**译 文**

# Spring MyBatis 集成示例

Contents

* [1. 为什么选择Mybatis?](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "1_Why_Mybatis)
  + [1.1.什么是MyBatis?](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "11_What_is_MyBatis)
* [2. Spring MyBatis 交互](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "2_Spring_MyBatis_interaction)
  + [2.1. 什么是 MyBatis-Spring?](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "21_What_is_MyBatis-Spring)
  + [2.2. 安装](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "22_Installation)
  + [2.3. Spring 应用](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "23_Spring_application_context_setup)上下文设置
  + [2.4. 注入映射](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "24_Injecting_Mappers)
    - [2.4.1. 服务层次](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "241_Service_Layer)
    - [2.4.2. 数据访问层](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "242_DAO_layer)
    - [2.4.2.1. 注册](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "2421_Register_mapper)映射
    - [2.4.2.1.1. 使用 XML 配置](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "24211_With_XML_Config)
  + [2.5. MyBatis 配置XML](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "25_MyBatis_configuration_XML)
    - [2.5.1. 类型别名](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "251_typeAliases)
    - [2.5.2. 映射](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "252_mappers)
  + [2.6. Mapper XML Files](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "26_Mapper_XML_Files)
    - [2.6.1. select查询](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "261_select)
    - [2.6.2. 结果集](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "262_Result_Map)
    - [2.6.2.1. id & 结果](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "2621_id_result)
* [3. 总结](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "3_Summary)
* [4. 资源](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "4_Resources)

## 1. 为什么使用Mybatis?

简单地说: 易用, 轻量, 开源, 动态 sql 和 sql 控制, 和之前的 iBATIS 类似。现在让我详细说明下这个项目。 回想以前使用 Podcastmania.ro的日子,浏览[Podcastpedia.org](http://www.codingpedia.org/ama/story-of-podcastpedia-org/" \o "Story of Podcastpedia.org" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)的故事,我使用我自己基于servlet的MVC框架来开发web 应用 及使用老的 [JDBC](http://en.wikipedia.org/wiki/Java_Database_Connectivity" \o "Wikipedia - JDBC" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank) 来访问数据库。在 Spring MVC升级后, 我开始使用Spring的JdbcTemplate 用来做数据库连接, 来移除部分冗余代码。 之后我在项目中使用iBATIS来访问数据库，Hibernate已经存在很久了，因为一些法律原因，没法使用数据库标准化，但是iBATIS是个附加选项。与此同时，MyBatis已经开始进入开发阶段，所以我阅读了它的文档，并且从Spring的JdbcTemplate转到使用MyBatis。在这段时间里，我使用Hibernate和JPA2.0来做数据的持久化，总体上来说，在当时我认为好的Java持久化框架有如下几种：

* JPA/Hibernate
* myBatis (前身为iBatis)
* Spring JDBC
* JDBC

凡是都有利弊, 在我使用了这些技术后，我选择使用MyBatis。你可以在网上找的很多资源，如教程，介绍等等，可以学习如何使用这个技术，并且可以在你项目中应用，来找出究竟哪个才是你做持久化操作的最好选择。我也建议观看教学视频，不幸的是视频声音比较糟糕，但是内容确实很有趣。

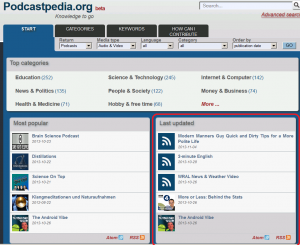
### 1.1. 什么是MyBatis?

所以现在我已经做出了我的选择，首先让我们来看看MyBatis。根据其官方的介绍，“MyBatis是第一个类持久化框架，并且支持自定义SQL语句、存储过程和高级映射。MyBatis消除了几乎所有的JDBC代码，而且可以手动进行参数设置和返回结果。MyBatis可以使用简单的XML或者注解来进行配置，使用Map接口和Java POJOs（简单Java对象）来记录数据库结果。”

说的够多了，让我们回到文章的主题，Spring和MyBatis是如何交互的。

## 2. Spring MyBatis 交互

为了突出易用性，在过去是使用简单的例子来解释Spring的MyBatis需要实现哪些来配置东西来重新恢复新特性（尤其是最新更新的）。你能通过访问 [Podcastpedia.org](https://github.com/Codingpedia/podcastpedia/" \o "Last updated podcasts from Podcastpedia.org" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank) 的主页来了解这些:

[](http://www.codingpedia.org/wp-content/uploads/2013/10/last_updated_podcasts.png)

展示下 Podcastpedia.org 的首页

如果你对于iBATIS很熟悉（MyBatis的前身），你可能会知道在Spring框架3版本之前，Spring框架直接提供了iBATIS的SQL结果集及其管理，数据访问层的接口实现和事务策略。 与此同时，iBATIS转型成MyBatis，Spring 3的开发也告一段落，Spring 的开发团队并没有发布集成MyBatis的Spring发行版，官方的Spring支持需要等一段时间。 在Spring开发团队对MyBatis产生兴趣后，MyBatis社区决定将MyBatis集成到Spring中。这就是MyBatis-Spring项目的诞生之初，现在已经被[Podcastpedia.org](https://github.com/Codingpedia/podcastpedia" \o "Podcastpedia.org, knowledge to go" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)广泛使用。

### 2.1. 什么是MyBatis-Spring?

MyBatis-Spring 是MyBatis 和Spring的无缝结合。此库允许MyBatis参与到Spring的事务中，能构建MyBatis映射和Sql会话，以及将其注入到其他beans中，转化MyBatis异常到Spring的数据异常，最终，它能使你不需要引入其他依赖便可构建MyBatis，Spring伙食MyBatis-Spring项目。

### 2.2. 安装

想要使用MyBatis-Spring模块，你只需要使用将mybais-spring-x.x.x.jar文件及其依赖导入到环境变量中即可。

如果你正在使用Maven作为构建管理工具，你只需要将如下依赖添加到你的pom.xml文件中即可:

<!-- MyBatis integration -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.2.1</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.2.3</version>

</dependency>

### 2.3. Spring 应用上下文设置

为了让MyBatis和Spring配合使用，你至少需要在Spring的应用上下文中定义如下2个东西：一个SqlSessionFactory及至少一个映射接口。

在MyBatis-Spring中，一个Sql会话工厂 Bean被用来创建一个Sql会话工厂。每个MyBatis应用中心围绕着一个Sql会话工厂。你用Sql会话中心去创建一个Sql会话。一旦你有一个会话后，你用它来执行语句的映射，提交或者回滚，最终，在你什么都不需要后，你就可以关闭会话。有了 MyBatis-Spring ，你不需要直接使用Sql会话工厂，因为你的beans能够被一个线程安全的Sql会话所注入，能够自动地进行提交，回滚及关闭会话操作，而这些只需要Spring的事务配置中进行配置即可。

为了配置工厂bean，你需要将如下代码写到XML配置文件中:

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="configLocation" value="classpath:config/mybatisV3-config.xml"/>

</bean>

不知你是否注意到了Sql会话工厂需要一个数据源没有。它可以是任何数据源，而且应该想Spring的数据源连接一样被配置。[Podcastpedia.org](https://github.com/Codingpedia/podcastpedia" \o "Podcastpedia.org, knowledge to go" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)的数据源配置示例如下：

<!-- ========================= DATASOURCE DEFINITION via JNDI ========================= -->

<!-- When resourceRef is true, the value of jndiName will be prepended with

server’s JNDI directory. Consequently, the actual name used will be

java:comp/env/jdbc/pcmDB. -->

<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean" scope="singleton">

<property name="jndiName" value="java:comp/env/jdbc/pcmDB" />

<property name="resourceRef" value="true" />

</bean>

想要了解我的Tomcat JDBC连接池的开发配置，或是了解数据源在开发/测试环境中配置，可以查看我的源代码。

MyBatis的配置文件的位置你可以通过阅读下个部分了解。阅读了MyBatis配置XML，你就知道你该如何配置它了。

### 2.4. 映射注入

不像其他数据访问层的代码需要手动使用Sql会话Dao支持或者是Sql会话模板，Mybatis-Spring 能创建一个线程安全的映射，能让你直接将其注入到其他bean中。我更倾向于称呼其为映射而非DAO。

#### 2.4.1. 服务层

服务层是基于数据访问层的。服务层的实现类需要加载实体类，使其在主页上被加载，能够被注入到数据访问层的映射中:

<!-- =================== Service beans configuration ================== -->

<bean id="startPageService" class="org.podcastpedia.service.impl.StartPageServiceImpl">

<property name="podcastDao" ref="podcastDao"/>

</bean>

一旦映射被注入，其能够在业务逻辑中被如下使用：

public class StartPageServiceImpl implements StartPageService {

private static final Integer NUMBER\_OF\_PODCASTS\_IN\_CHART = 5;

private static Logger LOG = Logger.getLogger(StartPageServiceImpl.class);

private PodcastDao podcastDao;

public void setPodcastDao(PodcastDao podcastDao) {

this.podcastDao = podcastDao;

}

@Cacheable(value="newestAndRecommendedPodcasts", key="#root.method.name")

public List<Podcast> getNewestPodcasts() {

List<Podcast> newestPodcasts = podcastDao.getNewestPodcasts(NUMBER\_OF\_PODCASTS\_IN\_CHART);

for(Podcast p : newestPodcasts) {

Episode lastEpisode = episodeDao.getLastEpisodeForPodcast(p.getPodcastId());

p.setLastEpisode(lastEpisode);

}

return newestPodcasts;

}

..............

}

注意到没有，上面并没有任何的Sql会话或是MyBatis对于服务层的指针。这里不需要任何会话的创建、打开或是关闭，MyBatis-Spring会搞定一切。你可以在下个部分了解到我是如何实现及配置PpdcastDao的。

**注意：如果你想要知道@Cacheable注解是如何被配置的，请访问我的另一篇文章**[Spring caching with Ehcache](http://www.codingpedia.org/ama/spring-caching-with-ehcache/" \o "Codingpedia.org - Spring caching with Ehcache" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)。

#### 2.4.2. 数据访问层

##### 2.4.2.1. 映射注册

你能通过使用传统的XML配置来注册映射或者是新的Java 3.0以上的配置（参考@Configuration）。我更倾向于使用传统的XML配置方法来进行配置，因为配置文件能和代码很好地进行分离:

###### 2.4.2.1.1. 使用XML进行配置

映射/dao在Spring中通过映射工厂Bean在应用上下文XML文件中进行配置：

<!-- =============== MyBATIS beans configuration ================== -->

<bean id="podcastDao" class="org.mybatis.spring.mapper.MapperFactoryBean">

<property name="sqlSessionFactory" ref="sqlSessionFactory"/>

<property name="mapperInterface" value="org.podcastpedia.dao.PodcastDao" />

</bean>

映射Bean工厂能够被Sql会话工厂配置，或者由一个Sql会话模板进行预配置。你能了解到Sql会话工厂是如何在安装/Spring应用上下文配置的。你也可以为数据访问层下载完整的Spring配置文件。

第二个参数-映射接口-即设置MyBatis映射的映射接口。注意到没有，映射类需要为一个接口特定做分类，而不是一个实际的接口实现:

package org.podcastpedia.dao;

import java.util.Date;

import java.util.List;

import java.util.Map;

import org.podcastpedia.domain.Comment;

import org.podcastpedia.domain.Podcast;

import org.podcastpedia.domain.Tag;

/\*\*

\* Interface for database access

\*

\* @author ama

\*

\*/

public interface PodcastDao {

/\*\*

\* Returns the newest podcasts (ORDER BY last\_updated DESC)

\*

\* @param numberOfPodcasts (number of podcasts to be returned)

\* @return

\*/

public List<Podcast> getNewestPodcasts(Integer numberOfPodcasts);

....................

}

如果是PodcastDao有一个与之相对的MyBatis XML映射文件在环境变量位置中，作为其映射接口，它能自动被映射工厂Bean解析。但是因为他们在不同的环境位置，需要配置相应的参数。

让我们看看如何配置sql语句以及如何在MyBatis中进行实现。

### 2.5. MyBatis XML 配置

正如文章刚开始提到的Sql会话工厂Bean有自己的配置参数，其同时也定义了MyBatis的配置。这里提供了一个示例配置文件：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE configuration

PUBLIC "-//[mybatis.org//DTD](http://mybatis.org//DTD) Config 3.0//EN"

"<http://mybatis.org/dtd/mybatis-3-config.dtd>">

<configuration>

<typeAliases>

<typeAlias type="org.podcastpedia.domain.Podcast" alias="Podcast"/>

.......

</typeAliases>

<mappers>

<mapper resource="maps/PodcastMapper.xml" />

........

</mappers>

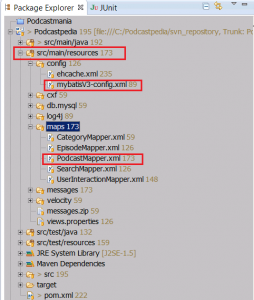
</configuration>

#### 2.5.1. 类型别名

一个类型别名仅仅只是为了让Java类型的名字更短。它纸盒XML配置相关，并且只已类名相关，减少类型的耦合度。你会在下个部分了解到什么是‘org.podcastpedia.domain.Podcast对象’，我将只用“Podcast”。

#### 2.5.2. 映射

配置元素只告诉MyBatis到哪里去找映射文件。Java并不会真的提供任何自动的映射文件。你能用路径资源指针或者url指向文件（如 file：//URLs）:

[](http://www.codingpedia.org/wp-content/uploads/2013/10/MyBatis-files.png)

MyBatis 配置文件及映射文件位置

**注意:有许多其他的参数你需要给MyBatis配置。请参考文档页面获取更多详细信息。**

### 2.6. 映射XML文件

MyBatis真正的力量是映射语句。这是最神奇的地方。其所有功能里，映射文件比起来其实非常简单。当你把它跟JDBC代码相比是，你会立马看到你节省了95%的代码量。MyBatis是为SQL聚焦而生，而且它比起你之前的方法来说做的更好。

这里我展示了一个PodcastMapper.xml小的代码片段，告诉各位PodcastDao接口方法是如何被映射的：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mapper PUBLIC "-//[mybatis.org//DTD](http://mybatis.org//DTD) Mapper 3.0//EN"

"<http://mybatis.org/dtd/mybatis-3-mapper.dtd>">

<mapper namespace="org.podcastpedia.dao.PodcastDao">

<!-- result maps -->

<resultMap id="podcastsMap" type="Podcast" >

<id column="podcast\_id" property="podcastId" />

<result column="url" property="url" />

<result column="rating" property="rating" />

<result column="numberRatings" property="number\_ratings" />

<result column="number\_visitors" property="numberOfVisitors" />

<result column="DESCRIPTION" property="description" />

<result column="PODCAST\_IMAGE\_URL" property="urlOfImageToDisplay" />

<result column="TITLE" property="title" />

<result column="last\_episode\_url" property="lastEpisodeMediaUrl" />

<result column="title\_in\_url" property="titleInUrl" />

<result column="publication\_date" property="publicationDate"/>

</resultMap>

<select id="getNewestPodcasts" resultMap="podcastsMap" parameterType="Integer">

SELECT

PODCAST\_ID,

URL,

NUMBER\_VISITORS,

DESCRIPTION,

PODCAST\_IMAGE\_URL,

TITLE,

last\_episode\_url,

title\_in\_url,

publication\_date

FROM

podcasts

WHERE

availability=200

ORDER BY publication\_date DESC

limit 0, #{value};

</select>

.....

</mapper>

请注意映射文件第5行的名字空间的值-org.podcastpedia.dao.PodcastDao 及第21行<select>元素。 如果你将其结合，它会使得org.podcastpedia.dao.PodcastDao.getNewestPodcasts方法及其数据访问层 定义的接口相结合。

#### 2.6.1. select查询

select语句是一个你在MyBatis中国非常常用的元素之一。你可以通过它将数据库中的数据取出，让应用的查询与数据紧密结合。每次插入、更新或者删除操作，其中就有可能伴随着查询。MyBatis的其中一个基础原则就是集中处理查询结果和查询效率。

被暴露出来的语句被称为getNewestPodcasts（和PodcastDao接口的方法相统一），使用一个Integer参数（和PodcastDao接口的输入方法相统一）和一个结果集参数接受，可以获得一个额外的结果集。

#### 2.6.2. 结果集

结果集元素在MyBatis中是最重要和最为强大的元素。它允许你不通过90%JDBC从ResultSets获取数据的代码，在某些情景下，允许你做一些JDBC甚至做不到的事情。事实上，写一个相同的代码 如join联合查询因素作为一个复杂的语句可能会产生上千行代码。这个结果集的设计师使得语句如此的简单甚至完全不需要使用结果集映射，而更复杂的语句也完全不需要描述其关系。

让我们谈一下数据集的一些结构。顶元素有一个id（“podcastMap”），在映射文件（名字空间）中唯一地辨识一个select语句，而且它在select中，和一个唯一辨识Java类名或是一个类型别名的类型（“Podcast”）。（“Podcast”成为MyBatis配置文件中的类型别名）

##### 2.6.2.1. id & 结果

<id column="podcast\_id" property="podcastId" />

<result column="url" property="url" />

有很多基础的映射。Id和列映射了一个简单的列值对应一个简单的属性或是一个简单的数据类型（String，int，double，Date等等）。

有个区别便是id将会作为一个辨识符标注结果，使其和别的对象相比较。它帮助提升整体性能，尤其是缓存和结果映射的性能（如join映射）。

在这集合中，PODCASTS表的列会和Podcast bean的属性发生些什么：

* 列属性展示了数据库的列名或是列标签的别名-这个是相同的字符串，可以通过resultSet.getString（列名）方法获得。

· 属性会展示域或是列结果映射。如果一个JavaBeans的属性和给出的名字相匹配，它就会碑额使用。相反，MyBatis会在域中寻找一个之前被给出的名字。两种情况中，你能使用经过复杂协调后的属性作为常用标记。举个例子，你能简单的映射一些东西，如：username或者更复杂的：address.street.number。

不管你信不信，这就是它。它可能看起来有很多东西，实际上只不是是一个简单的SELECT，不过一旦你学会了基础的使用方法，它会变得非常简单，你也会获得强力帮助，我会在未来提供的一些文章，在那时会提供一些学习指南。

## 3. 总结

好的，以上就是我要说的。你已经学习过了MyBatis的内容，了解了它是如何和Spring，MyBatis-Spring相结合的，也了解了如何在Spring的应用上下文中进行配置以及如何在Spring应用中使用它。

## 4. 资源

1. [MyBatis](http://mybatis.github.io/mybatis-3/" \o "MyBatis" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)
2. [MyBatis-Spring](http://mybatis.github.io/spring/index.html" \o "MyBatis-Spring" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)
3. [Spring Data Access with JDBC](http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/jdbc.html" \o "Data access with JDBC" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)
4. [MyBatis-Blog](http://blog.mybatis.org/" \o "MyBatis blog" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)

**[JPA 实现类型: 数据访问对象](https://xebia.com/blog/jpa-implementation-patterns-data-access-objects/)**

JPA是Java持久层的API,是Java EE 5的一部分，已经被Hibernate，TopLink，EclipseLink，OpenJPA及一部分的对象关系映射（ORM）框架所实现。因为JPA 是被设计成EJB3.0的起源，你能在EJB3.0应用中使用。但是他和EJB3.0外部一样有效，举个例子：Spring 应用。 甚至是Gavin King，Hibernate的设计者也推荐在Hibernate的第二版中使用JPA，使用Hibernate作为Java的持久层，明显比只用JPA要好得多。

一旦你克服了了你对注解的恐惧，你会发现在接口中大量的解释对象和方法语法，这些对象协同工作的方法以及你如何去实现他们。当你执着于hello-world形式的程序，它看起来就十分直白了。当你开始在真实的应用中写代码时，你回发现事情不是那么简单。JPA提供的抽象部分是有漏洞的，它是给你的应用中的更大的部分而非只是给你的数据访问对象（DAO）及你的实体对象。你需要下决定如何解决事务，慢加载，对象解除（think web框架），继承等等。它证明了书和文章对你来说不是那么的有用。

至少这是我开始使用JPA以来第一时间发现的。在接下来的几周内，我想讨论的是JPA的几种实现方式。

### **我们真的需要一个DAO吗?**

所以，在你的JPA应用中，你的开头应该是这样的：数据访问对象(DAO)。一个有趣的用具，在我们思考我们是否需要一个DAO来决定是否使用JPA的时候。在一年前，讨论的结果是“需要”，而且非常难以讨论出这样一个结果。我想出了一个将DAO放到JPA应用程序的主意。值得争议的是是否应该把一个小的层放到JPA的顶端，但是更重要的是创建一个DAO实体类来提供以下几个优点：

* 在你每次想要存储或者加载数据时，代替使用实体管理方法，你可以决定使用DAO，你和你的团队都会在这个决定中收益很多。
* 你能禁止很多实体类的具体操作。举个例子，你可能永远都不想让你的代码去移除日志实体。当你使用DAO的时候，你只需要不将移除日志实体的方法添加到日志DAO中即可。
* 理论上来说，提供使用DAO，你能够转换到另一个持久层系统（像是普通JDBC或者是iBATIS）。但因为JPA具有抽象性漏洞，我觉得它不适合在应用中处理一个复杂的场景。你可以在一个单独的实体中添加追踪功能或者是性能保持策略。
* 你能集中所有的查询到一个具体的实体类中而不是在你的代码中四处分离他们。你能使用命名好的查询来使其绑定到实体类中，但你仍然需要一个核心的地方来设置正确的参数。将查询和代码参数整合，在DAO中返回一个正确的类型看上去是一件很简单的事情，举个例子：

|  |  |
| --- | --- |
| 1  2  3  4  5 | public List<ChangePlan> findExecutingChangePlans() {        Query query = entityManager.createQuery(              "SELECT plan FROM ChangePlan plan where plan.state = 'EXECUTING'");        return (List<ChangePlan>) query.getResultList();  } |

所以当你决定是否使用DAO时，为何不先写一下他们？Spring的Jpa模板的Java文档强调，在使用特定类是没有那么多的要点。换句话说，通过使用 @PersistenceContext上下文注解，你能写你的JPA DAO作为一个POJO来获取一个实体管理类。它可以在EJB3.0容器和Spring 2.0及以上容器中使用，只要你添加[PersistenceAnnotationBeanPostProcessor](http://static.springframework.org/spring/docs/2.5.x/api/org/springframework/orm/jpa/support/PersistenceAnnotationBeanPostProcessor.html) bean到你的Spring上下文中。

### **类型安全的普通DAO形式**

因为每个DAO都对其他DAO分享了基础数据，看起来就像是继承了一个分享数据的基本类一样，子类都能共用父类的方法。外面有很多博客写了关于类型安全的呕吐DAO形式，你甚至能从谷歌下载到他们的源代码。当我们加这些资源中的元素相结合后，我们就能给DAO实现JPA接口。

### **实体类**

让我们看看我们如下的订单持久类：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | *@Entity*  *@Table*(name = "ORDERS")  public class Order {  *@Id*  *@GeneratedValue*  private int id;  private String customerName;  private Date date;    public int getId() { return id; }  public void setId(int id) { this.id = id; }    public String getCustomerName() { return customerName; }  public void setCustomerName(String customerName) { this.customerName = customerName; }    public Date getDate() { return date; }  public void setDate(Date date) { this.date = date;}  } |

不用对这个类的细节过于担心。我们会在JPA的实现形式中再次访问的。这里的使用@Table 注解是因为订单ORDER在SQL中是一个保留关键字。

### **DAO接口**

首先我们定义了一个普通的DAO接口来让所有的DAO获取数据：

|  |  |
| --- | --- |
| 1  2  3  4  5 | public interface Dao<K, E> {        void persist(E entity);        void remove(E entity);        E findById(K id);  } |

第一个参数K是key，第二个参数E是实体类。然后是persist、remove和findById方法，你可能想要添加一个返回List的findAll方法。但是就像实体类本身一样，我们会在之后JPA实现形式的DAO方法里再次访问。

然后我们为每个实体定义了一个子接口，写下我们想要的具体方法。举个例子，我们可能想要通过日期来查询信息，我们可以添加如下方法：

|  |  |
| --- | --- |
| 1  2  3 | public interface OrderDao extends Dao<Integer, Order> {  List<Order> findOrdersSubmittedSince(Date date);  } |

### **基本DAO 实现**

第三步是创建一个基本的JPA DAO实现。它可以在标准DAO接口中实现我们在第一步里创建的所有方法：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | public abstract class JpaDao<K, E> implements Dao<K, E> {  protected Class<E> entityClass;    *@PersistenceContext*  protected EntityManager entityManager;    public JpaDao() {  ParameterizedType genericSuperclass = (ParameterizedType) getClass().getGenericSuperclass();  this.entityClass = (Class<E>) genericSuperclass.getActualTypeArguments()[1];  }    public void persist(E entity) { entityManager.persist(entity); }    public void remove(E entity) { entityManager.remove(entity); }    public E findById(K id) { return entityManager.find(entityClass, id); }  } |

大部分的实现都比较直白，有几点需要注意：

* JPA DAO的构造函数需要包括获取实体类的反射。
* @PersistenceContext 注解可以让EJB3.0容器或者Spring容器注入实体控制。
* 实体管理类及实体类域是被保护的，所以只有继承才访问到他们。

### **特定的DAO实现**

最后我们来创建一个特定的DAO实现。它继承了JPA DAO的基本类来实现特定DAO接口：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class JpaOrderDao extends JpaDao<Integer, Order> implements OrderDao {  public List<Order> findOrdersSubmittedSince(Date date) {  Query q = entityManager.createQuery(  "SELECT e FROM " + entityClass.getName() + " e WHERE date >= :date\_since");  q.setParameter("date\_since", date);  return (List<Order>) q.getResultList();  }  } |

### **使用DAO**

你可以像这样获取一个订单DAO的实例，依靠使用EJB3.0或者Spring。在EJB3.0中，我们通常使用这样的注解：

|  |  |
| --- | --- |
| 1  2 | *@EJB*(name="orderDao")  private OrderDao orderDao; |

在Spring中，我们能使用XML来定义bean或者使用autowired注解，像是这样：

|  |  |
| --- | --- |
| 1  2 | *@Autowired*  public OrderDao orderDao; |

在任何情况下，一旦我们获取了DAO的实例，我们可以像这样使用：

|  |  |
| --- | --- |
| 1  2  3  4 | Order o = new Order();  o.setCustomerName("Peter Johnson");  o.setDate(new Date());  orderDao.persist(o); |

但是我们也可以使用实体来完成特定的查询，只要在订单DAO接口里添加如下方法：

|  |  |
| --- | --- |
| 1  2  3  4 | List<Order> orders = orderDao.findOrdersSubmittedSince(date);  for (Order each : orders) {  System.out.println("order id = " + each.getId());  } |

使用类型安全的DAO形式，我们能获得如下好处：

* 客户端代码不直接依赖JPA接口。
* 通过使用类型安全，我们仍能实现DAO来解决事情。
* 团队可以统一逻辑JPA代码。
* 可以定位事务，debug，文件处理等等。我们可以添加事务处理到我们应用的别的部分。
* 当需要测试数据库连通的时候，我们可以再次实现JPA接口来访问子类。

我希望这篇文章能教你如何使用JPA来写DAO。

文章提供了JPA实现的形式。在这个系列的下一篇博客里，我们会继续讨论这个例子。与此同时我很乐意听到你们写出了你们自己的DAO代码！

**原文说明**

题名：Spring MyBatis integration example

JPA implementation patterns: Data Access Objects

作者：Codingpedia

xebia

来源：

<http://www.codingpedia.org/ama/spring-mybatis-integration-example/>

<https://xebia.com/blog/jpa-implementation-patterns-data-access-objects/>

# Spring MyBatis integration example

# Contents

* [1. Why Mybatis?](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "1_Why_Mybatis)
  + [1.1. What is MyBatis?](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "11_What_is_MyBatis)
* [2. Spring MyBatis interaction](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "2_Spring_MyBatis_interaction)
  + [2.1. What is MyBatis-Spring?](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "21_What_is_MyBatis-Spring)
  + [2.2. Installation](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "22_Installation)
  + [2.3. Spring application context setup](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "23_Spring_application_context_setup)
  + [2.4. Injecting Mappers](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "24_Injecting_Mappers)
    - [2.4.1. Service Layer](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "241_Service_Layer)
    - [2.4.2. DAO layer](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "242_DAO_layer)
      * [2.4.2.1. Register mapper](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "2421_Register_mapper)
        + [2.4.2.1.1. With XML Config](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "24211_With_XML_Config)
  + [2.5. MyBatis configuration XML](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "25_MyBatis_configuration_XML)
    - [2.5.1. typeAliases](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "251_typeAliases)
    - [2.5.2. mappers](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "252_mappers)
  + [2.6. Mapper XML Files](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "26_Mapper_XML_Files)
    - [2.6.1. select](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "261_select)
    - [2.6.2. Result Map](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "262_Result_Map)
      * [2.6.2.1. id & result](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "2621_id_result)
* [3. Summary](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "3_Summary)
* [4. Resources](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "4_Resources)

## 1. Why Mybatis?

Short answer: simple, lightweight, open source, dynamic sql and sql control, previous iBATIS knowledge. Now let me elaborate a little bit on the subject. Back in the old days of Podcastmania.ro, see [Story of Podcastpedia.org](http://www.codingpedia.org/ama/story-of-podcastpedia-org/" \o "Story of Podcastpedia.org" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank), I used my own MVC like framwork based on servlets to develop the web application and plain old [JDBC](http://en.wikipedia.org/wiki/Java_Database_Connectivity" \o "Wikipedia - JDBC" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank) to access the database. After “upgrading” to Spring MVC, I started using Spring’s JdbcTemplate for database access, which removed some of the boilerplate code. Later I got involved in projects where database access occured via iBATIS – Hibernate was there for a long time, but because of legacy reasons and no database normalization whatsoever, iBATIS was the optimal choice. By about the same time MyBatis had been just launched, so I read the documentation, did a pilot, liked it and switched from Spring’s JdbcTemplate to MyBatis. In the mean time I’ve been working on projects with Hibernate and JPA 2.0 with Hibernate used for persistence, so I’d say I have a pretty good overview on the most popular Java Persistence Frameworks. You have currently four major options:

* JPA/Hibernate
* myBatis (former iBatis)
* Spring JDBC
* JDBC

Every approach has its pros and cons, but if I had to choose all over again a technology for Podcastpedia.org, I would choose MyBatis. You can find lots of resources on the web, lots of reviews, tutorials, pros and cons for using one technology over the other and so on – do your research and find out what is best persistence solution for your context. I also advise watching the following video, unfortunately the sound is pretty bad, but the content is really interesting :

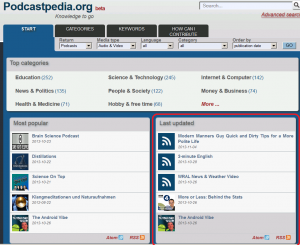
### 1.1. What is MyBatis?

So now that I have made my choice, let’s see first what is MyBatis. Well according to the official website, “MyBatis is a first class persistence framework with support for custom SQL, stored procedures and advanced mappings. MyBatis eliminates almost all of the JDBC code and manual setting of parameters and retrieval of results. MyBatis can use simple XML or Annotations for configuration and map primitives, Map interfaces and Java POJOs (Plain Old Java Objects) to database records.”

Enough with talking, let’s focus now on the main topic of this post, which is to find out how Spring and MyBatis can interact.

## 2. Spring MyBatis interaction

For the sake of simplicity, in this post I will present a simple example, which explains what needs to be implemented and configured to retrieve the newest(recently updated) podcasts from the database via MyBatis with Spring. You can experience the end result “live” by visiting the [homepage of Podcastpedia.org](https://github.com/Codingpedia/podcastpedia/" \o "Last updated podcasts from Podcastpedia.org" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank):

[](http://www.codingpedia.org/wp-content/uploads/2013/10/last_updated_podcasts.png)

Print screen home page Podcastpedia.org

If you are familiar with iBATIS (predecessor of MyBatis), you might know that until version 3, the Spring Framework provided direct integration with iBATIS SQL Maps in terms of resource management, DAO implementation support, and transaction strategies. But by the time iBATIS became MyBatis, Spring 3 development was already over, and the Spring team did not want to release with code based on a non-released version of MyBatis, official Spring support would have to wait. Given the interest in Spring support for MyBatis, the MyBatis community decided it was time to reunite the interested contributors and add Spring integration as a community sub-project of MyBatis instead. This is how MyBatis-Spring project was born, which is also used throughout [Podcastpedia.org](https://github.com/Codingpedia/podcastpedia" \o "Podcastpedia.org, knowledge to go" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)

### 2.1. What is MyBatis-Spring?

MyBatis-Spring integrates MyBatis seamlessly with Spring. This library allows MyBatis to participate in Spring transactions, takes care of building MyBatis mappers and SqlSessions and inject them into other beans, translates MyBatis exceptions into Spring DataAccessExceptions, and finally, it lets you build your application **code free** of dependencies on MyBatis, Spring or MyBatis-Spring.

### 2.2. Installation

To use the MyBatis-Spring module, you just need to include the mybatis-spring-x.x.x.jar file and its dependencies in the [classpath](http://en.wikipedia.org/wiki/Classpath_(Java)" \o "Wikipedia classpath java" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank).

If you are using Maven just add the following dependency to your pom.xml:

<!-- MyBatis integration -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.2.1</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.2.3</version>

</dependency>

### 2.3. Spring application context setup

To use MyBatis with Spring you need at least two things defined in the Spring application context: an SqlSessionFactory and at least one mapper interface.

In MyBatis-Spring, an SqlSessionFactoryBean is used to create an SqlSessionFactory. Every MyBatis application centers around an instance of SqlSessionFactory. You use the SqlSessionFactory to create an SqlSession. Once you have a session, you use it to execute your mapped statements, commit or rollback connections and finally, when it is no longer needed, you close the session. With MyBatis-Spring you don’t need to use SqlSessionFactory directly because your beans can be injected with a thread safe SqlSession that automatically commits, rollbacks and closes the session based on Spring’s transaction configuration.

To configure the factory bean, put the following in the Spring XML configuration file:

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="configLocation" value="classpath:config/mybatisV3-config.xml"/>

</bean>

Notice that the SqlSessionFactory requires a DataSource. This can be any DataSource and should be configured just like any other Spring database connection. For [Podcastpedia.org](https://github.com/Codingpedia/podcastpedia" \o "Podcastpedia.org, knowledge to go" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank) the DataSource is configured via [JNDI](http://en.wikipedia.org/wiki/Java_Naming_and_Directory_Interface" \o "Wikipedia - JNDI" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank):

<!-- ========================= DATASOURCE DEFINITION via JNDI ========================= -->

<!-- When resourceRef is true, the value of jndiName will be prepended with

server’s JNDI directory. Consequently, the actual name used will be

java:comp/env/jdbc/pcmDB. -->

<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean" scope="singleton">

<property name="jndiName" value="java:comp/env/jdbc/pcmDB" />

<property name="resourceRef" value="true" />

</bean>

See my post [Tomcat JDBC Connection Pool configuration for production and development](http://www.codingpedia.org/ama/tomcat-jdbc-connection-pool-configuration-for-production-and-development/" \o "Tomcat JDBC Connection Pool configuration for production and development" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank), to find out how the database resource is configured for production and development/testing environments.

The location parameter specifies the location of the MyBatis config file. You will see in a coming section, [MyBatis configuration XML,](http://www.codingpedia.org/ama/spring-mybatis-integration-example/" \l "mybatis_config_file) how this file looks like.

### 2.4. Injecting Mappers

Rather than code data access objects (DAOs) manually using SqlSessionDaoSupport or SqlSessionTemplate, Mybatis-Spring can create a thread safe mapper that you can inject directly into other bean. I like to call mappers also DAOs.

#### 2.4.1. Service Layer

I will start at the Service Layer, which lies before the DAO layer. The service layer implementation class – StartPageServiceImpl, which loads the newest podcasts in the model to be displayed on the homepage, will be injected the PodcastDao mapper:

<!-- =================== Service beans configuration ================== -->

<bean id="startPageService" class="org.podcastpedia.service.impl.StartPageServiceImpl">

<property name="podcastDao" ref="podcastDao"/>

</bean>

Once the mapper is injected it can then be used in the business logic:

public class StartPageServiceImpl implements StartPageService {

private static final Integer NUMBER\_OF\_PODCASTS\_IN\_CHART = 5;

private static Logger LOG = Logger.getLogger(StartPageServiceImpl.class);

private PodcastDao podcastDao;

public void setPodcastDao(PodcastDao podcastDao) {

this.podcastDao = podcastDao;

}

@Cacheable(value="newestAndRecommendedPodcasts", key="#root.method.name")

public List<Podcast> getNewestPodcasts() {

List<Podcast> newestPodcasts = podcastDao.getNewestPodcasts(NUMBER\_OF\_PODCASTS\_IN\_CHART);

for(Podcast p : newestPodcasts) {

Episode lastEpisode = episodeDao.getLastEpisodeForPodcast(p.getPodcastId());

p.setLastEpisode(lastEpisode);

}

return newestPodcasts;

}

..............

}

Notice that there are no SqlSession or MyBatis references at the service layer. Nor is there any need to create, open or close the session, MyBatis-Spring will take care of that. We’ll see in the next section how I implemented and configured the PodcastDao.

**Note:** If you want to find out how the @Cacheable annotation is configured, visit my post [Spring caching with Ehcache](http://www.codingpedia.org/ama/spring-caching-with-ehcache/" \o "Codingpedia.org - Spring caching with Ehcache" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank).

#### 2.4.2. DAO layer

##### 2.4.2.1. Register mapper

You can register the mapper either using a classical XML configuration or the new 3.0+ Java Config (a.k.a. @Configuration). I prefer the classical way with XML configuration, as I like to hold the configuration as separate as possible from the code:

###### 2.4.2.1.1. With XML Config

The mapper/dao is registered to Spring by including a MapperFactoryBean in the application context XML config:

<!-- =============== MyBATIS beans configuration ================== -->

<bean id="podcastDao" class="org.mybatis.spring.mapper.MapperFactoryBean">

<property name="sqlSessionFactory" ref="sqlSessionFactory"/>

<property name="mapperInterface" value="org.podcastpedia.dao.PodcastDao" />

</bean>

The Mapper BeanFactory can be set up with a SqlSessionFactory, as here, or a pre-configured SqlSessionTemplate. You saw how the SqlSessionFactory was configured in the section Installation/Spring application context setup. You can also download the complete Spring configuration file for the DAO layer.

The second parameter – mapperInterface – sets the mapper interface of the MyBatis mapper. Note that the mapper class specified **must** be an interface, NOT an actual implementation class:

package org.podcastpedia.dao;

import java.util.Date;

import java.util.List;

import java.util.Map;

import org.podcastpedia.domain.Comment;

import org.podcastpedia.domain.Podcast;

import org.podcastpedia.domain.Tag;

/\*\*

\* Interface for database access

\*

\* @author ama

\*

\*/

public interface PodcastDao {

/\*\*

\* Returns the newest podcasts (ORDER BY last\_updated DESC)

\*

\* @param numberOfPodcasts (number of podcasts to be returned)

\* @return

\*/

public List<Podcast> getNewestPodcasts(Integer numberOfPodcasts);

....................

}

If the PodcastDao had a corresponding MyBatis XML mapper file in the same classpath location as the mapper interface, it would have been parsed automatically by the MapperFactoryBean. But because they are different places in the classpath locations, the parameter had to be specified.

Let’s see now how the sql statement is configured and implemented in MyBatis.

### 2.5. MyBatis configuration XML

As mentioned in the beginning of the post the SqlSessionFactoryBean has the configLocation parameter that defines where the MyBatis configuration resides. Here is an extract from the configuration file that is relevant for the example presented here:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE configuration

PUBLIC "-//[mybatis.org//DTD](http://mybatis.org//DTD) Config 3.0//EN"

"<http://mybatis.org/dtd/mybatis-3-config.dtd>">

<configuration>

<typeAliases>

<typeAlias type="org.podcastpedia.domain.Podcast" alias="Podcast"/>

.......

</typeAliases>

<mappers>

<mapper resource="maps/PodcastMapper.xml" />

........

</mappers>

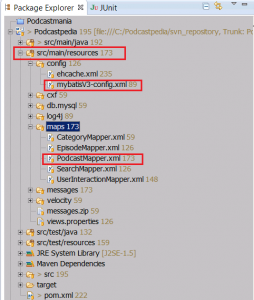
</configuration>

#### 2.5.1. typeAliases

A type alias is simply a shorter name for a Java type. It’s only relevant to the XML configuration and simply exists to reduce redundant typing of fully qualified classnames which include package names. As you’ll see in the following section, when referencing a org.podcastpedia.domain.Podcast object, I will just use “Podcast”.

#### 2.5.2. mappers

Configuration elements which tell MyBatis where to find the mappers. Java doesn’t really provide any good means of auto-discovery in this regard, so the best way to do it is to simply tell MyBatis where to find the mapping files. You can use class path relative resource references, or literal, fully qualified url references (including file:/// URLs):

[](http://www.codingpedia.org/wp-content/uploads/2013/10/MyBatis-files.png)

MyBatis configuration file and mappers location

**Note:** There are many other parameters you can set up in the configuration file for MyBatis. See the [Configuration](http://mybatis.github.io/mybatis-3/configuration.html" \o "MyBatis configuration" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank) documentation page for more details.

### 2.6. Mapper XML Files

The true power of MyBatis is in the Mapped Statements. This is where the magic happens. For all of their power, the Mapper XML files are relatively simple. Certainly if you were to compare them to the equivalent JDBC code, you would immediately see a savings of 95% of the code. MyBatis was built to focus on the SQL, and does its best to stay out of your way.

Here I will present a snippet from the PodcastMapper.xml that shows how the PodcastDao interface method is mapped:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mapper PUBLIC "-//[mybatis.org//DTD](http://mybatis.org//DTD) Mapper 3.0//EN"

"<http://mybatis.org/dtd/mybatis-3-mapper.dtd>">

<mapper namespace="org.podcastpedia.dao.PodcastDao">

<!-- result maps -->

<resultMap id="podcastsMap" type="Podcast" >

<id column="podcast\_id" property="podcastId" />

<result column="url" property="url" />

<result column="rating" property="rating" />

<result column="numberRatings" property="number\_ratings" />

<result column="number\_visitors" property="numberOfVisitors" />

<result column="DESCRIPTION" property="description" />

<result column="PODCAST\_IMAGE\_URL" property="urlOfImageToDisplay" />

<result column="TITLE" property="title" />

<result column="last\_episode\_url" property="lastEpisodeMediaUrl" />

<result column="title\_in\_url" property="titleInUrl" />

<result column="publication\_date" property="publicationDate"/>

</resultMap>

<select id="getNewestPodcasts" resultMap="podcastsMap" parameterType="Integer">

SELECT

PODCAST\_ID,

URL,

NUMBER\_VISITORS,

DESCRIPTION,

PODCAST\_IMAGE\_URL,

TITLE,

last\_episode\_url,

title\_in\_url,

publication\_date

FROM

podcasts

WHERE

availability=200

ORDER BY publication\_date DESC

limit 0, #{value};

</select>

.....

</mapper>

Notice the mapper’s namespace value at line 5 – org.podcastpedia.dao.PodcastDao and the id of the <select> element at line 21. If you combine, it results org.podcastpedia.dao.PodcastDao.getNewestPodcasts which is exactly the method of the interface defined at the DAO layer.

#### 2.6.1. select

The select statement is one of the most popular elements that you’ll use in MyBatis. Putting data in a database isn’t terribly valuable until you get it back out, so most applications query far more than they modify the data. For every insert, update or delete, there is probably many selects. This is one of the founding principles of MyBatis, and is the reason so much focus and effort was placed on querying and result mapping.

The statement exposed here is called getNewestPodcasts (same as corresponding method in the PodcastDao interface), takes a parameter of type Integer(same as input of the corresponding method in PodcastDao interface) and has a resultMap parameter of value "podcastsMap", which is a named reference to an external resultMap.

#### 2.6.2. Result Map

The resultMap element is the most important and powerful element in MyBatis. It’s what allows you to do away with 90% of the code that JDBC requires to retrieve data from ResultSets, and in some cases allows you to do things that JDBC does not even support. In fact, to write the equivalent code for something like a join mapping for a complex statement could probably span thousands of lines of code. The design of the ResultMaps is such that simple statements don’t require explicit result mappings at all, and more complex statements require no more than is absolutely necessary to describe the relationships.

Let’s talk a little about the structure of the resultMap. The top element has an id(=”podcastMap”) which uniquely identifies the select in the mapper (namespace), and is referenced in the **select**, and a type(=”Podcast”), which is a fully qualified Java class name, or a type alias (this is the case here, "Podcast" being the type alias defined in the MyBatis configuration file)

##### 2.6.2.1. id & result

<id column="podcast\_id" property="podcastId" />

<result column="url" property="url" />

These are the most basic of result mappings. Both id, and column map a single column value to a single property or field of a simple data type (String, int, double, Date, etc.).

The only difference between the two is that id will flag the result as an identifier property to be used when comparing object instances. This helps to improve general performance, but especially performance of caching and nested result mapping (i.e. join mapping).

What happens in this map is that some of the columns of the PODCASTS table are mapped to the properties of the Podcast bean:

* the column attribute represents the column name from the database, or the aliased column label – this is the same string that would normally be passed to resultSet.getString(columnName).

· the property attribute represents the field or property to map the column result to. If a matching JavaBeans property exists for the given name, then that will be used. Otherwise, MyBatis will look for a field of the given name. In both cases you can use complex property navigation using the usual dot notation. For example, you can map to something simple like: username, or to something more complicated like: address.street.number.

Believe it or not, that’s it. It might seem like a lot just do a simple SELECT, but once you learn the basics it becomes really simple and you get some powerful stuff, some of which I will present in a future post. And also is important to mention that the learning curve is pretty small.

## 3. Summary

Well, that’s it. You’ve learned about MyBatis, how it integrates with Spring via MyBatis-Spring, how to configure MyBatis in Spring’s application context and how to use it in a Spring application.

## 4. Resources

1. [MyBatis](http://mybatis.github.io/mybatis-3/" \o "MyBatis" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)
2. [MyBatis-Spring](http://mybatis.github.io/spring/index.html" \o "MyBatis-Spring" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)
3. [Spring Data Access with JDBC](http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/jdbc.html" \o "Data access with JDBC" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)
4. [MyBatis-Blog](http://blog.mybatis.org/" \o "MyBatis blog" \t "http://www.codingpedia.org/ama/spring-mybatis-integration-example/_blank)

**[JPA implementation patterns: Data Access Objects](https://xebia.com/blog/jpa-implementation-patterns-data-access-objects/)**

The [JPA](http://java.sun.com/developer/technicalArticles/J2EE/jpa/), short for Java Persistence API, is part of the Java EE 5 specification and has been implemented by [Hibernate](http://www.hibernate.org/397.html), [TopLink](http://www.oracle.com/technology/products/ias/toplink/jpa/index.html), [EclipseLink](http://www.eclipse.org/eclipselink/), [OpenJPA](http://openjpa.apache.org/), and a number of other [object-relational mapping](http://en.wikipedia.org/wiki/Object-relational_mapping) (ORM) frameworks. Because JPA was originally designed as part of the [EJB 3.0 specification](http://java.sun.com/products/ejb/docs.html), you can use it within an EJB 3.0 application. But it works equally well outside of EJB 3.0, for example [in a Spring application](http://static.springframework.org/spring/docs/2.5.x/reference/orm.html" \l "orm-jpa). And when even Gavin King, the designer of Hibernate, recommends using JPA in the second edition of Hibernate in Action, a.k.a. [Java Persistence with Hibernate](http://www.manning.com/bauer2/), it’s obvious that JPA is here to stay.

Once you [get over your fear of annotations](https://xebia.com/blog/2009/01/19/maybe-annotations-arent-that-bad-after-all/) ;-), you find that there is plenty of literature out there that explains the objects and methods within the API, the way these objects work together and how you can expect them to be implemented. And when you stick to hello-world-style programs, it all seems pretty straight forward. But when you start writing your first ****real****application, you find that things are not so simple. The abstraction provided by JPA is pretty[leaky](http://www.joelonsoftware.com/articles/LeakyAbstractions.html) and has ramifications for larger parts of your application than just your [Data Access Objects](http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html) (DAO’s) and your domain objects. You need to make decisions on how to handle transactions, lazy loading, detached object (think web frameworks), inheritance, and more. And it turns out that the books and the articles don’t really help you here.  
  
At least, that is what I discovered when I really started using JPA for the first time. In the coming weeks, I would like to discuss the choices I came up against and the decisions I made and why I made them. When I’m done, we’ll have a number of what I would like to not-too-modestly call JPA implementation patterns.

### **Do we really need a DAO?**

So, let’s start with the thing you would probably write first in your JPA application: the data access object (DAO). An interesting point to tackle before we even start is [whether you even need a DAO when using JPA](http://www.infoq.com/news/2007/09/jpa-dao). The conclusion of that discussion more than a year ago was “It depends” and while it is very hard to argue with such a conclusion :-), I would like to stick with the idea that a DAO does have its place in a JPA application. Arguably it provides only a thin layer on top of JPA, but more importantly making a DAO per entity type gives you these advantages:

* Instead of having to pick the right [EntityManager](http://java.sun.com/javaee/5/docs/api/javax/persistence/EntityManager.html) method every time you want to store or load data, you decide which one to use once and you and your whole team can easily stick to that choice.
* You can disallow certain operations for certain entity types. For example, you might never want your code to remove log entries. When using DAO’s, you just do not add a remove method to your LogEntry DAO.
* Theoretically, by using DAO’s you could switch to another persistence system (like plain JDBC or iBATIS). But because JPA is such a leaky abstraction I think that that is not realistically possible for even a slightly complex application. You do get a single point of entry where you can add tracing features or keep performance statistics.
* You can centralize all the queries on a certain entity type instead of scattering them through your code. You could use [named queries](http://java.sun.com/javaee/5/docs/api/javax/persistence/NamedQuery.html) to keep the queries with the entity type, but you’d still need some central place where the right parameters are set. Putting both the query, the code that sets the parameters, and the cast to the correct return type in the DAO seems a simpler thing to do. For example:

|  |  |
| --- | --- |
| 1  2  3  4  5 | public List<ChangePlan> findExecutingChangePlans() {        Query query = entityManager.createQuery(              "SELECT plan FROM ChangePlan plan where plan.state = 'EXECUTING'");        return (List<ChangePlan>) query.getResultList();  } |

So when you decide you are going to use DAO’s, how do you go about writing them? The highlighted (in bold) comment in the Javadoc for Spring’s [JpaTemplate](http://static.springframework.org/spring/docs/2.5.x/api/org/springframework/orm/jpa/JpaTemplate.html) seems to suggest that there’s not much point in using that particular class, which also makes [JpaDaoSupport](http://static.springframework.org/spring/docs/2.5.x/api/org/springframework/orm/jpa/support/JpaDaoSupport.html)superfluous. Instead you can write your JPA DAO as a POJO using the [@PersistenceContext](http://java.sun.com/javaee/5/docs/api/javax/persistence/PersistenceContext.html)annotation to get an EntityManager reference. It will work in an EJB 3.0 container and it will work in Spring 2.0 and up if you add the [PersistenceAnnotationBeanPostProcessor](http://static.springframework.org/spring/docs/2.5.x/api/org/springframework/orm/jpa/support/PersistenceAnnotationBeanPostProcessor.html) bean to your Spring context.

### **The type-safe generic DAO pattern**

Because each DAO shares a lot of functionality with the other DAO’s, it makes sense to have a base class with the shared functionality and then subclass from that for each specific DAO. There are [a](http://www.ibm.com/developerworks/java/library/j-genericdao.html) [lot](http://www.greggbolinger.com/blog/2008/04/17/1208457000000.html) [of](http://forum.springsource.org/showthread.php?t=66139) [blogs](https://xebia.com/blog/2009/02/07/acessing-generic-types-at-runtime-in-java/) out there about such a type-safe generic DAO pattern and you can even download [some code from Google Code](http://code.google.com/p/generic-dao/). When we combine elements from all these sources, we get the following JPA implementation pattern for DAO’s.

### **The entity class**

Let’s say we want to persist the following Order class:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | *@Entity*  *@Table*(name = "ORDERS")  public class Order {  *@Id*  *@GeneratedValue*  private int id;  private String customerName;  private Date date;    public int getId() { return id; }  public void setId(int id) { this.id = id; }    public String getCustomerName() { return customerName; }  public void setCustomerName(String customerName) { this.customerName = customerName; }    public Date getDate() { return date; }  public void setDate(Date date) { this.date = date;}  } |

Don’t worry too much about the details of this class. We will revisit the specifics in other JPA implementation patterns. The @Table annotation is there because ORDER is a reserved keyword in SQL.

### **The DAO interfaces**

First we define a generic DAO interface with the methods we’d like all DAO’s to share:

|  |  |
| --- | --- |
| 1  2  3  4  5 | public interface Dao<K, E> {        void persist(E entity);        void remove(E entity);        E findById(K id);  } |

The first type parameter, K, is the type to use as the key and the second type parameter, E, is the type of the entity. Next to the basic persist, remove, and findById methods, you might also like to add a List findAll() method. But like the entity class itself, we will revisit the DAO methods in later JPA implementation patterns.

Then we define one subinterface for each entity type we want to persist, adding any entity specific methods we want. For example, if we’d like to be able to query all orders that have been added since a certain date, we can add such a method:

|  |  |
| --- | --- |
| 1  2  3 | public interface OrderDao extends Dao<Integer, Order> {  List<Order> findOrdersSubmittedSince(Date date);  } |

### **The base DAO implementation**

The third step is to create a base JPA DAO implementation. It will have basic implementation of all the methods in the standard Dao interface we created in step 1:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | public abstract class JpaDao<K, E> implements Dao<K, E> {  protected Class<E> entityClass;    *@PersistenceContext*  protected EntityManager entityManager;    public JpaDao() {  ParameterizedType genericSuperclass = (ParameterizedType) getClass().getGenericSuperclass();  this.entityClass = (Class<E>) genericSuperclass.getActualTypeArguments()[1];  }    public void persist(E entity) { entityManager.persist(entity); }    public void remove(E entity) { entityManager.remove(entity); }    public E findById(K id) { return entityManager.find(entityClass, id); }  } |

Most of the implementation is pretty straight forward. Some points to note though:

* The constructor of the JpaDao includes [the method proposed by my colleague Arjan Blokzijl](https://xebia.com/blog/2009/02/07/acessing-generic-types-at-runtime-in-java/) to use reflection to get the entity class.
* The @PersistenceContext annotation causes the EJB 3.0 container or Spring to inject the entity manager.
* The entityManager and entityClass fields are protected so that subclasses, i.e. specific DAO implementations, can access them.

### **The specific DAO implementation**

And finally we create such a specific DAO implementation. It extends the basic JPA DAO class and implements the specific DAO interface:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class JpaOrderDao extends JpaDao<Integer, Order> implements OrderDao {  public List<Order> findOrdersSubmittedSince(Date date) {  Query q = entityManager.createQuery(  "SELECT e FROM " + entityClass.getName() + " e WHERE date >= :date\_since");  q.setParameter("date\_since", date);  return (List<Order>) q.getResultList();  }  } |

### **Using the DAO**

How you get a reference to an instance of your OrderDao depends upon whether we use EJB 3.0 or Spring. In EJB 3.0 we’d use a annotation like this:

|  |  |
| --- | --- |
| 1  2 | *@EJB*(name="orderDao")  private OrderDao orderDao; |

while in Spring we can use the XML bean files or we can use autowiring like this:

|  |  |
| --- | --- |
| 1  2 | *@Autowired*  public OrderDao orderDao; |

In any case, once we have a reference to the DAO we can use it like this:

|  |  |
| --- | --- |
| 1  2  3  4 | Order o = new Order();  o.setCustomerName("Peter Johnson");  o.setDate(new Date());  orderDao.persist(o); |

But we can also use the entity specific query we added to the OrderDao interface:

|  |  |
| --- | --- |
| 1  2  3  4 | List<Order> orders = orderDao.findOrdersSubmittedSince(date);  for (Order each : orders) {  System.out.println("order id = " + each.getId());  } |

With this type-safe DAO pattern we get the following advantages:

* No direct dependency on the JPA api from client code.
* Type-safety through the use of generics. Any casts that still need to be done are handled in the DAO implementation.
* One logical place to group all entity-specific JPA code.
* One location to add transaction markers, debugging, profiling, etc. Although as we will see later, we will need to add transaction markers in other parts of our applications too.
* One class to test when testing the database access code. We will revisit this subject in a later JPA implementation pattern.

I hope this convinces you that you do need DAO’s with JPA.

And that wraps up the first JPA implementation pattern. In the next blog of this series we will build on this example to discuss the next pattern. In the meantime I would love to hear from you how you write your DAO’s!