Seasar.NET入門

2007.5.27

Seasar.NET
杉本 和也

自己紹介

- 杉本 和也と申します
  - 高知県の株式会社アイビスに勤務しています
  - プログラミング歴 6年
  - オープンソース歴 2年
- Seasar.NETのリーダ
  - S2Container.NETのコミッタ
  - S2Dao.NETのリーダ

アジェンダ

- Seasar.NETについて
- S2Container.NETの使い方
- S2Dao.NETの使い方
- 業務ロジックのためのDI
- まとめ

Seasar.NETについて

- Seasar.NETについて
- S2Container.NETの使い方
- S2Dao.NETの使い方
- 業務ロジックのためのDI
- まとめ

Seasar.NETとは

- Seasarプロジェクトで.NET Frameworkに関するプロダクトを扱うプロジェクト

- Seasar.NETプロジェクトに参加するには
  - Sandboxプロジェクトにプロジェクト(プロダクト)を申請する
    - その後、Sandbox卒業したらSeasar.NETのプロダクトに
  - 既存のSeasar.NETプロダクトのコミッタになる
    - 希望者は連絡ください

Seasar.NETの目的

- 品質の高いソフトウェアを効率良く開発する為のプロダクトを開発・提供する

Seasar.NETの今

- S2Container.NET 1.2.9
  - AOPをサポートしたDIコンテナ
- S2Dao.NET 1.0.4
  - O/Rマッピングフレームワーク

S2Container.NETの使い方

Seasar.NETについて

S2Container.NETの使い方

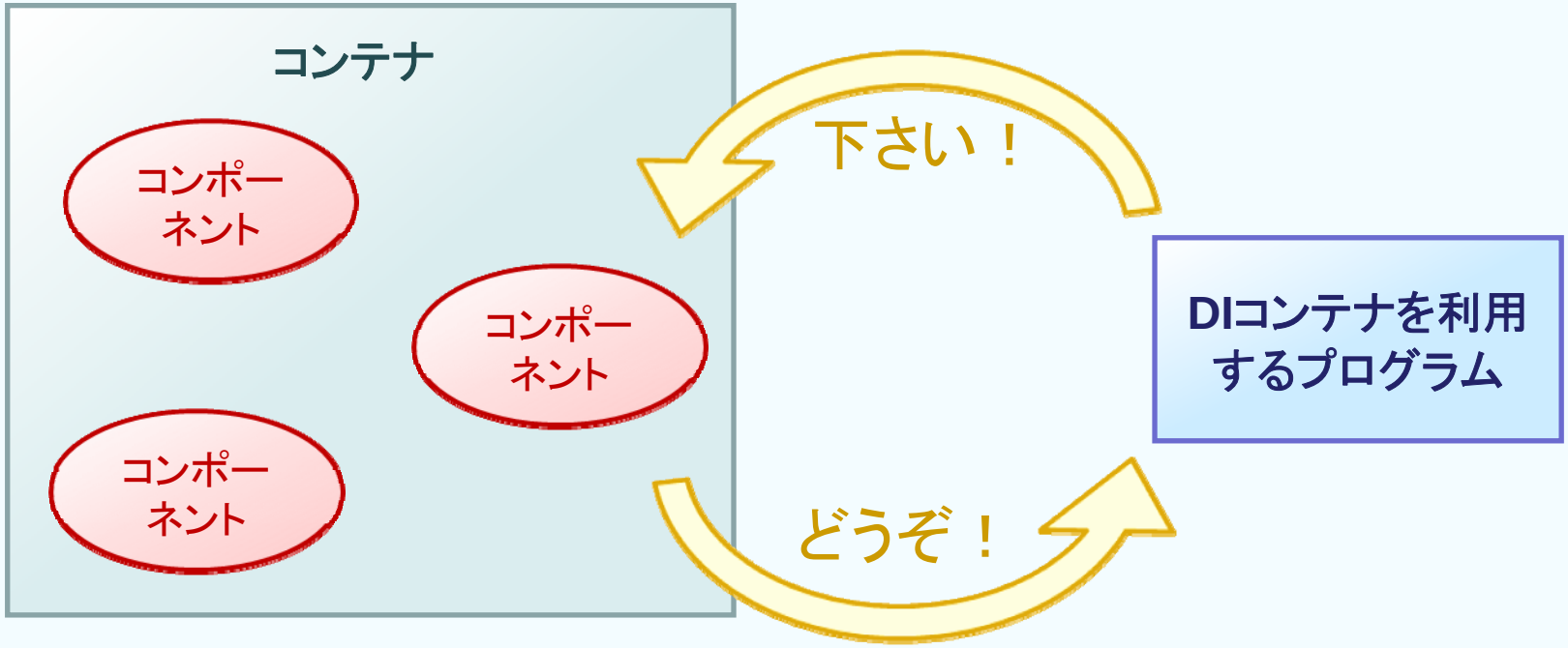
S2Dao.NETの使い方

業務ロジックのためのDI

まとめ

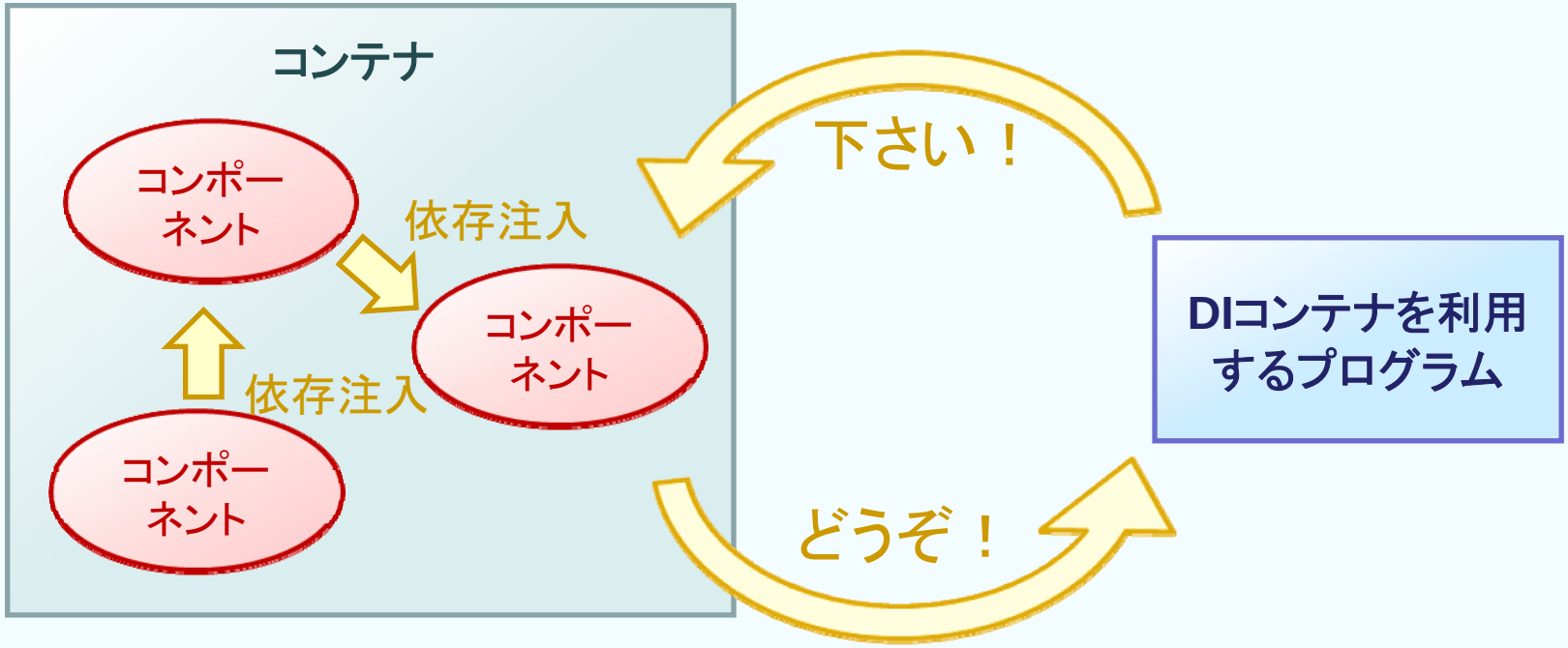
- AOPをサポートしたDIコンテナ
  - コンテナ
  - DI
  - AOP

- コンポーネントを格納する
  - クラスやインターフェース
- コンポーネントのインスタンスを管理する



- コンポーネントのインスタンスを管理する
- コンテナから利用側に「どうぞ！」するときにインスタンス化するかインスタンス化されたものを返す
- インスタンス化の種類（コンポーネントに設定できる）
  - singleton
    - 1度インスタンス化したら常に同じインスタンスを返す
  - prototype
    - 要求がある度に新しくインスタンス化したものを返す
  - request（ASP.NET連携時）
    - 同じrequest内であれば同じインスタンスを返す
  - session（ ASP.NET連携時）
    - 同じsession内であれば同じインスタンスを返す
  - outer
    - コンテナではインスタンスは管理しない

- DI (Dependency Injection)
  - 直訳すると依存注入
  - コンポーネントが別のコンポーネントを必要としていればセットしてくれる（具体的にはクラスのフィールドに）

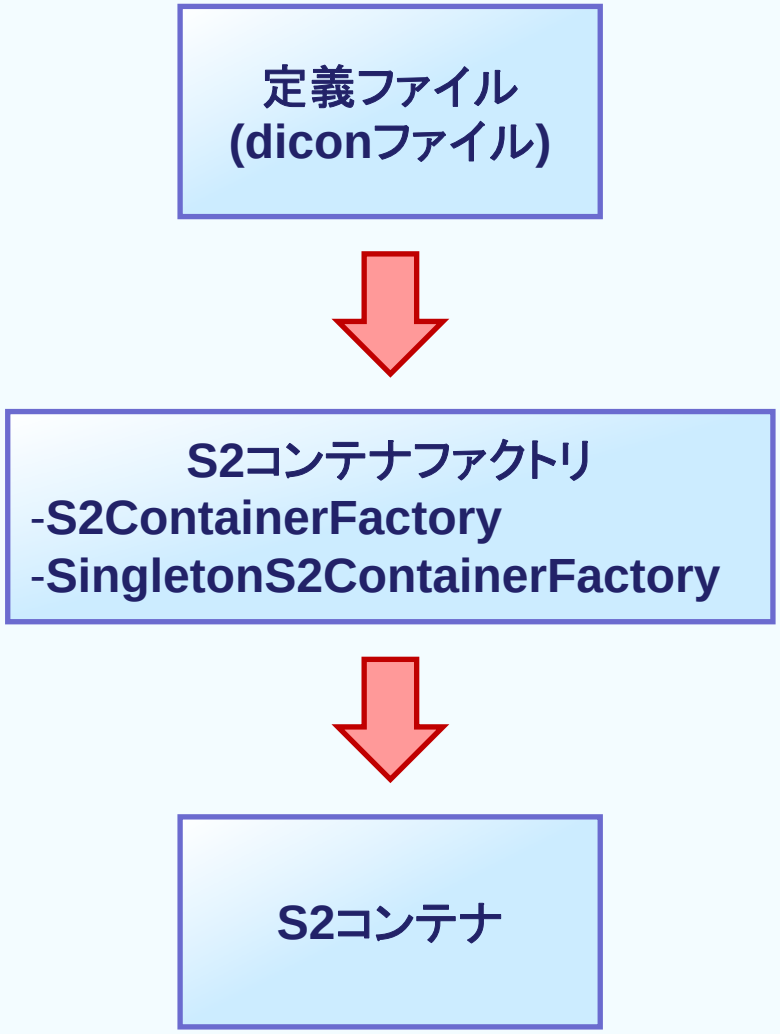


- DI (Dependency Injection)の種類
  - プロパティ・インジェクション
    - プロパティを使用しDIを行う
  - コンストラクタ・インジェクション
    - コンストラクタを使用してDIを行う
  - メソッド・インジェクション
    - メソッドを使用してDIを行う
- 自動バインディングを利用すると簡単

- S2Container.NETのコンテナをS2コンテナと呼ぶ

1. コンポーネントをS2コンテナに格納する
2. S2コンテナからコンポーネントを受け取る
3. コンポーネントを使う

S2Container.NET
コンポーネントの格納



1. diconファイルにコンポーネントの定義を記述する
2. S2コンテナファクトリにdiconファイルをセットする
3. S2コンテナファクトリからS2コンテナを作成する

Logic.dicon（埋め込まれたりソース）

```
<components>

   <component name="employeeLogic"
      class="EmployeeLogic" />

   <component class="DepartmentLogic" />

</components>
```

S2Container.NET
コンポーネントのサンプル

```
// 部署ロジックインターフェース
public interface IDepartmentLogic { ・・・ }
// 部署ロジック実装クラス
public class DepartmentLogic : IDepartmentLogic {・・・}

// 社員ロジックインターフェース
public interface IEmployeeLogic { ・・・ }
// 社員ロジック実装クラス
public class EmployeeLogic : IEmployeeLogic {
    private IDepartmentLogic departmentLogic;
    public IDepartmentLogic DepartmentLogic {
        set { departmentLogic = value; }
    }
     ・・・
}
```

S2Container.NET
コンポーネントを取得して使う

```
// S2コンテナファクトリに定義ファイルをセットする
SingletonS2ContainerFactory.ConfigPath = “Logic.dicon”;

// S2コンテナファクトリを初期化する
SingletonS2ContainerFactory.Init();

// S2コンテナファクトリからS2コンテナを作成する
IS2Container container = SingletonS2ContainerFactory.Container;

// S2コンテナからコンポーネントを取得する
IEmployeeLogic logic = (IEmployeeLogic)
   container.GetComponent(“employeeLogic”);
```

このlogicにはDepartmentLogicがセットされている

実際の使われ方

- Webアプリケーション(ASP.NET)
  - **S2HttpModuleが用意されており、そこでS2コンテナを扱う**
- Windowsアプリケーション等
  - スタートアップ部分で、自分でS2コンテナを扱う必要あり

AOP

- AOP (Aspect Oriented Programming)
  - アスペクト指向プログラミング
  - 本来の処理とは異なる処理をプログラムのソースコードに書かずに後から織り込みましょう！というプログラミング
  - ロギングやトランザクション処理等に使用される
  - S2Dao.NETのようなフレームワークを作成できる
    - インターフェースにAOPで実装を追加する

AOPを使わない場合

```
// 足し算を行うメソッド
int Plus(int x, int y) {
    int result = x + y;
    return result;
}
```


```
// 足し算を行うメソッド（ログ出力機能付き）
int Plus(int x, int y) {
    Console.WriteLine(“引数：” + x + “,” + y);
    int result = x + y;
    Console.WriteLine(“結果：” + result);
    return result;
}
```

- AOPを利用しない場合
  - コードの可動性が低下する
  - ロギングのような処理を追加したり取り除いたりする際にバグが混入する可能性がある

- →S2Container.NETのAOPの機能を利用して後から織り込もう！

S2Container.NET AOPの設定

```
<components>
    <component name="traceInterceptor"
    class="Seasar.Framework.Aop.Interfaces.TraceInt
    erceptor" />

    <component class="CulcLogic">
        <aspect>traceInterceptor</aspect>
    </component>
</components>
```

S2Dao.NETの使い方

Seasar.NETについて

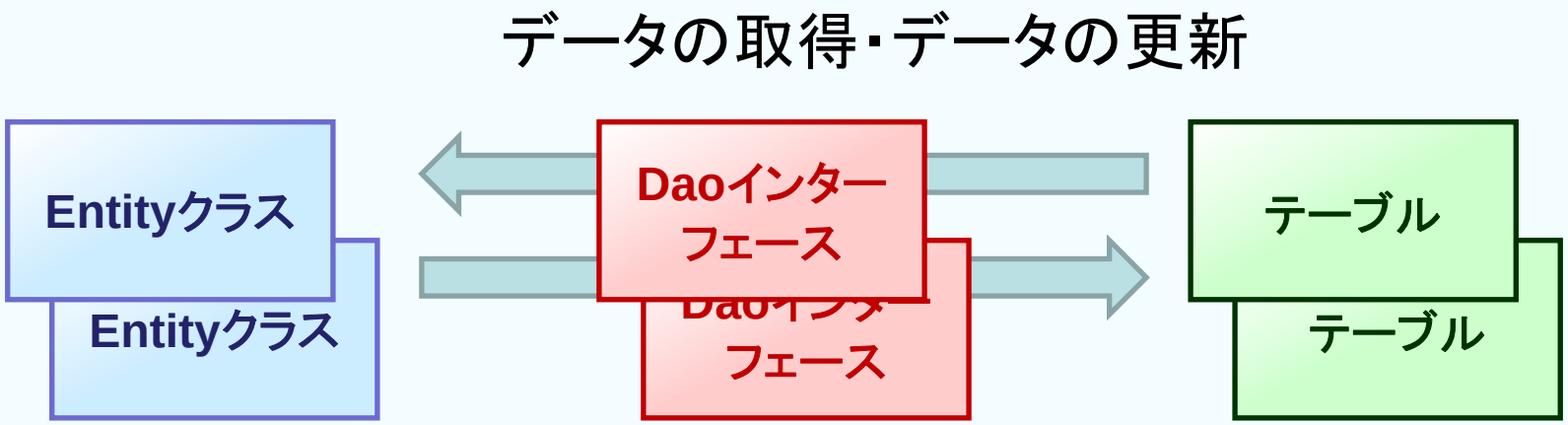
S2Container.NETの使い方

S2Dao.NETの使い方

業務ロジックのためのDI

まとめ

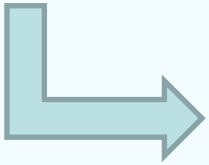
- S2Dao.NETはO/Rマッピングフレームワーク
  - マッピング情報はXMLに記述しない
- データベースへのSQL発行とオブジェクトへのマッピングを強力にサポート



1. テーブルに対応したEntityクラスを作る
2. Entityクラスに対応したDaoインターフェースを作る
3. 必要に応じてDaoインターフェースにメソッドを追加する
4. DiconファイルにDaoを登録する
5. DaoをロジッククラスにDIして使う

- 基本的にテーブルと1対1で作成する
  - 基本、テーブル名と同じ名前のクラス名にする
  - 基本、カラム名と同じ名前のプロパティ名にする

Employee（社員テーブル）	
PK	EmpID（社員ID）
U1	EmpCode（社員コード） EmpName（社員名）



```
public class Employee {
    private int empID;
    private int empCode;
    private string empName;

    public int EmpID
    {
        set { empID = value; }
        get { return empID; }
    }
    ...
```

Entityクラスに指定できる属性

- クラスに指定できる属性
  - Table属性 (テーブル名とクラス名が異なる場合に指定)
  - NoPersistentProps属性 (カラムとマッピングしないプロパティを指定)
  - VersionNo属性 , Timestamp属性 (排他制御を行うプロパティを指定)

- プロパティに指定できる属性
  - Column属性 (カラム名とプロパティ名が異なる場合に指定)
  - RelNo属性, RelKeys属性 (別テーブルとの結合を指定)
  - ID属性 (IDの自動生成を指定)

- Entityクラスと1対1でインターフェースを作成する
  - IEmployeeDao
- 発行するSQLと1対1でメソッドを追加する

- 更新系メソッド (メソッド名が下記で始まる)
  - Insert処理 (Insert, Add, Create)
  - Update処理 (Update, Modify, Store)
  - Delete処理 (Delete, Remove)
- 検索系メソッド
  - 更新系メソッド以外で戻り値の型を指定する

- 更新系メソッド
  - SQLを自動生成させる場合、引数はEntityクラス
  - SQLファイルや属性を使ってSQLをカスタマイズ
  - 戻り値の型はSystem.Int32かvoid
    - System.Int32であれば更新行数が戻り値
- 検索系メソッド
  - 戻り値の型がEntityクラスであれば1件分を取得
  - 戻り値の型がEntityクラスの配列, IList, IList<Entityクラス>であれば複数件を取得
  - 戻り値の型が上記以外であれば、1カラムの値を取得
  - 引数名からWHERE句を自動生成

Daoインターフェースサンプル

```
[Bean(typeof(Employee))]
public interface IEmployeeDao
{
    int InsertEmp(Employee emp);

    [Sql("delete from Employee where EmpCode=/*empCode*/")]
    void DeleteByEmpCode(int empCode);

    Employee GetByEmpCode(int empCode);

    [Query("order by EmpCode asc")]
    Employee[] GetAllEmployees();

    [Sql("select EmpName from Employee where EmpID=/*empID*/")]
    string GetEmpNameByEmpID(int empID);
}
```

Daoインターフェースに指定できる属性

- インターフェースに指定
  - Bean属性 (Entityクラスを指定する)
- メソッドに指定
  - Query属性 (Where句以降を指定)
  - Sql属性 (SQLをまるごと指定)
  - NoPersistentProps属性 (自動生成Update文で更新しないプロパティを指定する)
  - PersistentProps属性 (自動生成Update文で更新するプロパティを指定する)

- SQLをまるごと指定できる (SQL属性と同じ)
- Daoインターフェースと同じ名前空間に配置
- ビルドアクションプロパティを「埋め込まれたリソース」に設定
- ファイル名は”*インターフェース名_メソッド名*.sql”

IEmployeeDao_GetEmpNameByEmpID.sql

```
select
    EmpName
from
    Employee
where
    EmpID=/*empID*/3
```

- /*empID*/はバインド変数コメント
- バインド変数コメントの後ろにテスト用のダミーデータ
  - SQL発行ツールでSQLを実行するとEmpID=3という値でテストが実行できる

S2Dao.NET 設定

```
<!-- データプロバイダやDB接続文字列の設定はドキュメント参照 -->

<!-- S2Dao.NETのDaoInterceptorとそれに必要なコンポーネント -->
<component
    class="Seasar.Extension.ADO.Impl.BasicDataReaderFactory" />
<component
    class="Seasar.Extension.ADO.Impl.BasicCommandFactory" />
<component class="Seasar.Dao.Impl.DaoMetaDataFactoryImpl" />
<component name="DaoInterceptor"
           class="Seasar.Dao.Interceptors.S2DaoInterceptor"/>

<!-- 社員Dao -->
<component class="IEmployeeDao">
  <aspect>DaoInterceptor</aspect>
</component>
```

業務ロジックのためのDI

Seasar.NETについて

S2Container.NETの使い方

S2Dao.NETの使い方

業務ロジックのためのDI

まとめ

- ステートレスな業務ロジックの為の簡易DI機能
  - **Quill**
- 簡単DI機能
  - VSのツールボックスからFormにペタッと貼り付けるとDIが有効になる
  - FormのフィールドにDIすべきものがあればDIする
    - インターフェースの属性で実装クラスを指定する
    - 属性で実装クラスが指定されていればDIすべきとみなす
  - AOPは属性で指定する
  - S2コンテナと連携してS2コンテナのコンポーネントも扱える

- 制限された機能
  - DIのタイプはフィールド・インジェクションのみ
  - 扱うクラスには引数なしのコンストラクタが必要
  - インスタンスは管理しない(singletonのみ)
  - 1つのインターフェースに1つの実装クラスしか指定できない
- 簡単DIで小規模な開発でもDIの利用をしやすく
- Quillのソースコード量を最小限に保ち利用者が容易に確認できるようにする

① インタフェースを作る(Implementation属性で実装クラスを指定する)

```
[Implementation(typeof(CulcLogic)]
Public interface ICulcLogic
{
    int Plus(int x, int y);
}
```

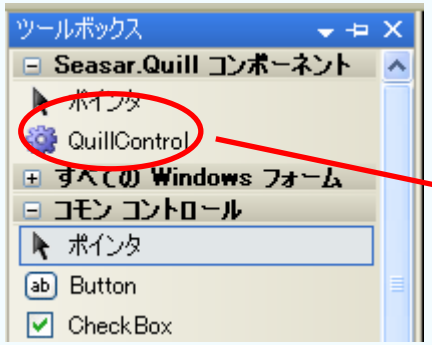
② 実装クラスを作る(Aspect属性でログ出力のAspectを適用する)

```
Public class CulcLogic : ICulcLogic
{
    [Aspect(typeof(ConsoleWriteInterceptor))]
    public int Plus(int x, int y)
    {
        Console.WriteLine( “Plusが呼ばれた” );
        return x + y;
    }
}
```

③ Formを作る(ICulcLogic型のフィールドを用意して使ってみる)

```
Public partial class Form1 : Form
{
    private ICulcLogic culcLogic = null;
    private void button1_Click(object sender, EventArgs s)
    {
        int ret = culcLogic.Plus(1, 2);
        Console.WriteLine(“戻り値：” + ret);
    }
}
```

④ デザイナからQuillControlをFormに貼り付ける



羽のアイコンはデフォルトで非表示なのでデザイナ上ではとりあえずじゃまにならないところに張っておく

⑤ 実行してボタンを押すとログが出力される
（DIとAOPが動いたことがわかる）

これだけで簡単DI＋AOP！

コンソールに出力されたログ

```
Start:Seasar.Quill.Examples.CulcLogic#Plus // (Interceptorによるログ)
Plusが呼ばれた
End  :Seasar.Quill.Examples.CulcLogic#Plus // (Interceptorによるログ)
戻り値：3
```

まとめ

Seasar.NETについて

S2Container.NETの使い方

S2Dao.NETの使い方

業務ロジックのためのDI

まとめ

まとめ

- S2Container.NET
  - 実装クラスに依存せずインターフェース経由でやりとりを行える
  - 変更、テスト、分業が行いやすい
- S2Dao.NET
  - マッピング情報をXMLに持たないので簡単に扱えることができ、劇的に生産性が向上する
  - マッピングミスやADO.NETのAPIの扱いのバグが無くなり品質が向上する
- Quill

ご静聴ありがとうございました

- S2Container.NET
  - http://s2container.net.seasar.org/
- S2Dao.NET
  - http://s2dao.net.seasar.org/

- sugimotokazuyaの日記
  - http://d.hatena.ne.jp/sugimotokazuya/