

DDBuilder ユーザガイド

第2版 2017年9月

目次

1. DDBuilder をはじめる	4
1.1 ダウンロードとインストール	4
1.2 動作確認済みの環境	5
1.3 求められる知識	5
1.4 サポート・保証等	5
1.5 アンインストール	5
2. 起動	5
3. 初期生成	6
3.1 生成	6
3.2 初期生成結果の内容	8
3.2.1 domain パッケージ	8
3.2.2 persistence パッケージ	8
3.2.3 service パッケージ	8
3.2.4 util パッケージ	9
3.2.5 view パッケージ	9
3.3 Eclipse に Web アプリケーションをインポート	9
3.4 Eclipse の Tomcat サーバを作成 (未作成の場合のみ)	11
3.5 テスト実行	13
4. イテレーションの流れ	16
5. イテレーションの例 1	16
5.1 モデル (例 : Product を追加)	16
5.2 実装	16
5.3 新しいモデルを Web アプリケーションに反映	18
5.4 リフレッシュ	19
5.5 内容確認	19
5.5.1 domain パッケージ	19
5.5.2 persistence パッケージ	20
5.5.3 service パッケージ	20
5.5.4 util パッケージ	20
5.5.5 view パッケージ	20

5.6	テスト実行.....	20
6.	イテレーションの例 2	23
6.1	モデル (例 : StockService, Warehouse, Stock を追加)	23
6.2	実装.....	23
6.3	新しいモデルを Web アプリケーションに反映.....	24
6.4	リフレッシュ	24
6.5	テスト実行.....	24
6.6	実装コード (例 : StockService, Warehouse, Stock)	29
7.	Web アプリケーションのカスタマイズ.....	37
7.1	システム名称のカスタマイズ.....	37
7.2	一覧画面の表示内容のカスタマイズ.....	38
7.3	JPA 永続化方式のカスタマイズ.....	39
7.4	テストデータ一括登録サービスメソッドのカスタマイズ.....	40
8.	制限事項.....	43
8.1	プリミティブ型ラッパークラスを使用	43
8.2	使用できる Java 標準クラス	43
8.3	コレクション型とマップ型	43
8.4	同じ組み合わせのクラス間で定義できる ManyToMany 関連の数.....	44
8.5	List の要素型.....	44
8.6	クラス名 (テーブル名に SQL 予約語は使用不可)	44
8.7	双方向関連.....	44
8.7.1	@OneToMany の場合.....	44
8.7.2	@ManyToMany の場合.....	44
9.	DDBuilder が上書きするファイル.....	45
9.1	domain.....	45
9.2	service.....	45
9.3	view	45
9.4	lib.....	45
10.	データベース	45
10.1	ログ	45
10.2	Derby データファイルの場所.....	46
10.3	Derby 以外のデータベースを使用する場合.....	46
10.4	データベース (テーブル) の参照.....	47
11.	Web アプリケーションの説明.....	48
11.1	構造図.....	48
11.2	永続化に関して.....	48
11.2.1	エンティティマネージャ	48
11.2.2	トランザクション.....	48
11.2.3	設定定義ファイル.....	48
11.2.4	データベースファイル	49
11.3	テスト用初期データの作成.....	49
11.4	サインイン機能.....	49

11.5	log4j2 ログ設定.....	49
12.	トラブル時の対応.....	49
12.1	Eclipse のデータ・ソース・エクスプローラでデータベースに接続できない.....	49
12.2	JPA (Hibernate)に関連する実行時エラーが解消しない	49
13.	補足：データベーステーブルの確認方法	49

1. DDBuilder をはじめる

1.1 ダウンロードとインストール

下図の場所からダウンロードします。

<http://www.nextdesign.co.jp/ddd/index.html>



The screenshot shows the website for DDBuilder. At the top, there is a navigation bar with links for 'セミナー', 'システム開発', '製品情報', '人材募集', and '会社情報'. Below this, there are three main sections, each with a 'Download' button highlighted in a red box:

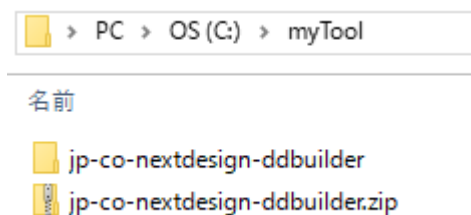
- DDBuilder (40 MB)**: A button labeled 'Download'. Below it, text reads: 'JavaコードファーストでDDDドメインモデルをイテレーティブに洗練するためのプログラムです。ダウンロード後のZIPファイルを解凍してください。操作手順は説明書をご覧ください。' and a link 'トップページへ戻る'.
- Jクラスレポート (16 MB)**: A button labeled 'Download'. Below it, text reads: 'Javaソースから各種情報、メトリクスを抽出するためのプログラムです。ダウンロードファイルを適当な場所に保存した後、解凍してください。起動方法は、解凍先の「はじめにお読みください(使い方).txt」をご覧ください。' and a link 'トップページへ戻る'.
- Let's アンケート・小テスト (30 MB)**: A button labeled 'Download'. Below it, text reads: '簡易的にアンケートや小テストを実施するためのプログラムです。ダウンロード後のZIPファイルを解凍すると 1つの warファイルと 2つの pdfファイル があります。Tomcat7相当のコンテナにデプロイして稼働できます。手順などは操作説明書をご覧ください。' and a link 'トップページへ戻る'.

At the bottom of the page, there is a copyright notice: 'Copyright 2001-2017, All Rights Reserved'.

インストールは、ダウンロードしたファイルを解凍するだけです。

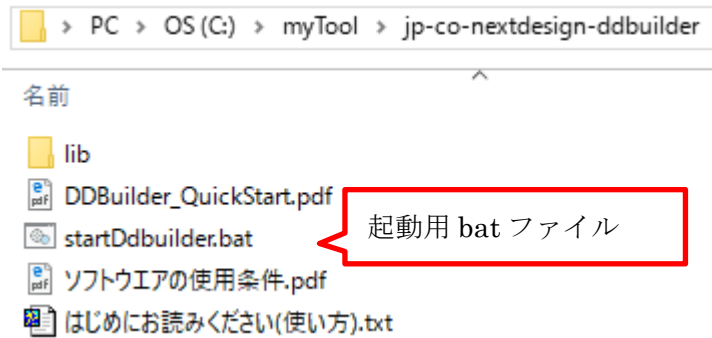
ダウンロードしたファイル (jp-co-nextdesign-ddbuilder.zip) を解凍すると

下図のように jp-co-nextdesign-ddbuilder が作成されます。



※解凍先の場所は任意です。

jp-co-nextdesign-ddbuilder の下は下図のようになっています。



1.2 動作確認済みの環境

- Windows7 Professional 32/64bit
- Windows10 Professional 64bit
- Java8
- Eclipse IDE for Java EE Developers 4.4, 4.5, 4.6
- Apache Tomcat 7.0, 8.0, 8.5, 9.0

1.3 求められる知識

- オブジェクト指向モデリング技術（考え方）
- DDD ドメイン駆動設計（考え方）
- 繰り返し型開発（考え方）
- Java 言語
- JPA（Java Persistence API）入門レベル
- Eclipse の基本操作

1.4 サポート・保証等

DDBuilder は無料・無保証のソフトウェアです。

DDBuilder の使用に関しお客様に生じた損害に対する賠償及びその他の責任は一切負わないものとします。

DDBuilder では、Apache Wicket, Java EE JPA/Hibernate, Apache Derby, Eclipse を使用しています。

1.5 アンインストール

jp-co-nextdesign-ddbuilder 以下をすべて削除してください。

2. 起動

jp-co-nextdesign-ddbuilder¥startDdbuilder.bat を起動します（ダブルクリック）

下図の画面が表示されたら正常です。

ステータス: 操作できます。

(1) 出力先:
[説明] Webアプリケーションの出力先を指定します。 [例] C:\MyWebApplications

(2) 現在固定

(3) GroupId:
[説明] パッケージ名を指定します。 [例] jp.co.nextdesign

(4) ArtifactId:
[説明] システム名を指定します。 [例] app1

操作例

- ① 入力項目(1)(3)(4)に各[例]を参考にを入力します。
- ② [作成/更新]ボタンを押下します。
- ③ (1)出力先にWebアプリケーションフォルダが作成されます。 [例]C:\MyWebApplications¥app1
- ④ 作成されたWebアプリケーションフォルダの内容はEclipseプロジェクトとして構成されています。
- ⑤ Eclipseの[既存のプロジェクトをワークスペースへ]操作でインポートします。2回目以降の繰り返しでは[リフレッシュ]操作です。
- ⑥ Eclipseでドメインモデルの振る舞いを確認し、必要に応じてドメインモデルを追加・更新し、②に戻り繰り返します。

※ インポートしたプロジェクトはEclipse上で[サーバで実行]や[サーバでデバッグ]できます。

※ ①時点ですでにフォルダが存在する場合は、⑥で追加・更新されたドメインクラスをもとに内容は上書き更新されます。

※DDBuilder は無料・無保証のソフトウェアです。"

※本ソフトウェアの使用に関し、お客様に生じた損害に対する賠償およびその他の責任は一切負わないものとします。

※本ソフトウェアでは、Apache Wicket, Java EE JPA/Hibernate, Apache Derby, Eclipse を使用しています。

※開発者: <http://www.nextdesign.co.jp>

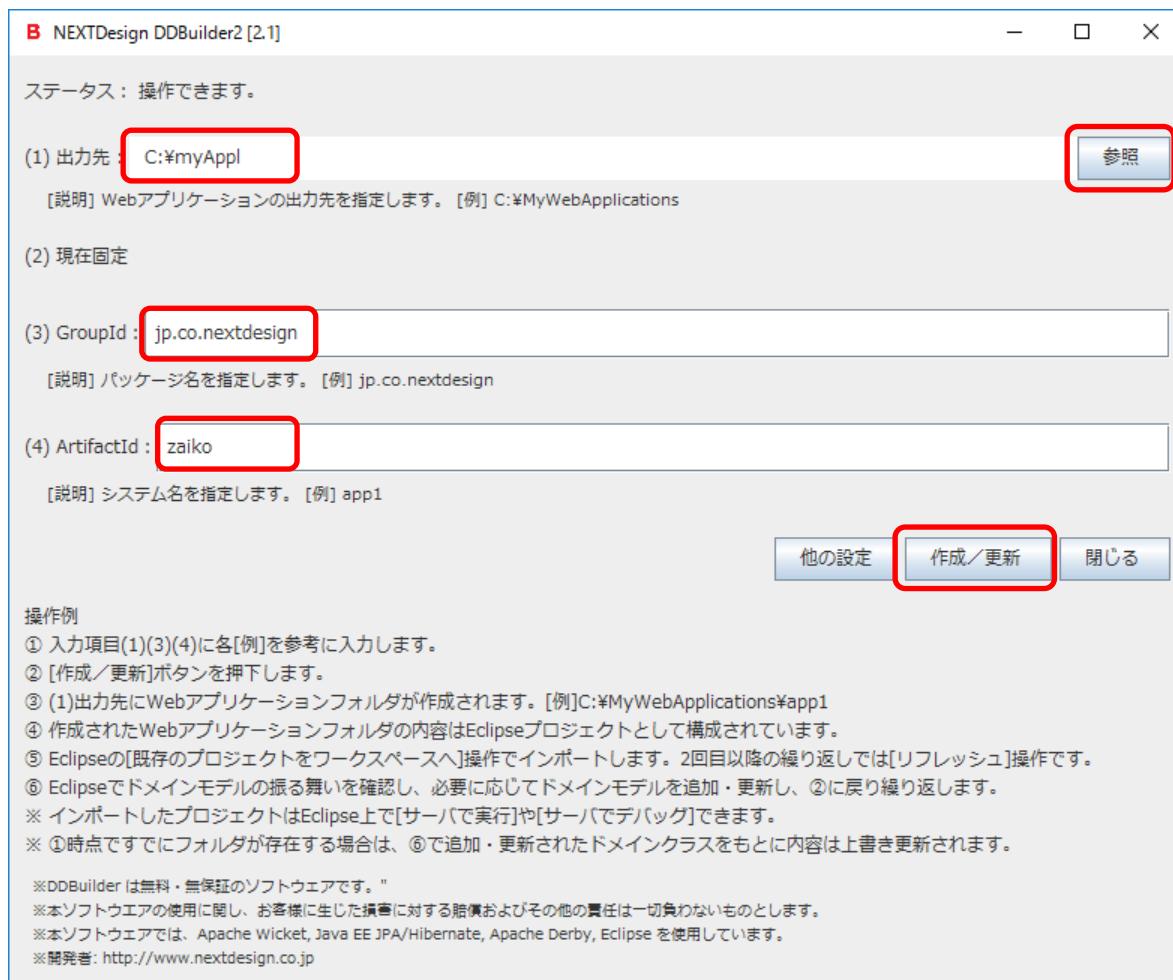
※背後の黒いウィンドウ（コマンドプロンプト画面）は閉じないでください。

3. 初期生成

3.1 生成

最初に、空の Web アプリケーションを作成します。

「空」とは、利用者が作成したクラスをまったく含まないという意味で、実際には、Eclipse にインポート可能な Web アプリケーションとして複数のファイルを含んでいます。



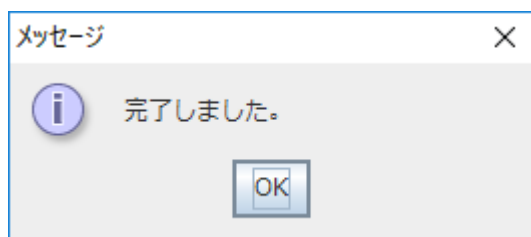
(1) 必須情報を入力します。

出力先：任意のフォルダ名。このフォルダの下に Web アプリケーションが生成されます。

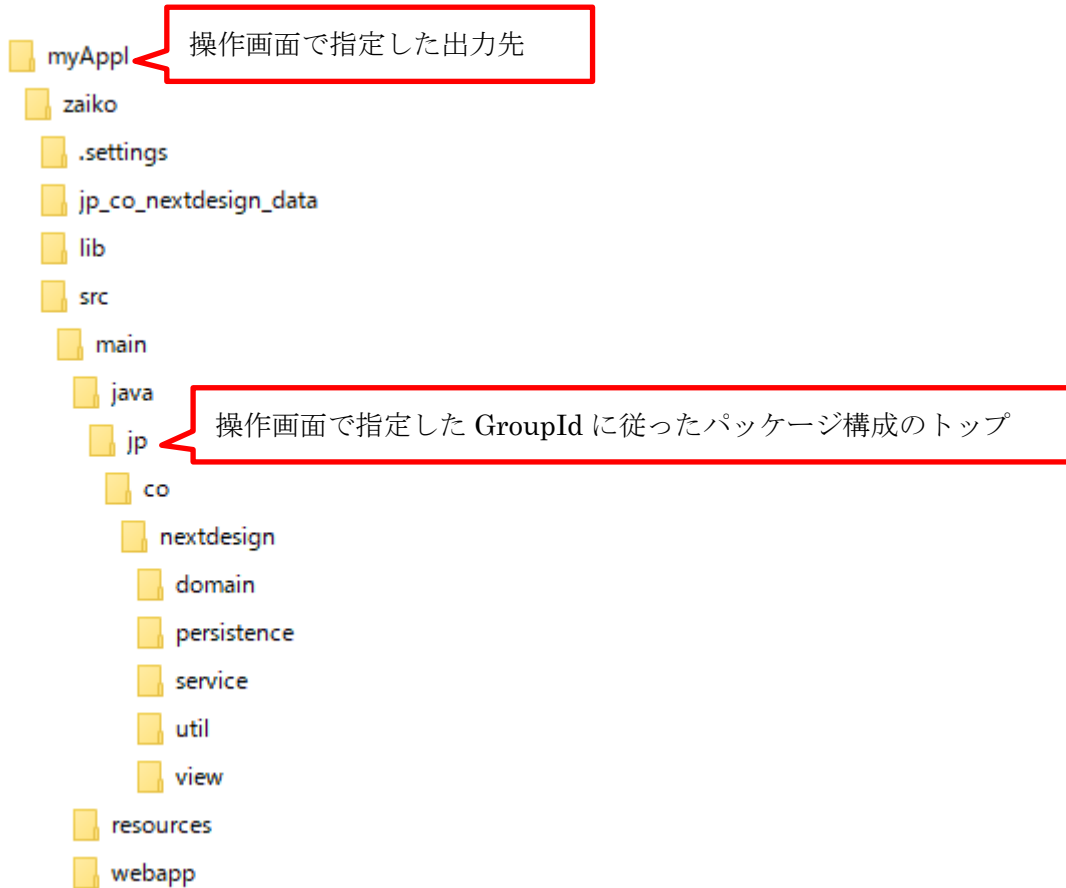
GroupId：最上位のパッケージ名

ArtifactId：アプリケーション名

(2) 作成/生成ボタンを押下し、下図のダイアログが表示されたら生成完了です。OK を押下します。



3.2 初期生成結果の内容



3.2.1 domain パッケージ

(1) ddb サブパッケージ

DDBuilder が用意した基底クラスを含みます。

例 : DdBaseEntity.java

(2) g サブパッケージ

空。

3.2.2 persistence パッケージ

DDBuilder が用意した JPA に関するクラスを含みます。

例 : DdEntityManagerFactory.java

persistence パッケージ内で利用者が作業することはありません。

3.2.3 service パッケージ

(1) ddb サブパッケージ

DDBuilder が用意した基底クラスを含みます。

例 : DdBaseService.java

(2) g サブパッケージ

TestDataService.java を含みます。テストデータを登録するために使用します (後述)

3.2.4 util パッケージ

DDBuilder が用意した utility クラスを含みます。
利用者が作成した utility クラスも格納できます。

3.2.5 view パッケージ

(1) ddb サブパッケージ

DDBuilder が用意した基底クラスと最小限必要なビューを含みます。

例 : DdBaseEditPage.java

例 : DdBaseEditPage.html

※Wicket のビューは同名の java ファイルと html ファイルのペアで構成されます。

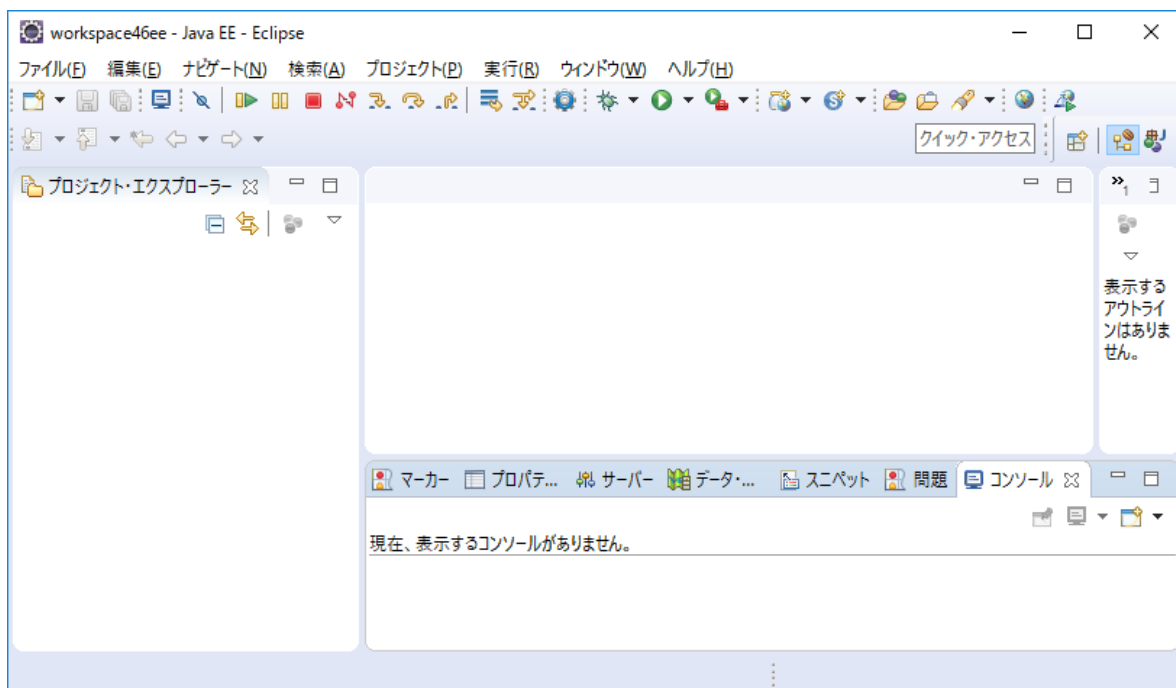
(2) g サブパッケージ

DDBuilder が用意した最小限必要なビューを含みます。

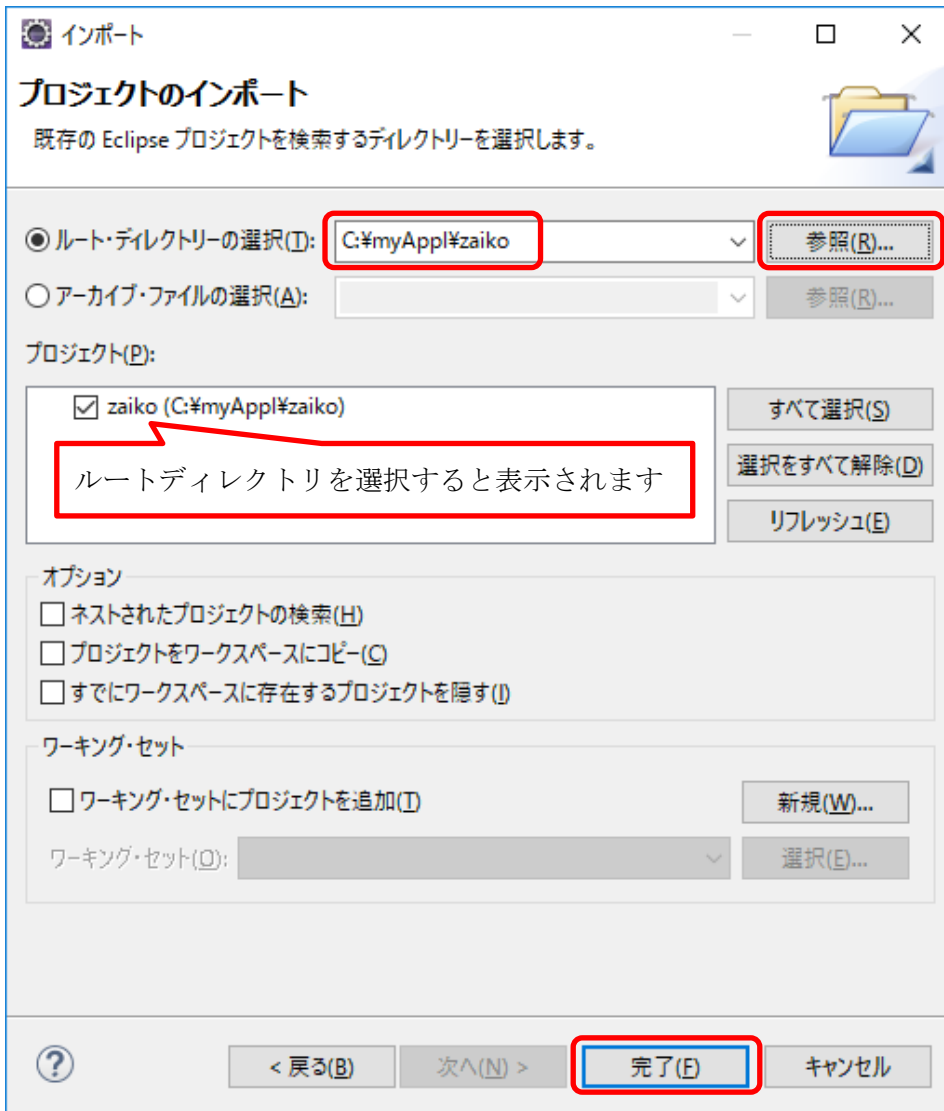
例 : DomainClassListPage.java

例 : DomainClassListPage.html

3.3 Eclipse に Web アプリケーションをインポート

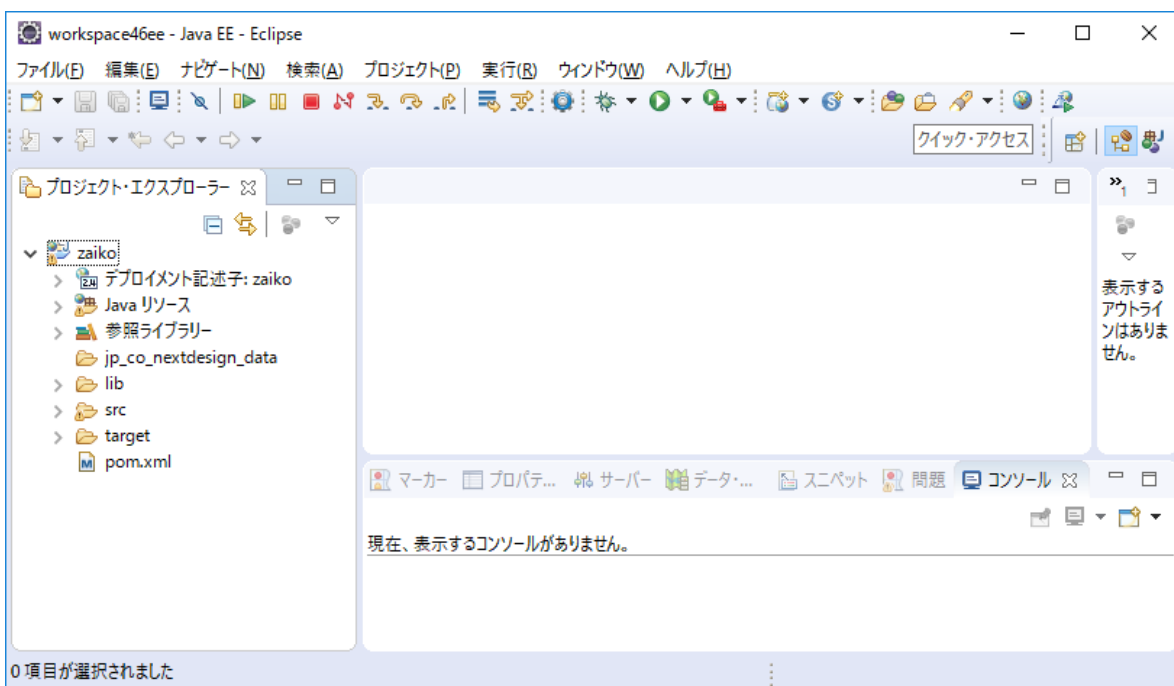


ファイルメニュー → インポート → 一般 → 既存のプロジェクトをワークスペースへ → 次へ



下図のようにプロジェクトエクスプローラに表示されれば正常です。

下図は zaiko を展開した状態です。

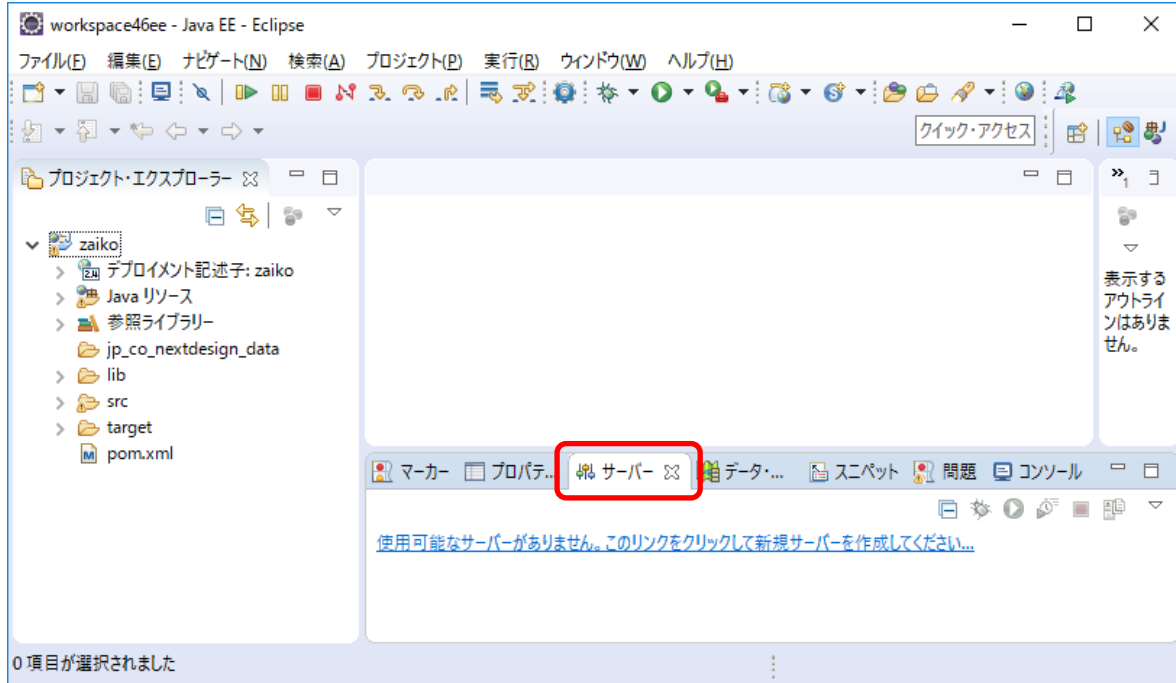


3.4 Eclipse の Tomcat サーバを作成（未作成の場合のみ）

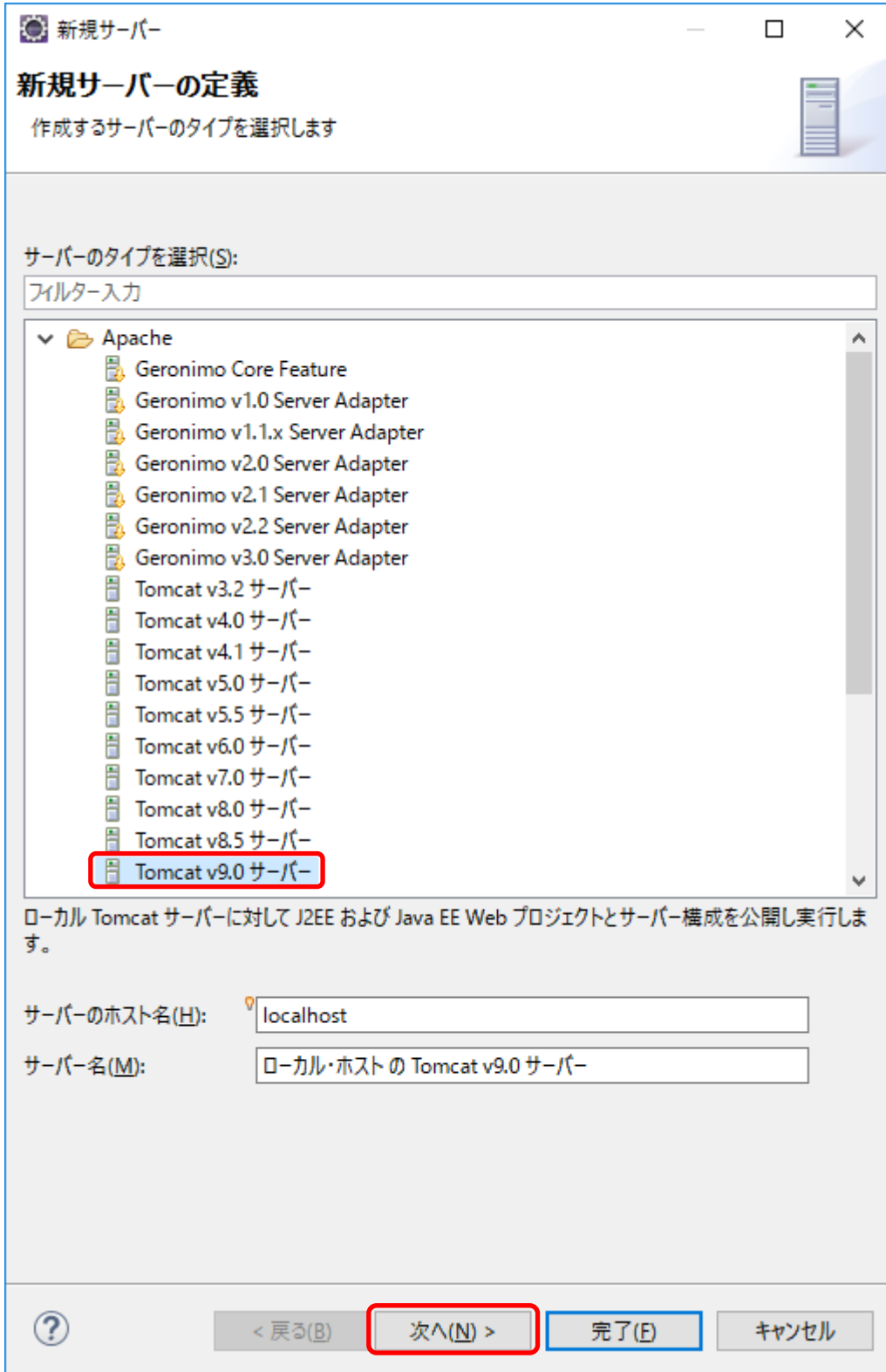
テスト実行には Eclipse に Tomcat サーバまたは相当サーバが作成済みであることが必要です。サーバビュー（ウィンドウメニュー → ビューの表示 → サーバ）で確認できます。

Apache Tomcat サーバが未作成の場合は作成してください。

すでに作成済みであればこの手順は不要です。

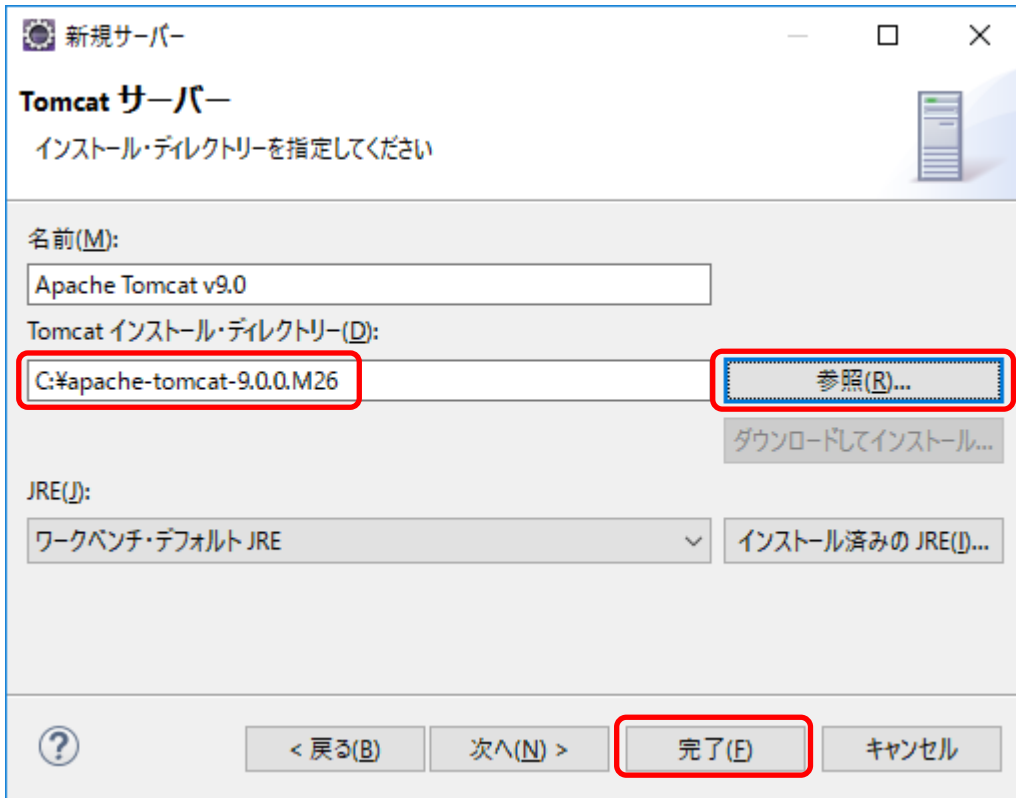


上図はサーバが未作成の例です。作成する方法はいくつかありますが、一例を示します。サーバビューの「使用可能なサーバがありません。…」リンクをクリックします。

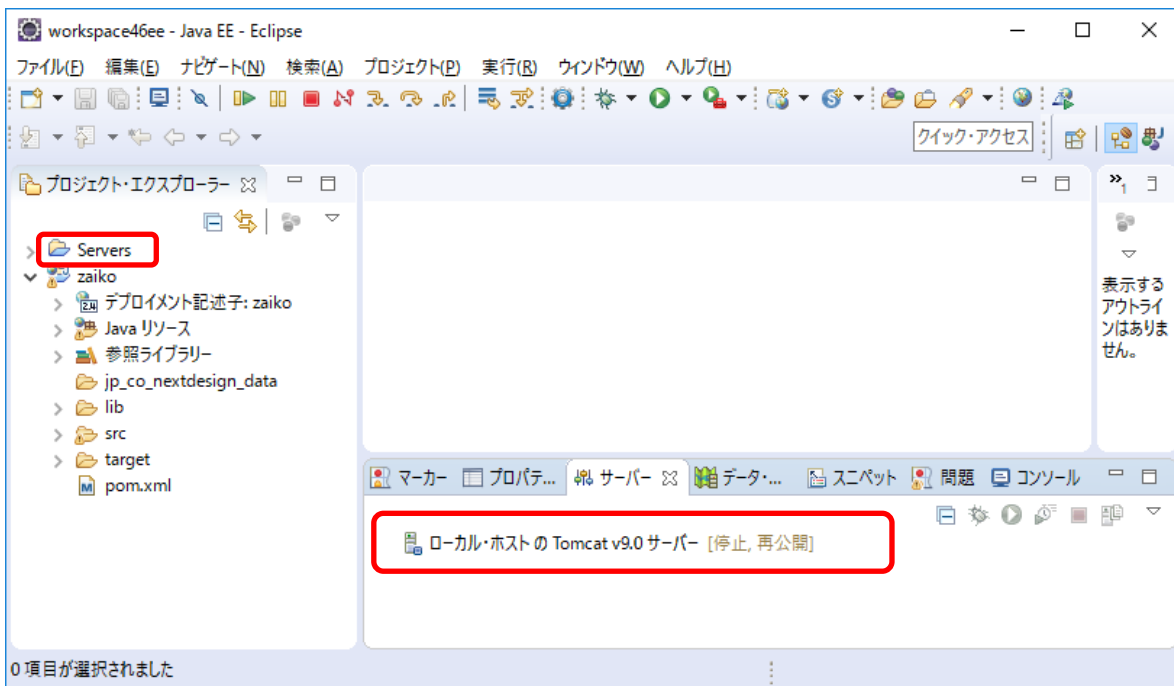


次へ

この例の環境では、Apache Tomcat サーバが C:\¥apache-tomcat-9.0.0.M26 に格納されています。

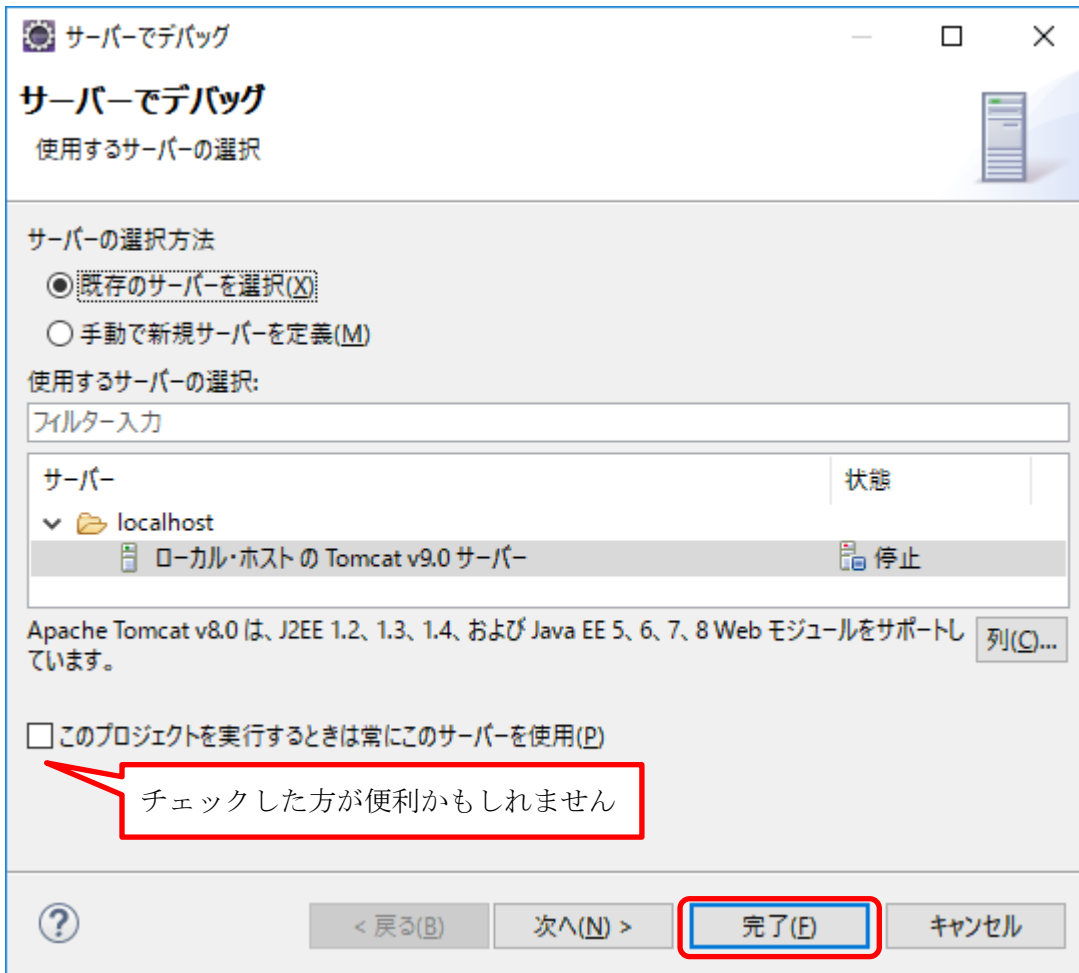


完了

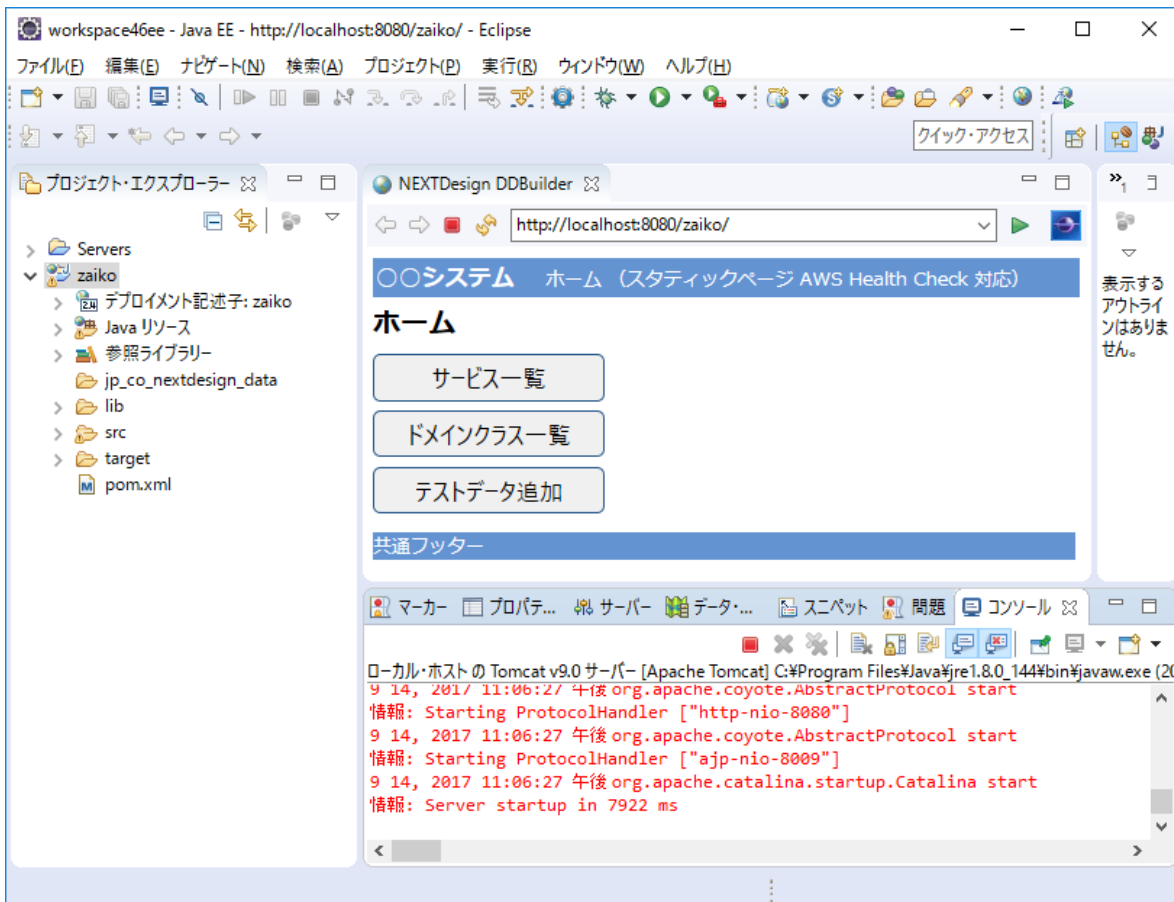


3.5 テスト実行

プロジェクトエクスプローラでプロジェクト（例：zaiko）を
右クリック → デバッグ → サーバでデバッグとします。



下図のように表示されれば正常です。

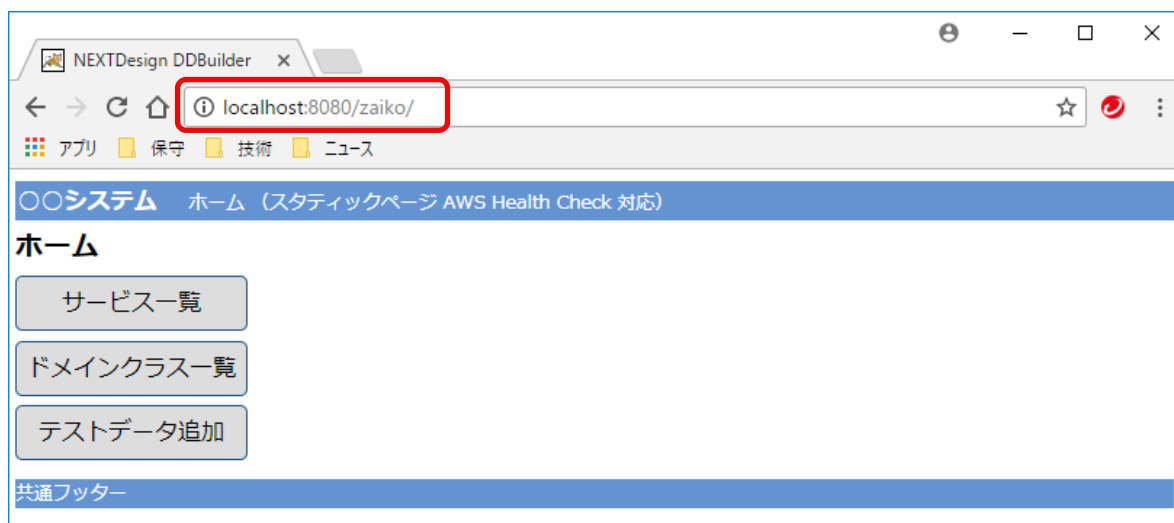


利用者作成のドメインモデルは、この時点ではまだありませんが、画面は操作できます。

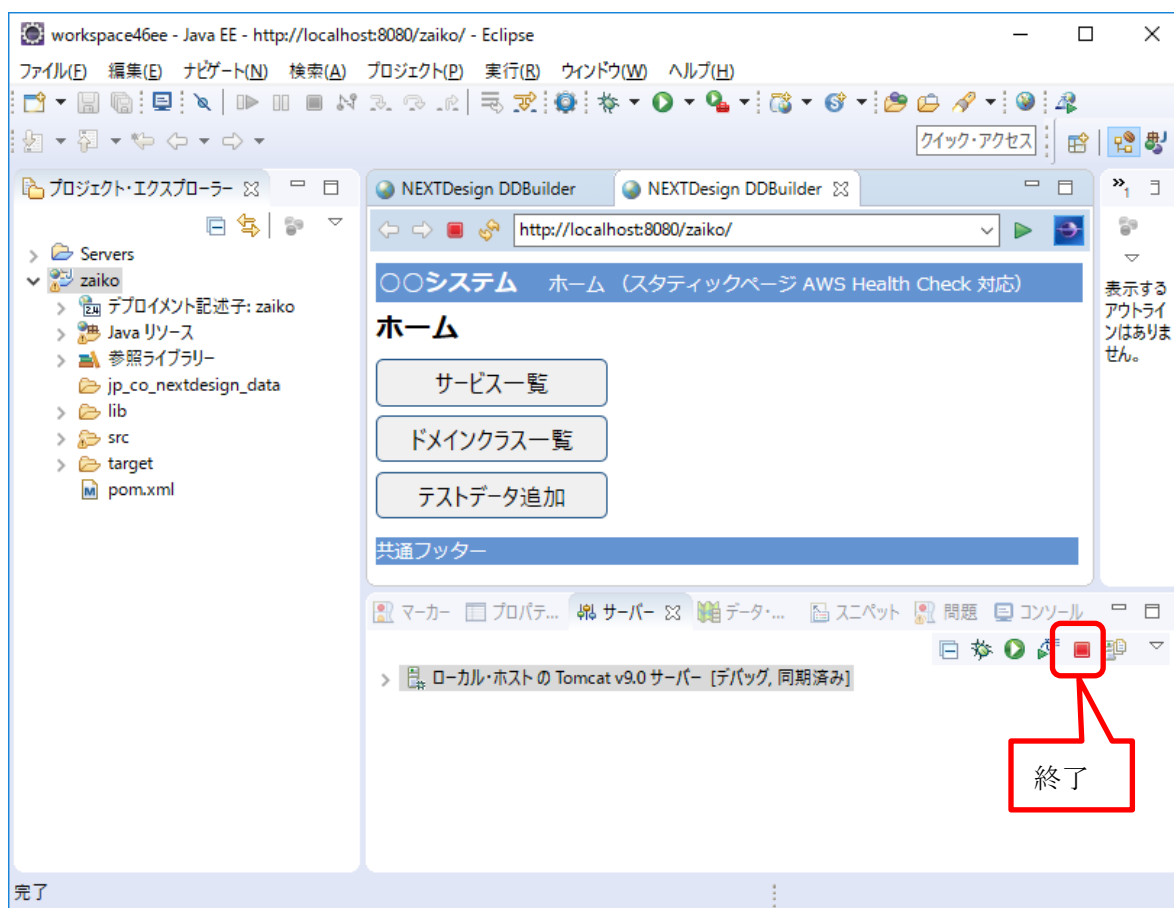
また、サーバが起動中の状態であれば、Eclipse からではなく、他のブラウザからアクセスできます。

下図は Chrome でアクセスした例です。

例 : <http://localhost:8080/zaiko/>

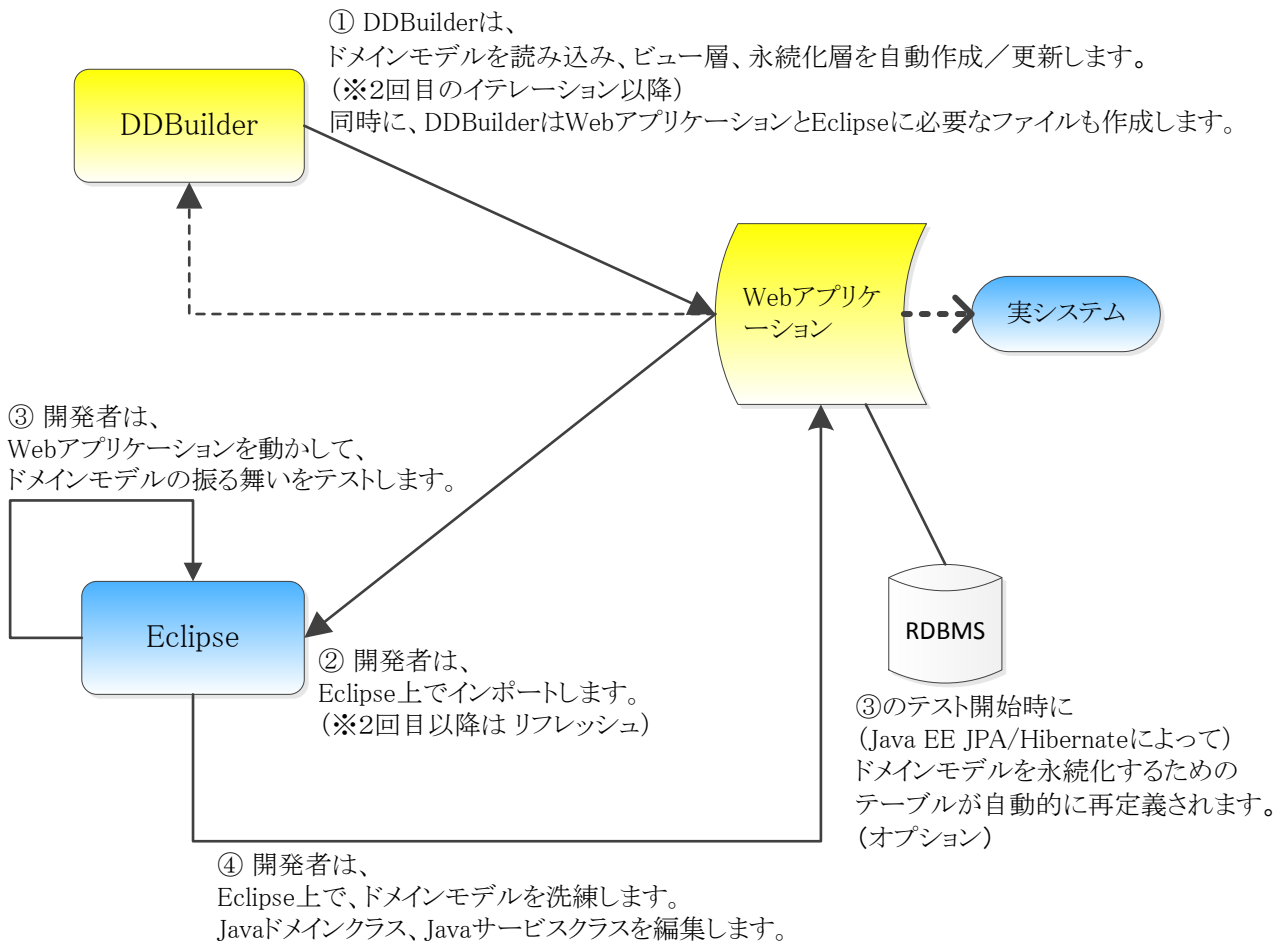


下図の赤いボタンを押下し、デバッグを終了します。



4. イテレーションの流れ

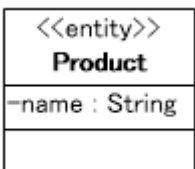
1回のイテレーション ① → ② → ③ → ④



5. イテレーションの例 1

5.1 モデル (例：Product を追加)

例：モデリングを行い、下図の Product クラスが 1 つだけ抽出されたとします。



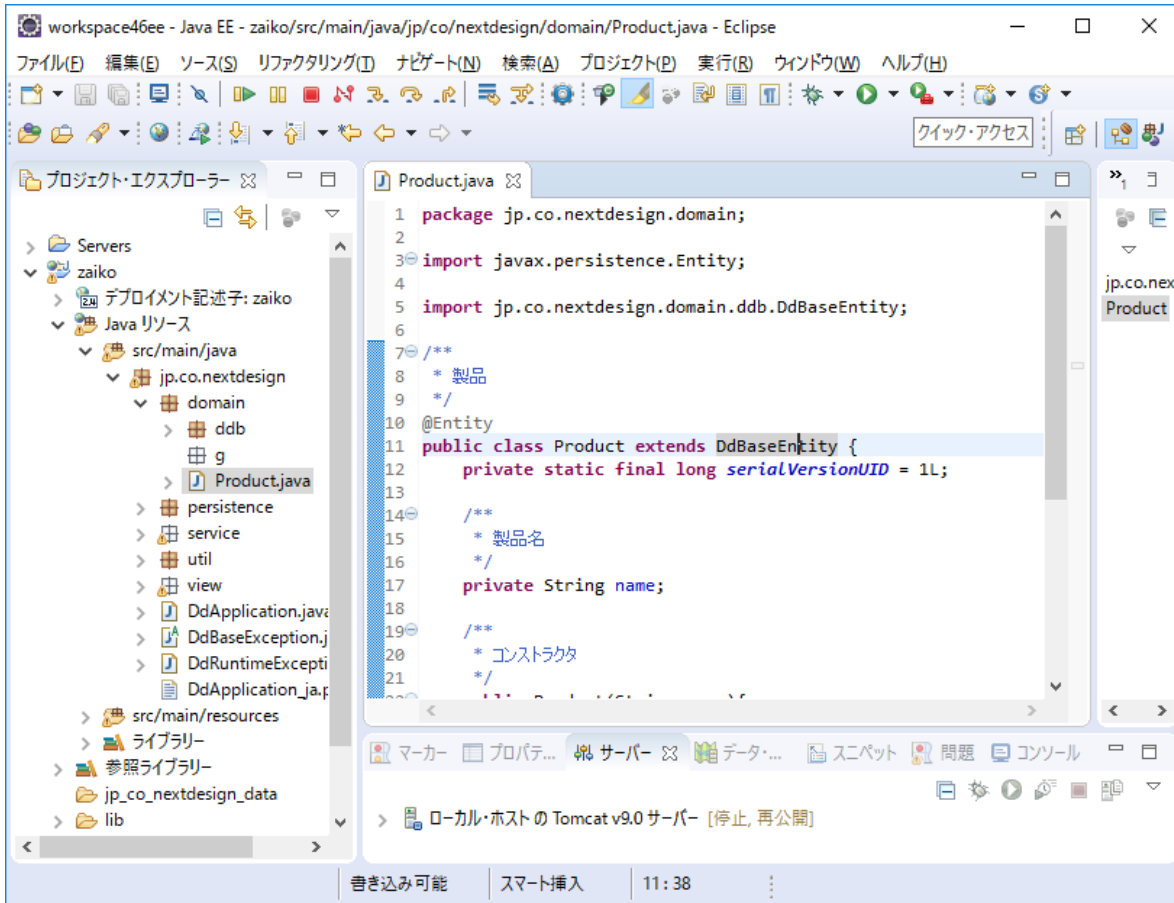
5.2 実装

例：Eclipse で domain パッケージの下に Product クラスを追加します。

(注) コード中の DdBaseEntity は JPA エンティティ用の基底クラスで、DDBuilder が提供します。JPA の要件を満たせば、自作の基底クラスや基底クラスを使用しない選択も可能です。


```
/**
 * 製品
 */
@Entity      <----- JPA アノテーション
public class Product extends DdBaseEntity {    <----- DDBuilder エンティティ基底クラス
    private static final long serialVersionUID = 1L;
    /**
     * 製品名
     */
    private String name;

    /**
     * コンストラクタ
     */
    public Product(){
        super();
        this.name = "";
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```



5.3 新しいモデルを Web アプリケーションに反映

Product クラスを追加したところで、DDBuilder を使って Web アプリケーションを更新します。DDBuilder を起動し、作成／更新ボタンを押下します。



作成/更新の結果、Product に関するビューなどが追加されたことになります。
 ただし、Eclipse は変更を検知していませんので、次のようにリフレッシュ操作してください。

5.4 リフレッシュ

DDBuilder の作成/更新処理が完了したら、Eclipse プロジェクトをリフレッシュします。
 方法: Eclipse のプロジェクトエクスプローラの zaiko を右クリック → リフレッシュ

5.5 内容確認

以下のように更新されていれば正常です。

5.5.1 domain パッケージ

Product.java は利用者が追加したクラスです。

(1) ddb サブパッケージ

追加・更新ありません。

(2) g サブパッケージ

DdProductManager.java が追加されています。

5.5.2 persistence パッケージ

追加・更新ありません。

5.5.3 service パッケージ

追加・更新ありません。

(1) ddb サブパッケージ

追加・更新ありません。

(2) g サブパッケージ

DdProductService.java が追加されています。

5.5.4 util パッケージ

追加・更新ありません。

5.5.5 view パッケージ

(1) ddb サブパッケージ

追加・更新ありません。

(2) g サブパッケージ

次の 2 ファイルが更新されています。

DomainClassListPage.java

DomainClassListPage.html

次の 4 ファイルが追加されています。

ProductEditPage.java (製品編集画面ビュー)

ProductEditPage.html (製品編集画面ビュー)

ProductInstanceListPage.java (製品インスタンス一覧画面ビュー)

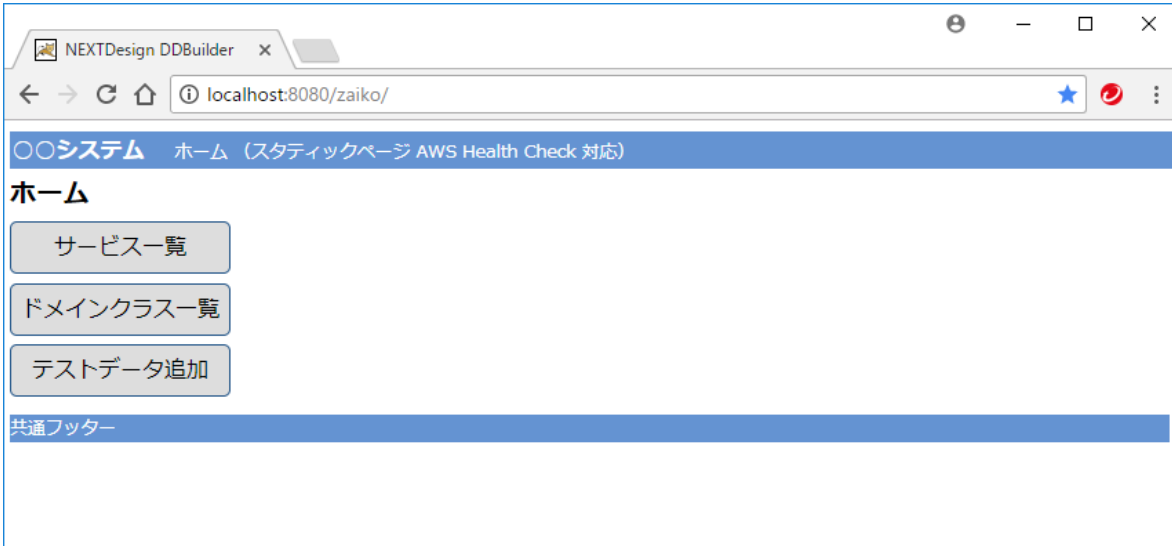
ProductInstanceListPage.html (製品インスタンス一覧画面ビュー)

5.6 テスト実行

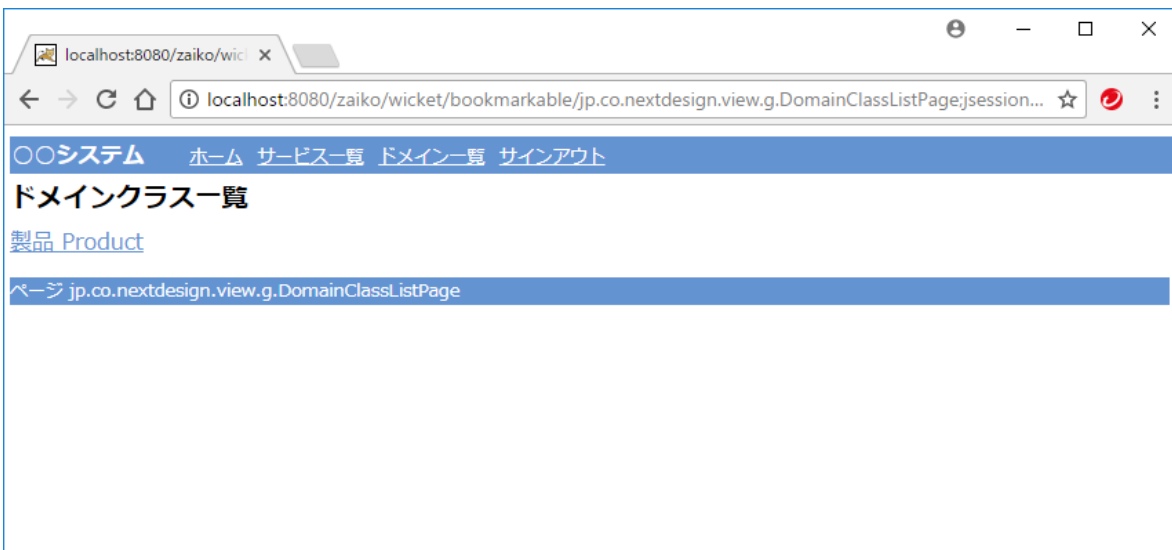
zaiko を右クリック → デバッグ → サーバでデバッグ

例：ここで、製品(Product)クラスのインスタンスを 1 つ登録してみます。

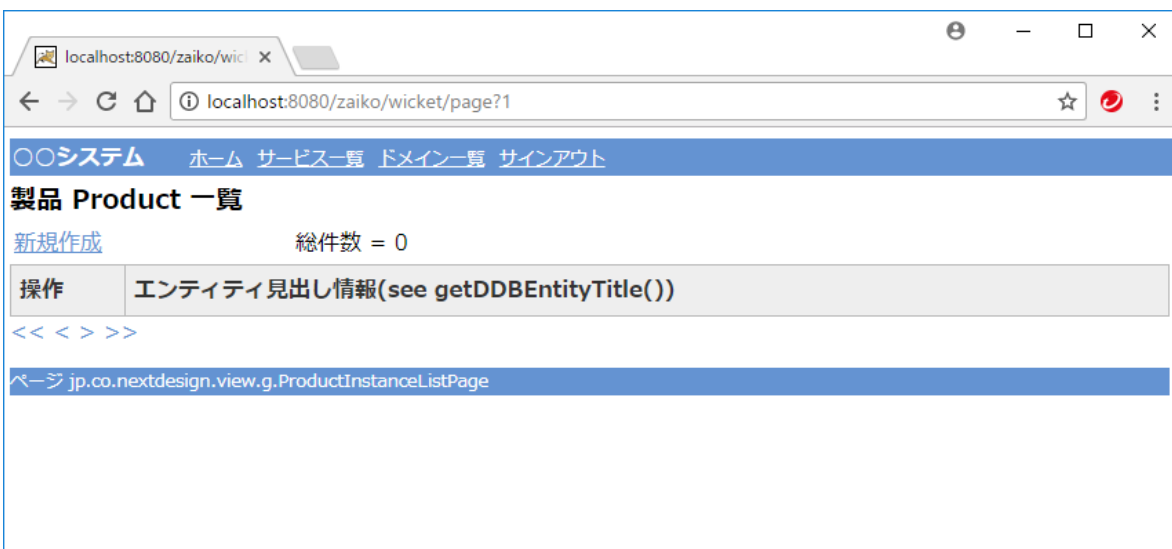
以下はブラウザ (Chrome) でアクセスしたときの画面です。



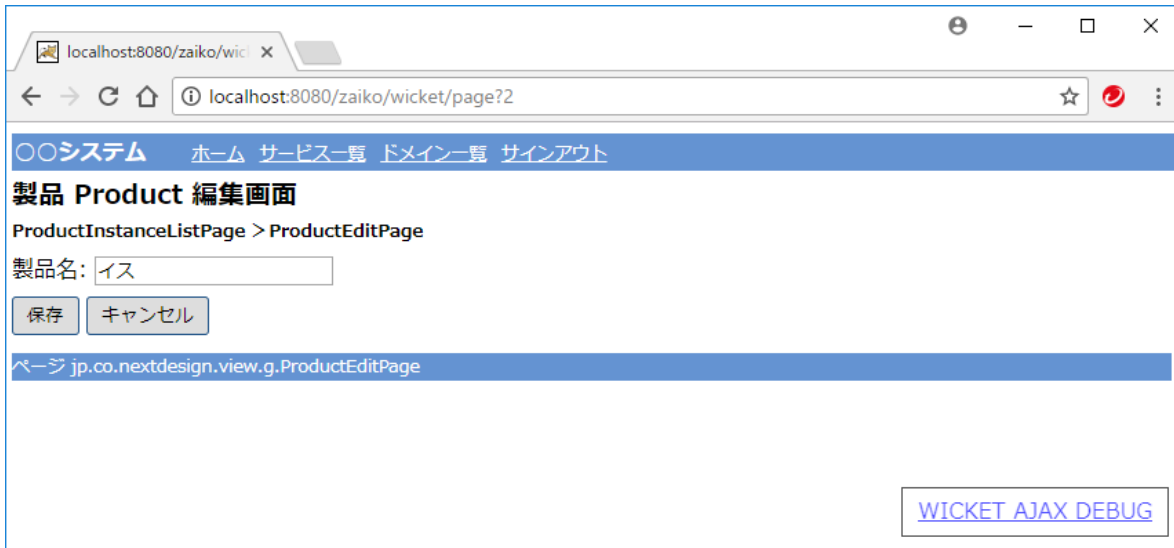
ドメインクラス一覧ボタンを押下



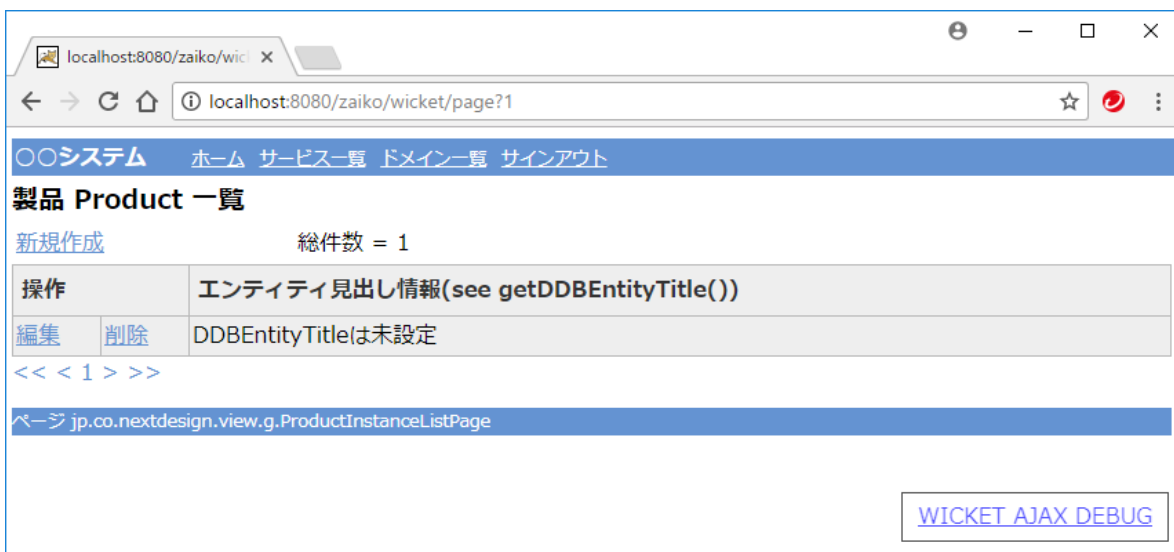
製品 Product リンクをクリック



新規作成リンクをクリック

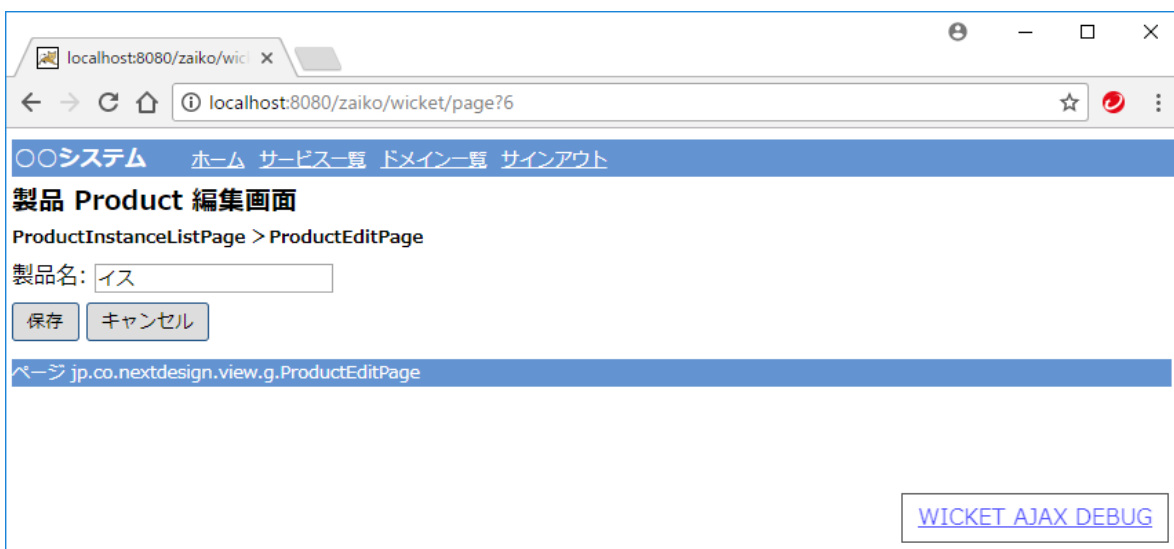


製品名を入力し保存ボタンを押下

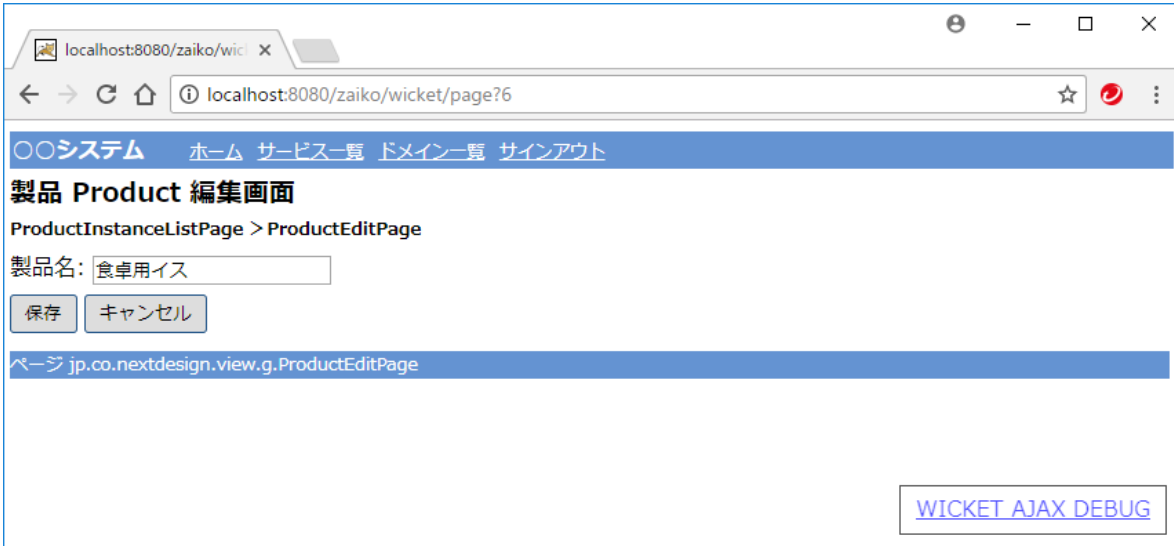


1 件登録されました。

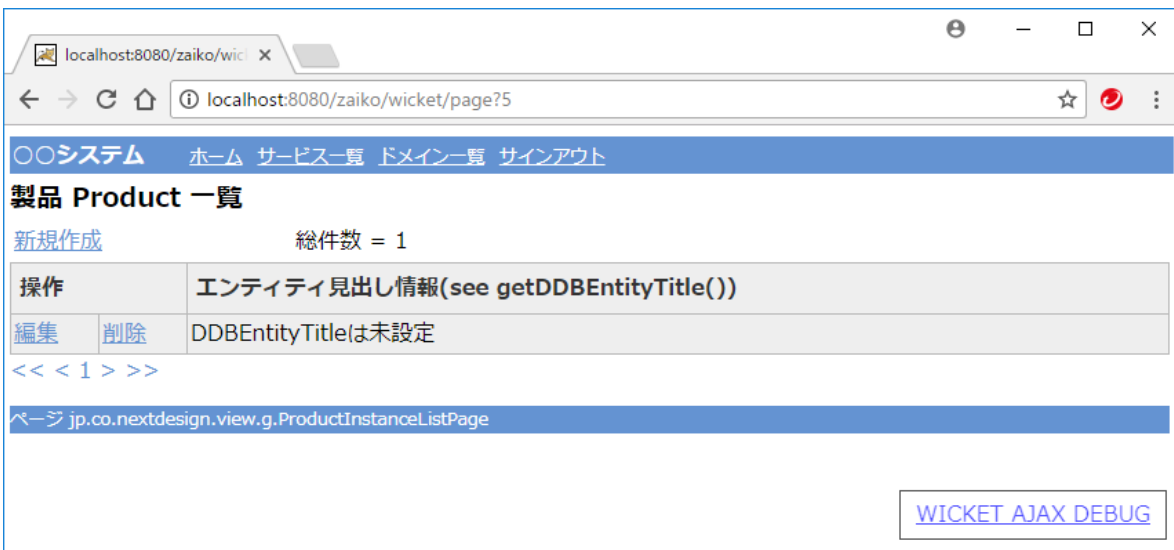
編集リンクをクリック



製品名を変更



保存

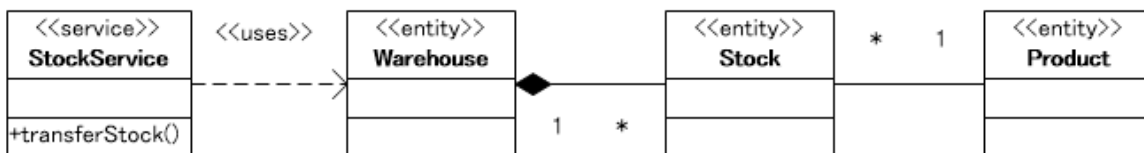


6. イテレーションの例 2

イテレーションの流れとしては、Product クラスを追加した時と同様です。

6.1 モデル (例 : StockService, Warehouse, Stock を追加)

例 : Product クラスだけのモデルを以下のように洗練したとします。



6.2 実装

例 : 後述のコード例を参照してください。

6.3 新しいモデルを Web アプリケーションに反映

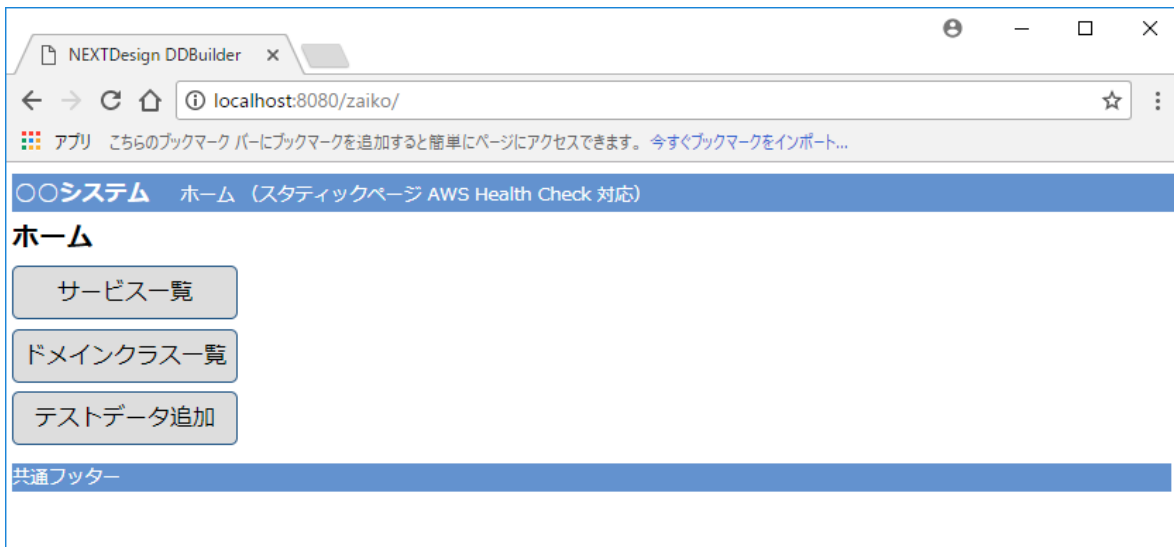
最初のイテレーションの場合と同様です。

6.4 リフレッシュ

最初のイテレーションの場合と同様です。

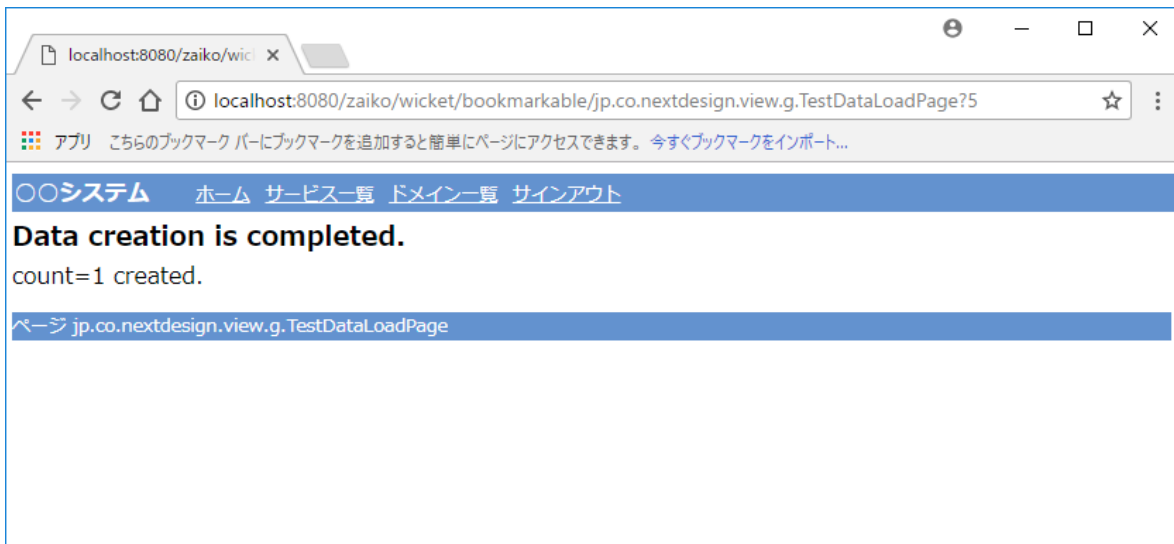
6.5 テスト実行

Web アプリケーションをサーバで開始します。



テストデータ追加ボタンをクリックします。

サインイン画面が表示されたら ID= 123 パスワード=123。

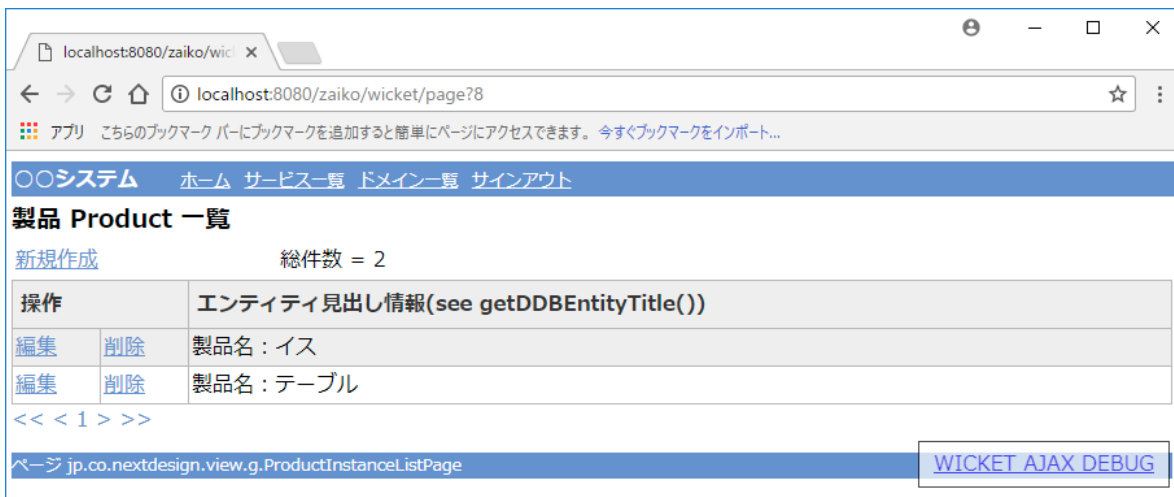


製品、倉庫、在庫の各クラスのインスタンス（テストデータ）が登録されました。

ドメイン一覧リンクから確認できます。



例：製品 Product リンクをクリックします。



例・2行目の編集リンクをクリックします。



編集して保存するか、キャンセルで戻ります。

例：在庫 Stock をクリックします。

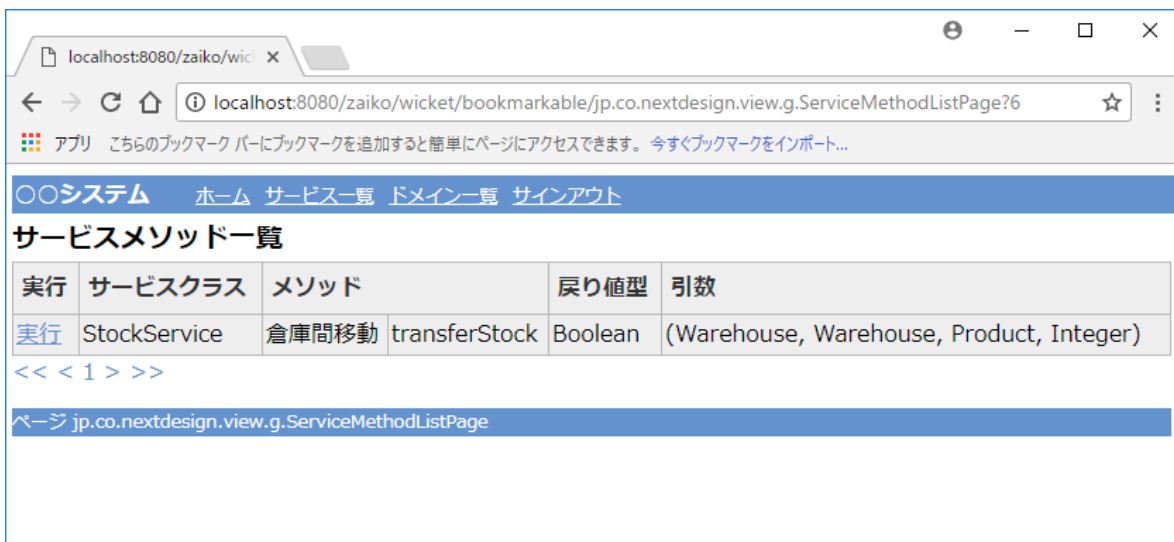


※ここで「エンティティ見出し情報(see getDDBEntityTitle())」列の表示内容が実際と違っているはずです。これは、Product クラスで DdBaseEntity の getDDBEntityTitle() メソッドをオーバーライドしているからです。このルールについては後述します。



次にサービスメソッドを実行してみます。

サービス一覧またはホーム → サービス一覧で下図の画面が表示されます。



実行リンクをクリックします。

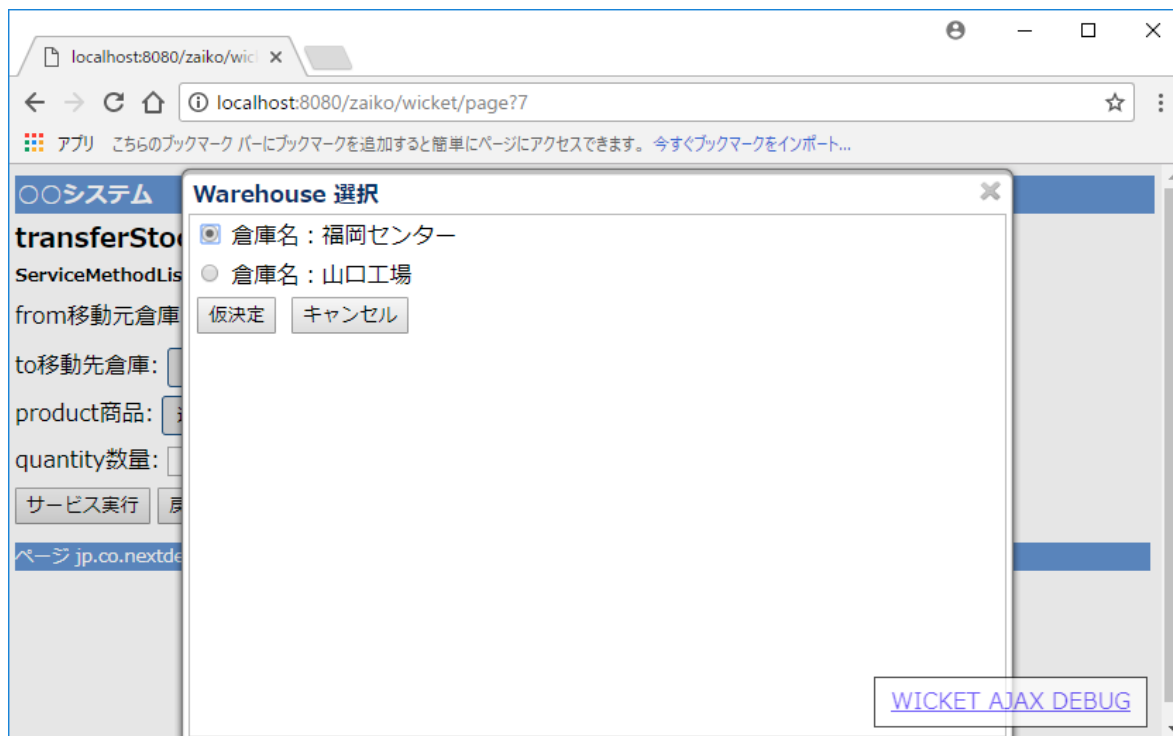
下図の画面が表示されたら、このサービスメソッドに渡す引数を指定します。

from 移動元倉庫：選択ボタンを押下します。

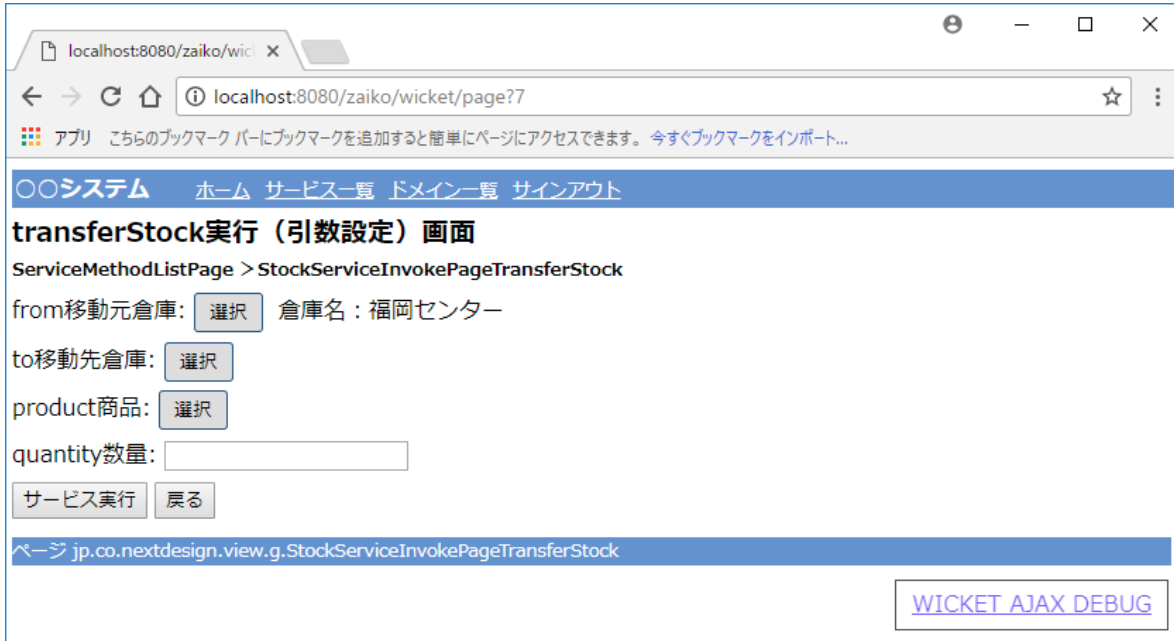
※「from 移動元倉庫」というラベルは、StockService # transferStock メソッドの Javadoc コメントから作成されています。



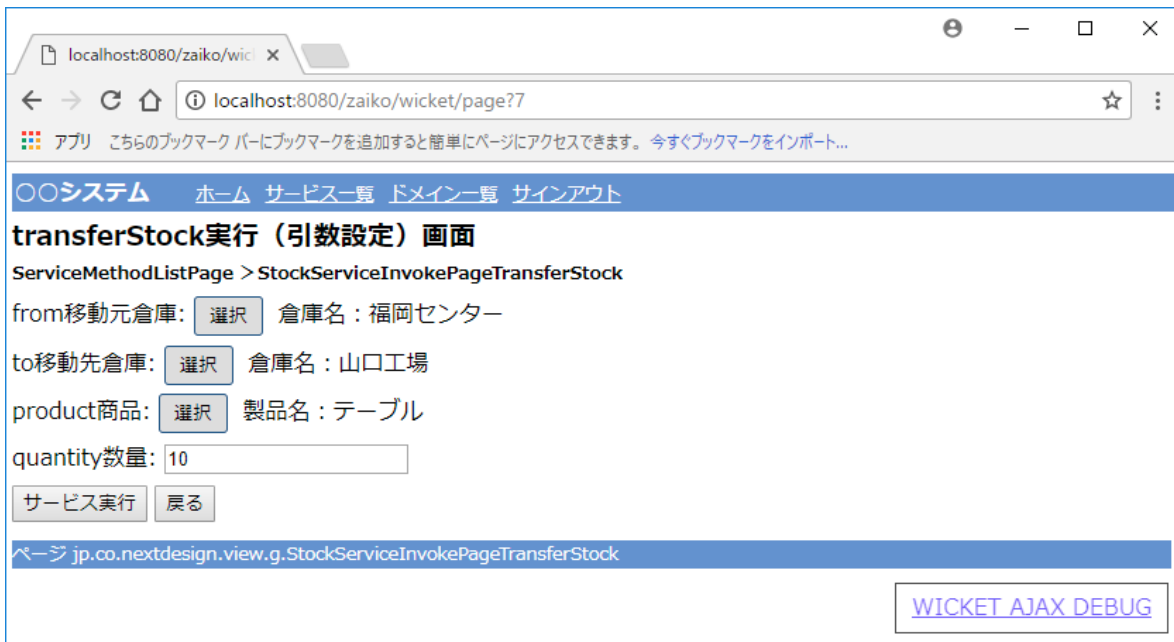
下図の画面で、例として「福岡センター」を選択します。



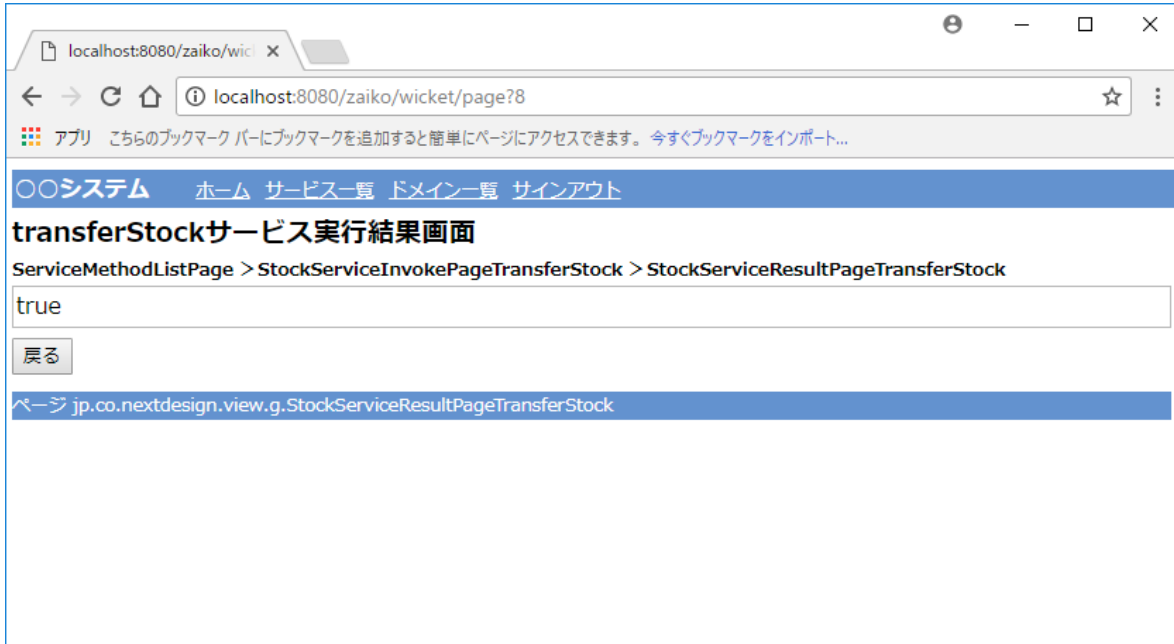
仮決定ボタンを押下します。



同様に他も選択します。quantity 数量は入力します。



サービス実行を押下します。下図が完了画面です。倉庫間移動メソッドの詳細は実装しないで、戻り値 `true` のみを応答していますので、下図のように `true` が表示されます。



倉庫間移動メソッドを具体的に実装した後であれば、上図の戻り値で確認したり、ドメイン一覧画面から倉庫クラスや在庫クラスの内容を参照してサービスの実行結果を確認することができます。

また自作のページを view パッケージに追加することができます。自作ページは DDBuilder 操作画面で作成／更新しても上書きされません。

6.6 実装コード（例：StockService, Warehouse, Stock）

```
package jp.co.nextdesign.domain;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.OneToMany;
import javax.persistence.Transient;
import jp.co.nextdesign.domain.ddb.DdBaseEntity;
/**
 * 製品
 */
@Entity
public class Product extends DdBaseEntity {
    private static final long serialVersionUID = 1L;

    /** 製品名 */
    private String name;

    /** 在庫 */
    @OneToMany(mappedBy="product", cascade=CascadeType.ALL, orphanRemoval=true)
    private List<Stock> stockList = new ArrayList<Stock>();
}
```

```
/** コンストラクタ */
public Product(){
    super();
    this.name = "";
}

//OneToMany で双方向関連を維持するためのコードを
//含む getStockList(),setStockList(List<Stock> stockList)の例
@Transient
private ArrayList<Stock> latestStockList = new ArrayList<Stock>();
public List<Stock> getStockList() {
    return this.stockList;
}

public void setStockList(List<Stock> stockList) {
    for(Stock newStock : stockList){
        if (!latestStockList.contains(newStock)){
            newStock.setProduct(this);
        }
    }
    for(Stock oldStock : latestStockList){
        if (!stockList.contains(oldStock)){
            oldStock.setProduct(null);
        }
    }
    this.stockList = stockList;
    latestStockList = new ArrayList<Stock>(this.stockList);
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

/** DDB 一覧表示用タイトル */
@Override
public String getDDBEntityTitle(){
    return "製品名 : " + this.getName();
}
}
```

```
package jp.co.nextdesign.domain;
import java.util.Calendar;
import java.util.Date;
import javax.persistence.Entity;
import javax.persistence.ManyToOne;
import jp.co.nextdesign.domain.ddb.DdBaseEntity;
/**
 * 在庫
 */
@Entity
public class Stock extends DdBaseEntity {
    private static final long serialVersionUID = 1L;

    /** 数量 */
    private Integer quantity;

    /** 製品 */
    @ManyToOne
    private Product product;

    /** 倉庫 */
    @ManyToOne
    private Warehouse warehouse;

    /** コンストラクタ */
    public Stock(){
        super();
        this.quantity = 0;
    }

    /**
     * 次月の在庫予定日を応答する
     * @return 次月の在庫予定日
     */
    public Date getNextMonthWarehousingDate(){
        Date result = Calendar.getInstance().getTime();
        //何らかの実装
        return result;
    }

    /** DDB 一覧表示用タイトル */
```

@Override

```
public String getDDBEntityTitle(){
    String result = "製品名=";
    result += this.getProduct() != null ? this.getProduct().getName() : "";
    result += " 倉庫名=";
    result += this.getWarehouse() != null ? this.getWarehouse().getName() : "";
    result += " 数量=" + this.getQuantity();
    return result;
}

public Integer getQuantity() {
    return quantity;
}

public void setQuantity(Integer quantity) {
    this.quantity = quantity;
}

public Product getProduct() {
    return product;
}

public void setProduct(Product product) {
    this.product = product;
}

public Warehouse getWarehouse() {
    return warehouse;
}

public void setWarehouse(Warehouse warehouse) {
    this.warehouse = warehouse;
}
}
```

```
package jp.co.nextdesign.domain;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.OneToOne;
import javax.persistence.Transient;
import jp.co.nextdesign.domain.ddb.DdBaseEntity;
/**
```

```
 * 倉庫
```

```
 */
```

@Entity

```
public class Warehouse extends DdBaseEntity {
```



```
private static final long serialVersionUID = 1L;

/** 倉庫名 */
private String name;

/** 在庫リスト */
@OneToMany(mappedBy="warehouse", cascade=CascadeType.ALL, orphanRemoval=true)
private List<Stock> stockList = new ArrayList<Stock>();

/** コンストラクタ */
public Warehouse(){
    super();
    this.name = "";
}

/** 製品在庫を追加する */
public boolean addStock(Product product, int quantity){
    //処理 (この例では省略)
    return true;
}

/** 製品在庫を削減する */
public boolean removeStock(Product product, int quantity){
    //処理 (この例では省略)
    return true;
}

//OneToMany で双方向関連を維持するためのコードを
//含む getStockList(),setStockList(List<Stock> stockList)の例
@Transient
private ArrayList<Stock> latestStockList = new ArrayList<Stock>();
public List<Stock> getStockList() {
    return this.stockList;
}

public void setStockList(List<Stock> stockList) {
    for(Stock newStock : stockList){
        if (!latestStockList.contains(newStock)){
            newStock.setWarehouse(this);
        }
    }
    for(Stock oldStock : latestStockList){
        if (!stockList.contains(oldStock)){

```

```
        oldStock.setWarehouse(null);
    }
}
this.stockList = stockList;
latestStockList = new ArrayList<Stock>(this.stockList);
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

/** DDB 一覧表示用タイトル */
@Override
public String getDDBEntityTitle(){
    return "倉庫名 : " + this.getName();
}
}

package jp.co.nextdesign.service;
import jp.co.nextdesign.domain.Product;
import jp.co.nextdesign.domain.Warehouse;
import jp.co.nextdesign.service.ddb.DdBaseService;
/**
 * 在庫管理サービス
 */
public class StockService extends DdBaseService {

    /**
     * 倉庫間移動
     * @param from 移動元倉庫
     * @param to 移動先倉庫
     * @param product 移動する商品
     * @param quantity 移動する数量
     * @return
     */
    public Boolean transferStock(Warehouse from, Warehouse to, Product product, Integer quantity) {
        boolean result = false;
        try {
            startService(); //サービス初期化処理
        }
    }
}
```

```
begin(); //トランザクション開始

// 倉庫間移動処理
if (from.removeStock(product, quantity)){
    if (to.addStock(product, quantity)){
        result = true;
    }
}
if (result){
    commit(); //トランザクション commit
} else {
    rollback(); //トランザクション rollback
}
} catch (Exception e) {
    rollback();
} finally {
    endService(); //サービス終了処理
}
return result;
}
}
```

```
/*
```

```
 * もとは DDBuilder2.1 が生成したファイルです。
```

```
*/
```

```
package jp.co.nextdesign.service.g;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import jp.co.nextdesign.domain.Product;
import jp.co.nextdesign.domain.Stock;
import jp.co.nextdesign.domain.Warehouse;
import jp.co.nextdesign.domain.g.DdProductManager;
import jp.co.nextdesign.domain.g.DdStockManager;
import jp.co.nextdesign.domain.g.DdWarehouseManager;
import jp.co.nextdesign.service.ddb.DdBaseService;
```

```
/**
```

```
 * テスト用データを作成するサービス
```

```
 * このクラスは DDBuilder によって上書きされません。存在しない場合のみ新規作成します。
```

```
 * Service for creating test data.
```

```
 * This class is not overwritten by DDBuilder. It will only be created if it does not exist.
```

```
 *
```

```
*/
```

```
public class TestDataService extends DdBaseService{

    public void addTestData(int count){
        try {
            startService();
            begin();

            //製品
            DdProductManager bookManager = new DdProductManager();
            Product product1 = new Product();
            product1.setName("イス");
            bookManager.persist(product1);
            //
            Product product2 = new Product();
            product2.setName("テーブル");
            bookManager.persist(product2);
            //倉庫
            DdWarehouseManager warehouseManager = new DdWarehouseManager();
            Warehouse warehouse1 = new Warehouse();
            warehouse1.setName("福岡センター");
            warehouseManager.persist(warehouse1);
            //
            Warehouse warehouse2 = new Warehouse();
            warehouse2.setName("山口工場");
            warehouseManager.persist(warehouse2);
            //在庫
            DdStockManager stockManager = new DdStockManager();
            Stock stock1 = new Stock();
            stock1.setProduct(product1);
            stock1.setWarehouse(warehouse1);
            stock1.setQuantity(10);
            stockManager.persist(stock1);
            //
            Stock stock2 = new Stock();
            stock2.setProduct(product2);
            stock2.setWarehouse(warehouse2);
            stock1.setQuantity(20);
            stockManager.persist(stock2);

            commit();
        } catch (Exception e) {
            rollback();
        }
    }
}
```

```

    } finally {
        endService();
    }
}
}
}

```

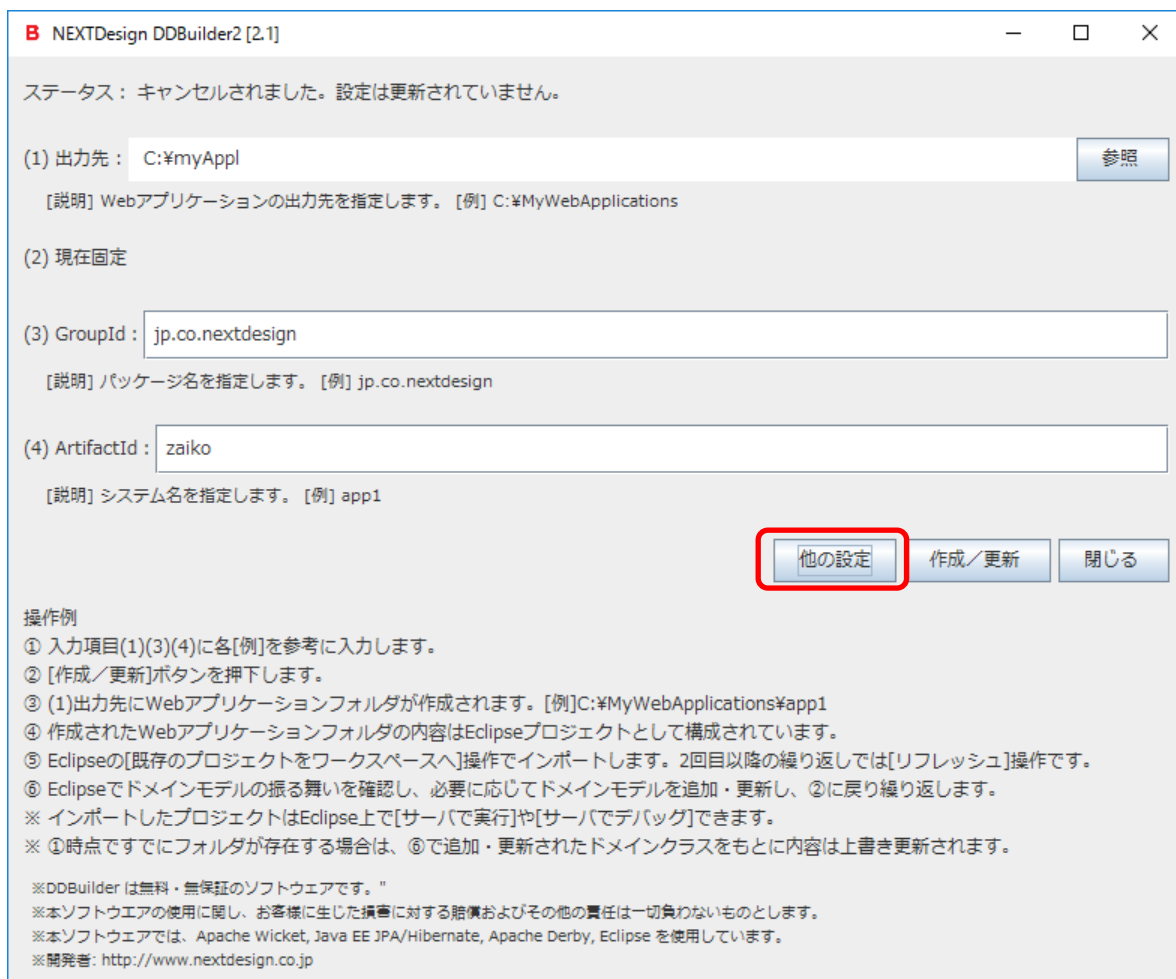
7. Web アプリケーションのカスタマイズ

7.1 システム名称のカスタマイズ

各ページのヘッダー部のシステム名を設定します。



DDBuilder 操作画面で他の設定を押下します。



B プロパティ設定 ×

アプリケーション名: 〇〇システム 例：在庫管理システム

会社名: 〇〇株式会社

スタートページクラス完全名:

完了 キャンセル

7.2 一覧画面の表示内容のカスタマイズ

localhost:8080/zaiko/wic ×

localhost:8080/zaiko/wicket/page?5

〇〇システム ホーム サービス一覧 ドメイン一覧 サインアウト

製品 Product 一覧

[新規作成](#) 総件数 = 1

操作	エンティティ見出し情報(see getDDBEntityTitle())
編集 削除	DDBEntityTitleは未設定

<< < 1 > >>

ページ jp.co.nextdesign.view.g.ProductInstanceListPage

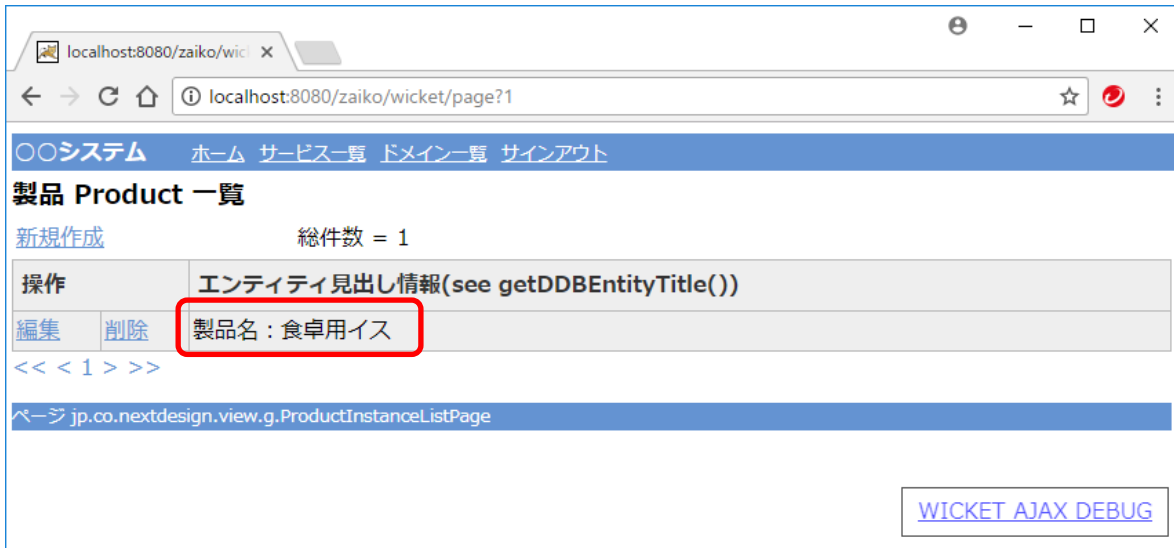
WICKET AJAX DEBUG

上図赤枠の内容をカスタマイズします。

例：Product.java に次のようなコードを追加して、DdBaseEntity#getDDBEntityTitle()メソッドをオーバーライドします。

@Override

```
public String getDDBEntityTitle(){
    return "製品名：" + this.getName();
}
```

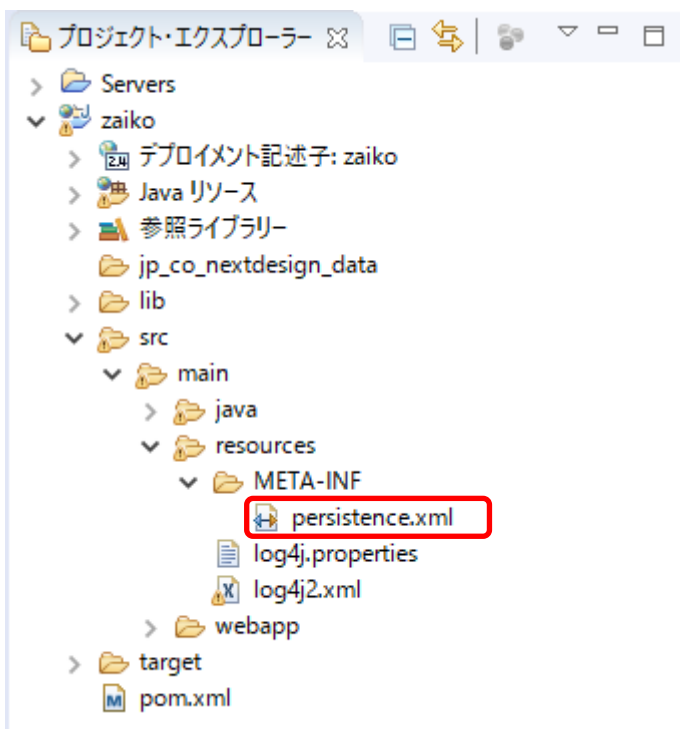


7.3 JPA 永続化方式のカスタマイズ

サーバを起動 (Tomcat を起動) すると、初期設定の場合、登録したデータは消えます。

起動する度に RDB テーブルが新しく定義されるからです。

RDB テーブルを毎回定義しないで、前回の状態を復元するようにするためには、下図の赤枠にある persistence.xml ファイルを変更します。



変更方法 : persistence.xml を開き、下記の「create-drop」を「none」に変更します。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0"
```

```
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
```

```
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
```

```
    <persistence-unit name="nextdesign_pu" transaction-type="RESOURCE_LOCAL">
```

```

<description>Derby Persistence Unit</description>
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

<!-- This is where we tell JPA/Hibernate about our @Entity objects -->

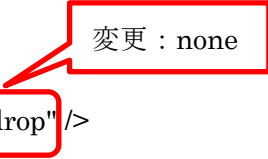
<properties>
  <property name="javax.persistence.jdbc.driver" value="org.apache.derby.jdbc.EmbeddedDriver"
/>

  <!-- create Database or not -->
  <property name="javax.persistence.jdbc.url"
value="jdbc:derby:jp_co_nextdesign_zaiko_data;create=true" />
  <property name="javax.persistence.jdbc.user" value="" />
  <property name="javax.persistence.jdbc.password" value="" />
  <property name="hibernate.dialect" value="org.hibernate.dialect.DerbyTenSixDialect" />

  <!--
value =
"create-drop" or "create" AT REBUILD,
"update" IN DEVELOPMENT,
"none" IN OPERATION
-->
  <property name="hibernate.hbm2ddl.auto" value="create-drop" />

  <property name="hibernate.show_sql" value="true" />
  <property name="hibernate.format_sql" value="true" />
  <property name="hibernate.transaction.flush_before_completion" value="true" />
</properties>
</persistence-unit>
</persistence>

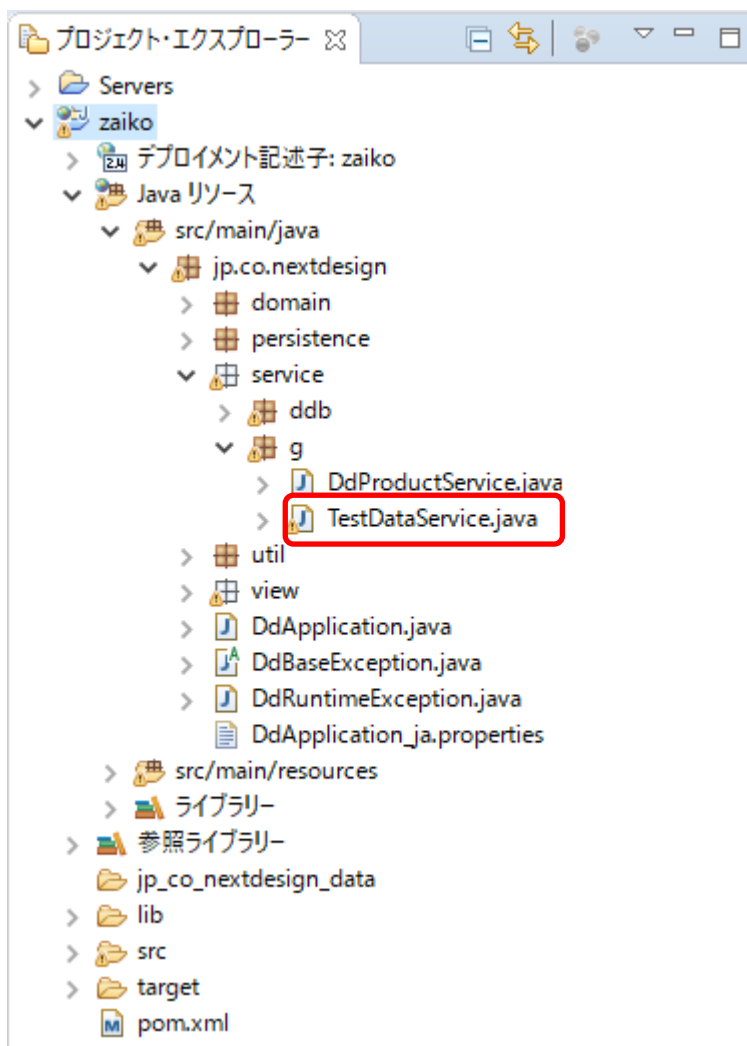
```


 変更 : none

ただし、`create-drop` の場合は、実行開始時にドメインモデル（`Product` など）に合わせて RDB のテーブルが定義されますが、`none` の場合はいっさい変更されません。そのため、次のイテレーションで既存のエンティティに属性を追加した場合や、新たなエンティティを追加した場合などでは、永続化クラスと RDB テーブル定義の間で不整合が発生し、実行できません。従って、エンティティの変更を頻繁に繰り返すような段階では `create-drop` が適しています。しかし、イテレーションの度に登録画面からテストインスタンスを登録することも現実的ではありません。そこで、DDBuilder としては次のサービスクラスを用意しています。

7.4 テストデータ一括登録サービスメソッドのカスタマイズ

下図の `testDataService.java` をカスタマイズします。



例えば、下記赤枠のように実装し、Product クラスのインスタンスを 2 件登録するようにします。

```
public class TestDataService extends DdBaseService{
    public void addTestData(int count){
        try {
            startService();
            begin();
            DdProductManager bookManager = new DdProductManager();
            Product product = new Product();
            product.setName("食卓用イス");
            bookManager.persist(product);
            product = new Product();
            product.setName("食卓用テーブル");
            bookManager.persist(product);
            commit();
        } catch (Exception e) {
            rollback();
        } finally {
            endService();
        }
    }
}
```

```
}
```

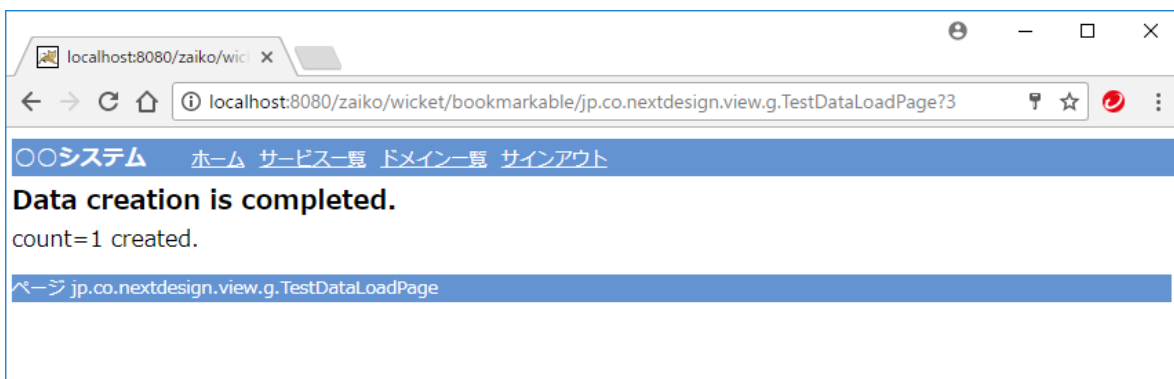
サーバを再起動してホーム画面の「テストデータ追加」ボタンを押下します。



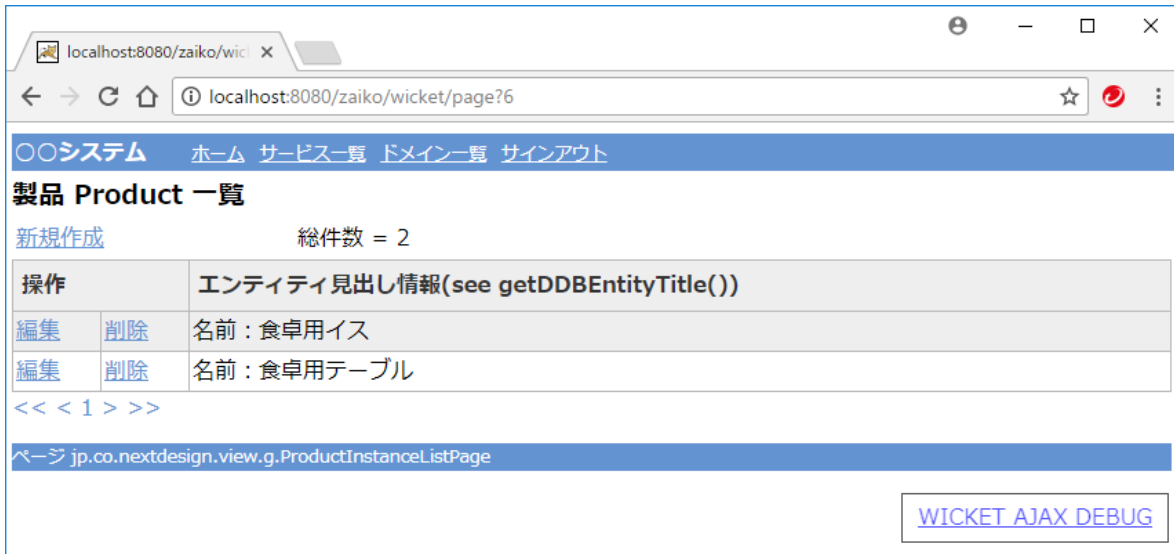
サインイン画面が出ます。サインイン認証は仮実装なので、123/123 でサインインできます。



サインインボタンを押下すると、`TestDataService# addTestData(count)`が実行されます。



常に `count=1` と表示されますが、`addTestData` の実装通りに 2 件登録されます。一覧画面で確認できます。



8. 制限事項

8.1 プリミティブ型ラッパークラスを使用

Boolean

Character

Byte

Short

Integer

Long

Float

Double

8.2 使用できる Java 標準クラス

java.lang.String,

java.util.Date,

java.math.BigDecimal

8.3 コレクション型とマップ型

JPA 仕様では、

- java.util.Collection
- java.util.Set
- java.util.List
- java.util.Map

を使用できますが、DDBuilder では、

- java.util.List<T>

のみ使用できます。

また、DDBuilder の生成コード内で使用するコレクションの具象クラスは、`java.util.ArrayList<T>` です。

8.4 同じ組み合わせのクラス間で定義できる ManyToMany 関連の数

同じ組み合わせのクラス間（クラス A<*>-----<*>クラス B）での ManyToMany 関連の数は 1 個までです。この場合、クラス A、クラス B にそれぞれ `add` メソッド、`remove` メソッドの定義が必要です。

```
A#addB(B b)
A#removeB(B b)
B#addA(A a)
B#remove(A a)
```

Java で実装したドメインクラスの例を提供していますので、ご参考ください。

8.5 List の要素型

`List<T>` の T は、`domain` パッケージ内の `@Entity` が付いたクラスのみ使用できます。

8.6 クラス名（テーブル名に SQL 予約語は使用不可）

例 1 使用不可

```
@Entity
```

```
public class Order extends DdBaseEntity {
```

上記の場合、RDB 上のテーブル名はクラス名と同じ `Order` とされます。しかし、`Order` は SQL 予約語であるため使用できません。（SQL 構文エラーになります）

例 2 使用可能

```
@Entity
```

```
@Table(name="ORDER_TABLE")
```

```
public class Order extends DdBaseEntity {
```

例 1 の問題は、`@Table` アノテーションを使って、テーブル名として SQL 予約語以外の名前を指定することで回避できます。

8.7 双方向関連

8.7.1 @OneToMany の場合

`get` メソッドは単純な `getter` で問題ありませんが、`set` メソッドには追加の実装が必要です。

8.7.2 @ManyToMany の場合

単純な `getter/setter` に加えて、`add` メソッド、`remove` メソッドが必要です。

次を参考にしてください。

参考：Hibernate ドキュメント「[Hibernate ORM 5.2 User Guide](#)」の 2.7.2 Bidirectional

参考:ネクストデザインサイトの <http://www.nextdesign.co.jp/ddd/index.html> にあるドメインクラスのサンプルに含まれている Book クラスを中心に参考にしてください。

9. DDBuilder が上書きするファイル

DDBuilder で Web アプリケーションを更新した場合に、上書きされるファイルは以下の通りです。

9.1 domain

ddb 上書き

g 上書き

他 DDBuilder は参照のみします。

9.2 service

ddb 上書き

g 上書き (TestDataService.java は上書きしません)

他 DDBuilder は参照のみします。

9.3 view

ddb 上書き

g 上書き

他 上書きしません。

9.4 lib

各 jar は初回のみ出力します。

理由は、derby-xxx.jar は Eclipse のデータ・ソース・エクスプローラでドライバとして参照設定した場合、次に DDBuilder が出力するときに上書き不可エラーになるためです。

10. データベース

10.1 ログ

サーバでデバッグすると Eclipse のコンソールに次のようなログが表示されます。

ログ例:

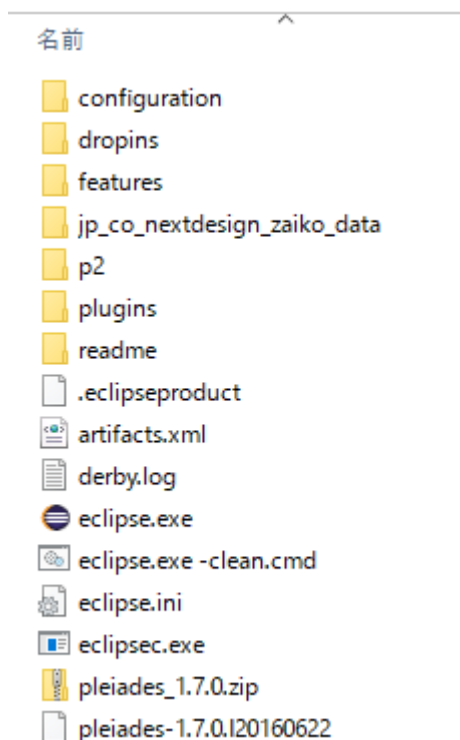
```
Hibernate:
    drop table Book
Hibernate:
    drop table hibernate_sequences
Hibernate:
    create table Book (
        id bigint not null,
        createdAt timestamp,
        name varchar(255),
        primary key (id)
    )
Hibernate:
    create table hibernate_sequences (
        sequence_name varchar(255),
        sequence_next_hi_value integer
    )
```

10.2 Derby データファイルの場所

DDBuilder が生成した Web アプリケーションは、デフォルトでは Derby データベースを使用します。

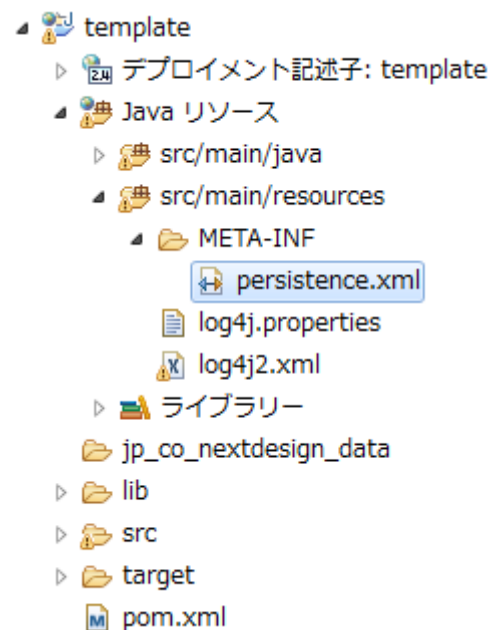
Derby のデータファイルは以下の場所に作成されます。

Eclipse フォルダ (eclipse.exe が在るフォルダ) 直下の jp_co_nextdesign_zaiko_data フォルダの中です。



10.3 Derby 以外のデータベースを使用する場合

Derby 以外のデータベースを使用する場合や、テーブル再定義のタイミングを変更する場合は、persistence.xml を変更してください。指定方法は JPA や Hibernate の使用をご確認ください。



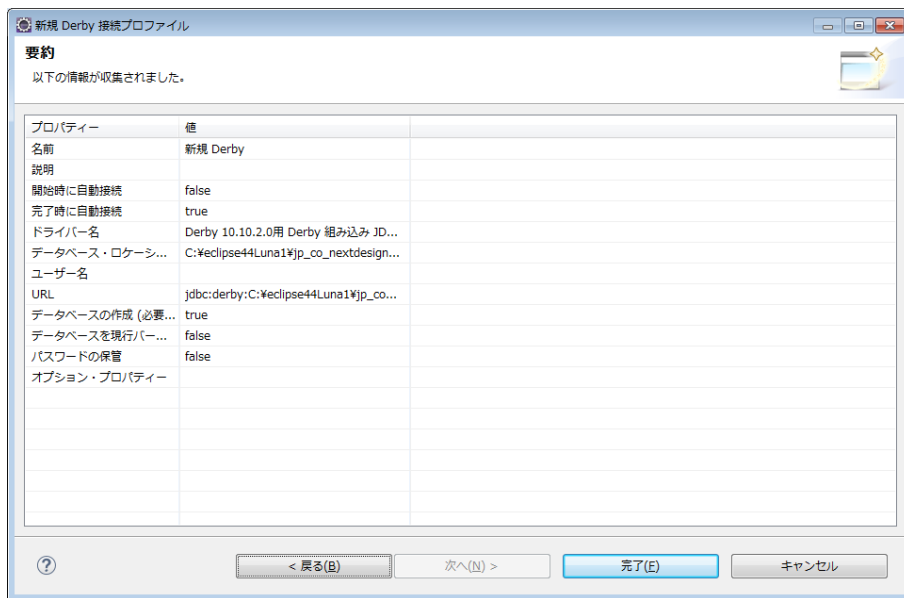
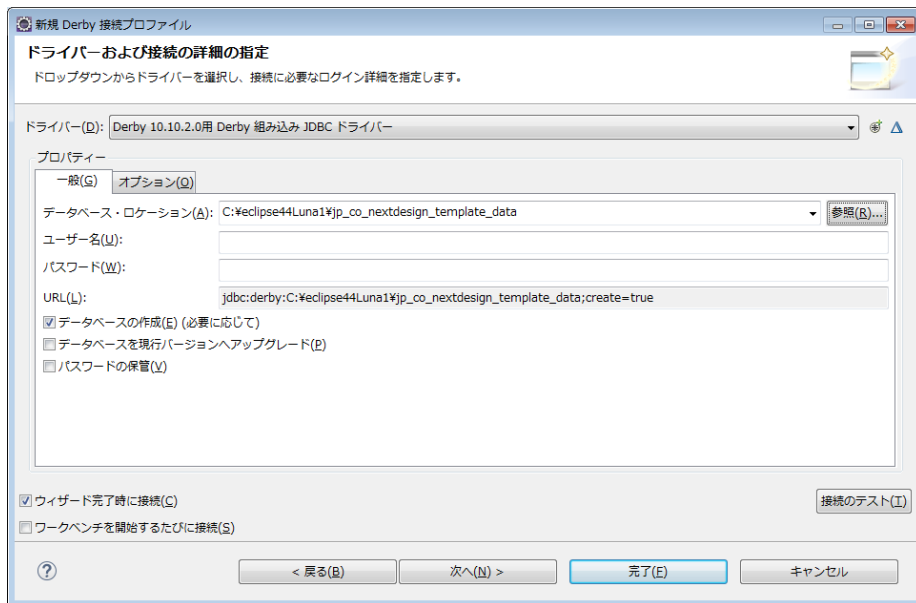
10.4 データベース（テーブル）の参照

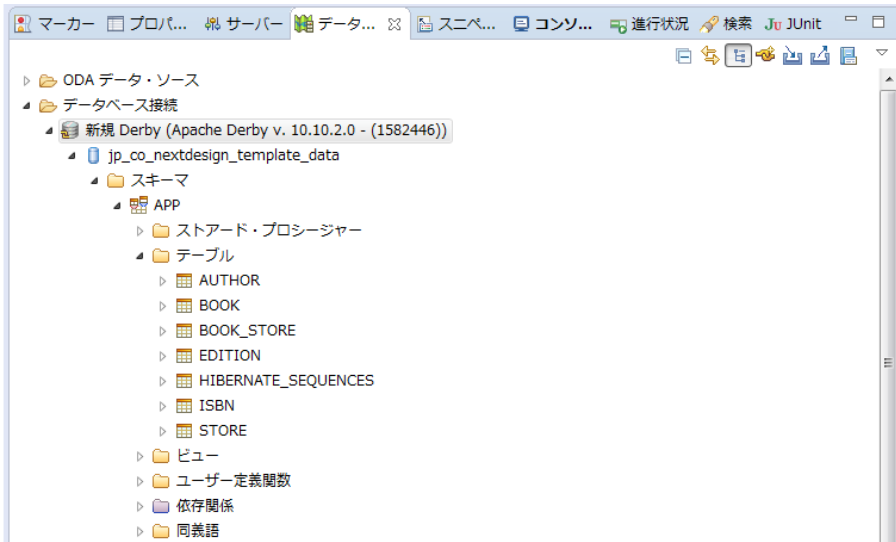
Eclipse で実行している場合は、次の場所にデータベースが作成されています。

Eclipse フォルダ（eclipse.exe が在るフォルダ）¥jp_co_nextdesign_zaiko_data
 （zaiko は DDBuilder 操作画面の ArtifactId で指定した文字列です）

Eclipse のデータ・ベース・エクスプローラで参照できます。

データ・ベース・エクスプローラ → 新規 → Derby → 以下参照（ドライバーのバージョンに注意）

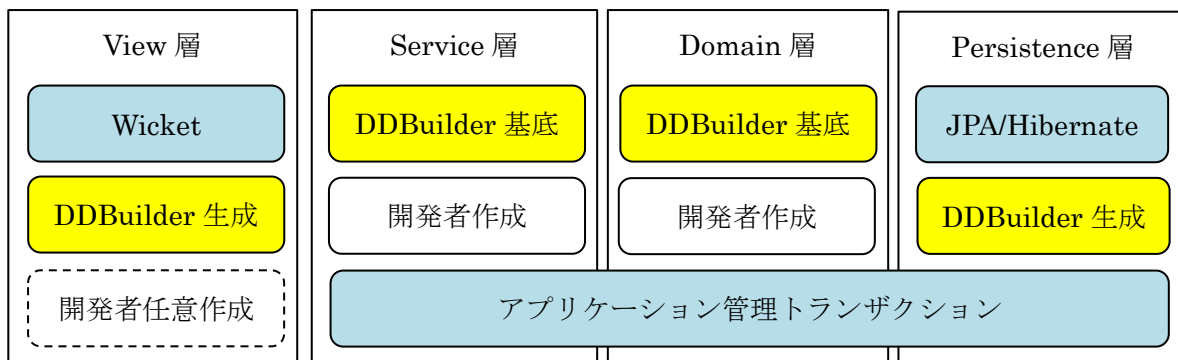




後述の【補足】データベーステーブルの確認方法も参考ください。

11. Web アプリケーションの説明

11.1 構造図



11.2 永続化に関して

11.2.1 エンティティマネージャ

- コンテナ管理エンティティマネージャ
- アプリケーション管理エンティティマネージャ

本アプリケーションでは、アプリケーション管理のエンティティマネージャを使用します。

11.2.2 トランザクション

- JTA トランザクション
- リソースローカルトランザクション

本アプリケーションでは、リソースローカルトランザクションを使用します。

11.2.3 設定定義ファイル

persistence.xml

11.2.4 データベースファイル

DDBuilder が生成した Web アプリケーションは、デフォルトでは、JavaDB（組込モード）を使用します。データベースファイルは、

- Eclipse 上で実行した場合は、Eclipse のワークスペース直下
- Tomcat にデプロイして実行した場合は、apache-tomcat フォルダの bin 直下のフォルダ jp_co_nextdesign_zaiko_data に作成されます。

11.3 テスト用初期データの作成

次のクラスに実装します。初期作成時は内容は空です。このクラスは再作成しても上書きされません。
service.g.TestDataService#addTestData(int count)

11.4 サインイン機能

初期作成すると次のクラスに簡易実装しています。このクラスは再作成すると上書きされます。
view.ddb.DdSession#authenticate(final String userName, final String password)

11.5 log4j2 ログ設定

Web アプリケーション出力フォルダ¥src¥main¥resources¥log4j2.xml

12. トラブル時の対応

12.1 Eclipse のデータ・ソース・エクスプローラでデータベースに接続できない

データ・ソース・エクスプローラの接続時に使用するドライバーを、DDBuilder が使用している derby.jar にします。データ・ソース・エクスプローラが使用するドライバーが古いバージョンの derby.jar になっている可能性があります。

12.2 JPA (Hibernate)に関連する実行時エラーが解消しない

データベースファイル（フォルダ）を一旦削除してみます。

データベースファイルは、

- Eclipse 上で実行した場合は、Eclipse のワークスペース直下
- Tomcat にデプロイして実行した場合は、apache-tomcat フォルダの bin 直下のフォルダ jp_co_nextdesign_zaiko_data に作成されます。

13. 補足：データベーステーブルの確認方法

データ・ソース・エクスプローラビューでデータベースの状態を確認できます。

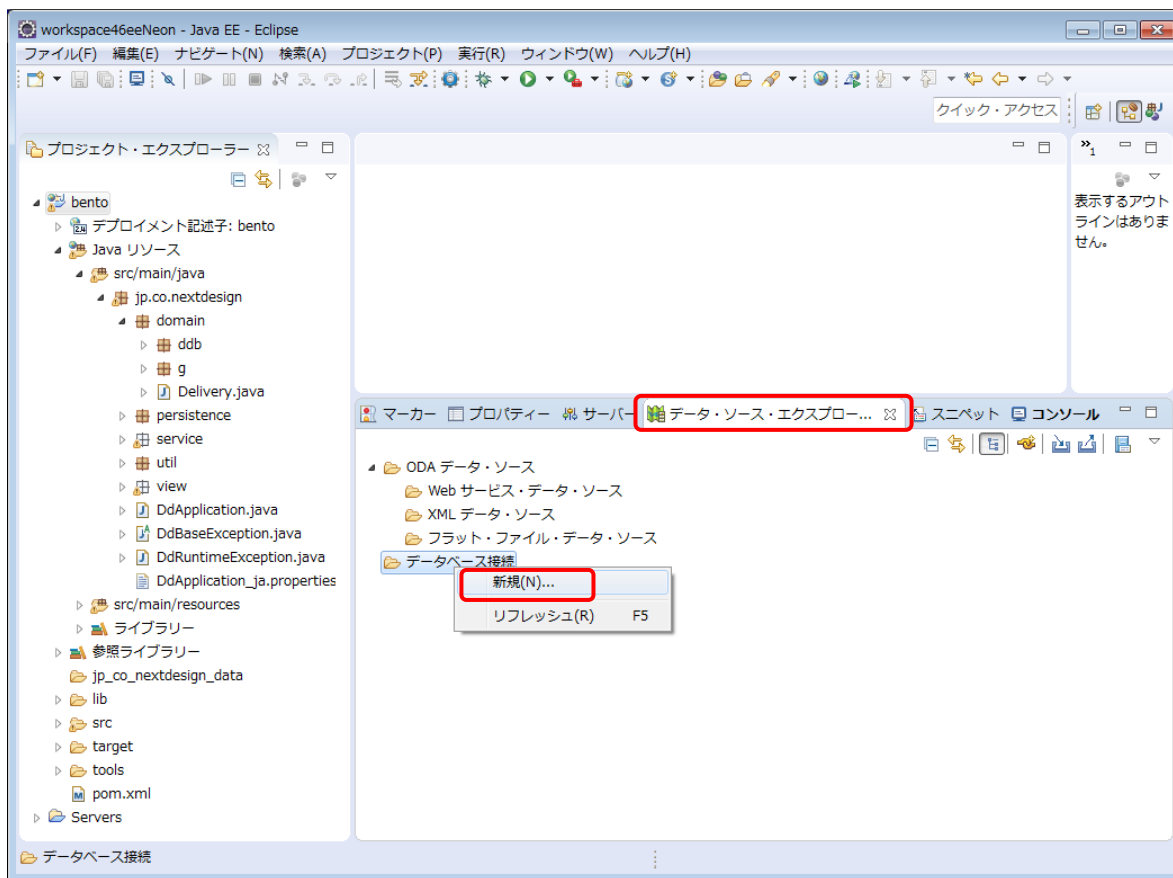
この章は以下の環境で説明します。

Windows 7 Pro 64bit

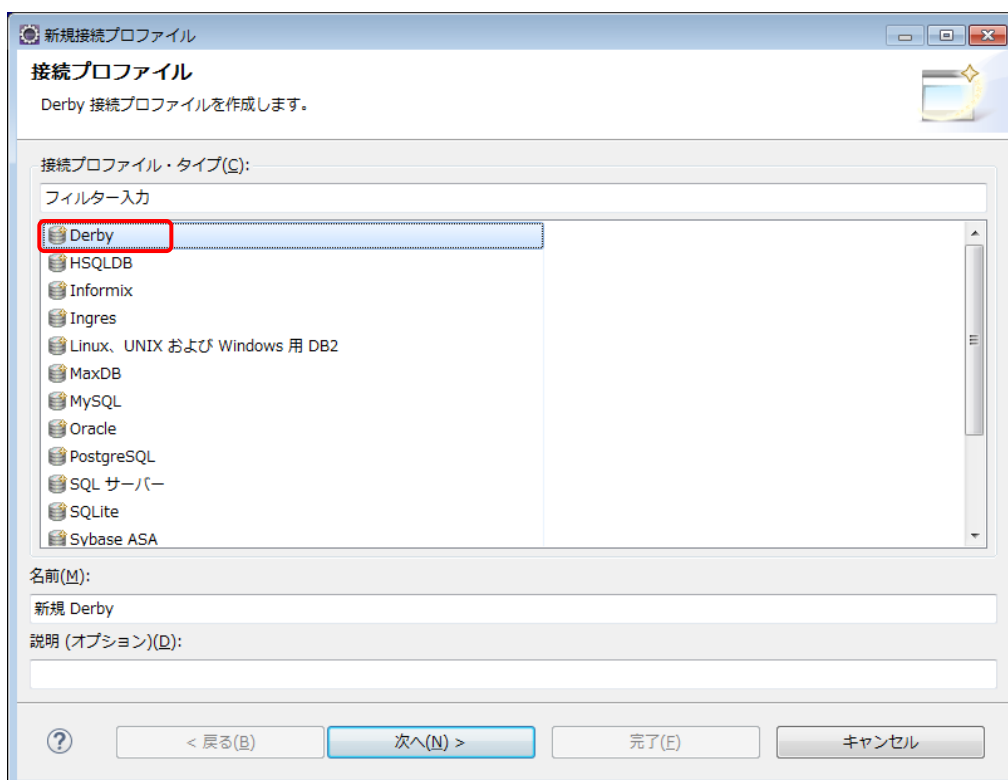
Eclipse Neon 2 Packages – Eclipse IDE for Java EE Developers + Pleiades(pleiades-1.7.13)

java version "1.8.0_111"

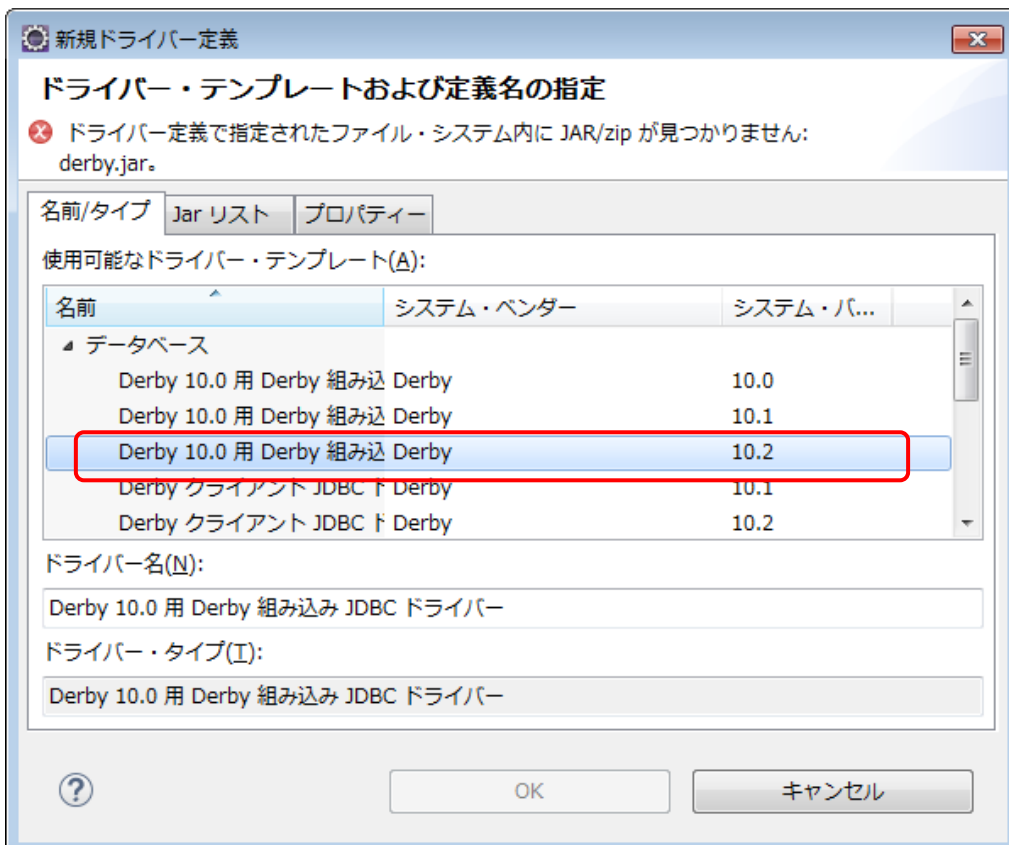
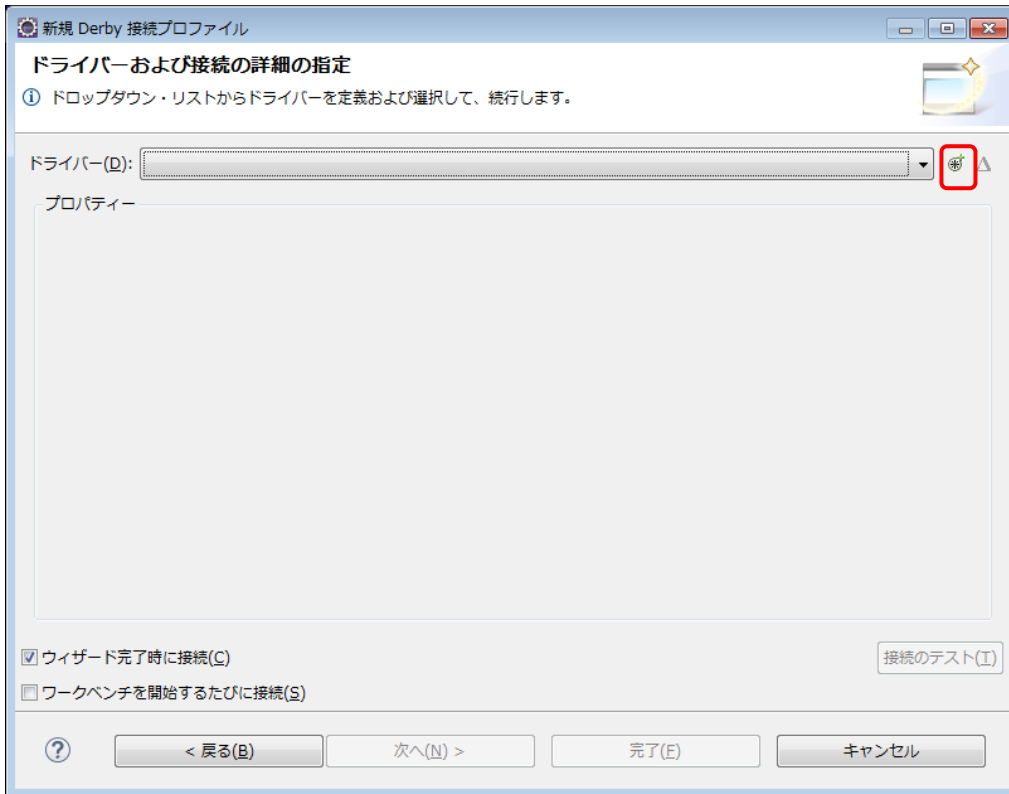
Java(TM) SE Runtime Environment (build 1.8.0_111-b14)



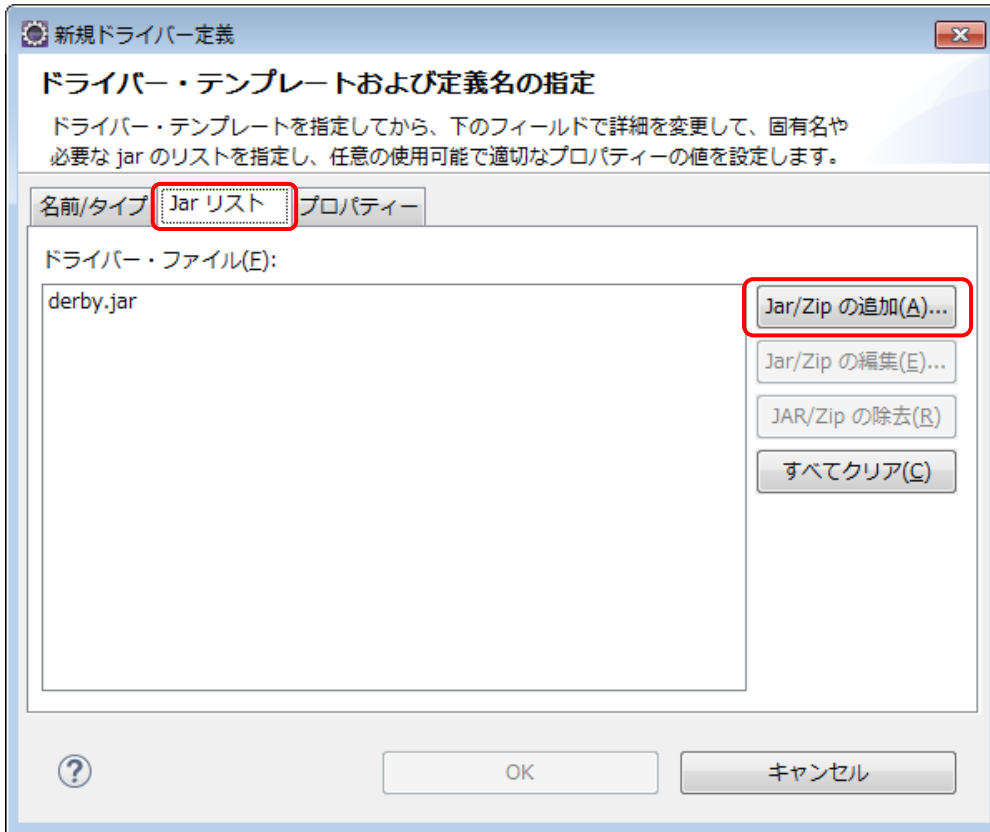
デフォルトの場合、JDK に組み込まれている JavaDB (Derby) を使用されます。



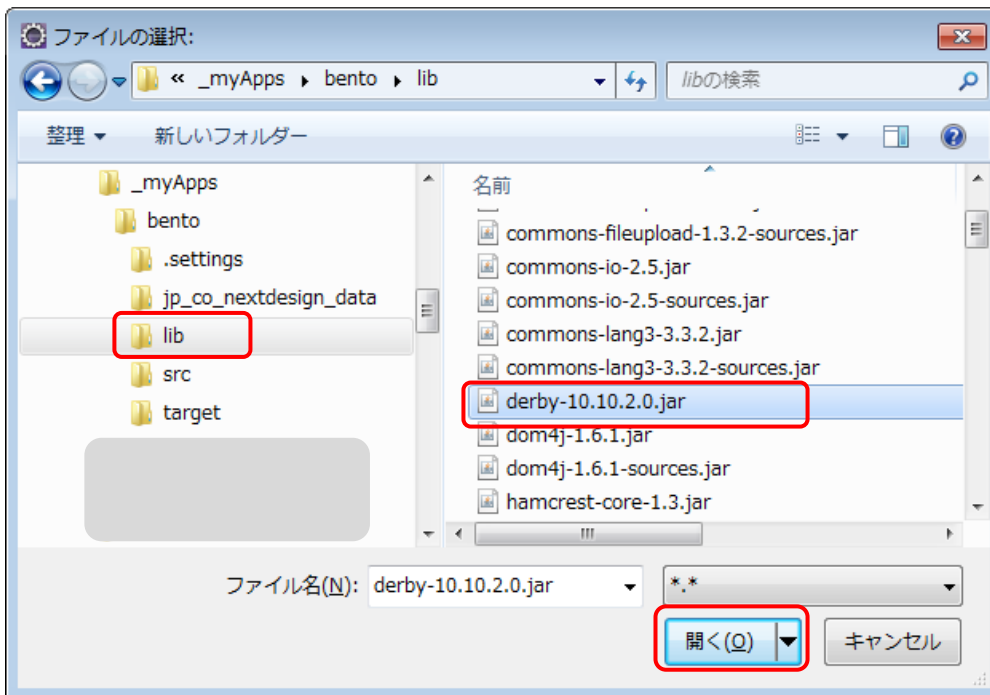
新規ドライバー定義します。次図の赤枠にあるボタンをクリックします。



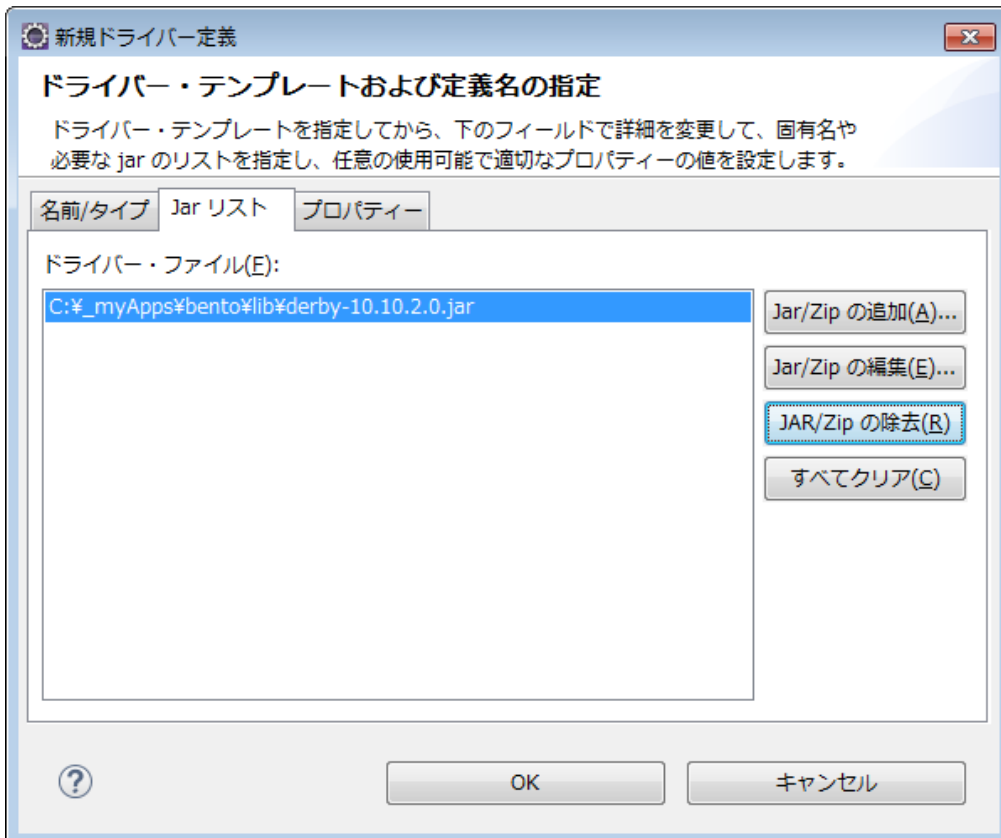
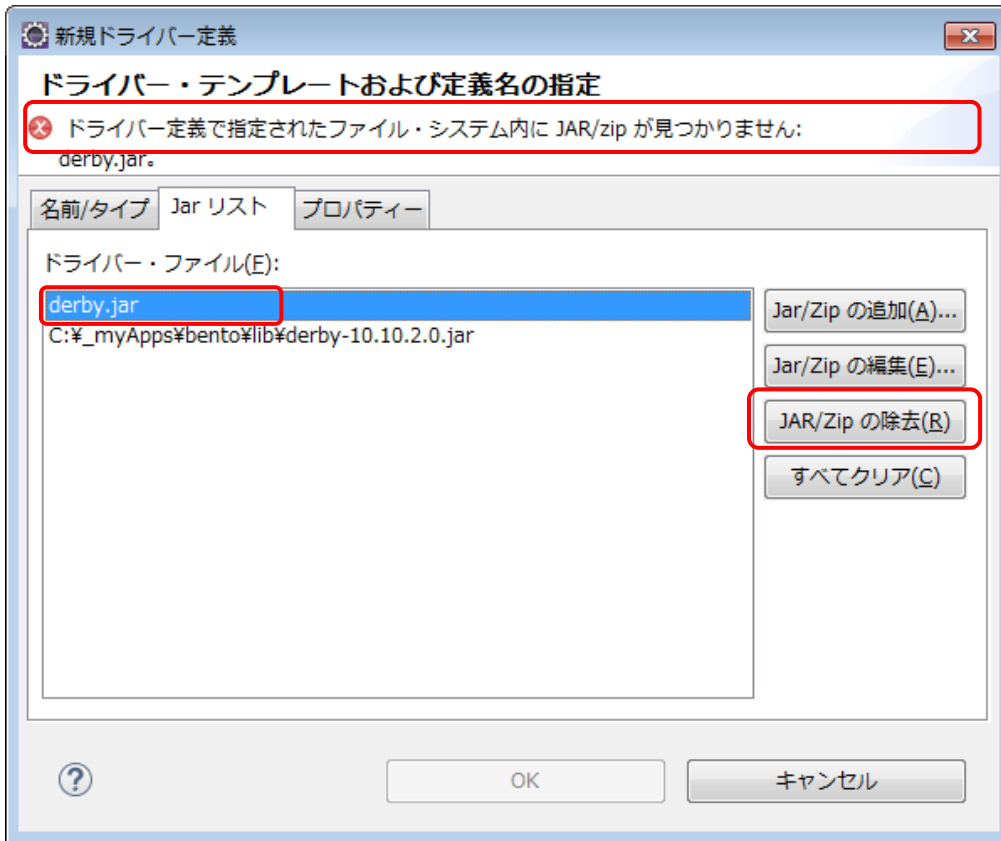
「Jar リスト」タブを選び、「Jar/Zip の追加」を押下します。



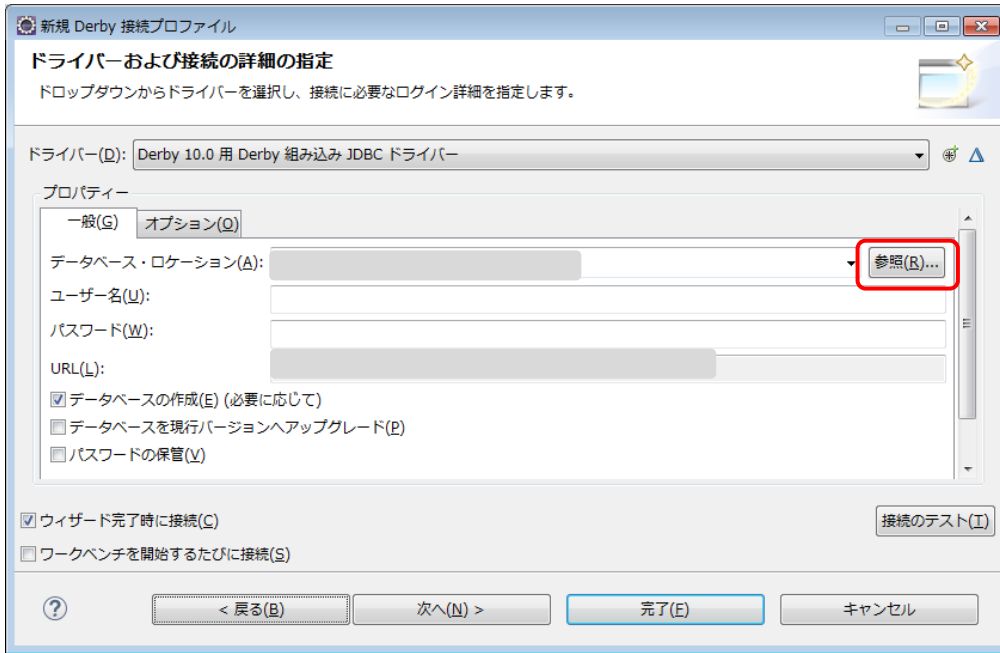
プロジェクト内の lib フォルダの derby-10.10.2.0.jar を指定します。



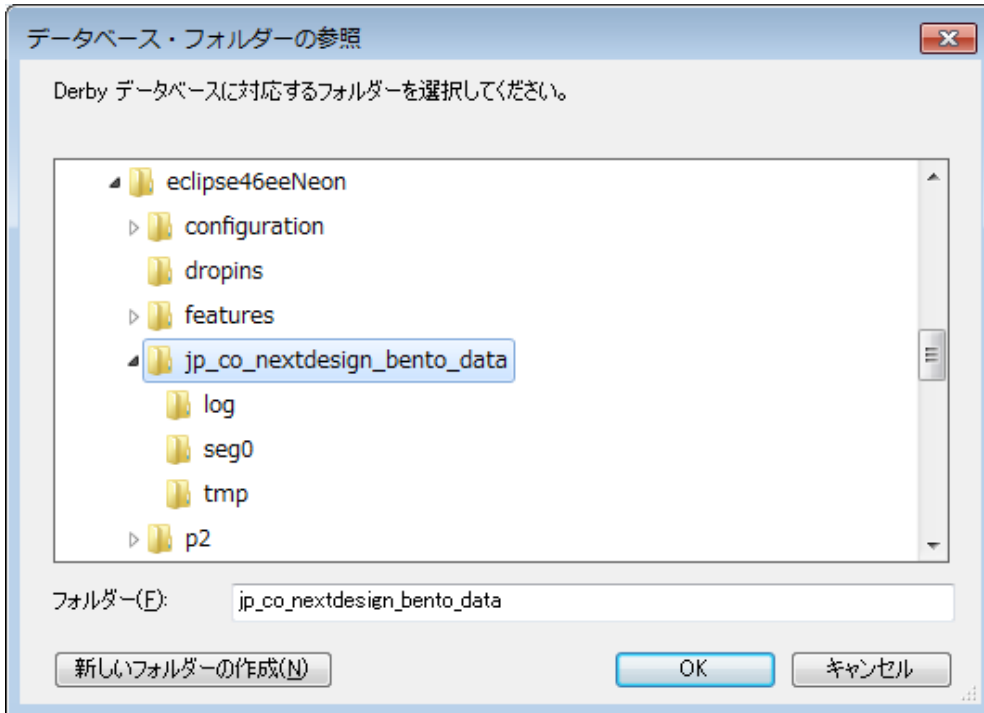
derby.jar は除去します。



OK

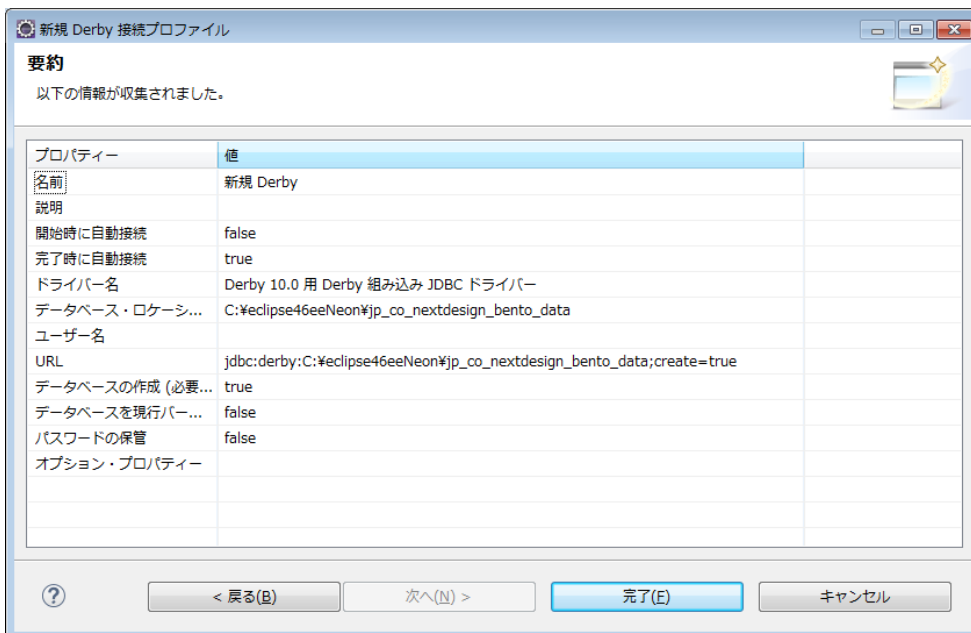


データベースロケーションとして、Eclipse フォルダ () 直下の `jp_co_nextdesign_bento_data` を指定いします。

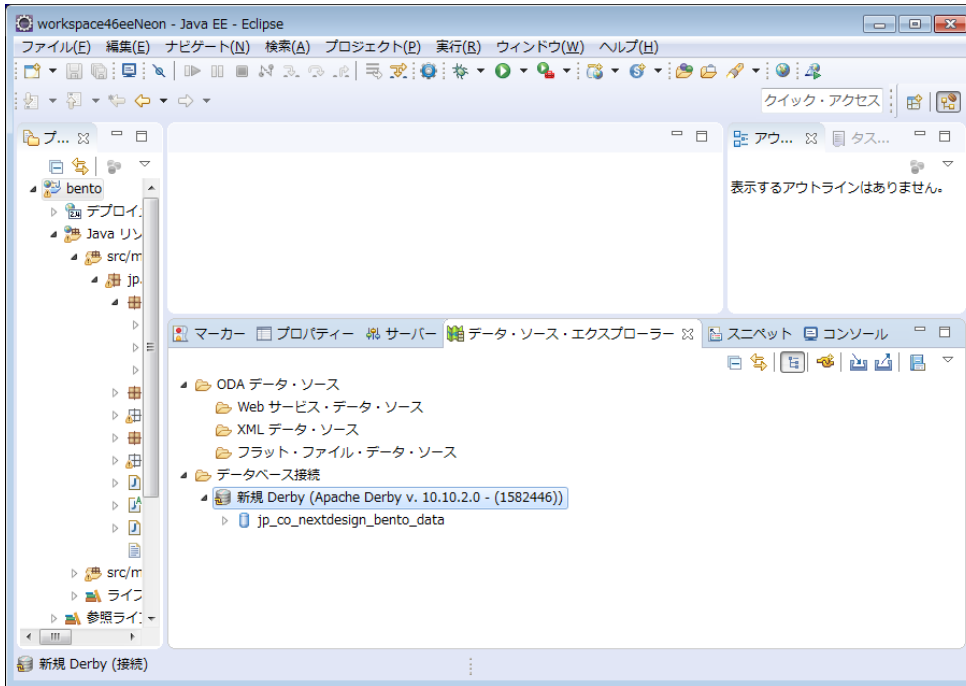




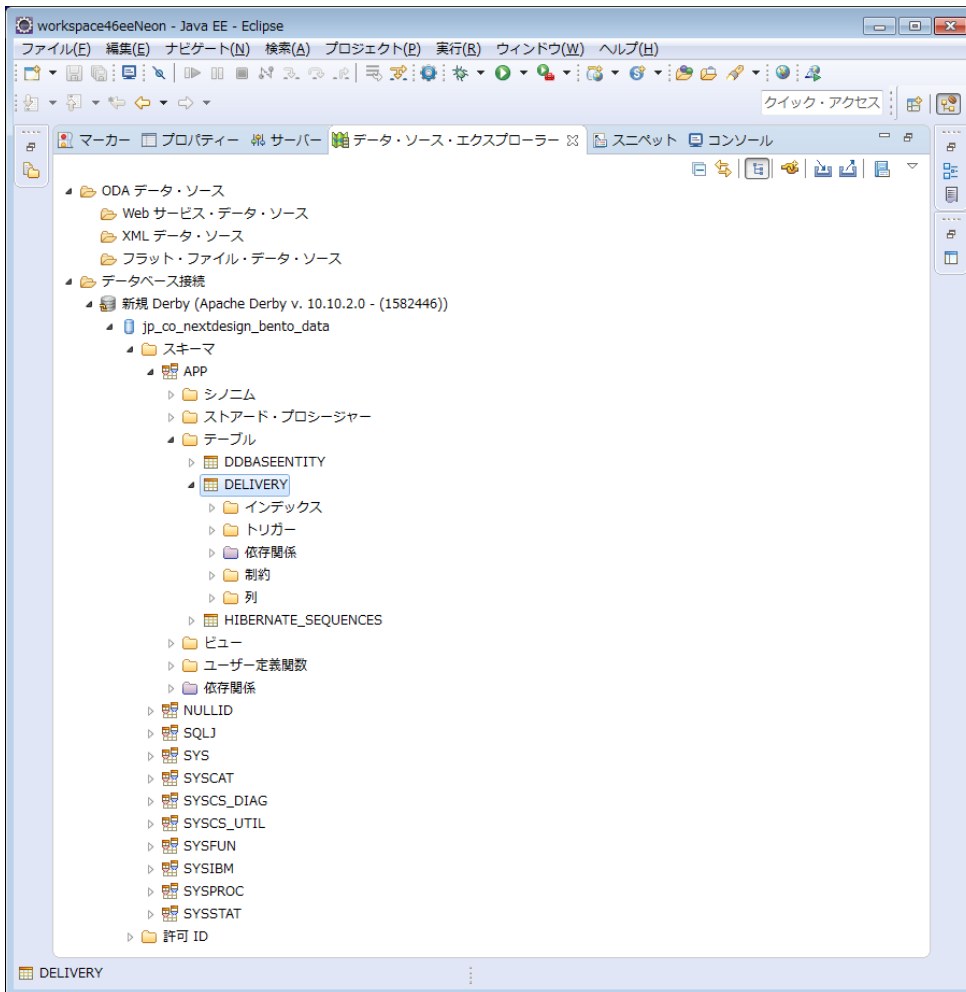
次へ



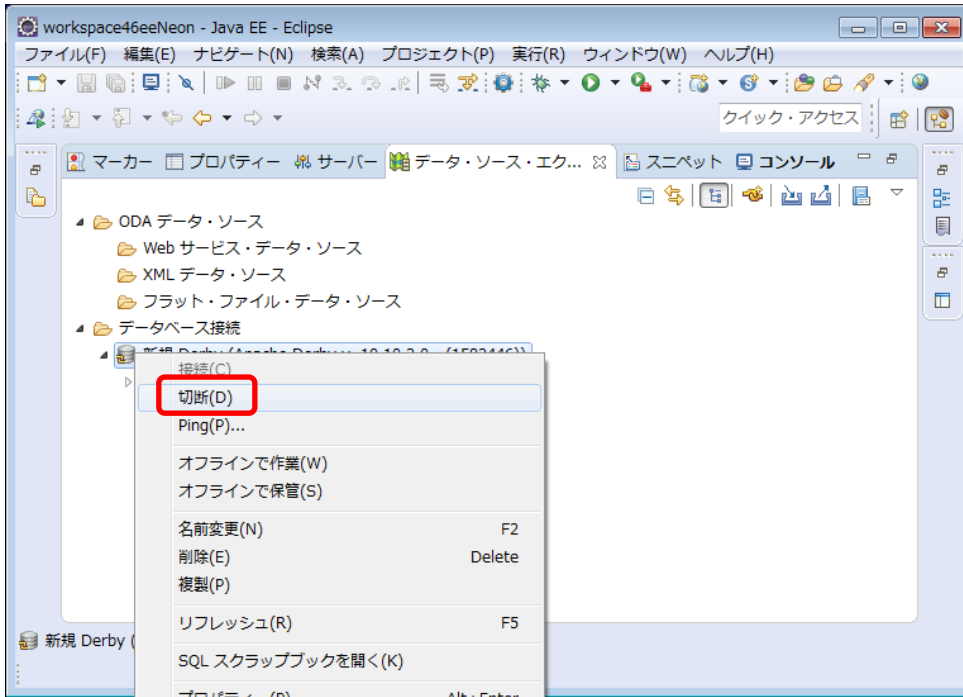
完了



展開すると次図のように確認できます。



最後に必ずデータベース接続を切断します。
接続したままで動作確認すると例外が発生します。



-終-