック番号： 4444 が削除されていることを確認してください。

FUNCTION=S の GROSSSALES 機能は登録された全書籍情報の売上額を集計します。

```
NativeCOBOL

FUNCTION: [S] READ=1, ADD=2, DELETE=3 NEXT=4, GROSSSALES=S, QUIT=9
ストック番号: [ ]
タイトル: [*****]
ジャンル: [*****]
著者: [*****]
小売価格: [ 0] 仕入れ: [ 0]
売上: [017000000]

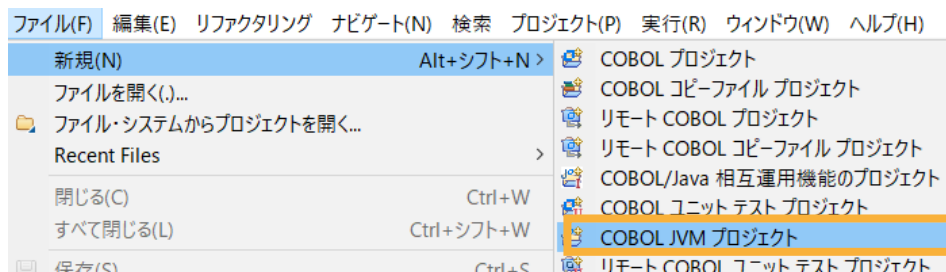
-----
STATUS: [00]
```

確認後、「FUNCTION」に “9” を入力し、Enter キーを押してプログラムを終了します。

3.2 JVM COBOL プロジェクトの作成

本節では、さきほど確認した従来の COBOL プログラムを JVM COBOL としてコンパイルを行います。

- 1) Visual COBOL for Eclipse メニューより、[ファイル(F)] > [新規(N)] > [COBOL JVM プロジェクト] を選択します。



- 2) 「プロジェクト名」に “BookLibrary” を入力して、[終了(F)] をクリックします。

COBOL JVM プロジェクト
ワークスペースまたは外部の場所に COBOL JVM プロジェクトを作成します。

プロジェクト名(P):

☒ デフォルト・ロケーションの使用(D)
ロケーション(L): 参照(R)...

プロジェクトテンプレートを選択
Micro Focus テンプレート

☐ テンプレートの参照
場所: 参照...
ファイルシステムを選択: default

JRE
☒ 実行環境 JRE の使用(V):
☐ プロジェクト固有の JRE を使用(S):
☐ Use default JRE 'AdoptOpenJDK' and workspace compiler preferences [JRE を構成...](#)

ワーキング・セット
☐ ワーキング・セットにプロジェクトを追加(T) 新規(W)..
ワーキング・セット(O): 選択(E)...

< 戻る(B) 次へ(N) > **終了(F)** キャンセル

- 3) BookLibrary プロジェクトを選択し、マウスの右クリックにてコンテキストメニューを表示したうえで、[プロパティ(R)] を選択します。



- 4) [Micro Focus] > [ビルド構成] を選択し、以下の入力をしたうえで、[適用して閉じる] をクリックします。

[ソースエンコーディング]: “ANSI”

[追加指令]: “ASSIGN(EXTERNAL) ILSMARTLINKAGE ILNAMESPACE(com.microfocus.sample)”

フィルタ入力

- > リソース
 - Coverage
 - Micro Focus
 - JVM ビルド パス
 - > SQL プリプロセッサ
 - コピーファイル
 - ビルダー
 - ビルド環境
 - ビルド構成**
 - ビルド優先順位

ビルド構成

フィルタテキストを入力

設定	値
▼ 一般	
文字セット	ASCII
ソース エンコーディング	ANSI
COBOL 方言	Micro Focus
ソース フォーマット	可変

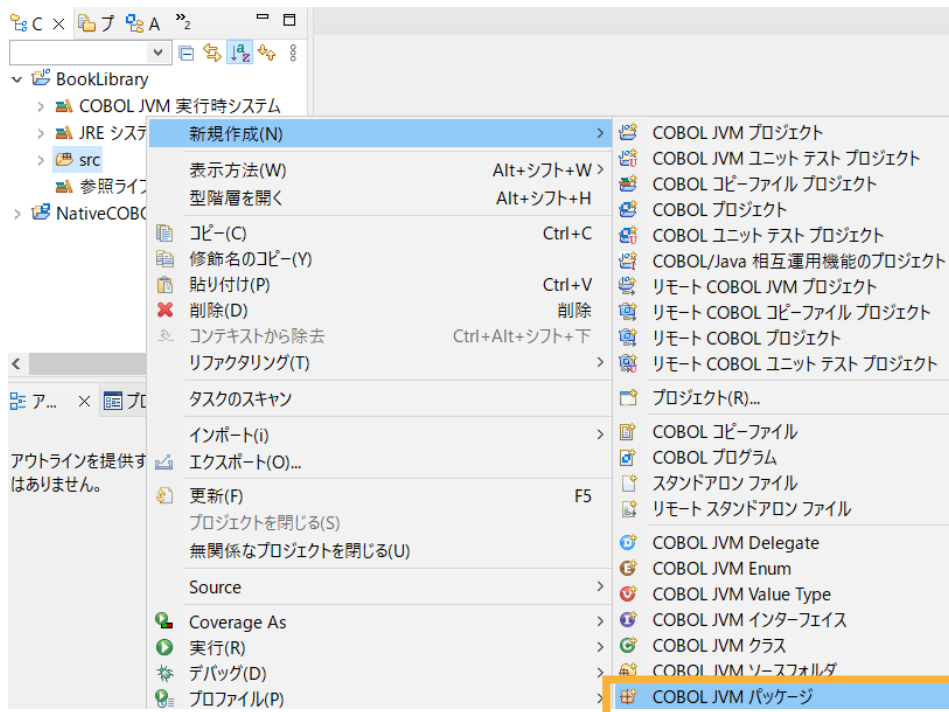
▼ エラー/警告	
警告レベル	重大なエラーだけ(レベル 5)
最大エラー数	100
▼ 追加指令	
追加指令	ASSIGN(EXTERNAL) ILSMARTLINKAGE ILNAMESPACE(com.microfocus.sample)
ソース エンコーディング SOURCE-ENCODING はソース プログラムのエンコーディングをコンパイラに渡します。その後、RUNTIME-ENCODING 指令が指定されていない COBOL コンパイル設定: CHARSET"ASCII" SOURCE-ENCODING"UTF8" DIALECT"MF" SOURCEFORMAT"variable" NOLIST anim EXITPROGRAM"ANSI" packageToFolderMapping.jarFolder=dist BookLibrary.jar outputDirectory=bin NOWARNING MAX-ERROR"100" ASSIGN (EXTERNAL) ILSMARTLINKAGE ILNAMESPACE(com.microfocus.sample)	
デフォルトの復元(T) 適用(L) 適用して閉じる キャンセル	

補足)

ILSMARTLINKAGE コンパイラ指令により、COBOL と Java 言語でのデータ型の差異を吸収するラッパークラスがコンパイル時に自動生成されます。本指令の詳細については、製品マニュアルトップより、[リファレンス] > [コンパイラ指令] > [コンパイラ指令 - アルファベット順一覧] > [ILSMARTLINKAGE] を参照してください。

ILNAMESACE コンパイラ指令は、JVM COBOL の出力先のパッケージ階層を指定するもので、今回は、com.microfocus.sample パッケージ配下に出力します。本指令の詳細については、製品マニュアルトップより、[リファレンス] > [コンパイラ指令] > [コンパイラ指令 - アルファベット順一覧] > [ILNAMESPACE] を参照してください。

- 5) BookLibrary プロジェクト配下の src フォルダを選択し、マウスの右クリックにてコンテキストメニューを表示した上で、[新規作成(N)] > [COBOL JVM パッケージ] を選択します。



- 6) 「名前」に “com.microfocus.sample” を入力して、[終了(F)] をクリックします。

COBOL JVM パッケージ

COBOL JVM パッケージを新規作成します。

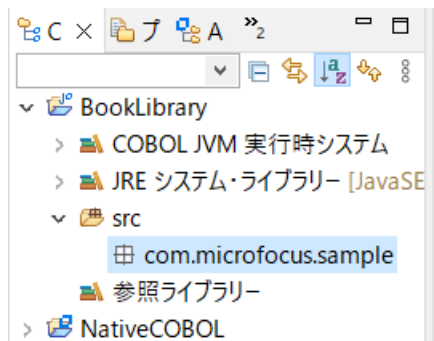
パッケージに対応するフォルダを作成します。

ソース フォルダ(D): BookLibrary/src 参照(O)...

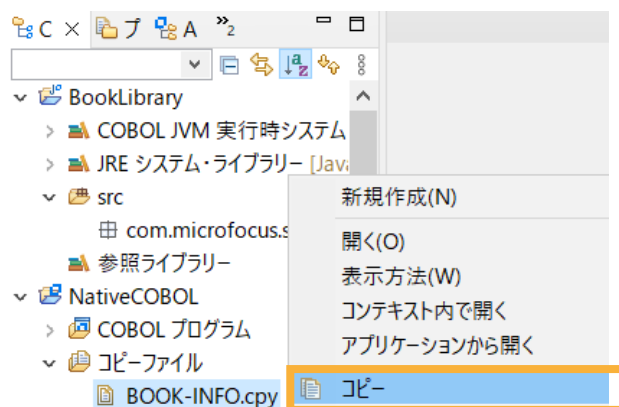
名前(M): com.microfocus.sample

終了(E) キャンセル

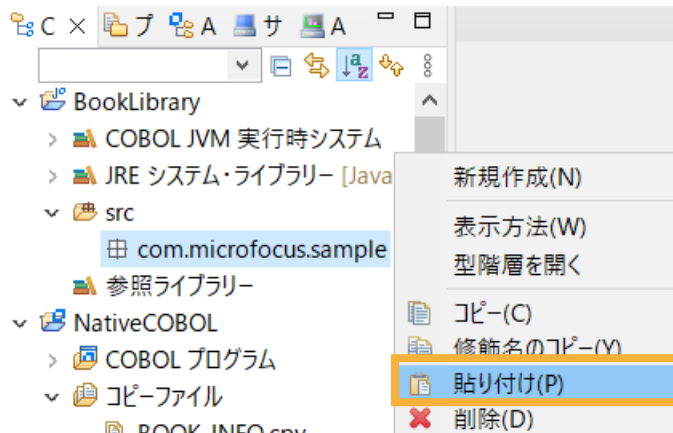
src フォルダー配下に、空のパッケージが作成されます。



- 7) さきほどの NativeCOBOL プロジェクトの「コピーファイル」配下の “BOOK-INFO.cpy” を選択し、マウスの右クリックにてコンテキストメニューを表示したうえで、[コピー] を選択します。

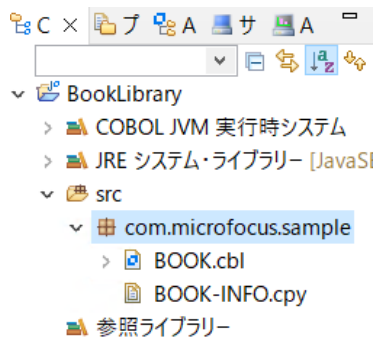


- 8) BookLibrary プロジェクト配下の src¥com.microfocus.sample フォルダーを選択し、マウスの右クリックにてコンテキストメニューを表示したうえで、[貼り付け(P)] を選択します。



この操作により、BOOK-INFO.cpy が com.microfocus.sample フォルダ配下にコピーされます。

- 9) さきほどと同様の手順で、NativeCOBOL プロジェクトの「COBOL プログラム」配下の “BOOK.cbl” を BookLibrary プロジェクト配下の src¥com.microfocus.sample フォルダにコピーしてください。コピー後は以下のように表示されます。



デフォルトで自動的にビルドが有効になっている場合、BOOK.cbl をコピーした段階でコンパイルが行われ、プロジェクトが保存されたフォルダ配下の bin¥com¥sample フォルダに、以下の .class ファイル、すなわち、Java バイトコードが生成されます。

« workspace-jvmws > BookLibrary > bin > com > microfocus > sample

```

BOOK$_MF_LCTYPE_1.class
BOOK.cbldat
BOOK.class
LnkBDetails.cbldat
LnkBDetails.class
LnkFileStatus.cbldat
LnkFileStatus.class
LnkFunction.cbldat
LnkFunction.class

```

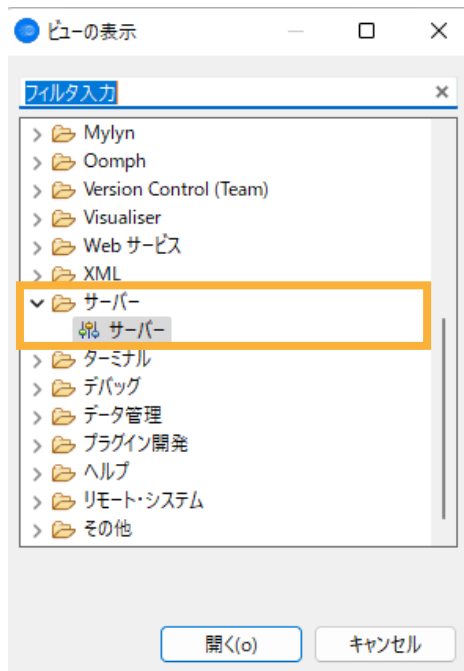
補足)

自動的にビルドする設定は、[プロジェクト(P)] > [自動的にビルド(M)] で確認できます。チェックされている場合は自動的にビルドが行われます。

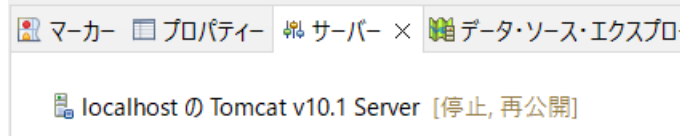


3.3 Apache Tomcat 上で動作する RESTful Web サービスの作成

本節を実行するにあたり、Visual COBOL for Eclipse 上で、予めサーバー設定を実施してください。この設定は、以降の手順で使用する J2EE パースペクティブのデフォルトで表示されるサーバービューで行うことができます。また、異なるパースペクティブを使用している際でも、Eclipse のメニューより、[ウィンドウ(W)] > [ビューの表示(V)] > [その他(O)] を選択した上で表示されるダイアログ上で、以下のビューを選択することで行えます。

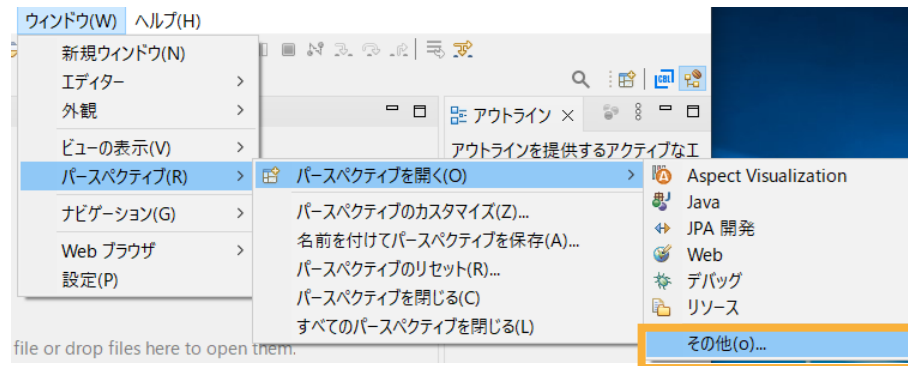


本手順では 以下のように Tomcat 10.1 のサーバー構成を行っています。

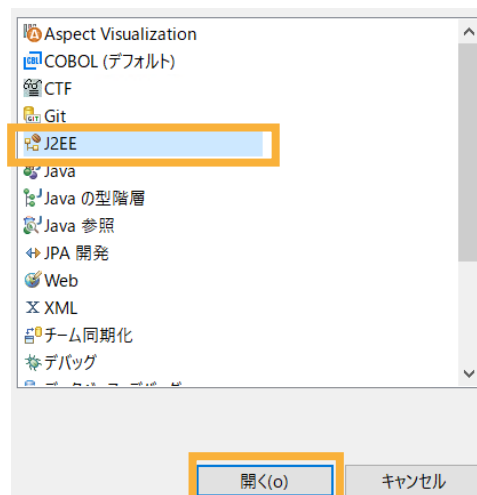


3.3.1 Java Web アプリケーションプロジェクトの作成

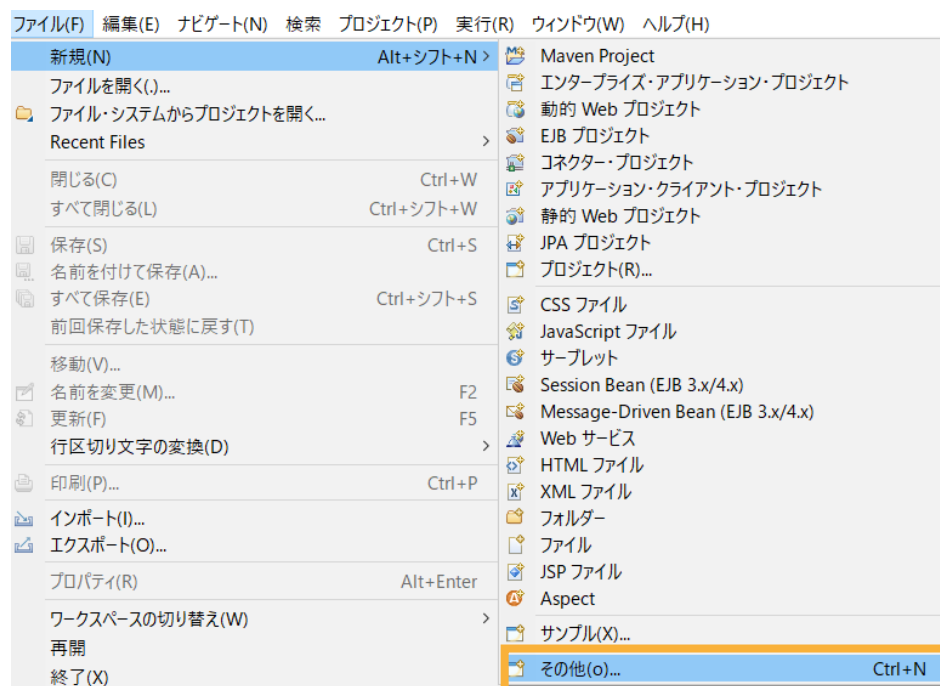
- 1) [ウィンドウ(W)] > [パースペクティブ(R)] > [パースペクティブを開く(O)] > [その他(o)] を選択します。



- 2) 「J2EE」を選択して、[開く(o)] をクリックします。



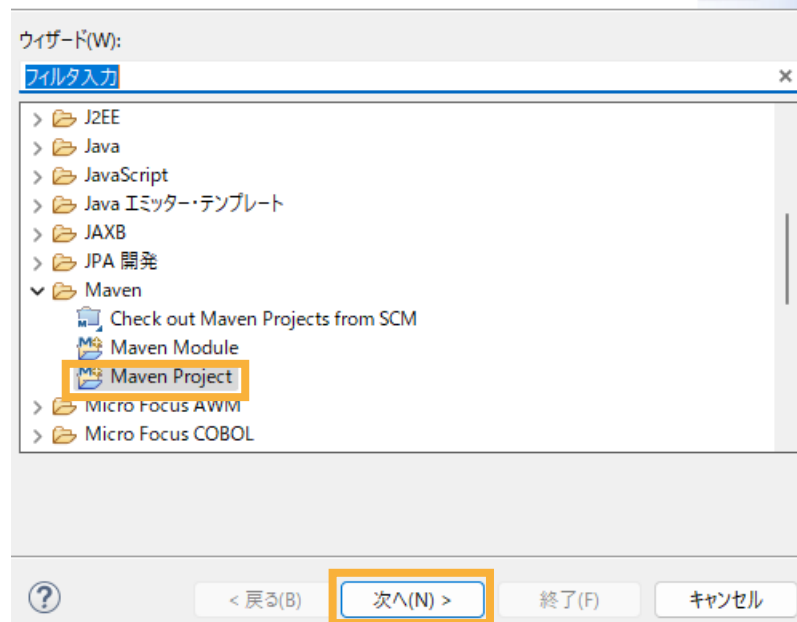
- 3) [ファイル(F)] > [新規(N)] > [その他(o)] を選択します。



- 4) [Maven] > [Maven Project] を選択し、[次へ(N)]をクリックします。

ウィザードを選択

Create a Maven project



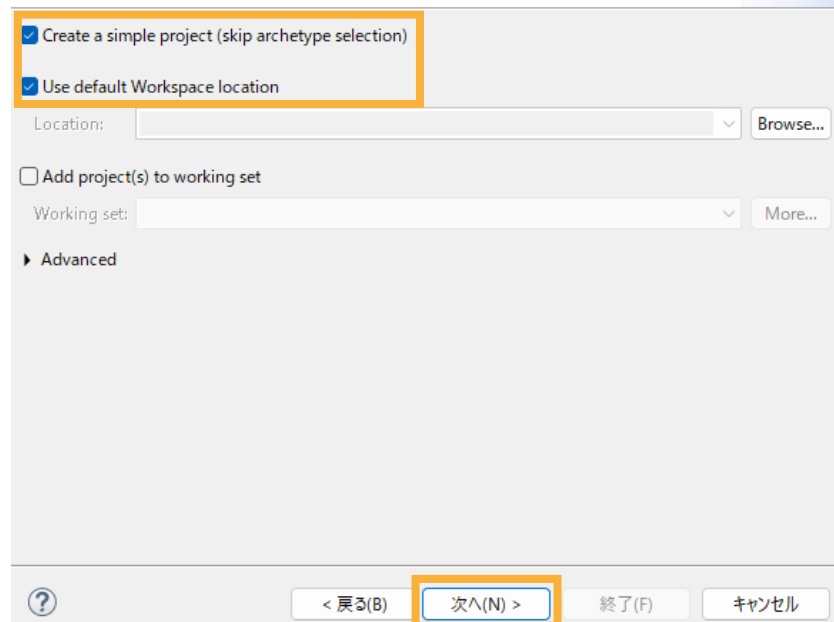
- 5) 以下の設定を行い、[次へ(N)] をクリックします。

Create a simple project (skip archetype selection) にチェック

Use default Workspace location にチェック

New Maven project

Select project name and location



以下の入力を行い、[終了(F)] をクリックします。

Group Id: "com.microfocus.sample"

Artifact Id: "JVMRESTfulWS"

Version: "0.0.1-SNAPSHOT"

Packaging: "war"

New Maven project

Configure project



Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

Parent Project

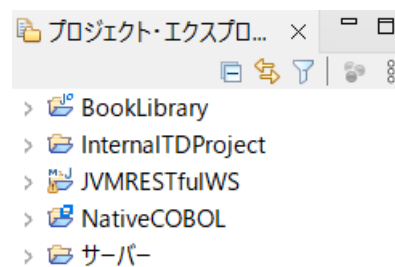
Group Id:

Artifact Id:

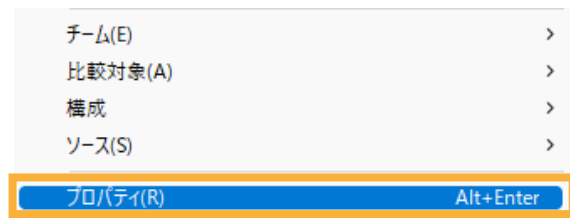
Version:

▶ Advanced

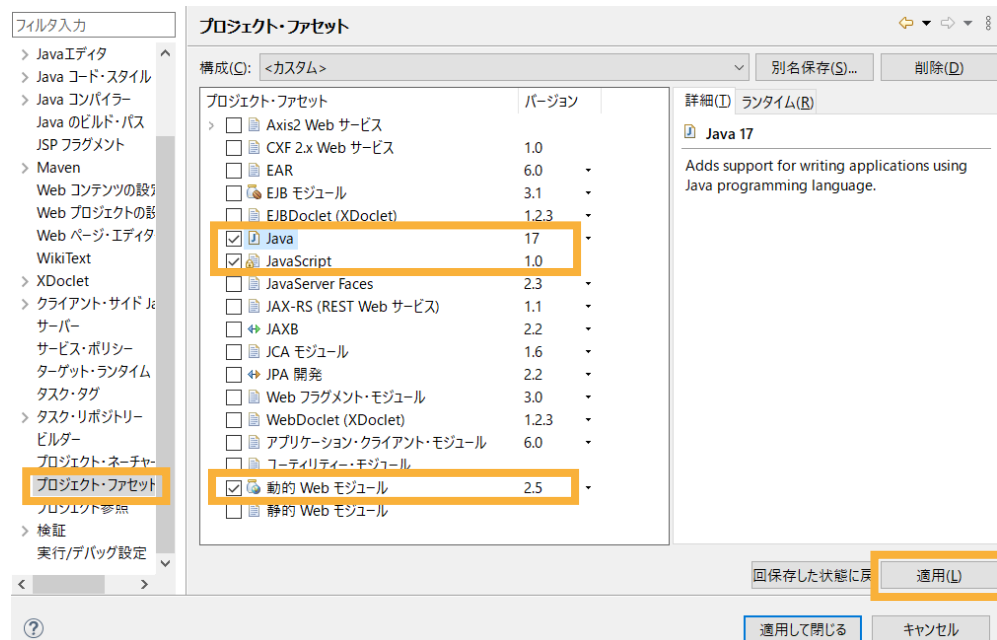
JVMRESTfulWS プロジェクトが作成されます。



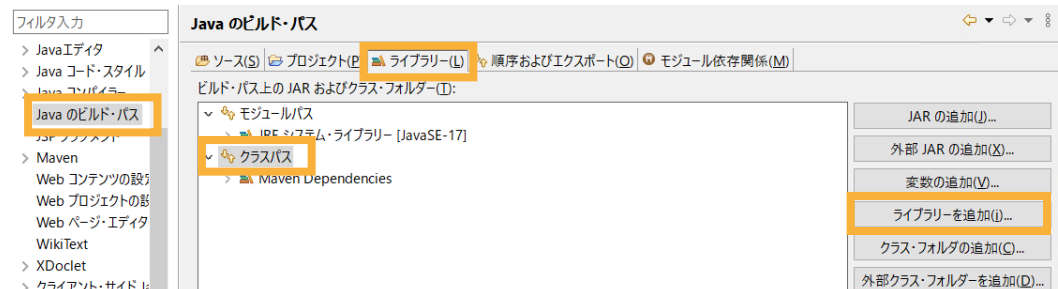
- 6) JVMRESTfulWS プロジェクトを選択し、マウスの右クリックにてコンテキストメニューを表示したうえで、[プロパティ(R)] を選択します。



- 7) [プロジェクト・ファセット]を選択し、「Java」に“17”を選択し、「JavaScript」と「動的 Web モジュール」にチェックを入れて[適用(L)]をクリックします。※[プロジェクト・ファセット]を選択した際、「Convert to faceted from…」リンクが表示される場合は、リンクをクリックしてください。



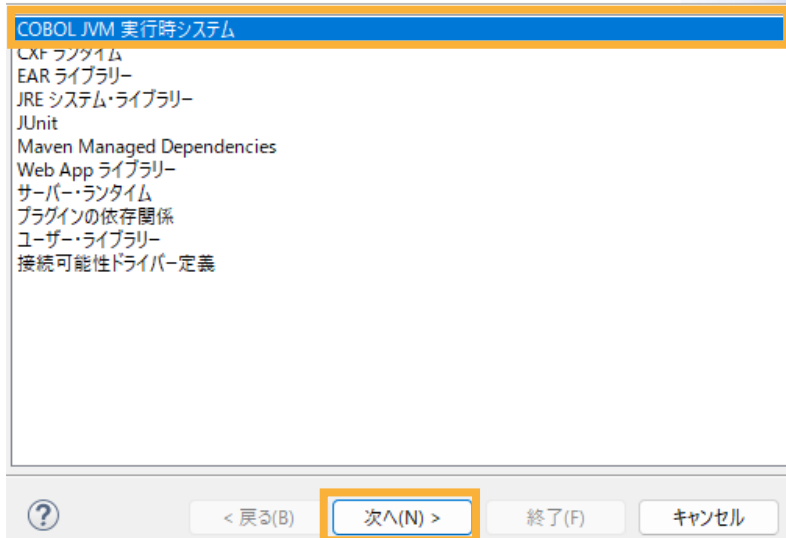
- 8) [Java のビルド・パス] から [ライブラリー] タブを選択します。「クラスパス」を選択したうえで、[ライブラリーの追加(i)]をクリックします。



- 9) [COBOL JVM 実行時システム] を選択したうえで、[次へ(N)] をクリックします。

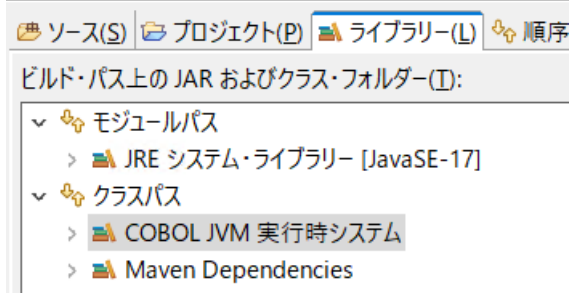
ライブラリーの追加

追加するライブラリー・タイプを選択します。

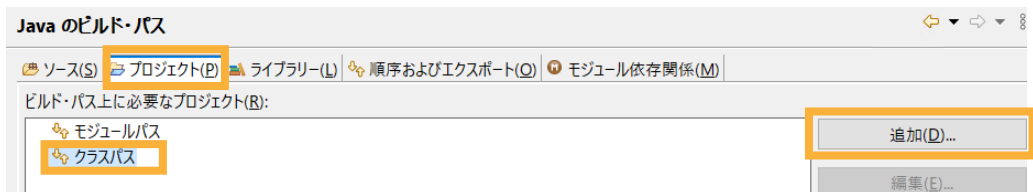


次の画面では、そのまま [終了(F)] をクリックすると、「COBOLJVM 実行時システム」が追加されます。

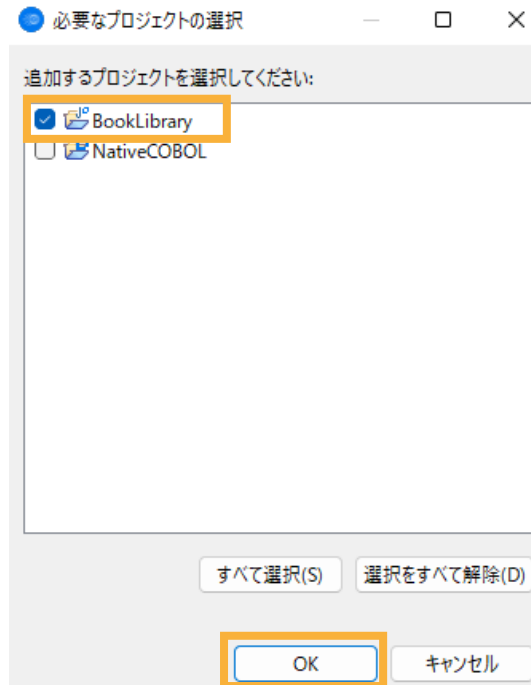
Java のビルド・パス



10) [プロジェクト]タブを選択し、「クラスパス」を選択したうえで、[追加(D)] をクリックします。



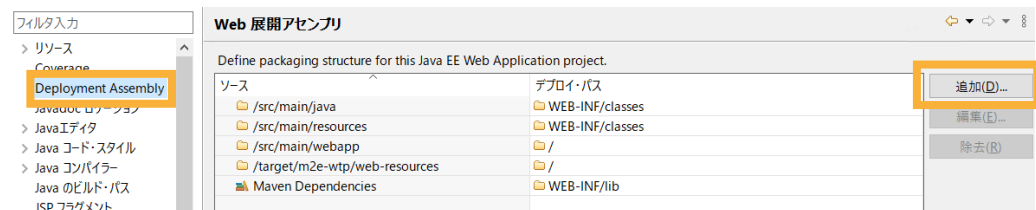
11) 「BookLibrary」プロジェクトにチェックを入れたうえで、[OK] をクリックします。



BookLibrary プロジェクトが追加されたことを確認したうえで、[確認(L)] をクリックします。



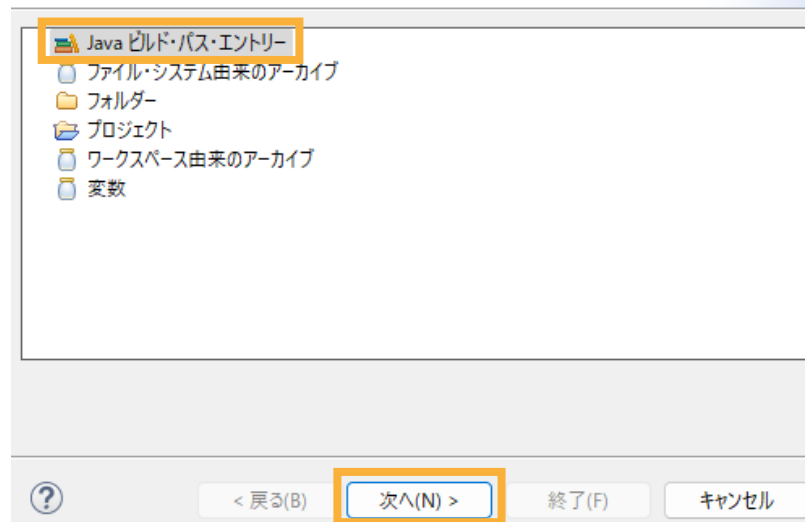
12) [Deployment Assembly] を選択し、[追加(D)] をクリックします。



13) [Java ビルド・パス・エントリ] を選択し、[次へ(N)] をクリックします

ディレクトティブの種類を選択

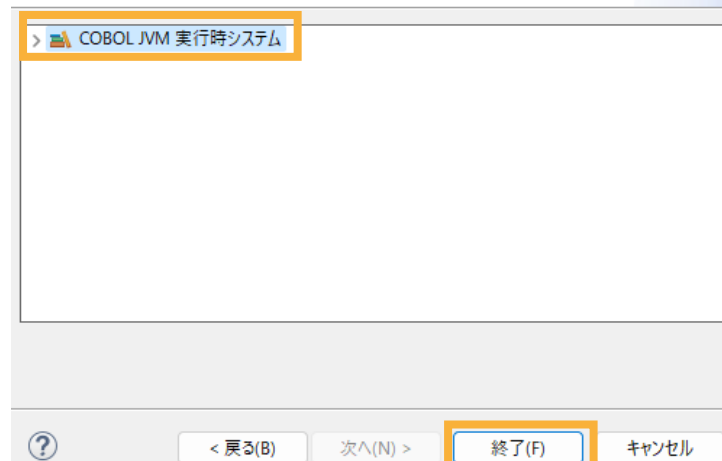
アセンブリディレクトティブを新規追加します。



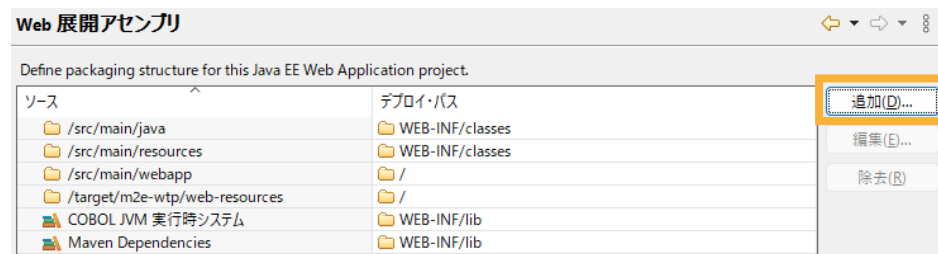
[COBOL JVM 実行時システム] を選択し、[終了(F)] をクリックします。

Java ビルド・パス・エントリー

Select build path entries to include in the deployment assembly.



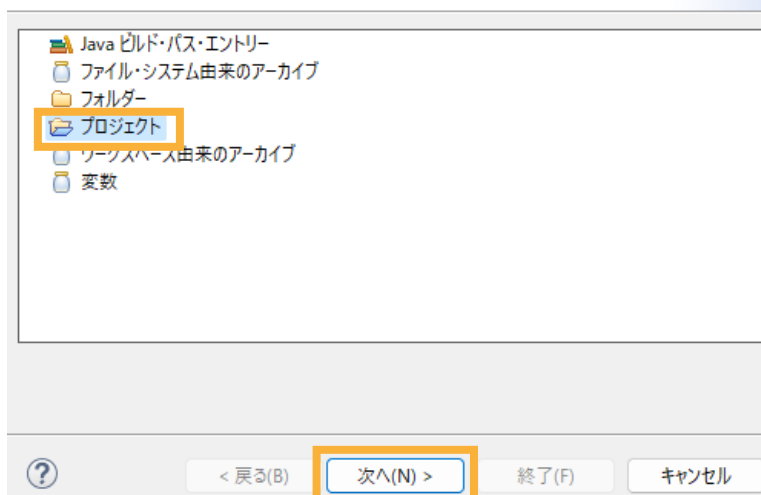
14) さきほど同様、[追加(D)] をクリックします。



15) 「プロジェクト」を選択し、[次へ(N)] をクリックします。

ディレクトティブの種類を選択

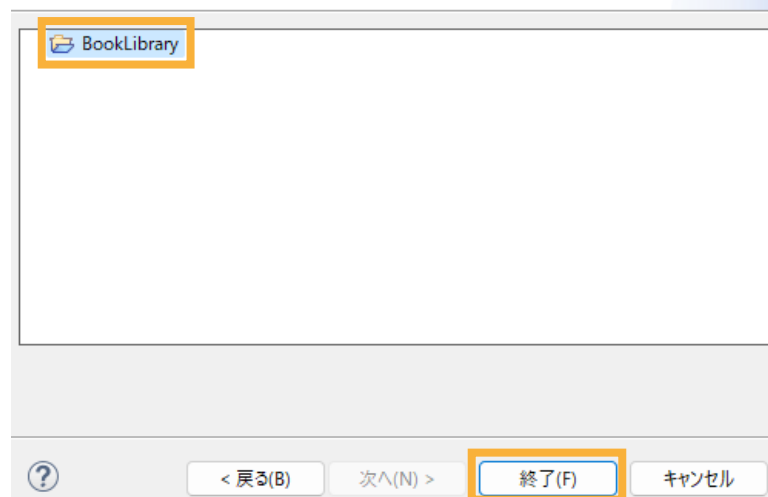
アセンブリディレクトティブを新規追加します。



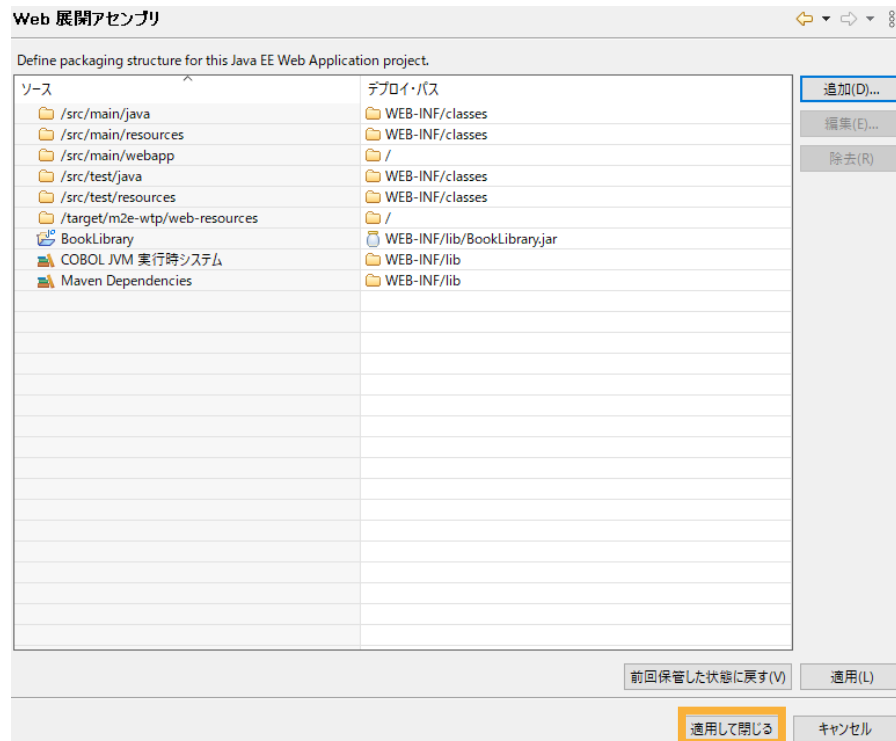
「BookLibrary」を選択し、[終了(F)] をクリックします。

プロジェクト

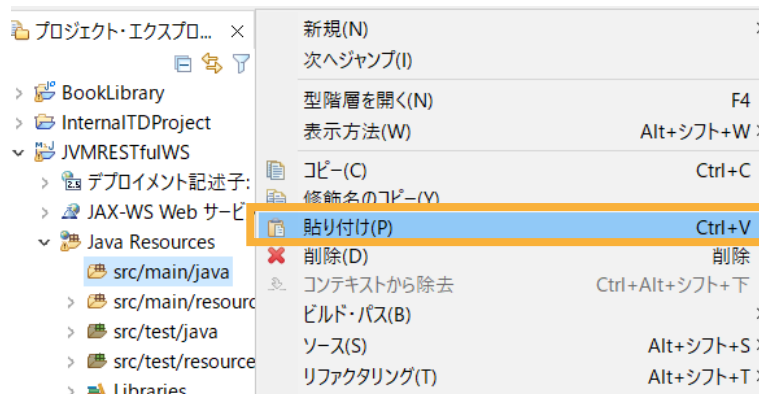
Select projects to include in the deployment assembly.



以下のように追加した項目が表示されていることを確認して、[適用して閉じる] をクリックします。

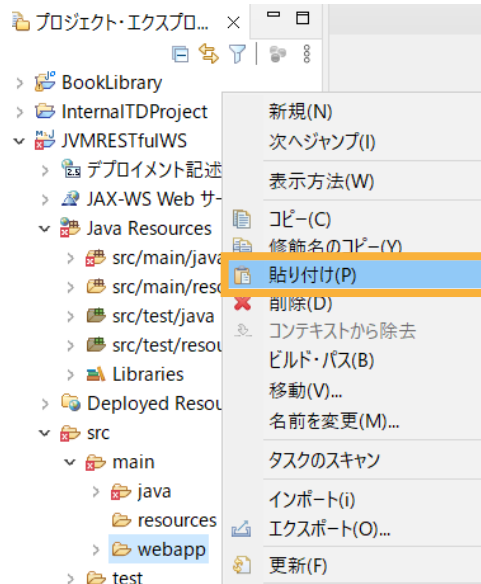


- 16) C:\jvmRESTfulWSTutorial\Java\src 配下の com フォルダをクリップボードにコピーしたうえで、JVMREStfulWS プロジェクト配下の [Java Resources] > [src/main/java] を選択した状態でマウスの右クリックにてコンテキストメニューを表示して、[貼り付け(P)] を選択します。



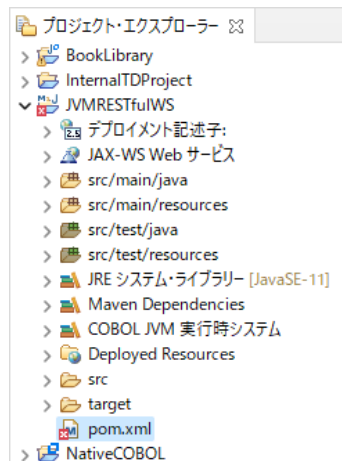
この時点ではエラーになりますが、ここでは無視します。

- 17) C:\jvmRESTfulWSTutorial\Java\webapps 配下の WEB-INF フォルダをクリップボードにコピーしたうえで、JVMREStfulWS プロジェクト配下の src\main\webapps フォルダを選択し、マウスの右クリックにてコンテキストメニューを表示したうえで、[貼り付け(P)] を選択します。

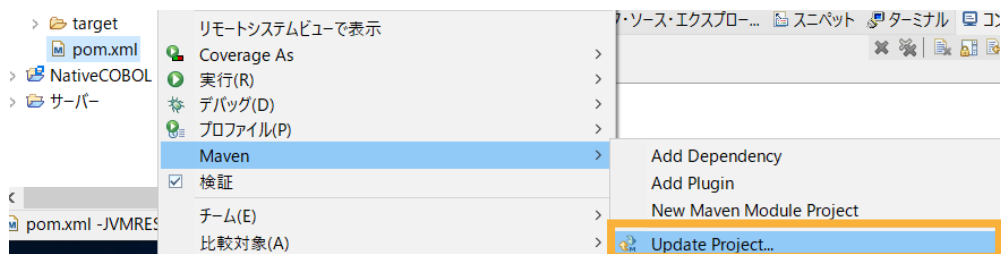


18) JVM RESTfulWS プロジェクト配下の pom.xml の中身を以下のファイルで上書き保存します。

C:\¥jvmRESTfulWSTutorial¥Java¥pom.xml



19) pom.xml を選択し、マウスの右クリックにてコンテキストメニューを表示したうえで、[Maven] > [Update Project] を選択します。



JVMRESTfulWS にチェックがついていることを確認したうえで [OK] をクリックしてください。

Update Maven Project

Select Maven projects and update options

Available Maven Codebases

☒ JVMRESTfulWS	Select All
	Add out-of-date
	Deselect All
	Expand All
	Collapse All

☐ Offline
☒ Update dependencies
☐ Force Update of Snapshots/Releases
☒ Update project configuration from pom.xml
☒ Refresh workspace resources from local filesystem
☒ Clean projects

20) サーバービューを開き、Tomcat サーバーをダブルクリックします。

localhost の Tomcat v10.1 Server [停止, 再公開]

21) [起動構成を開く] リンクをクリックします。

概要

一般情報

ホスト名とその他の共通設定を指定します。

サーバー名:	localhost の Tomcat v10.1 Server
ホスト名:	localhost
ランタイム:	Apache Tomcat v10.1
構成パス:	/サーバー/localhost の Tomcat v10.1 Server-config
起動構成を開く	

22) [環境]タブを開き、[追加(A)] をクリックします。

起動構成プロパティの編集



名前(N): localhost の Tomcat v10.1 Server

サーバー (S) 引数 (A) クラスパス (C) ソース (S) 環境 (E) 共通 (G)

設定する環境変数(S)

変数	値

追加(A)...
選択(L)...
編集(D)...

23) 以下の入力を行い、[OK] をクリックします。

名前: "BOOKINFO"

値: "C:/jvmRESTfulWSTutorial/DAT/BOOKINFO.DAT"

名前(N): BOOKINFO

値(V): C:/jvmRESTfulWSTutorial/DAT/BOOKINFO.DAT 変数(B)...

OK キャンセル

BOOKINFO 変数が追加されたことを確認して、[OK] をクリックします。

名前(N): Tomcat 10

サーバー (S) 引数 (A) クラスパス (C) ソース (S) 環境 (E) 共通 (G)

設定する環境変数(S)

変数	値
BOOKINFO	C:/jvmRESTfulWSTutorial/DAT/BOOKINFO.DAT

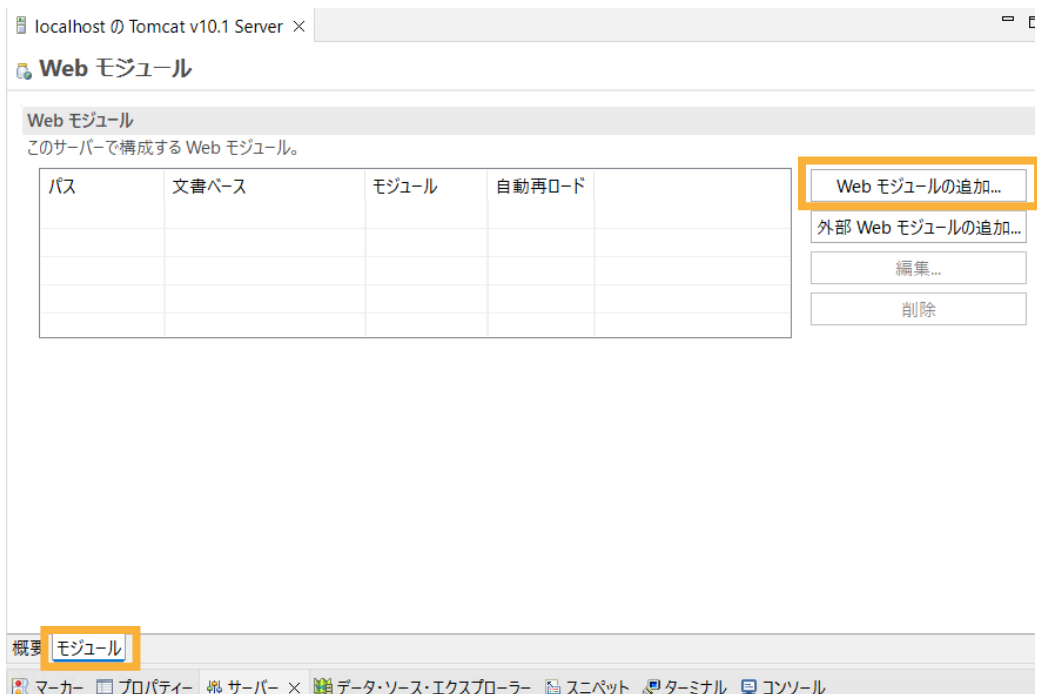
追加(A)...
選択(L)...
編集(D)...
削除(O)
コピー(C)
貼り付け(P)

☒ ネイティブ環境への環境の追加(A)
☐ ネイティブ環境を指定された環境と置換(P)

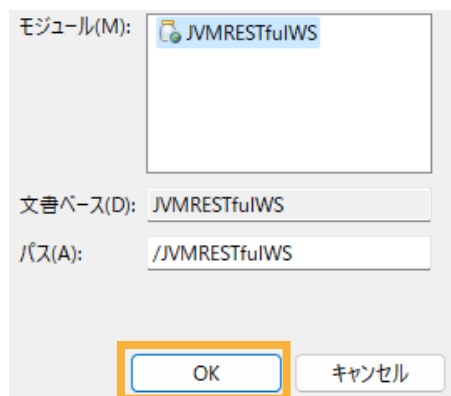
前回保管した状態に戻す(V) 適用(Y)

OK キャンセル

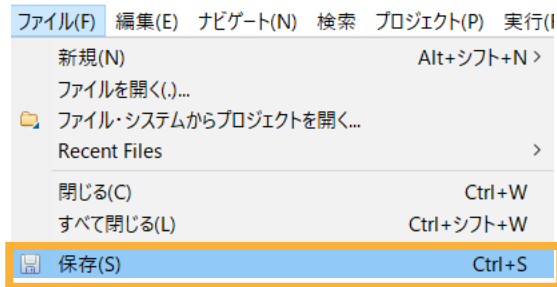
24) [モジュール]タブを選択し、[Web モジュールの追加] をクリックします。



JVMRESTfulWS を選択し、[OK] をクリックします。

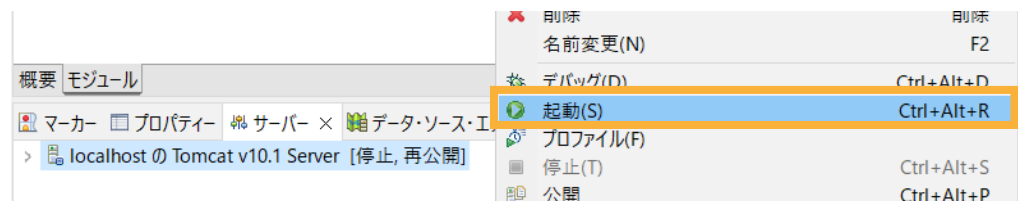


Eclipse IDE のメニューより [ファイル(F)] > [保存(S)] を選択し、変更を保存してください。

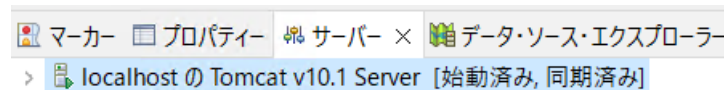


3.3.2 RESTful Web サービスの確認

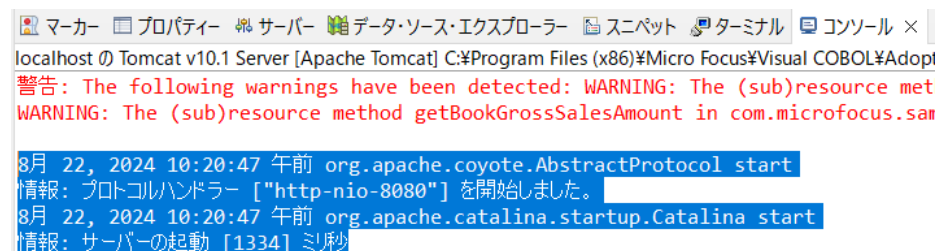
- 1) サーバービューより、Tomcat サーバーを選択し、マウスの右クリックにてコンテキストメニューを表示したうえで、[起動 (S)] を選択します。



起動すると、ステータスが“始動済み”に変更されます。



また、コンソールビューでもサーバーの起動を確認できます。



- 2) C:\jvmRESTfulWSTutorial¥WebClient 配下の Search.html を Web ブラウザーで開きます。

Micro FocusDemo
HOME
書籍検索
在庫登録
在庫情報削除
売上集計
CONTACT



ABC 書店在庫管理システム

書籍情報の検索

ストック番号を入力の上、[検索]ボタンを押下してください。

システムメニュー

- Home Page
- 書籍検索**
- 在庫情報登録
- 在庫情報削除
- 売上集計
- Contact

- 3) ストック番号に“1111”を入力し、[検索]をクリックすると、先に確認したように結果が画面上に戻されます。

書籍情報の検索

ストック番号を入力の上、[検索]ボタンを押下してください。

システムメニュー

- Home Page
- 書籍検索**
- 在庫情報登録
- 在庫情報削除
- 売上集計
- Contact

ストック番号	1111
タイトル	LOAD OF THE RING
ジャンル	FANTASY
著者	TOLKIEN
小売価格	1500
仕入れ数	2000
売り上げ数	1000

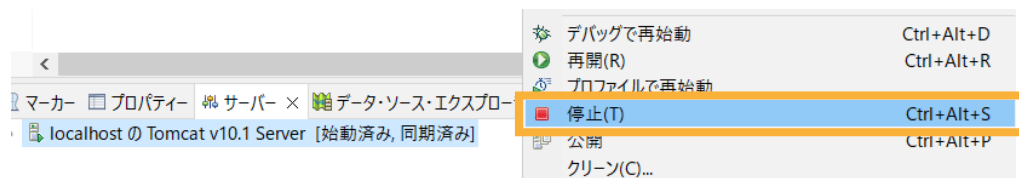
```

IF表示
api EndPoint
http://localhost:8080/JVMRESTfulWS/bookmgnt/books/1111
Request
(NO DATA)
Response
{
  "bookInfo": {
    "stockNo": "1111",
    "title": "LOAD OF THE RING",
    "author": "TOLKIEN",
    "genre": "FANTASY"
  }
}

```

このように、Web 画面から Java で作成した RESTful Web サービスを介して COBOL ロジックを呼び出すことが確認できます。このインターフェースは、一般的な RESTful Web サービス仕様に基いているため、COBOL 資産を活用した他システムとのデータ連携が容易に行えるようになります。

Eclipse IDE のサーバービューに戻り、Tomcat サーバー選択し、マウスの右クリックにてコンテキストメニューを表示したうえで、[停止(T)] をクリックしてサーバーを停止します。



補足)

今回は、Eclipse IDE 上から Tomcat アプリケーションサーバーを起動し、RESTful Web サービスを稼働させていましたが、Eclipse IDE を使用せず、Tomcat をはじめとしたアプリケーションサーバー上でも今回のサービスを稼働させることができます。

免責事項

ここで紹介したソースコードは、機能説明のためのサンプルであり、製品の一部ではございません。ソースコードが実際に動作するか、御社業務に適合するかなどに関しまして、一切の保証はございません。ソースコード、説明、その他すべてについて、無謬性は保障されません。ここで紹介するソースコードの一部、もしくは全部について、弊社に断りなく、御社の内部に組み込み、そのままご利用頂いても構いません。本ソースコードの一部もしくは全部を二次的著作物に対して引用する場合、著作権法に基づき、適切な扱いを行ってください。