DAOを繰り返すな！

HibernateとSpring AOPで、汎用性と型安全性を備えたDAOを作る

Per Mellqvist (per@mellqvist.name)
System architect
自由职业者

2006年 5月 12日

JavaTM 5ジェネリックの採用により、汎用の（ジェネリックな）型安全なDAO（Data Access Object：データ・アクセス・オブジェクト）実装というアイデアが現実のものになりました。この記事では、システム・アーキテクトであるPer Mellqvistが、Hibernateに基づくジェネリックDAO実装クラスについて解説します。また、Spring AOPイントロダクションを使用して、このクラスに、クエリー実行のための型安全なインターフェースを追加する方法も示します。

ほとんどの開発者にとって、システム内のすべてのDAOについてほぼ同じコードを書くのが、今では習慣になっています。このような繰り返しを「コード・スメル（コードの匂い）」として嗅ぎ分ける人もいるかもしれませんが、私たちのほとんどはそれを我慢することに慣れてしまいました。対策はあります。さまざまなORMツールを使用して、コードの繰り返しを避けることができます。たとえば、Hibernateでは、永続的ドメイン・オブジェクトのすべてにセッション操作を直接使用することができます。この方法の欠点は、型安全性が失われることです。

なぜデータ・アクセス・コード用の型安全なインターフェースが必要なのでしょうか。私なら、最近のIDEツールと組み合わせて使用することで、プログラミングの誤りを減らし、生産性を高めることができるから、と答えます。なにより、型安全なインターフェースでは、どのドメイン・オブジェクトの永続的ストレージが使用可能かが明確に示されます。第二に、エラーが起きやすい型変換の必要がなくなります。（CRUDよりもクエリー操作で一般的な問題です。）最後に、今日のほとんどのIDEに見られるオート・コンプリーション機能を活用できます。オート・コンプリーションは、特定のドメイン・クラスで使用可能なクエリーを記憶する高速な方法です。

この記事では、型安全なインターフェースのメリットを失わずに、DAOを何度も繰り返すのを避ける方法を示します。実際、新しいDAOごとに書く必要があるのは、Hibernateマッピング・ファイル、平凡な古いJavaインターフェース、そしてSpring構成ファイルの10行だけです。

DAO実装

DAOのパターンは、企業向けJava開発者なら誰でもよく知っているはずです。ただし、パターンの実装にはかなり幅があるので、この記事で示すDAO実装の前提条件を明確にしておきましょう。

- システム内のすべてのデータベース・アクセスは、カプセル化を達成するためにDAOを通じて行われます。
- DAOの各インスタンスは、1つのプライマリー・ドメイン・オブジェクトまたはエンティティーを担当します。ドメイン・オブジェクトが独立したライフサイクルを持つ場合は、専用のDAOが必要です。
- DAOは、ドメイン・オブジェクトの作成（Creation）、（主キーによる）読み取り（Read）、更新（Update）、および削除（Deletion）、すなわちCRUDを処理します。
- DAOでは、主キー以外の条件に基づくクエリーも可能です。私はこれらをファインダー・メソッドまたはファインダーと呼んでいます。ファインダーのリターン値は、通常、DAOが担当するドメイン・オブジェクトの集合です。
- DAOは、トランザクション、セッション、または接続の処理は担当しません。これらは、柔軟性を達成するために、DAOの外部で処理されます。

ジェネリックDAOインターフェース

ジェネリックDAOの基礎は、CRUD操作にあります。次のインターフェースは、ジェネリックDAOのメソッドを定義しています。

リスト1. ジェネリックDAOインターフェース

```
public interface GenericDao <T, PK extends Serializable> {

/** Persist the newInstance object into database */
PK create(T newInstance);

/** Retrieve an object that was previously persisted to the database using
*   the indicated id as primary key
*/
T read(PK id);

/** Save changes made to a persistent object.  */
void update(T transientObject);

/** Remove an object from persistent storage in the database */
void delete(T persistentObject);
}
```

インターフェースの実装

リスト1のインターフェースをHibernateで実装するのは、実に簡単です（リスト2）。基盤のHibernateメソッドを呼び出して、キャストを追加するだけです。Springは、セッション管理とトランザクション管理を担当します。（もちろん、これらの関数が正しくセットアップされていることが前提ですが、このテーマについては、HibernateとSpringフレームワークのマニュアルで詳しく説明されています。）

リスト2. 最初のジェネリックDAO実装

```
public class GenericDaoHibernateImpl <T, PK extends Serializable>
implements GenericDao<T, PK>, FinderExecutor {
private Class<T> type;

public GenericDaoHibernateImpl(Class<T> type) {
this.type = type;
}

public PK create(T o) {
```

```
return (PK) getSession().save(o);
}

public T read(PK id) {
return (T) getSession().get(type, id);
}

public void update(T o) {
getSession().update(o);
}

public void delete(T o) {
getSession().delete(o);
}

// Not showing implementations of getSession() and setSessionFactory()
}
```

Spring構成

最後に、Spring構成で、GenericDaoHibernateImplのインスタンスを作成します。GenericDaoHibernateImplのコンストラクターに対して、DAOインスタンスが担当するドメイン・クラスを指定する必要があります。これが必要なのは、DAOによって管理されるオブジェクトのタイプについて、Hibernateにランタイムの知識を持たせるためです。リスト3では、サンプル・アプリケーションのドメイン・クラスPersonをコンストラクターに渡し、インスタンス化されたDAOへのパラメーターとして、構成済みのHibernateセッション・ファクトリーを設定しています。

リスト3. DAOの構成

```
<bean id="personDao" class="genericdao.impl.GenericDaoHibernateImpl">
<constructor-arg>
<value>genericdaotest.domain.Person</value>
</constructor-arg>
<property name="sessionFactory">
<ref bean="sessionFactory"/>
</property>
</bean>
```

便利なジェネリックDAO

まだ終わりではありませんが、ここまででもすでに便利です。リスト4に、この状態でのジェネリックDAOの使用例を示します。

リスト4. DAOの使用

```
public void someMethodCreatingAPerson() {
...
GenericDao dao = (GenericDao)
beanFactory.getBean("personDao"); // This should normally be injected

Person p = new Person("Per", 90);
dao.create(p);
}
```

この時点で、型安全なCRUD操作ができるジェネリックDAOになっています。GenericDaoHibernateImplをサブクラス化して、各ドメイン・オブジェクトのクエリー機能を追加してもいいでしょう。しかし、この記事の目的は、クエリーごとに明示的なJavaコードを書か

ずにこれを達成する方法を示すことなので、さらに2つのツールを使用して、DAOにクエリーを導入することにします。すなわち、Spring AOPとHibernate名前付きクエリーです。

Spring AOPイントロダクション

Spring AOPのイントロダクション（introduction）を使用すると、既存のオブジェクトをプロキシでラップし、実装すべき新しいインターフェースを定義し、以前はサポートされていなかったメソッドのすべてを単一のハンドラーに委託することによって、既存のオブジェクトに機能を追加することができます。私のDAO実装では、イントロダクションを使用して、既存のジェネリックDAOクラスに多数のファインダー・メソッドを追加します。ファインダー・メソッドは各ドメイン・オブジェクトに固有なので、ジェネリックDAOの型付きインターフェースに適用されます。

このためのSpring構成をリスト5に示します。

リスト5. FinderIntroductionAdvisorのSpring構成

```
<bean id="finderIntroductionAdvisor" class="genericdao.impl.FinderIntroductionAdvisor"/>

<bean id="abstractDaoTarget"
class="genericdao.impl.GenericDaoHibernateImpl" abstract="true">
<property name="sessionFactory">
<ref bean="sessionFactory"/>
</property>
</bean>

<bean id="abstractDao"
class="org.springframework.aop.framework.ProxyFactoryBean" abstract="true">
<property name="interceptorNames">
<list>
<value>finderIntroductionAdvisor</value>
</list>
</property>
</bean>
```

リスト5の構成ファイルでは、3つのSpringビーンズ（beans）を定義しています。最初のビーン、FinderIntroductionAdvisorは、DAOに導入されたすべてのメソッドのうち、GenericDaoHibernateImplクラスでは使用できないものを処理します。Advisorビーンについては、もう少し後で詳しく説明します。

2番目のビーンは、「abstract」です。Springでは、これは、このビーンを他のビーン定義で再利用できるが、インスタンス化されないことを意味します。abstractプロパティー以外に、このビーン定義が指定しているのは、GenericDaoHibernateImplのインスタンスが必要なことと、SessionFactoryの参照が必要なことだけです。GenericDaoHibernateImplクラスは、引数としてドメイン・クラスを取る1つのコンストラクターだけを定義していることに注目してください。このビーン定義はabstractなので、後で何度でも再利用することができ、コンストラクター引数を適切なドメイン・クラスに設定することができます。

最後に、3番目の最も興味深いビーンは、GenericDaoHibernateImplのバニラ・インスタンス（基本機能のみのインスタンス）をプロキシでラップして、ファインダー・メソッドを実行する能力を与えます。このビーン定義もabstractであり、バニラDAOに導入したいインターフェースを指定していません。インターフェースは、具象インスタンスごとに異なります。リスト5に示されている構成全体が一度しか行われないことに注目してください。

ジェネリックDAOの拡張

各DAOのインターフェースは、もちろん、GenericDaoインターフェースに基づきます。必要なのは、インターフェースを特定のドメイン・クラスに適応させて、ファインダー・メソッドを含むように拡張することだけです。リスト6に、特定の目的に合わせて拡張されたGenericDaoインターフェースの例を示します。

リスト6. PersonDaoインターフェース

```
public interface PersonDao extends GenericDao<Person, Long> {
List<Person> findByName(String name);
}
```

明らかに、リスト6で定義されているメソッドの目的は、Personを名前で検索することです。必要なJava実装コードはすべてジェネリック・コードなので、さらにDAOを追加するときも更新は不要です。

PersonDaoの構成

Spring構成は、すでに定義した「abstract」ビーンに依存するので、かなりコンパクトになります。DAOがどのドメイン・クラスを担当するかを指定する必要があり、また、DAOが実装すべきインターフェース (一部のメソッドは直接、その他のメソッドはイントロダクションを使用して) をSpringに対して指示する必要があります。リスト7に、PersonDAOのSpring構成ファイルを示します。

リスト7. PersonDaoのSpring構成

```
<bean id="personDao" parent="abstractDao">
<property name="proxyInterfaces">
<value>genericdaotest.dao.PersonDao</value>
</property>
<property name="target">
<bean parent="abstractDaoTarget">
<constructor-arg>
<value>genericdaotest.domain.Person</value>
</constructor-arg>
</bean>
</property>
</bean>
```

リスト8に、この更新版のDAOの使用例を示します。

リスト8. 型安全なインターフェースの使用

```
public void someMethodCreatingAPerson() {
...
PersonDao dao = (PersonDao)
beanFactory.getBean("personDao"); // This should normally be injected

Person p = new Person("Per", 90);
dao.create(p);

List<Person> result = dao.findByName("Per"); // Runtime exception
}
```

リスト8のコードは型安全なPersonDaoインターフェースを使用する正しい方法ですが、DAOの実装はまだ完全ではありません。findByName()を呼び出すと、ランタイム例外が発生します。問題は、findByName()の呼び出しに必要なクエリーをまだ実装していない点にあります。残っているのは、クエリーを指定することだけです。これを是正するために、Hibernate名前付きクエリーを使用します。

Hibernate名前付きクエリー

Hibernateでは、Hibernateマッピング・ファイル（hbm.xml）でHQLクエリーを定義して、名前を付けることができます。このクエリーを後でJavaコード内で使用するには、付けた名前を参照するだけです。この方法のメリットの1つは、コードを変更しなくても、デプロイ時にクエリーをチューニングできることです。すぐ後で示すように、もう1つのメリットは、新しいJava実装コードを書かなくても「完全な」DAOを実装できる可能性があることです。リスト9は、名前付きクエリーを含んだマッピング・ファイルの例です。

リスト9. 名前付きクエリーを含んだHibernateマッピング・ファイル

```
<hibernate-mapping package="genericdaotest.domain">
<class name="Person">
<id name="id">
<generator class="native"/>
</id>
<property name="name" />
<property name="weight" />
</class>

<query name="Person.findByName">
<![CDATA[select p from Person p where p.name = ? ]]>
</query>
</hibernate-mapping>
```

リスト9では、ドメイン・クラスPersonのHibernateマッピングをnameとweightという2つのプロパティーで定義しています。Personは、これらのプロパティーを持つ単純なPOJOです。このファイルには、データベース内のPersonのすべてのインスタンスのうち、「name」が指定されたパラメーターに等しいものを検索するクエリーも含まれています。Hibernateには、名前付きクエリーについて実際の名前空間機能はありません。この記事の目的から、私はすべてのクエリー名の前にドメイン・クラスの短い（修飾されていない）名前を付けています。現実の状況では、パッケージ名も含めた完全なクラス名を使用する方がよいでしょう。

ステップ・バイ・ステップの概要

任意のドメイン・オブジェクトについて新しいDAOを作成・構成するために必要なすべてのステップを説明しました。すなわち、次の3つの単純なステップです。

1. GenericDaoを拡張して、必要なファインダー・メソッドを含めるインターフェースを定義します。
2. ドメイン・オブジェクトのhbm.xmlマッピング・ファイルに、各ファインダー・メソッドの名前付きクエリーを追加します。
3. DAOに関する10行のSpring構成ファイルを追加します。

説明のまとめとして、ファインダー・メソッドを実行するコード（1回書いただけ！）を示します。

再利用可能なDAOクラス

使用されているSpringアドバイザーとインターセプターは単純なものであり、事実、GenericDaoHibernateImplClassを逆参照するのが仕事です。メソッド名が「find」で始まる呼び出しはすべて、DAOと単一のメソッドexecuteFinder()に渡されます。

リスト10. FinderIntroductionAdvisorの実装

```
public class FinderIntroductionAdvisor extends DefaultIntroductionAdvisor {
public FinderIntroductionAdvisor() {
super(new FinderIntroductionInterceptor());
}
}

public class FinderIntroductionInterceptor implements IntroductionInterceptor {

public Object invoke(MethodInvocation methodInvocation) throws Throwable {

FinderExecutor genericDao = (FinderExecutor) methodInvocation.getThis();

String methodName = methodInvocation.getMethod().getName();
if (methodName.startsWith("find")) {
Object[] arguments = methodInvocation.getArguments();
return genericDao.executeFinder(methodInvocation.getMethod(), arguments);
} else {
return methodInvocation.proceed();
}
}

public boolean implementsInterface(Class intf) {
return intf.isInterface() && FinderExecutor.class.isAssignableFrom(intf);
}
}
```

executeFinder()メソッド

リスト10の実装に不足しているのは、executeFinder()の実装だけです。このコードは、呼び出されたクラスとメソッドの名前を見て、構成上の規則を使用して、これらをHibernateクエリーの名前と照合します。FinderNamingStrategyを使用して、別の命名法でクエリーに名前を付けることもできます。デフォルトの実装では、「ClassName.methodName」という名前のクエリーを探します。ここでのClassNameは、パッケージ名を含まない短い名前です。リスト11で、私のジェネリックで型安全なDAO実装が完成します。

```
public List<T> executeFinder(Method method, final Object[] queryArgs) {
final String queryName = queryNameFromMethod(method);
final Query namedQuery = getSession().getNamedQuery(queryName);
String[] namedParameters = namedQuery.getNamedParameters();
for(int i = 0; i < queryArgs.length; i++) {
Object arg = queryArgs[i];
Type argType =  namedQuery.setParameter(i, arg);
}
return (List<T>) namedQuery.list();
}

public String queryNameFromMethod(Method finderMethod) {
return type.getSimpleName() + "." + finderMethod.getName();
}
```

まとめ

Java 5より前のJava言語は、型安全性と汎用性の両方を備えたコードの作成をサポートしていませんでした。開発者は、どちらか1つを選ばなければなりませんでした。この記事では、Java 5のジェネリックをSpringやHibernate（およびAOP）などのツールと組み合わせて生産性を高める一例を紹介しました。ジェネリックで型安全なDAOクラスは、比較的簡単に書くことができます。必要なのは、1つのインターフェース、いくつかの名前付きクエリー、そしてSpring構成への10行の追加だけです。これだけで、エラーを大幅に減らし、時間も節約することができます。

この記事のほとんどすべてのコードが再利用可能です。この記事では実装されていないタイプのクエリーや操作（おそらく、一括操作）をDAOクラスに含めることもでき、この記事で例示した技法を使えば、少なくとも、そのうちのいくつかを実装できるはずです。ジェネリックで型安全なDAOクラスのその他の実装については、「参考文献」を参照してください。

謝辞

ジェネリックでありながら型安全なDAOというコンセプトは、Java言語にジェネリックが登場して以来のテーマでした。私は2004年のJavaOneで、ジェネリックDAOの実現性についてDon Smithと簡単なディスカッションを行いました。この記事で使用されているDAO実装クラスは実装例であり、他の実装も存在します。たとえば、Christian BauerはCRUD操作と条件検索を備えた実装を公開していますし、Eric Burkeもこの分野での仕事を行っています。いずれ、さらに多くの実装が登場すると思います。Christianは、ジェネリックで型安全なDAOを書くという私の最初の試みに目を通し、改善のヒントを与えてくれました。最後に、この記事のレビューにおいて貴重な助力を与えてくれたRamnivas Laddadに感謝します。

ダウンロード

内容	ファイル名	サイズ
Full source code	j-genericdao.zip	12MB

著者について

Per Mellqvist

Per Mellqvistは、スウェーデン・ストックホルム市を本拠地とするシステム・アーキテクトです。企業向けJavaのあらゆる側面を熟知し、1つのことを完璧に処理するフレームワークやツールを高く評価しています。