内容

はじめに

Spring ファミリのプロジェクトには Spring Boot、Spring Security、Spring Data、Spring Cloud、Spring Batch その他[1] があります。基盤になっているのは Spring Framework で更にその中核技術は IoC(Inversion of Control)コンテナによる DI(Dependency Injection = 依存性注入)です。

全 Spring プロジェクトの土台になっている DI の機能や設定について、Spring Framework 5.3.7 を基に必要最低限に絞って纏めます。

1. DI（依存性注入）とは

Spring のドキュメント[2]には以下のように書かれています。

『依存性注入（DI）は、オブジェクトが、コンストラクター引数、ファクトリ メソッドへの引数、またはオブジェクト インスタンスの構築または作成後に設定されたプロパティを通じてのみ依存関係（つまり、それらが連携する他のオブジェクト）を定義するプロセスです。～中略～コンテナは、Bean を作成するときにそれらの依存関係を挿入します。』

＜Bean とは＞※Spring プロジェクトの正式な説明ではなく個人的な解釈です（念の為、以下同）

Spring が管理対象にするクラスで、構成メタデータ(Configuration Metadata)に bean として指定（bean 定義）したものです。bean 定義は xml ファイル、Java アノテーション等の方法があります。

bean(クラス)はオブジェクトまたはインスタンスと呼ばれます(明確な使い分けはありません)。

＜依存性とは＞

Bean は機能を遂行するために別の Bean の機能を必要とする場合があります。例えば、請求金額算出オブジェクトが“消費税算出オブジェクト”を必要とする場合、「請求金額算出は消費税算出に依存している」といえます。この依存している関係が“Dependency(依存性)”です。

＜注入とは＞

DI を使わない場合、上記の例では請求金額算出オブジェクトが「new 消費税算出()」として小税算出のインスタンスを取得するのが一般的です。DI を使うと、IoC コンテナが消費税算出のインスタンスを必要とする側の Bean に渡してくれます。

---

[1] 詳細は https://spring.io/projects 参照 全てのプロジェクトを俯瞰した説明は見当たりませんが、Spring Framework の機能をベースにして各プロジェクトが個別に活動しているようです

[2] https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-dependencies

2. DI（依存性注入）の方法

依存性は、構成メタデータ(Configuration metadatabean)に bean を定義し、コンストラクタ、セッター、フィールドのいずれかを使って注入を行います。

2.1. 構成メタデータ

構成メタデータの定義方法にはアノテーション（Java）構成、XML 構成があります。

（1）アノテーション構成

アノテーション構成ファイルは、いくつかの注釈を追加した Java オブジェクトです。

```
@Configuration
@ComponentScan("jp.co.focs.explain")
public class Config {

    @Bean
    public Sky sky() {
        return new Sky(“blue”);
    }
}
```

上記例は、アノテーションでこのクラスが bean 定義（*@Bean* アノテーション）のプロバイダーであり、パッケージ *jp.co.focs.explain* で追加の Bean のコンテキスト スキャンを実行する必要があることを Spring に通知しています。

（2）XML 構成

以下はコンストラクタで依存性注入を行う、XML 構成ファイルです。

```
<bean id="focs" class="jp.co.focs.explain.Company">
    <constructor-arg index="0" ref="sky"/>
</bean>

<bean id="sky" class=" jp.co.focs.explain.Sky ">
    <constructor-arg index="0" value="blue"/>
</bean>
```


2.2. 注入箇所

（1）コンストラクター注入

```
@Component
public class Company {

    @Autowired //このアノテーションは現在は不要です
    public Company(Sky sky) {
        this.sky = sky;
    }
}
```

Spring は、2.1（1）アノテーション構成で指定した@ComponentScan の実行中に Company クラスに遭遇し、*@Autowired* 注釈付きコンストラクターを呼び出してインスタンスを初期化します。

Sky のインスタンスは、*Config* クラスの *@Bean* アノテーション付きメソッドを呼び出すことによって取得されます。最後に、*ApplicationContext* を通して依存性を注入済の Company オブジェクトを取得します。

```
ApplicationContext context = new AnnotationConfigApplicationContext(Config.class);
Company company = context.getBean(Company.class);
```

※Spring Framework 4.3/Spring Boot 1.4 以降、
単一のコンストラクターを持つクラスは *@Autowired* アノテーションを省略できます。

（2）setter ベースの依存性注入

setter ベースの DI は、Bean をインスタンス化した後に、Spring が setter メソッドを呼び出すことで実現します。

```
---------＜構成メタデータ＞----------
<bean id="exampleBean" class="examples.ExampleBean">
    <!-- setter injection using the nested ref element -->
    <property name="beanOne">
        <ref bean="anotherExampleBean"/>
    </property>
</bean>
<bean id="anotherExampleBean" class="examples.AnotherBean"/>
---------＜注入先クラス＞----------
public class ExampleBean {
    private AnotherBean beanOne;
    public void setBeanOne(AnotherBean beanOne) {
        this.beanOne = beanOne;
    }
}
```

（3）フィールド注入

```
public class Company {

    @Autowired
    private Sky sky;
}
```

※フィールドに直接注入できますが、コンストラクタ／セッター注入と異なり null が設定されても感知できません

3. DI の使い方（フレームワークへの組込み）

依存性の注入が一番使われるのは、フレームワークに自分が開発したクラスを組込む形です。フレームワークが想定している「型」で業務処理を設計し、フレームワークに呼び出してもらいます。

Spring Framework は J2EE/Web アプリケーションの MVC フレームワークでしたが、コアの部分はバッチ・フレームワーク（Spring Batch）への組込みでも使われています。

4. 「@」アノテーションと JDK 互換性

Spring が DI（依存性の注入）や AOP（アスペクト指向）を実現するための中心的な道具として、@Configuration、@Component、@Autowired 等の“@”付キーワード=アノテーション（注釈）を使っています。

4.1. アノテーションとは

アノテーション自体は動作を持たず、外から読み取られることで機能します。読み取られるタイミングは java.lang.annotation.RetentionPolicy の中に「注釈を保持するためのさまざまなポリシー」として以下のように定義されています。

SOURCE	注釈はコンパイラによって破棄されます。…@Override 等、コンパイラがチェックに使います
CLASS	注釈はコンパイラによってクラスファイルに記録されますが、実行時に VM によって保持される必要はありません。
RUNTIME	注釈は、コンパイラによってクラスファイルに記録され、実行時に VM によって保持されるため、反射的に読み取られる可能性があります。 …<反射的=reflectively>ReflectionAPI を使い他のクラスから読み取れます

Spring の使用例

① 起動時に <context:component-scan base-package="*package*" /> の指示があると、*package* に入っている全クラスファイルを走査し、“@Component”が書かれたクラスを探しだして bean 定義を創出します。

② @Autowired と書かれたフィールドに、id が一致する bean 定義のインスタンスをセットします。

4.2. Spring と JDK

Spring の DI 関連の仕様が Java 7 に取り込まれ[3]、Spring 3 と Java 7(JSR-330 仕様)はアノテーションの一部[4]で互換があります。

下記以外のアノテーションについては Spring-Core Technologies を参照してください。

Spring	**javax.inject.***	**javax.inject の制限 / コメント**
@Autowired	@Inject	@Inject には 'required' 属性がありません。代わりに Java 8 の Optional で使用できます。
@Component	@Named/@ManagedBean	JSR-330 は構成可能なモデルを提供せず、名前付きコンポーネントを識別する方法のみを提供します。

[3] https://docs.oracle.com/javaee/7/tutorial/cdi-basic.htm#GIWHB

[4] https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-standard-annotations

5. プロジェクトの開始（Spring Boot）

Spring Framework を使ったアプリケーションの開発を行うには、Spring の関係だけでも依存ライブラリの把握が大変です。更に RDB 等のミドルウェアと連係するためには大量のライブラリに依存することになります。これらの依存性を解決しながら IDE プロジェクトを作ることができます。

5.1. 空プロジェクトの作成

Spring のサイトでライブラリが揃ったプロジェクトを得ることができます。