今さら聞けないDIコンテナ入門

～ 自作DIコンテナのすすめ ～

By 黒龍＠わんくま同盟

今さら聞けないDIコンテナ入門

- DIコンテナってどういうもの？
- どういった機能があるの？？
- で、どう便利なの？？？

DIコンテナってどういうもの？

- Dependency Injection(依存性の注入)という用語の頭文字をとったもの
- 軽量コンテナと呼ばれるいくつかの実装で採用されていたIoC(Inverse of Control、制御の逆転)と呼ばれていたコンポーネントの依存性解決の手段に名称を付けたものです

EJBではダメ？

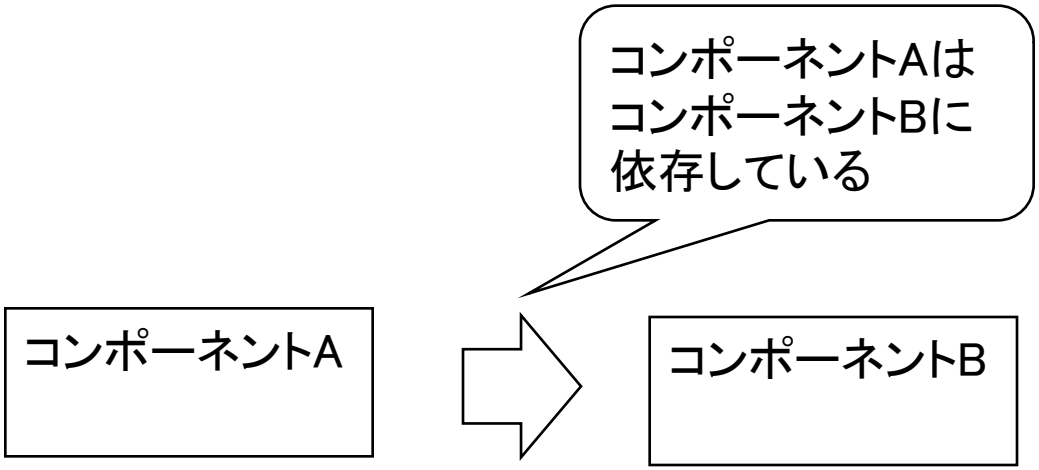
- コンポーネントとEJBコンテナが密につながっている（コンポーネントの再利用不可）
- 各種機能の実装が必須（敷居が高い）
- Deployが必須（テストがやりにくい）

軽量コンテナ

- コンテナに依存しないシンプルなコンポーネント（通常のオブジェクト）
- 実装必須なものはなし（一部のコンテナではあり）
- Deployは基本的には不要（コンテナ自体は依存せず）

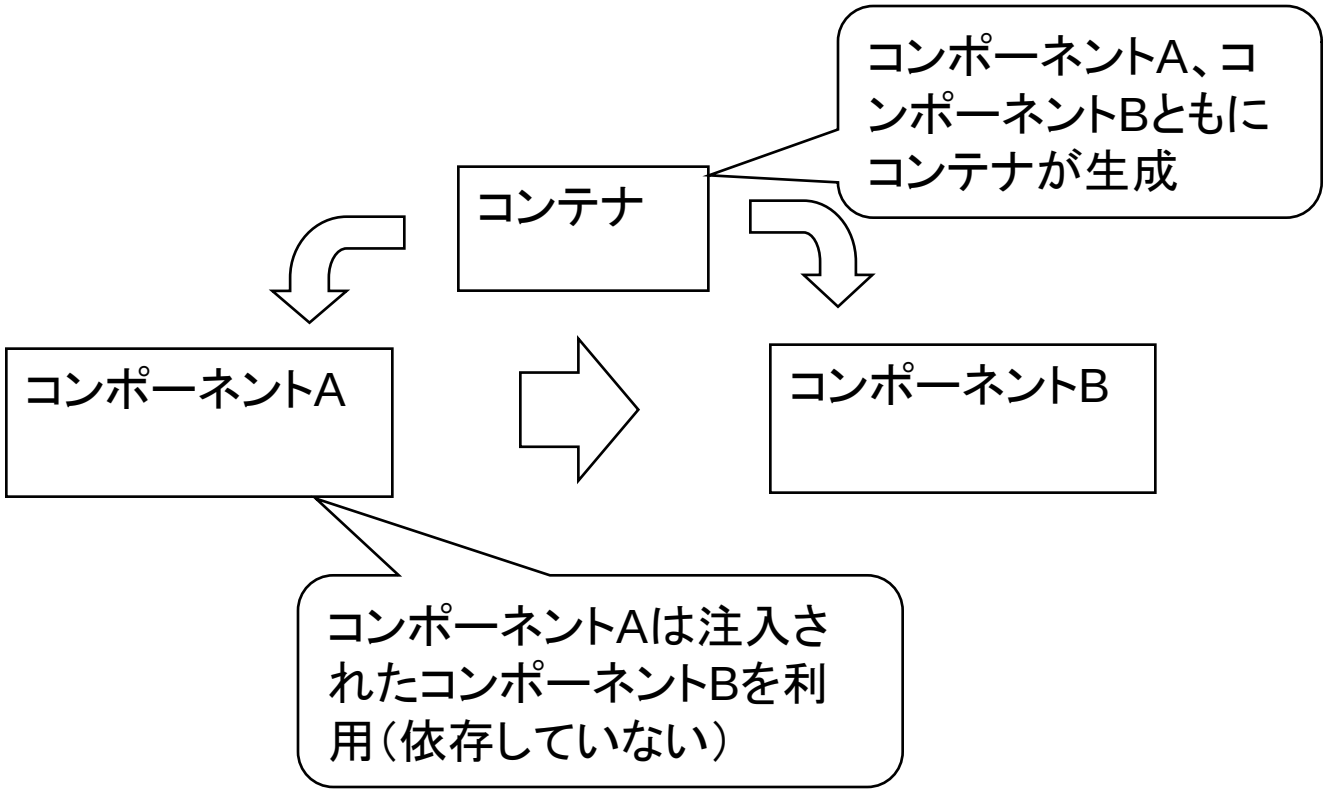
IoC(DI)って？？

- コンポーネントAがコンポーネントBを利用する場合（従来のケース）



IoC(DI)って？？

- コンポーネントBはコンテナが生成してコンポーネントAへ設定(制御の逆転)



JAVAだけの話？

- とっても汎用的な技術です
- EJBの分野での問題(複雑性)解決に大変効果を発揮します
- が、.Netも複雑ですよね

代表的な機能として

- オブジェクトの生成と管理（オブジェクトファクトリ＋コンテナ）
- 依存性の注入（Dependency Injection）
- などがあります

これって割とよくある概念です

- オブジェクトファクトリ＋コンテナ
  - コネクション
    - コネクションオブジェクトを生成、管理する
  - スレッド
    - スレッドオブジェクトを生成、管理する
- 等々すでによく使われています

これって割とよくある概念です

- 依存性の注入、設定とコードの分離
  - リモーティング
    - 接続先、種別、セキュリティの有無
  - コネクションストリング
    - 接続先、種別、セキュリティの有無
  - 動的プロパティ
    - 対応する様々なプロパティ
- 等々こちらも普段利用しています

設定値はどのように分離？

- 多くの実装ではXML形式のファイルを採用
- コードじゃダメなの？
- Demoで順を追ってみてみましょう

いたって普通に実装した例

- フォームからクラスを生成して呼び出す
  - TestClass1 testObj = new TestClass1();
  - MessageBox.Show(testObj.AddString(textBox1.Text,textBox2.Text));
- Demo

いたって普通に実装した例

- 問題点
  - 必ず実装が必要
  - テスト時に生成箇所、使用箇所の変更が必要

ひとまずInterfaceを使いましょう

- インタフェースを使ってみる
  - ITestClass testObj = new TestClass1();
  - MessageBox.Show(testObj.AddString(textBox 1.Text, textBox2.Text));
- Demo

ひとまずInterfaceを使いましょう

- 改善点
  - 実装に依存しない
  - 使用箇所は変更せずに差し替え可能
  - ファクトリを使えば生成箇所も変更不要
- 問題点
  - やっぱりコードの変更が必要
  - リコンパイルも必要

で、DIコンテナ

- DIコンテナを使ってみる
  - XMLで設定（App.Configに記載）
  - オブジェクトはコンテナから取得
    - ITestClass testClass = (ITestClass)DIContainer.DIContainer.instance.GetControl(typeof(ITestClass));
- Demo

で、DIコンテナ

- 改善点
  - ファクトリ（DIコンテナ）が実装（クラス）に依存しないので実装不要
  - テスト時も容易にモックに切り替え可能
  - リコンパイル不要でクラスの切り替えが可能
- 並行開発の加速！！
- デメリット
  - 最適化に若干影響など

で、DIコンテナ

- 忘れずにDIを見てみましょう
- 設定ファイルで依存性（関連）を記述
  - <add name="test3" type="TestClass3.DITestClass,TestClass3" propertys="Component" />
    - 今回はプロパティにインジェクションしてみました
    - プロパティ名から型はリフレクションで解決

自作のススメ

- 今回は
  - XMLで設定
  - オブジェクトは型から自動判別
- ほかに見られる実装
  - 名前で紐付け
  - XMLでの設定
  - 型から判別などなど

自作のススメ

- 例えば
  - データベースで管理
  - 属性を利用して設定レスに
  - 型での解決をメインにして設定コードレス
  - などなど
- 設計に関わるため既製品ではいまいち
- 自分で作れば自由にできます

AOPとDIの秘密の関係

- AOPって？
  - アスペクト指向プログラミングといって横断的関心事を分離する設計＆実装の手法です
  - 横断的関心事って？？
    - ログ出力
    - トランザクション参加
    - 認証、承認
    - などなど

AOPとDIの秘密の関係

- .NETは実はやりやすいんです
  - Proxyの仕組みがある
  - 動的コンパイルも可能
- Demo

AOPとDIの秘密の関係

- いろいろな実装例
  - ContextBoundsObjectと属性を用いる方法
    - Newに割り込めるのがメリット
    - ContextBoundsObjectの継承がデメリット
  - RealProxyを用いる方法
    - Newに割り込めないのでファクトリで生成が必要
    - MarshalByRefObjectの継承でよい（対象が多い）
  - 動的コンパイルによる派生クラス生成
    - 型が一致しなくなる
    - 速度的なデメリットがない

AOPとDIの秘密の関係

- 他にもいろいろありますが基本的には実際の型の代わりにダミーを渡す仕組みなためDIコンテナと組み合わせることが多いです
  - Seasar
  - Spring
  - などなど

AOP+DIコンテナで何ができるか？

- いろいろ出来ます
- 例えば…
  - キャッシュ
  - 例外のマッピング（DB系Exceptionからアプリ例外に）
  - エラー表示
  - ユーザの偽装
  - などなど

DEMO

- 属性を用いたAOP＋DIコンテナの例

作ってみたくなりました？？

ありがとうございました