Lucyユーザーガイド

Lucy - the lightweight DI container -

最近のJavaEE標準やフレームワークは本当に多種多様のニーズにこたえています。
しかし、本当にわたしたちアプリケーション開発者は楽になっているでしょうか？
JavaEE標準やフレームワークに振り回されてしまって、自分たちが本当に作らなくてはいけないものにフォーカスできていなかったりしませんか？
Javaアプリケーションの開発は、本当はもっともっとシンプルで楽にできるはずです。

そういう思いを形にした、小さくて扱いやすいフレームワーク、それがLucyです。
Lucyは軽量コンテナ(DIコンテナ)と呼ばれるフレームワークです。その中でも軽量を目指すため、軽量DIコンテナと私たちは呼んでいます。
「アプリケーション開発をもっと良く、もっと楽に。」
Lucyはそのために存在します。

目次

Lucy概要

Lucyとは何か

Lucyは軽量コンテナと呼ばれる部類のフレームワークです。
主にオブジェクトの生成と管理を行うためのフレームワークで、DIコンテナと呼ばれます。
オブジェクトの生成と管理をアプリケーション側でしてしまうと、生成するタイミングやそのライフサイクルを
管理できなくなってしまい、コードはメンテナンスしにくくなります。これを解消するためのフレームワークの一種がLucyです。
DIとはDependency Injectionの略で、その意味は下記文献を参照してください。

かくたにさんによる日本語の翻訳
http://kakutani.com/trans/fowler/injection.html

Martin Fowlerによる原書
http://www.martinfowler.com/articles/injection.html

このDIを行うためのフレームワークの軽量なものがLucyです。

Lucyの特徴

Lucyは以下のような特徴を持っています.

- アノテーションまたは設定ファイルによるインジェクション
- 型によるインジェクトを優先する
- JavaBeansにこだわらない自由なmethodインジェクト
- 小さなコア機能と宣言的な拡張機能の提供
- 明確なライフサイクル

稼動環境

JDKバージョン

Lucyでは**JDK1.6**以上必須です.

JavaEEの依存している仕様

LucyはJavaEEの仕様には依存していません.

Lucyの依存しているフレームワーク**/**ライブラリ

必須ライブラリ

T2は以下のフレームワーク/ライブラリが必須です.

- commons：Lucyの共通ライブラリ
- slf4j-api：ロギング用インタフェース(実際のロギングライブラリは選択可能)
- mvel2：式言語を解釈するためのライブラリ
- javassist：AOPを実現するためのライブラリ

オプションライブラリ

以下のフレームワーク/ライブラリはオプションですが必要になる場合があります.

- logback-core：ロギングのコアライブラリ
- logback-classic：slf4j-apiの実装ロギングライブラリ
- jxl: Excelを読み込むためのライブラリ
- junit：テスト用ライブラリ

Lucyの機能

ディペンデンシーインジェクション**(DI)**機能

Lucyはアノテーションまたは設定ファイルによるDI機能を持っています。
LucyのDI機能はどちらも明示的にインジェクトするポイントを指定するようになっています。
インジェクションは、JavaBeansプロパティまたはメソッドによって実行されます。

アノテーションによるインジェクション機能

早速アノテーションによるインジェクトの例をみてみましょう。
覚えるべきアノテーションは、ただ一つ、**@Inject**だけです。
@Injectはインジェクトするポイントを示すアノテーションでメソッドにつける事が出来ます。

シンプルな例でみてみましょう。以下の例では、ClientにGreetingインスタンスを設定しています。
@Injectアノテーションのついている、injectメソッドに注目してください。
（このサンプルのすべてのソースコードはlucy-examplesのlucy.examples.example1 パッケージにあります）

```
package lucy.examples.example1;

import lucy.annotation.core.Inject;
import commons.annotation.composite.PrototypeScope;

/**
 * {@.en Simple client class for greeting. }
 *
 * {@.ja Greetingクラスを単純に呼ぶクライアントクラスです。 }
 *
 * @author shot
 *
 */
/**
 * {@.en This class is annotated with @PrototypeScope that means this
instance
 * is created by everytime getting through Lucy. }
 *
 * {@.ja このクラスは@PrototypeScopeがつけられているので、Lucyから取得されるたびに毎回インスタンスが作られます。 }
 */
@PrototypeScope
public class Client {

    protected Greeting greeting;

    /**
     * {@.en Simply print out greeting.hello(). }
     *
     * {@.ja 単純にGreeting.hello()の結果を出力します。 }
     */
    public void sayHello() {
        System.out.println(this.greeting.hello());
    }

    /**
     * {@.en @Inject means this is a injection point for Lucy.You clearly
see
     * this, right:)? }
     *
     * {@.ja @Injectで、Lucyによるインジェクトポイントを示します。
     * ここではGreetingインタフェースの実装クラスがインジェクトされます。 }
     */
    @Inject
    public void inject(Greeting greeting) {
        this.greeting = greeting;
    }
}
```

また、複数インジェクトする場合はどうするのでしょうか。
複数@Injectを記述してもかまいませんが、どうせであればまとめてインジェクトしたいところでしょう。
Lucyは1つのメソッドで複数同時のインジェクトをすることが出来ます。
下記の例のようになります。injectメソッドをみてください。Messageインタフェースの実装クラスと、Printerインタフェースの実装クラスが

同じメソッドでインジェクトされるので、非常にシンプルにインジェクトするポイントがまとまっており見やすいです。
(このサンプルのすべてのソースコードはlucy-examplesのlucy.examples.example2 パッケージにあります)

```
package lucy.examples.example2;

import commons.annotation.composite.SingletonScope;

import lucy.annotation.core.Inject;

/**
 * {@.en Simple client class for showing how to use multiple injects with
 * @Inject. }
 *
 * {@.ja @Inject を使った複数のインジェクションのサンプルです. }
 *
 * @author shot
 */
@SingletonScope
public class Client2 {

    protected Message message;

    protected Printer printer;

    public void execute() {
        printer.print(message.getMessage());
    }

    /**
     * {@.en Multiple injection. }
     *
     * {@.ja MessageクラスとPrinterクラスをインジェクションします. }
     *
     * @param message
     * @param printer
     */
    @Inject
    public void inject(Message message, Printer printer) {
        this.message = message;
        this.printer = printer;
    }
}
```

設定ファイルによるインジェクション機能

Lucyの設定ファイルによるインジェクションもアノテーションのそれと同じように簡単です。
@Injectの代わりに**inject**タグでインジェクトポイントを指定します。

設定ファイルは下記のようになります。injectタグの部分に注目してください。
injectタグには2つ設定すべきポイントがあります。

- method属性でインジェクトに使うメソッドを指定する。
- ボディ部を使って、設定するコンポーネントや文字列などを指定する。例えば以下の例なら、**"HEADER :"**などがそれにあたる。

(このサンプルのすべてのソースコードは、lucy-examplesのlucy.examples.example3 にあります。)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lucy-config PUBLIC "-//LUCY//DTD LUCY//EN"
"http://www.lucy.org/dtd/lucy-config.dtd">
<lucy-config>

    <!--
        <ja>コンポーネントタグ(<component>)はコンポーネントを定義します.
            クラス、インスタンス属性、コンポーネント名などを定義します.</ja>
    -->
    <component name="sender"
class="lucy.examples.example3.impl.MessageSenderImpl"
instance="singleton">

        <!--
            <ja>インジェクトタグ(<inject>)はインジェクトポイントを定義します.
method属性でインジェクションに使うメソッドを定義します.
                インジェクトする対象はinjectタグのボディ部に記述します.
                以下の例では、setHeaderメソッドで文字列をインジェクトしていま
す.</ja>
        -->
        <inject method="setHeader">"HEADER :"</inject>
    </component>

    <component class="lucy.examples.example3.Client3"
instance="singleton">

        <!--
            <ja>nameをつけたコンポーネントをそのままインジェクトすることも出来ます.
                以下の例では、senderという名前のついた、
lucy.examples.example3.impl.MessageSenderImplを
                インジェクトしています.</ja>
        -->
        <inject method="inject">sender</inject>
    </component>
</lucy-config>
```

コンポーネントのライフサイクル管理機能

DIコンテナの持つべき機能として、コンポーネントのライフサイクル管理があります。
スコープはアノテーションまたは設定ファイルによって定義します。
Lucyはデフォルトではシングルトンスコープ、プロトタイプスコープの2つしかありません。
しかし、スコープを外部から簡単に差し込めるようにしています。

例えばWebアプリケーションではリクエストスコープ、セッションスコープなどを新たに外部から定義することが出来ます。

シングルトンスコープ

シングルトンスコープは、Lucyから取得するたびに毎回同じインスタンスが返ってくるスコープです。
アノテーションでは**@SingletonScope**、設定ファイルではcomponentタグの**instance="singleton"**で表現されます。

アノテーションの場合、下記のようになります。

```
@SingletonScope
public class GreetingImpl implements Greeting {

    /**
     * {@.en hello(). }
     *
     * {@.ja helloメソッドです.  }
     */
    public String hello() {
        return "Hello Lucy!";
    }

}
```

設定ファイルの場合、下記のようになります。

```
<lucy-config>
    <component name="sender"
class="lucy.examples.example3.impl.MessageSenderImpl"
instance="singleton">
</lucy-config>
```

プロトタイプスコープ

プロトタイプスコープは、Lucyから取得するたびに毎回新しいインスタンスが生成されるスコープです。
アノテーションでは@PrototypeScope、設定ファイルではinstance="prototype"で表現されます。

アノテーションの場合、下記のようになります。

```
@PrototypeScope
public class Client {

    protected Greeting greeting;

    /**
     * {@.en Simply print out greeting.hello(). }
     *
     * {@.ja 単純にGreeting.hello()の結果を出力します.  }
     */
    public void sayHello() {
        System.out.println(this.greeting.hello());
    }

    /**
     * {@.en @Inject means this is a injection point for Lucy.You clearly
see
     * this, right:)? }
     *
     * {@.ja @Injectで、Lucyによるインジェクトポイントを示します.
```

```
    * ここではGreetingインタフェースの実装クラスがインジェクトされます.
    * Clientクラスではインタフェースしか意識していない点を注目してください. }
    */
   @Inject
   public void inject(Greeting greeting) {
      this.greeting = greeting;
   }
}
```

設定ファイルの場合、下記のようになります。

```
<lucy-config>
   <component name="sender" class="hoge.foo.bar.impl.SampleImpl"
instance="prototype">
</lucy-config>
```

そのほかのスコープについて

Lucyではスコープを拡張することが出来ます。例えばT2では以下のようなスコープを外部からLucyに足しこんでいます。

- リクエストスコープ：HttpRequestと同じライフサイクルを持つスコープ
- セッションスコープ：HttpSessionと同じライフサイクルを持つスコープ
- アプリケーションスコープ：ServletContextと同じライフサイクルを持つスコープ

@RequestScope/@SessionScope/@ApplicationScopeを
外部で定義してLucyにConfigurationクラスで渡しています。

AOP機能

LucyはAOP機能を持っています。AOPとはAspect Oriented Programming（アスペクトオリエンテッドプログラミング）の略で、
横断的な関心事をアスペクトとして定義することで、非機能要件を機能要件から完全に分離させることのできることが特徴的です。
簡単に言えば、ロギングやトランザクションなどの業務アプリケーションとして必須処理だが面倒で煩雑なものを業務ロジックから分離させることが出来ます。

アノテーションによるアスペクト

まずはアノテーションによるアスペクトを仕掛けてみましょう。アノテーションでは覚えるべきアノテーションは一つだけ、**@Aspect**です。
@Aspectの属性値で下記の項目を設定します。

設定メソッド	必須/オプション	仕様	デフォルト値
interceptBy	必須	処理をインタセプトする実体であるInterceptorの実装クラスを指定する	なし
pointcut	オプション	どのメソッドにアスペクトを適用するかを記述する	全てのpublicメソッドに適用する

最も簡単なサンプルは以下のようになります。以下のサンプルではexecuteメソッドに対して、SysoutInterceptorが処理をインタセプトするようになります。
(このサンプルのすべてのソースコードは、lucy-examplesのlucy.examples.example4 にあります。)

```
package lucy.examples.example4.impl;

import commons.annotation.composite.SingletonScope;
import commons.annotation.core.Aspect;

import lucy.examples.example4.ClientTarget;
import lucy.examples.example4.SysoutInterceptor;

/**
 * {@.en  }
 *
 * <br />
 *
 * {@.ja アスペクトをかけられる対象クラスです. @Aspectによってインタセプタを指定しています.
 * @Aspect のinterceptBy属性で設定するインタセプタのクラスを指定します.
 * また、pointcut属性で、アスペクトを仕掛けるメソッドを正規表現で指定します.
 * }
 * @author shot
 *
 */
@SingletonScope
@Aspect(interceptBy = SysoutInterceptor.class, pointcut =
"execute")
public class ClientTargetImpl implements ClientTarget {

   /**
    * {@.en  }
    *
    * <br />
    *
    * {@.ja サンプルの実行メソッドです. }
    */
   public void execute(String str) {
      System.out.println(str);
   }

}
```

設定ファイルによるアスペクト

アノテーション同様のことを設定ファイルでも可能です。
@Aspectの代わりに**aspect**タグで処理を割り込ませることが出来ます。設定ファイルは下記のようになります。
インタセプタも通常通りコンポーネントとして定義しておき、name属性をつけておきます(青字の部分)。AOPをかけるほうにはaspectタグで、
先ほどname属性指定したコンポーネントの名前をボディ部に記載します。

(このサンプルのすべてのソースコードは、lucy-examplesのlucy.examples.example5 にあります。)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lucy-config PUBLIC "-//LUCY//DTD LUCY//EN"
"http://www.lucy.org/dtd/lucy-config.dtd">
<lucy-config>

    <component name="sample1"
class="lucy.examples.example5.SysoutInterceptor"
instance="singleton">
    </component>

    <component class="lucy.examples.example5.impl.ClientTarget2Impl"
instance="singleton">
        <aspect pointcut="execute">sample1</aspect>
    </component>
</lucy-config>
```

アノテーションの仕様

この章では、Lucyのアノテーションの仕様について学びます.

@Inject

@Injectはインジェクト先を指定するアノテーションです。アノテーション指定での場合、@Injectは必須です。
特に指定が無い場合、**@Inject**ではタイプによるインジェクトを試みます。

属性の仕様は下記のとおりです。

属性名	必須/オプション	仕様	デフォルト値
name	オプション	インジェクトを名前ベースで行う(※1)	なし
types	オプション	インジェクトの型を補う	なし

name属性はコンポーネントを名前で検索したい場合に使います。
types属性は複数の同一インタフェースを持つコンポーネントを並べてインジェクトしたい場合などの補足情報として使います。

※1:名前で探したコンポーネントが無い場合、Lucyがコンポーネントが無い場合の挙動に従う(デフォルトの場合、nullが戻る)

//TODO Lucy-examplesへのサンプルの追加と説明

@InitMethod

@InitMethodはコンポーネントが作成されてから、初期化するためのメソッドを指定できます。
引数を取ることは出来ません。

//TODO Lucy-examplesへのサンプルの追加と説明

@DestroyMethod

@DestroyMethodはコンポーネントが破棄されるタイミングで呼ばれるメソッドを指定できます。
引数を取ることは出来ません。

//TODO Lucy-examplesへのサンプルの追加と説明

@SingletonScope

@SingletonScopeはコンポーネントをシングルトンであると明記するためのアノテーションです。
このアノテーションがつけられている場合、Lucyから何回取得されても同じインスタンスが返ります。

@PrototypeScope

@PrototypeScopeはコンポーネントをプロトタイプであると明記するためのアノテーションです。
このアノテーションがつけられている場合、Lucyから取得される度に新しいインスタンスが返ります。

@Aspect

@AspectはAOPのためのアノテーションです。
属性値に下記の項目を設定します。

設定メソッド	必須/オプション	仕様	デフォルト値
interceptBy	必須	処理をインタセプトする実体であるInterceptorの実装クラスを指定する	なし
pointcut	オプション	どのメソッドにアスペクトを適用するかを記述する	全てのpublicメソッドに適用する

設定ファイルの仕様

Lucyの設定はアノテーションだけでなく、XMLに宣言的に記述することも出来ます。

lucy-configタグ
Lucy用の設定ファイルの親タグです。

サンプルは以下のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lucy-config PUBLIC "-//LUCY//DTD LUCY//EN"
"http://www.lucy.org/dtd/lucy-config.dtd">
<lucy-config>
    ...
</lucy-config>
```

componentタグ

componentタグはLucyで管理させるコンポーネントを定義するタグです。
ひとつのcomponentタグで1コンポーネントを管理します。

componentタグには以下のような属性があります。

属性名	必須/オプション	仕様	デフォルト値
class	必須	管理する対象のクラスを記述する	なし
name	オプション	コンポーネントに名前をつける	なし
instance	オプション	コンポーネントのスコープを記述する	デフォルトは**singleton**(シングルトン)

最もシンプルなサンプルは以下のようになります。class属性は必須なので、必ず記載します。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lucy-config PUBLIC "-//LUCY//DTD LUCY//EN"
"http://www.lucy.org/dtd/lucy-config.dtd">
<lucy-config>
    <component
class="hoge.foo.bar.BazServiceImpl"></component>
</lucy-config>
```

name属性はinstance属性も記載すると以下のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lucy-config PUBLIC "-//LUCY//DTD LUCY//EN"
"http://www.lucy.org/dtd/lucy-config.dtd">
<lucy-config>
    <component name="bazService"
class="hoge.foo.bar.BazServiceImpl"
instance="singleton"></component>
</lucy-config>
```

injectタグ

injectタグはコンポーネントをインジェクトするためのタグです。
injectする先をmethod属性で指定して、インジェクトしたいコンポーネントはボディ部に記載します。
ボディ部は式言語と呼ばれる形式ですが、それについては後述するので今は気にしないでください。

属性は下記のようになります。method属性は必須なので必ず記載する必要があります。

属性名	必須/オプション	仕様	デフォルト値
method	必須	インジェクトするメソッドを指定する	なし

parameterTypes	オプション	インジェクトするコンポーネントの型情報を補う	なし

以下に一番簡単なサンプルを示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lucy-config PUBLIC "-//LUCY//DTD LUCY//EN"
"http://www.lucy.org/dtd/lucy-config.dtd">
<lucy-config>
  <component class="lucy.InjectCounter" instance="prototype">
    <inject method="setValue">"aaa"</inject>
  </component>
</lucy-config>
```

initタグ

initタグはあるコンポーネントが初期化されるタイミングで実行されるメソッドを指定するタグです。
このタグを利用することでコンポーネントの初期化イベント時に処理を差し込むことが出来ます。
method属性を使って、初期化時に実行するメソッドを指定します。

属性の仕様は下記のようになります。

属性名	必須/オプション	仕様	デフォルト値
method	必須	初期化時に実行するメソッドを指定する	なし

最もシンプルな例は下記のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lucy-config PUBLIC "-//LUCY//DTD LUCY//EN"
"http://www.lucy.org/dtd/lucy-config.dtd">
<lucy-config>
  <component class="lucy.config.stax.LifeCycleMethodTagTarget">
    <init method="exec" />
  </component>
</lucy-config>
```

引数を取る場合には、引数をボディ部に並べます。
以下の例では、配列として渡す引数を並べています。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lucy-config PUBLIC "-//LUCY//DTD LUCY//EN"
"http://www.lucy.org/dtd/lucy-config.dtd">
<lucy-config>
  <component name="ary"
class="lucy.config.stax.LifeCycleMethodTagTargetWithParams2">
    <init method="exec">{"aaa", 123}</init>
  </component>
</lucy-config>
```

destroyタグ

destroyタグはあるコンポーネントが破棄されるタイミングで実行されるメソッドを指定するタグです。
このタグを利用することで、コンポーネントが破棄されるタイミングで処理差し込むことが出来ます。

属性の仕様は下記のようになります。

属性名	必須/オプション	仕様	デフォルト値
method	必須	破棄時に実行するメソッドを指定する	なし

シンプルな例は下記のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lucy-config PUBLIC "-//LUCY//DTD LUCY//EN"
"http://www.lucy.org/dtd/lucy-config.dtd">
<lucy-config>
    <component class="lucy.config.stax.LifeCycleMethodTagTarget">
        <destroy method="exec" />
    </component>
</lucy-config>
```

includeタグ

includeタグはコンポーネントをまとめてインクルードするためのタグです。
includeタグではlucy用のxmlへのパスを指定します。パスはタグのボディ部に記述します。
includeタグで登録されたコンポーネントは、同一のLucyインスタンスに展開されます。Lucyが階層構造をもつことはありません。

以下に簡単なサンプルを示します。lucy/config/stax/include1.xmlを
読み取り、コンポーネントを登録します。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lucy-config PUBLIC "-//LUCY//DTD LUCY//EN"
"http://www.lucy.org/dtd/lucy-config.dtd">
<lucy-config>
    <include>"lucy/config/stax/include1.xml"</include>
</lucy-config>
```

もう少しプログラマチックなサンプルだと以下のようにボディ部に式言語を記載する事も出来ます。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lucy-config PUBLIC "-//LUCY//DTD LUCY//EN"
"http://www.lucy.org/dtd/lucy-config.dtd">
<lucy-config>
    <include>"lucy/config/stax/include1.xml"</include>
    <include>"lucy/config/stax/include2.xml"</include>
    <include><![CDATA[
    env = System.properties["env"];
```

```
   path = "lucy/config/stax/test";
   if(env == "ut") {
      path = path + "3";
   } else if(env == "it") {
      path = path + "4";
   }
   path = path + ".xml";
   path;
   ]]></include>
</lucy-config>
```

aspectタグ

aspectタグはAOPでエンハンスするためのタグです。
pointcut属性を使って、正規表現でエンハンスする先を指定します。

属性の仕様は下記のようになります。

属性名	必須/オプション	仕様	デフォルト値
pointcut	オプション	エンハンスする対象メソッドを正規表現で指定する	なし

シンプルな例は下記のようになります。下記の例だと、ServiceImplクラスのbeginまたはexecメソッドに
requiredTxという名前で登録されているインタセプタをポイントカットとしてウィービングします。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lucy-config PUBLIC "-//LUCY//DTD LUCY//EN"
"http://www.lucy.org/dtd/lucy-config.dtd">
<lucy-config>
   ...
   <component class="lucy.tx.ServiceImpl">
      <aspect pointcut="begin|exec">requiredTx</aspect>
   </component>
</lucy-config>
```

Lucyの規約・制約

Lucyの規約

Lucyの暗黙的な規約は以下のとおりです。

- アノテーションの設定とXMLの設定がある場合、XMLの設定が優先される。

Lucyの制約

Lucyの制約は以下のとおりです。

- コンストラクターインジェクションの機能が無いため、コンポーネントにデフォルトコンストラクタが必須である。

アドバンスドトピック

式言語

Lucyでは式言語として、デフォルトでMvel を使用しています。主にXMLのボディ部分でMvelを使っています。
Mvelでは、ちょっとした処理など様々な記述が可能ですが、これについては今後記載していきます。

現状は以下をご覧ください。
http://mvel.codehaus.org/Language+Guide+for+2.0

Lucyのコンポーネント登録の仕組み

Lucyのコンポーネント登録の仕組みは以下のようになっています。

コンポーネントの登録（**Lucy.register(Class)**）
↓
コンポーネントの**bind**とその結果**(ConfigBindResult)**の登録。
ConfigBindResultには、アノテーションや**XML**に従い、メソッドやプロパティごとに**Behavior**を登録しておく。

Lucyからコンポーネントを取得するときの仕組み

Lucyからコンポーネントを取り出すときの仕組みは以下のようになっています。
コンポーネントの取り出し**(Lucy.get(Class))**
↓
バインド結果**(ConfigBindResult)**を使って、インジェクト開始。
↓
Lucyのライフサイクルに従い、コンポーネントを組み立てる**(ComponentBuilder.build())**。各フェーズごとに**Behavior**を実行。
（最もよくあるケースは、**AOP**実施**->**シングルトン／プロトタイプのコンポーネント作成**->@Inject**で必要コンポーネントのインジェクトとなる）

Lucyのライフサイクル

Lucyはコンポーネントの管理に明確なライフサイクルをもっています。ユーザがLucyからコンポーネントを取得すると、Lucyは下記の表のようなフェーズごとに
コンポーネントを処理していきます。

フェーズ名	フェーズ説明	具体的な処理内容
CLASS_LOADING	クラスのロード処理	アスペクトをかけるなど、クラスそのものに対しての処理をする。具体例：**@Aspect**
COMPONENT_CREATING	コンポーネントをインスタンス化する	コンポーネントを設定されたスコープで生成する。 具体例：**@SingletonScope**

COMPONENT_CREATED	作成したコンポーネントへの操作	作成されたコンポーネントに対してインジェクトなどの操作をする。具体例:**@Inject**
COMPONENT_INTIALIZE	コンポーネントの初期化	コンポーネントの初期化をする。 具体例:**@InitMethod**
COMPONENT_DESTROY	作成したコンポーネントの廃棄	コンポーネントを廃棄する。 具体例:**@DestroyMethod**

Lucyサンプル

Lucyのサンプルはlucy-examplesとして、プロジェクトがあります。そちらをダウンロードしてみてください。

ライセンス

Lucyのライセンスとして、ASL2.0を採用します.
その他使用しているライブラリのライセンスは別途記載します.

開発体制とリリース

（下記ドキュメントはユーザガイドから切り離す予定です.）

バージョン管理

開発はsvnのbranchをリリース版とする、「release branch」の方法を採用しています.
ある機能追加をしてリリースする場合には、branchを作成し、そちらで行います. バグ修正は、リリース版を修正した後にtrunkにマージを行います.

バージョニング
・プロダクトのバージョンについて
プロダクトのバージョンについては以下のような形式をとります.

lucy-xx.yy.zz(Q)

例えば、lucy-1.0.1-GAのようになります. xxは、メジャーバージョンです.数値で表現されます.
アーキテクチャの変更や稼動条件の変更、アプリケーションの作り方を大幅に変えてしまう変更はこのリリースになります.
yyは、マイナバージョンです.数値で表現されます.公開インタフェースの変更を伴う新しい機能の追加、ライブラリの新規追加・削除などはこのリリースになります.
zzは、バグフィックスバージョンです.数値で表現されます.バグ修正、または公開インタフェースの変更を伴わない機能追加、ライブラリのバージョンアップなどはこのリリースになります.
Q*は、識別番号です.アルファベットと数値で表現され、以下のような順番に変化していきます.

- SNAPSHOT：開発途中のリリースです.どのような変更が入っても不思議ではありません.
- Alpha：基本機能は出来上がっている状態です.ただしinterfaceの変更などの変更は以前として入る可能性があります.
- Beta：interfaceはほぼ固定され、機能の基本部分は確定されています.ただし最適化・リファクタリングなどが発生する可能性があります.
- RC：interface及び機能も固定されます.何らかのバグが発生した場合、RC番号をインクリメントします.

- GA：正式リリース版です.
- SP：正式リリース版に深刻なバグがあった場合にリリースされます.これは最新版にしか適用されません.

これらのルールは1.0以降に適用されます. 1.0以前は識別番号はSNAPSHOTとGAの簡易系統とします. バージョン番号も以下のとおりとします.

lucy-xx.yy.zz.aa(SNAPSHOT|GA)

xx/yy/zzの意味は上記と同じです.aaはリビジョンアップのための数字で、どのように使ってもかまいません.

リリース
・シングルリリースプラン

lucyでは安定版/開発版とわけることはせず、安定版のみのリリースします.

・リリース間隔

リリース間隔は、
1.0以前

マイナーバージョンアップで1-2ヶ月に1度を目安とする.
バグフィックスバージョンアップは適宜行う.

1.0以後

マイナーバージョンアップで4ヶ月に1度を目安とする.
バグフィックスバージョンアップは適宜行う.

・互換性について

互換性は以下の部分で守られます.

- @Publishedとついたインタフェース・クラスは、公開インタフェース/公開クラスとして互換性を保つ.
- 公開インタフェース/公開クラスはメソッドの追加する分には問題ないが、それ以外での変更の場合はマイナバージョンリリース以上になる.
- 公開インタフェース/公開クラス上の使われなくなったメソッドは@Deprecatedとするのみとする.

リリースプラン

現在のところ、Lucyの大幅な機能追加によるリリースプランは検討されていませんが、下記のような課題はあります。

- LucyからAOP機能を切り離す。
- Lucyのスコープ拡張機能の改善
- 依存jarの削減とLucy拡張機能との統合

課題が終わり次第適切にリリースします。