第3章 Hibernate 配置和 SessionFactory 创建

本章要点

- Hibernate SessionFactory 的配置。
- Hibernate 连接数据库配置。
- Hibernate 生成 SQL 配置。
- Hibernate 检索数据库策略配置。

Hibernate 框架的核心之一就是 Hibernate 配置，通过配置文件确定 Hibernate 运行时的行为参数，避免了使用硬编码形式，提高了应用的可维护性。将来需要修改某些属性，直接修改 Hibernate 配置文件内容即可，不需要修改程序代码，再重新编译。

Hibernate 的配置内容主要包括 Hibernate 自身功能参数配置和类-表映射配置，本章只讲解 Hibernate 自身参数配置，第 4 章开始讲解类映射配置。通过本章的学习，读者应该理解 Hibernate 的配置项以及每项取值的含义，熟练掌握 Hibernate 的基本配置。

Hibernate 编程的核心是取得 SessionFactory 对象，在 Hibernate 开发中，可以使用多种方式取得此对象，对实际项目开发中，一般使用 Spring 集成 Hibernate 方式，通过 Spring 管理 Hibernate 的 SessionFactory 对象的创建。本章主要介绍使用 Hibernate 方式创建 SessionFactory 对象，重点了解其基本原理，掌握学习 Hibernate 编程，因为企业实际项目基本不用本章讲述的这些方式。

3.1 Hibernate 配置的功能

Hibernate 配置完成 Hibernate 连接数据库的参数设定、数据库类型的设定、查询二级缓存的配置、映射文件的引入、注释映射类的引入、各种检索策略的设置等等。

在完成 Hibernate 的配置信息后，就可以使用 Hibernate 的 Configuration 对象读取配置的信息，为创建 SessionFactory 做准备。

Hibernate 框架的核心配置信息和映射配置信息，是 Hibernate 的核心，在此基础上使用 Hibernate API 完成项目的持久化功能。

3.2 Hibernate 配置的方式

Hibernate 支持 XML 格式配置、属性文件（Properties）和编程方式三种配置方式。实际项目开发中推荐的方式是使用 XML 格式，以其语法格式清晰、功能强大、便于修改等优点使其成为配置的首选。

1．XML 格式文件配置

Hibernate 应用开发时，最常见的配置方式就是使用 XML 文件格式，既可以使用默认的 hibernate.cfg.xml，也可以使用自定义的配置文件名。使用 Configuration 配置对象的方法 configure()用于读取配置文件并对其进行解析，为创建 SessionFactory 提供所需信息。

2．属性文件配置

属性文件配置方式具有简洁、配置代码编写少的优点，因为不需像 XML 文件那样需要众多的<property>标记，而是直接以 propertyname=value 进行配置即可，因此属性文件通常要比 XML 文件要小得多，但是属性文件的缺点是无法进行映射文件或映射类的引入。在实际项目中可以结合属性文件和 XML 文件配置，在属性文件中配置 Hibernate 属性参数，而在 XML 文件中引入映射文件和映射类。

3．编程方式配置

此种配置方式是不推荐的方式，但可在项目的测试阶段，通过编程方式加载需要的配置参数和映射文件，检查 ORM 的映射语法是否正确。通过编程逐步增加映射文件或映射类，可以测试出哪个映射文件或映射类有问题。

3.3 Hibernate XML 方式配置

Hibernate XML 配置方式通过使用 XML 格式的配置文件对 Hibernate 的各种特征属性进行配置，默认情况下，启动 Hibernate 的 Configuration 配置对象后，调用其 configure()方法会自动读取 classpath 根目录下的 hibernate.cfg,xml 文件，其读取的编程代码如下：

```
Configuration cfg=new Configuration();
cfg.configure();
```

也可以使用其他名称的配置文件，甚至将配置文件保存在一个指定包的文件夹内。假如创建的配置文件名为 hibernate.oa.xml, 将此保存在包 com.city.oa.config 中，则创建配置对象后，通过带参数的 configure()方法实现对此配置文件的读取，其示意代码如下：

```
Configuration cfg=new Configuration();
cfg.configure("com.city.oa.config.Hibernate.oa.xml");
```

企业实际项目开发中，如果采用 Spring 整合 Hibernate，一般不需要此 Hibernate 配置文件，直接在 Spring 配置文件中配置 Hibernate 的配置信息。单独开发 Hibernate 应用时，通常直接使用 hibernate.cfg.xml 配置文件。

如果使用 Eclipse 开发工具开发 Hibernate Java 项目，在项目的 Java Resources 的 src 目录下创建 hibernate.cfg.xml，项目的目录结构和配置文件的位置参见图 3-1。

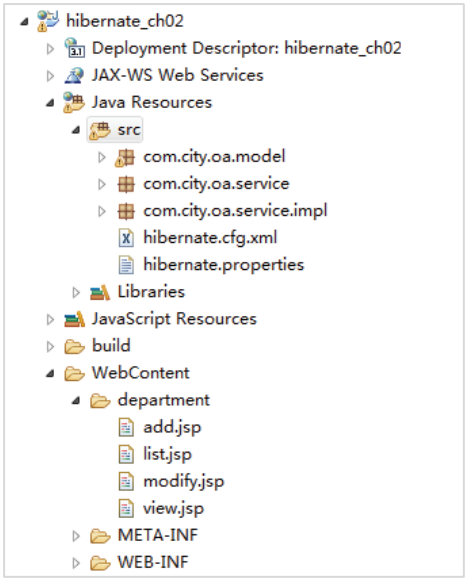

图 3-1　Hibernate 项目目录结构和配置文件位置

创建的 hibernate.cfg.xml 代码案例参见程序 3-1。其文件结构主要包含如下内容。

（1）XML 文件头标志。

（2）Hibernate 配置文件的 DTD。

（3）根元素标记<hibernate-configuration>。

（4）SessionFactory 配置元素标记<session-factory>。

（5）各种 Hibernate 的配置属性标记。

程序 3-1　hibernate.cfg.xml //Hibernate 核心配置 XML 文件

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
       "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
       "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
       <!-- 使用 Tomcat 数据源连接 MSSQL 数据库 -->
       <property name="hibernate.connection.datasource">java:comp/env/cityoa
       </property>
       <!-- SQL 方言 -->
       <property  name="hibernate.dialect">org.hibernate.dialect.MySQLDialect
       </property>
       <!-- 启用 Hibernate 自动会话环境管理-->
       <property name="current_session_context_class">thread</property>
       <!-- 不启用 Hibernate 二级缓存  -->
```

```
        <property  name="cache.provider_class">org.hibernate.cache.internal.
        NoCacheProvider</property>
        <!-- 显示生成的 SQL 语句 -->
        <property name="show_sql">true</property>
    <!-- 引入类与表映射文件 -->
        <mapping resource="com/city/oa/value/DepartmentValue.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Hibernate XML 配置文件的属性配置的语法是<property name="属性名" value="属性值" />，属性名是固定的，由 Hibernate 框架确定，取值需要根据 Hibernate 的文档进行配置。

Hibernate XML 配置文件除属性配置外，还可以配置引入映射信息的标记<mapping>，通过此标记，可以引入 XML 的映射文件和包含了映射注释信息的持久类，如下代码演示了如何引入 XML 映射文件和包含映射注释的类文件。

```
<mapping resource="com/city/oa/value/DepartmentModel.hbm.xml"/>
<mapping class="com.city.oa.value.DepartmentModel"/>
```

3.4 Hibernate 属性方式配置

Hibernate 还支持使用属性文件的方式配置，配置的文件名是 hibernate.properties，并存储在类路径(classpath)根目录内。当创建 Hibernate 的 Configuration 配置对象时，该对象会自动读取类根目录下的 hibernate.properties 文件，读取其配置的属性参数值，对 Hibernate 的 SessionFactory 的特性进行定制。

配置文件内使用 name=value 的方式配置 Hibernate 启动时需要的参数，其简单的案例代码参见程序 3-2。

程序 3-2　hibernate.properties //Hibernate 属性配置文件

```
hibernate.connection.driver_class=com.mysql.jdbc.Driver
hibernate.connection.url=jdbc:mysql://localhost:3306/cityoa
hibernate.connection.username=root
hibernate.connection.password=root
hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

属性文件配置方式的缺陷是无法指定映射文件，在使用 Configuration 类创建 SessionFactory 时，不得不手动编程引入类映射文件，因此在实际项目中很少使用属性文件配置方式，都倾向于使用 XML 的配置文件。由于 Hibernate 的众多属性都有其默认的参数值，一般情况下，需要配置的属性参数不是很多。

3.5 Hibernate 编程配置方式

Hibernate 支持无须配置文件的纯编程方式进行 Hibernate 配置，该方式通过

Configuration 对象的各种方法完成属性的设定，映射 XML 文件的引入和采用 Java 注释方式的持久类的引入。

Configuration 类提供的设置属性的方法如下。

1．setProperty（String name,String value）

该方法每次设定一个 Hibernate 属性，参数 name 指定 Hibernate 的属性名，value 指定属性的取值。如下代码演示了使用此方法设置数据库连接信息属性。

```
cfg.setProperty("hibernate.connection.driver_class","com.mysql.jdbc.Driver");
cfg.setProperty("hibernate.connection.url","jdbc:mysql://localhost:3306
/cityoa");
cfg.setProperty("hibernate.connection.username","root");
cfg.setProperty("hibernate.connection.password","root");
cfg.setProperty("connection.pool_size","1");
cfg.setProperty("hibernate.dialect","org.hibernate.dialect.MySQLDialect");
```

2．setProperties（properties）

该方法通过传入的 Properties 的集合对象，可以同时设定多个 Hibernate 属性参数。将上面单次设定属性的方法改造为使用 Properties 对象的示意代码如下。

```
Configuration cfg=new Configuration();
Properties props=new Properties();
props.setProperty("hibernate.connection.driver_class",  "com.mysql.jdbc.
Driver");
props.setProperty("hibernate.connection.url",  "jdbc:mysql://localhost:3306
/cityoa");
props.setProperty("hibernate.connection.username", "root");
props.setProperty("hibernate.connection.password", "root");
props.setProperty("connection.pool_size", "1");
props.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQLDialect");
cfg.setProperties(props);
cfg.addResource("com/city/oa/model/DepartmentModel.hbm.xml");
```

通过比较，使用单次设定属性方式与使用 Properties 对象的编程量基本相当，使用任何一种方式均可。

3.6 Hibernate 配置的内容

Hibernate 配置文件的核心内容是数据库连接配置，支持各种方式连接数据库，在此基础上，Hibernate 的配置包括缓存的设置、检索机制的配置等，其配置的内容包括如下项目。

（1）数据库连接配置。

（2）数据库方言配置。

（3）生成数据表配置。

（4）显示生成 SQL 语句配置。

（5）事务处理的配置。

（6）Hibernate 缓存的配置。

3.7　数据库连接配置

Hibernate 提供了许多属性用于设定与数据库的连接，因为 Hibernate 是持久层框架，与数据库连接是核心的配置。Hibernate 支持多种连接数据库的配置方式，包括 Hibernate 自己管理的数据库连接，或使用其他著名的连接池框架（如 C3P0、Proxocol 等），也可以使用 JavaEE 应用服务器管理的数据库连接池。

3.7.1　使用 JDBC 驱动类连接数据库

Hibernate 框架自身可以直接使用 JDBC 驱动连接数据库，并有内置的简化的连接池管理机制，Hibernate 在使用数据库连接时，都是以连接池方式进行管理的。但此内置连接池管理机制缺少其他连接池框架拥有的各种特性和高性能，无法适应高并发和对性能要求较高的场合，因此在实际项目部署场合，不能使用其内置的这种数据库连接池，而是使用服务器管理的连接池或第三方著名的连接池框架。

Hibernate 提供的使用 JDBC 确定的属性名和取值参见表 3-1。

表 3-1　Hibernate JDBC 数据库连接属性

属　性　名	说　　明	取　　值
hibernate.connection.driver_class	驱动类	连接数据库的 JDBC 驱动类名
hibernate.connection.url	连接 URL	JDBC 连接地址
hibernate.connection.username	数据库用户名	数据库用户名
hibernate.connection.password	用户密码	数据库用户的密码
hibernate.connection.pool_size	连接池个数	初始的连接个数，整数
hibernate.connection.isolation	连接隔离级别	取值 1，2，4 或 8

使用 JDBC 方式配置数据库连接的 XML 配置代码示意如下。

```
<session-factory>
 <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
 <property name="connection.url">jdbc:mysql://localhost:3306/cityoa</property>
 <property name="connection.username">root</property>
 <property name="connection.password">city62782116</property>
 <property name="connection.pool_size">1</property>
</session-factory>
```

3.7.2　使用 JavaEE 服务器管理的连接池连接数据库

对于开发要部署到应用服务器上的 JavaEE 企业级应用，Hibernate 应该优先使用服务

器配置的数据库连接池，因为服务器配置的数据库连接池通常具有非常优秀的性能，与服务器的融合也是最佳的。

JavaEE 服务器都提供了自身特有的数据库连接池配置机制，一些大型的企业级应用服务器如 Oracle WebLogic、IBM WebSphere、Oralce GlassFish 等都提供了 Web 操作界面来实现连接池的配置，而一些开源的小型应用服务器基本都使用配置文件方式，在指定的配置文件中实现对数据库连接池的配置。以著名的 Tomcat 服务器为例，其数据库连接池的配置在安装目录的/config/context.xml 文件中完成，其配置的语法参见如下代码：

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>
<Resource
     name="cityoa"
     auth="Container"
     type="javax.sql.DataSource"
     driverClassName="com.mysql.jdbc.Driver"
     maxIdle="2"
     maxWait="5000"
     url="jdbc:mysql://localhost:3306/cityoa"
     username="root"
     password="root"
     maxActive="20"
     removeAbandoned="true"
     removeAbandonedTimeOut="5"
     logAbandoned="true" />
</Context>
```

上面的代码中在<Context>根元素下使用<Resource>元素配置一个连接到 MySQL 数据库 cityoa 的数据库连接池，最大连接个数为 50，默认连接个数为 2，注册到服务器命名服务中的注册名为 cityoa，Hibernate 框架可通过此注册名使用配置的数据库连接池。

Hibernate 使用 JNDI 查找 JavaEE 应用服务器命名服务中注册的数据库连接池管理对象 DataSource 的相应属性名和参数值如表 3-2 所示。

表 3-2　Hibernate 使用服务器管理连接池的属性

属　性　名	说　　明	取　　值
hibernate.connection.datasource	数据源注册名	注册的数据源名
hibernate.jndi.class	JNDI 实现类	Context 的实现类名
hibernate.jndi.url	JNDI 服务器 URL	命名服务器的位置 url
hibernate.jndi	JNDI 参数	创建 Context 实现类对象的参数

项目开发中部署到应用服务器的 Hibernate 应用项目通常使用服务器内置的命名服务器，即本地 JNDI 命名服务器，此时只需指定 hibernate.connection.datasource 属性即可，其值指定服务器连接池配置的 JNDI 服务名，如下示意代码演示了使用 Tomcat8 配置的数据

库连接池的 Hibernate 配置案例。

```
<session-factory>
 <property name="connection.datasource">java:comp/env/cityoa</property>
</session-factory>
```

代码中的连接池数据源注册名是在配置的<Resource>的 name 属性值 cityoa 的基础上增加 Tomcat 命名服务前缀 java:comp/env 结合而成的，这是 Tomcat 的语法，对应其他应用服务器是不需要的。每个服务器都有自己的 JNDI 命名语法，具体配置信息需要参阅不同服务器的文档，如 GlassFish 直接使用配置的数据源名称即可，不需要添加任何前缀。

通过数据源方式配置连接数据库，不再需要配置其他的账号、密码和连接池大小等信息，因为这些信息已经在服务器的配置文件中。

3.7.3　使用连接池框架 C3P0 连接数据库

如果开发的项目不部署在 JavaEE 应用服务器上，当然这种情况是很少见的，因为现代企业级项目基本都运行在各种应用服务器上。或者在项目的测试阶段，为加快项目的开发进度，通常使用单元测试类，测试使用 Hibernate 编程完成的各种对象的方法，不会把项目部署到服务器再测试，而是单独编写的测试类进行测试。

此时由于代码不是在服务器上运行，因此不能使用服务器配置的数据库连接池和 JNDI 命名服务，项目开发人员会使用第三方提供的连接池开源框架（如 C3P0、Hikaricp、Pproxool 等）。从前经常使用的 Apache 的连接池管理框架 DBCP 由于其缺陷和性能较差，最新版的 Hibernate 已经不再支持使用 DBCP，这点开发者一定要注意。

C3P0 框架由于其全面支持 JDBC4 版本的新特性和优良的连接池管理性能，被 Hibernate 框架选为最佳搭档，在 Hibernate 类库中已经包含了 C3P0 的类库文件和 Hibernate 整合文件，保存在 Hibernate 的压缩文件包里的\lib\optional\c3p0 文件夹内，其包含的与 C3P0 有关的类库文件参见图 3-2。

图 3-2　Hibernate 内置的 C3P0 类库文件

Hibernate 配置使用 C3P0 时，只需在原来配置使用 JDBC 驱动方式基础上，增加使用 C3P0 的属性即可，Hibernate 框架启动时会检测到 C3P0 属性，自动启动 C3P0 连接池框架，同时终止 Hibernate 内部自带的连接池管理机制。下面给出了 Hibernate 配置 C3P0 连接池框架的配置代码。

```
<session-factory>
 <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
 <property name="connection.url">jdbc:mysql://localhost:3306/cityoa</property>
```

```
 <property name="connection.username">root</property>
 <property name="connection.password">root</property>
 <property name="connection.pool_size">1</property>
 <!-- 最大连接数 -->
 <property name="hibernate.c3p0.max_size">20</property>
 <!-- 最小连接数 -->
 <property name="hibernate.c3p0.min_size">5</property>
 <!-- 获得连接的超时时间,如果超过这个时间,会抛出异常，单位秒 -->
 <property name="hibernate.c3p0.timeout">120</property>
 <!-- 最大的 PreparedStatement 的数量 -->
 <property name="hibernate.c3p0.max_statements">100</property>
 <!-- 每隔 120 秒检查连接池里的空闲连接，单位是秒-->
 <property name="hibernate.c3p0.idle_test_period">120</property>
 <!-- 当连接池里面的连接用完的时候，C3P0 一下获取的新的连接数 -->
 <property name="hibernate.c3p0.acquire_increment">2</property>
 <!-- 每次都验证连接是否可用 -->
 <property name="hibernate.c3p0.validate">true</property>
</session-factory>
```

3.7.4 使用连接池框架 Proxool 连接数据库

Proxool 是一种 Java 数据库连接池技术，是 sourceforge 下的一个开源项目，这个项目提供一个健壮、易用的连接池，最为关键的是这个连接池提供监控功能，方便易用，便于发现连接泄露的情况。

由于 Proxool 框架是在其他框架基础上开发的，吸收了其他连接池框架的优点，增加了监控连接内存泄露功能，因此得到了开发者的青睐。

Hibernate 使用 Proxool 框架，需要首先配置 Proxool 的配置文件 Proxool.xml，并将其保存在与 hibernate.cfg.xml 相同的 classpath 根目录下。

在 Proxool.xml 文件内，配置了管理数据库连接的基本信息，包括驱动、URL、账号和密码信息，其配置信息参见程序 3-3。

程序 3-3 Proxool.xml //Proxool 连接池配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- the proxool configuration can be embedded within your own application's.
Anything outside the "proxool" tag is ignored. -->
<something-else-entirely>
  <proxool>
    <alias>CITYOAPool</alias>
<driver-class>com.mysql.jdbc.Driver</driver-class>
<driver-url>jdbc:mysql://localhost:3306/cityoa</driver-url>
    <driver-properties>
      <property name="user" value="root"/>
      <property name="password" value="root"/>
    </driver-properties>
```

```
    <maximum-connection-count>10</maximum-connection-count>
    <house-keeping-test-sql>select CURRENT_DATE</house-keeping-test-sql>
  </proxool>
</something-else-entirely>
```

Proxool 框架在启动时，自动检索 classpath 根目录下的 Proxool.xml 配置文件，设置数据库连接信息。

Proxool 配置后，在 Hibernate 的配置文件 hibernate.cfg.xml 中配置使用 Proxool 连接池框架的信息，指定其配置文件。程序 3-4 演示了二者的整合配置信息。

程序 3-4　hibernate.cfg.xml //使用 Proxool 的 Hibernate XML 配置文件

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
       "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
       "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
       <!-- Database connection settings -->
       <property  name="hibernate.connection.provider_class">org.hibernate.
       connection.ProxoolConnectionProvider</property>
       <property name="hibernate.proxool.pool_alias">CITYOAPool</property>
       <property name="hibernate.proxool.xml">Proxool.xml</property>
       <!-- SQL dialect -->
       <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
       <!-- Echo all executed SQL to stdout -->
       <property name="show_sql">false</property>
       <!-- Drop and re-create the database schema on startup -->
       <property name="hbm2ddl.auto">create</property>
       <!-- 引入持久类映射文件 -->
       <mapping resource="com/city/oa/model/DepartmentModel.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

在 Hibernate 与 Proxool 整合的类库文件 hibernate-proxool-5.2.3.Final.jar 中定义了 Hibernate 集成 Proxool 的属性如下。

（1）hibernate.proxool.pool_alias。

该属性指定了 Proxool 配置的连接池的别名，此处是 CITYOAPool。

（2）Hibernate.proxool.xml。

该属性指定 Proxool 配置文件的名称，此处为 Proxool.xml，由于将其保存在 classpath 根目录下，不需要指定文件夹，直接使用文件名即可。

3.8　Hibernate 数据库类型属性配置

由于不同数据库在 SQL 语法上有所区别，需要 Hibernate 生成针对不同数据库生成不同的 SQL 语句，Hibernate 使用方言属性设定数据库的类型。方言的属性名称是：

hibernate.dialect。Hibernate 框架内置了不同数据库类型的方言实现类，表 3-3 是 Hibernate 内置的不同数据库的方言实现类。设定该属性时，指定其方言类名即可，配置的语法代码如下所示。

```
hibernate.dialect=数据库方言实现类名
```

表 3-3　Hibernate 框架数据库方言实现类

数据库类型	方言实现类
DB2	org.hibernate.dialect.DB2Dialect
DB2 AS/400	org.hibernate.dialect.DB2400Dialect
DB2 OS390	org.hibernate.dialect.DB2390Dialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
MySQL	org.hibernate.dialect.MySQLDialect
MySQL with InnoDB	org.hibernate.dialect.MySQLInnoDBDialect
MySQL with MyISAM	org.hibernate.dialect.MySQLMyISAMDialect
Oracle	org.hibernate.dialect.OracleDialect
Oracle 9i/10g	org.hibernate.dialect.Oracle9Dialect
Sybase	org.hibernate.dialect.Oracle9Dialect
Sybase Anywhere	org.hibernate.dialect.SybaseDialect
Microsoft SQL Server	org.hibernate.dialect.SybaseAnywhereDialect
SAP DB	org.hibernate.dialect.SQLServerDialect
Informix	org.hibernate.dialect.SAPDBDialect
HypersonicSQL	org.hibernate.dialect.InformixDialect
Ingres	org.hibernate.dialect.HSQLDialect
Progress	org.hibernate.dialect.IngresDialect
Mckoi SQL	org.hibernate.dialect.ProgressDialect

在本书中使用 MySQL 数据库，其方言的配置代码如下：

```
<property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

3.9 Hibernate 处理检索属性

Hibernate 提供了执行 SQL 批处理操作时策略的设定，通过这些检索策略属性的设定，可以提高 Hibernate 执行批处理的执行速度，进而提高系统的性能。Hibernate 配置中与检索有关的属性及其取值和含义参见表 3-4。

表 3-4　Hibernate 设置 DML 批处理性能的属性

属　性　名	取　　值	说　　明
hibernate.jdbc.batch_size	数值（如 10）	设定批处理个数
hibernate.order_inserts	true|false	设定是否将批处理 insert 语句进行排序，默认为 false
hibernate.order_updates	true|false	设定是否将批处理 update 语句进行排序，默认为 false
hibernate.jdbc.batch_versioned_data	true|false	是否指定在实体中增加版本信息，默认为 true

3.10　Hibernate 查询批处理设定属性

Hibernate 对执行查询的操作也提供了批处理设定属性，用于提高 Hibernate 执行检索时的速度，这些属性的设定需要根据不同的数据库和服务器的内存情况进行设定，一般要在项目测试和运行过程中进行不断的调整，以达到最佳的处理性能。Hibernate 用于设定查询批处理的参数属性参见表 3-5。

表 3-5　Hibernate 设置查询检索批处理性能的属性

属　性　名	取　　值	说　　明
hibernate.jdbc.fetch_size	0 或者数值	设定 JDBC 的 Statement 读取数据的时候每次从数据库中取出的记录条数。当 Fetch Size=50 的时候，性能会提升 1 倍之多，当 Fetch Size=100，性能还能继续提升 20%，Fetch Size 继续增大，性能提升的就不显著了
hibernate.max_fetch_depth	0～3	设定当执行对一关联的查询时，外关联抓取树结构的最大深度。值为 0 意味着将关闭默认的外连接抓取。设为 1 或更高值能启用 one-to-one 和 many-to-oneouter 关联的外连接抓取，它们通过 fetch="join"来映射
hibernate.jdbc.use_scrollable_resultset	true|false	当读取结果集数据时，是否启动可滚动类型的结果集，默认是 false，启用只读向前的结果集类型
hibernate.jdbc.use_streams_for_binary	true|false	当读取 binary 类型字段数据时，是否启用 Stream 流方式，默认是 false
hibernate.jdbc.use_get_generated_keys	true|false	是否启用 JDBC3 预编译 SQL 执行对象的生成主键值的方法
hibernate.jdbc.wrap_result_sets	true|false	启用是否保证查询结果集以加快查询处理速度。默认为 false
hibernate.enable_lazy_load_no_trans	true|false	在事务之外初始化一个代理对象或关联的容器。通常不要设定为 true

通常使用属性文件方式设定 Hibernate 的这些批处理增加、修改、删除或检索策略的属性，而在 XML 文件中设定连接数据库的属性。当不需要这些属性定义时，可直接修改属性文件的名称，如果将 hibernate.properties 文件修改为 hibernate01.properties，则 Hibernate 框架将不再读取此属性文件。

3.11　SQL 日志追踪属性

在测试 Hibernate 应用时，经常需要查看 Hibernate 生成的 SQL 语句进行代码的检查和验证，Hibernate 提供了生成 SQL 日志的选项属性。有关 SQL 日志的属性参见表 3-6。

表 3-6 Hibernate 设置 SQL 日志的属性

属 性 名	取 值	说 明
hibernate.show_sql	true|false	是否启用SQL生成日志，启用后将向日志文件生成select语句，如果没有指定日志设定，则在JVM控制台输出生成的SQL语句
hibernate.format_sql	true|false	是否生成格式化的SQL语句。每个子句单独为一行，便于查看复杂的SQL语句
hibernate.use_sql_comments	true|false	是否启用SQL中的注释生成。有利于开发者进行错误的调试

3.12 缓存策略设定属性

Hibernate 提供处理性能的关键是其提供了执行持久化操作时的数据缓存机制，如在查询操作时，如果多次检索同一个实体对象，且缓存中该实体对象已经存在，则 Hibernate 不会执行对数据库表的 Select 操作，而是直接取得缓存中保存的对象并返回，因而性能得以显著提高。

Hibernate 有二级缓存机制，当操作持久对象时，先在一级缓存中查找此对象，如果没有再到二级缓存中查找，如果这二个缓存中都没有要操作的持久对象，才对数据库执行 select 语句，检索出需要的持久对象。

Hibernate 一级缓存又称为“Session 的缓存”。Session 不能被卸载，Session 的缓存是事务范围的缓存（Session 对象的生命周期通常对应一个数据库事务或者一个应用事务），一级缓存是无法通过配置参数进行配置的，Hibernate 只提供了对二级缓存的配置参数。

Hibernate 二级缓存是一个可插拔的缓存插件，它由 SessionFactory 负责管理。由于 SessionFactory 对象的生命周期和应用程序的整个过程对应，因此第二级缓存是进程范围或者集群范围的缓存。这个缓存中存放的对象的松散数据。第二级对象有可能出现并发问题，因此需要采取适当的并发访问策略，该策略为被缓存的数据提供了事务隔离级别。缓存适配器用于把具体的缓存实现软件与 Hibernate 集成。第二级缓存是可选的，可以在每个类或每个集合的粒度上配置第二级缓存。

Hibernate 的一级缓存是其框架内部设定，不能终止一级缓存，而二级缓存是专门用于查询时保存检索对象的缓存，可以使用多个缓存框架产品用于 Hibernate 的二级缓存的实现。有关缓存方面的设定属性参见表 3-7。

表 3-7 Hibernate 设置缓存机制的属性

属 性 名	取 值	说 明
hibernate.cache.use_query_cache	true|false	是否启动查询缓存
hibernate.cache.use_second_level_cache	true|false	是否启动二级缓存机制
hibernate.cache.query_cache_factory	缓存实现类的全名	设定二级缓存机制的实现机制

如下代码演示了 Hibernate 中启用二级缓存机制，使用 EhCache 作为缓存实现机制，并指定使用 oa.ehcache.xml 文件作为缓存配置文件。

```
<!-- 开启二级缓存 -->
<property name="hibernate.cache.use_second_level_cache">true</property>
<!-- 二级缓存的提供类 在 hibernate4.0 版本以后我们都是配置这个属性来指定二级缓存的提
供类-->
<property  name="hibernate.cache.region.factory_class">org.hibernate.cache.
ehcache.EhCacheRegionFactory</property>
<!-- 二级缓存配置文件的位置 -->
<property name="hibernate.cache.provider_configuration_file_resource_path">
oa.ehcache.xml</property>
```

3.13　事务处理和并性控制设定属性

Hibernate 内部使用 JDBC API 完成数据库的操作，将持久类对象保存到对应的数据表中或从表中读取记录的字段值到持久对象中。默认情况下 Hibernate 使用 JDBC Connection 接口提供的事务处理机制通过数据库提供的事务处理服务完成数据处理的事务操作。

在 JavaEE 服务器环境下开发企业级应用时，Hibernate 也可以使用 JavaEE 提供的 JTA 事务机制完成事务处理，而不是使用 JDBC API 提供的。JTA 提供了在分布式环境下跨越多个数据库的分布式事务机制，而这是 JDBC API 所不具备的。

在 Hibernate 配置中通过表 3-8 所示的与事务处理相关的属性，实现对 Hibernate 事务处理的配置，包括使用何种事务处理方式以及事务处理平台的选择。由于与事务处理有关的配置参数较多，本书只重点展示了最常用的几个配置参数，其他参数请参阅 Hibernate 的在线文档。

表 3-8　Hibernate 设置事务处理的配置属性

属　性　名	取　　值	说　　明
hibernate.transaction.coordinator_class	jdbc|jta	设定选择事务处理的机制类型，默认是 jdbc
hibernate.transaction.jta.platform	Borland|Bitronix|JBossAS|JBossTS|JOTM|Weblogic|WebSphere|Resin	当选择事务处理为 jta 时，选择提供 JTA 服务的服务器类型

Hibernate 在没有配置事务处理类型选项 hibernate.transaction.coordinator_class 参数时，自动使用 JDBC API 提供的事务处理机制，此机制下，会使用使用数据库提供的内置的事务完成 Hibernate 应用的事务处理。

如果选择 JTA 事务处理类型，可根据 Hibernate 应用具体使用哪种服务器，通过设定 hibernate.transaction.jta.platform 对 JTA 的实现机制进行详细的指定，例如如果应用运行 WebLogic 服务器上，则选择此参数值为 Weblogic，各个服务器的取值参见表 3-8。

3.14　取得 Hibernate SessionFactory 的方式

完成 Hibernate 配置后，Hibernate 编程最重要的步骤就是读取配置信息，然后创建

SessionFactory 对象。SessionFactory 负责创建 Session 实例，可以通过 Configuration 实例构建 SessionFactory。

Configuration 实例 config 会根据当前的数据库配置信息，构造 SessionFacory 实例并返回。SessionFactory 一旦构造完毕，即被赋予特定的配置信息。配置信息的任何变更将不会影响到已经创建的 SessionFactory 实例 sessionFactory。

SessionFactory 保存了对应当前数据库配置的所有映射关系，同时也负责维护当前的二级数据缓存和 SQL 执行对象 Statement 的缓存池，加快对象的持久化操作。SessionFactory 的创建过程非常复杂，创建时间长，对应用的性能影响巨大。

在系统设计中一定要使用 SessionFactory 的重用策略，由于 SessionFactory 采用了线程安全的设计，可由多个线程并发调用。如果项目开发中选择如程序 3-5 所示的方式取得 Hibernate 的 SessionFactory 对象，在每个业务方法中都单独创建配置对象再读取配置信息，最后调用 Configuration 对象的 buildSessionFactory()创建 SessionFactory，如此每个业务方法都会对配置文件和映射文件进行解析，这些操作都需要很长时间才能完成，将会极大地减慢应用的运行速度，影响系统的性能。

程序 3-5 DepartmentServiceImpl.java //部门业务实现类

```
//使用 Hibernate API 完成部门增加功能
package com.city.oa.service.impl;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.city.oa.model.DepartmentModel;
import com.city.oa.service.IDepartmentService;
//部门业务实现类
//使用 Hibernate API 完成
public class DepartmentServiceImpl implements IDepartmentService {
    //增加部门业务方法
    public void add(DepartmentModel dm) throws Exception {
        Configuration cfg=new Configuration();
        cfg.configure();
        SessionFactory sf=cfg.buildSessionFactory();
        Session session=sf.openSession();
        Transaction tx=session.beginTransaction();
        session.save(dm);
        tx.commit();
        session.close();
        sf.close();
    }
}
```

实际项目开发时整个应用使用一个单例的 SessionFactory 对象即可，不需要每个线程都创建一个单独的 SessionFactory 对象。下面将分别讲述不同创建或取得 SessionFactory 的

方法，并比较这些创建方式的优点和对性能的影响。

3.14.1　原型模式取得 SessionFactory 对象

使用原型模式取得 SessionFactory 对象，就是每次调用 Hibernate API 对象完成持久化功能时，都创建单独的 SessionFactory 对象，使用完后就将其销毁，程序 3-5 已经演示了在业务方法内使用原型模式创建 SessionFactory 对象，在实际应用编程中，应该避免使用此种模式创建该对象，应该使用单例共享模式，使得整个应用项目共享一个 SessionFactory 对象。

3.14.2　单例工厂模式取得 SessionFactory 对象

由于 SessionFactory 对象的创建时间长，占用的资源多，如果每个业务对象的方法内当使用此对象时都创建新的对象实例，会极大地影响系统的性能，应该采用单例模式编程，保证整个系统公用此对象。程序 3-6 演示了通过单例模式取得 SessionFactory 方法。

程序 3-6　SessionFactoryUitl.java　//SessionFactory 单例模式工厂

```
package com.city.oa.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
//SessionFactory 单例工厂
public class SessionFactoryUtils {
    //类变量 SessionFactory
    private static SessionFactory sessionFactory=null;
    static{
        Configuration cfg=new Configuration();
        cfg.configure();
        sessionFactory=cfg.buildSessionFactory();
    }
    //取得 SessionFactory
    public static SessionFactory getSessionFactory(){
        return sessionFactory;
    }
}
```

通过定义类的静态属性变量，并使用 static 静态代码块，只创建一个 SessionFactory 对象，所有需要 SessionFactory 对象的代码只需调用此类的静态方法即可取得单例的、可供多线程共享的实例对象，新的取得 SessionFactory 对象的代码示意如下。

```
SessionFactory sessionFactory=SessionFactoryUtils.getSessionFactory();
```

取得单例的 SessionFactory 对象后，就可以创建非单例的 Session 对象，实现并发的持久化操作。

3.14.3 使用 Hibernate 内置的 Session-Facotry-Name 属性配置的 JNDI 取得 SessionFactory 对象

使用上述单例工厂模式取得 SessionFactory 对象，依然存在无法持久取得单例的对象问题。依据 Java 对象的管理机制，当一个对象在 JVM 内存创建后，如果没有其他对象引用时候，会被 Java 虚拟机的垃圾收集器标注，在线程空闲时，被垃圾收集器回收，对象被销毁，内存被释放，下次再使用 SessionFactory 对象时，还需要载入其工厂类，创建新的单例对象。如何能保证创建的 SessionFactory 对象永久被引用，需要一定的机制实现对象该对象的引用，JavaEE 提供了命名服务机制实现对指定对象的永久引用。

应用开发人员可以手动编程，在创建 SessionFactory 对象后，将其注册到应用服务器的命名服务中，以后当需要此对象时，通过 JNDI API 从命名服务中取得已经绑定的对象，不需重新创建，进而快速取得该对象，提高系统的性能。

一般要保证在应用启动时，自动创建 SessionFactory 对象并注册到命名服务中，要使用自启动机制实现。在 Java Web 应用项目中，最佳的实现方式是使用服务器启动监听器，当服务器启动后，立即执行创建 SessionFactory 对象并注册到命名服务中的任务。程序 3-7 给出通过 Java Web ServletContextListener 监听器完成这一任务的代码实现。

程序 3-7 WebApplicationStartupListener.java //Web 站点启动监听器

```
package com.city.oa.listenenr;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
@WebListener
public class WebApplicationStartupListener implements ServletContextListener {
    //服务器启动监听方法
    public void contextInitialized(ServletContextEvent event){
        Configuration cfg=new Configuration();
        cfg.configure();
        //取得创建的 SessionFactory 对象
        SessionFactory sessionFactory=cfg.buildSessionFactory();
        try {
            //连接到本地命名服务
            Context ctx=new InitialContext();
            //注册创建的 SessionFactory 对象到命名服务
            ctx.bind("sessionFactory",sessionFactory);
            ctx.close();

        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
    //服务器停止监听方法
    public void contextDestroyed(ServletContextEvent event){
    }
}
```

当 Web 应用启动后，SessionFactory 对象被注册到命名服务中，可以修改程序 3-7 中的 SessionFactoryUitl 工厂类，将取得 SessionFactory 对象的方法改造为使用 JNDI 从命名服务中取得，而不再是从读取配置信息阶段开始，又进一步提高了系统的性能，因为查找对象的时间要比创建对象少得多，速度更快，性能更好，程序 3-8 为使用 JNDI 后查找 SessionFactory 改进后的实现代码。

程序 3-8　SessionFactoryUtilsWithJNDI.java

```
package com.city.oa.util;
import javax.naming.Context;
import javax.naming.InitialContext;
import org.hibernate.SessionFactory;
public class SessionFactoryUtilsWithJNDI {
    //取得 SessionFactory
    public static SessionFactory getSessionFactory() throws Exception{
        Context ctx=new InitialContext();
        SessionFactory sessionFactory=(SessionFactory)ctx.lookup
        ("sessionFactory");
        ctx.close();
        return sessionFactory;
    }
}
```

在业务方法内要实现对象的持久化，可以使用此工厂类取得 SessionFactory 对象，再使用 Hibernate 其他 API 对象完成，下面给出了重新改造后的部门增加的业务方法代码。

```
public class DepartmentServiceImpl implements IDepartmentService {
    //增加部门业务方法
    public void add(DepartmentModel dm) throws Exception {
        SessionFactory sf=SessionFactoryUtilsWithJNDI.getSessionFactory();
        Session session=sf.openSession();
        Transaction tx=session.beginTransaction();
        session.save(dm);
        tx.commit();
        session.close();
        sf.close();
    }
}
```

经过以上代码升级改造后，应用系统在持久化的处理性能要比原来每次直接读取配置文件后创建 SessionFactory 的方式有巨大的提高。由于将 SessionFactory 对象注册到命名服务的方式已经成为流行的处理方式，Hibernate 框架提供了配置 SessionFactory 注册到命名服务的配置参数：hibernate.session_factory_name，当在配置文件中配置此参数后，当首次创建 SessionFactory 对象后，该对象被 Hibernate 框架自动注册到命名服务中，不再需要手动编程，如下代码演示了将创建的 SessionFactory 对象以 OASessionFactory 注册到命名服务中。

```
hibernate.session_factory_name=OASessionFactory
```

如果配置此参数，则程序 3-7 的 Web 服务器启动监听器的监听代码就不需要手动连接命名服务了，此部分代码可以改为如下代码所示。

```
public void contextInitialized(ServletContextEvent event){
       Configuration cfg=new Configuration();
       cfg.configure();
       //取得创建的 SessionFactory 对象
       SessionFactory sessionFactory=cfg.buildSessionFactory();
}
```

如此原来连接命名服务和注册对象的代码就可以省略了，首次创建 SessionFactory 对象后，Hibernate 负责自动完成对象的命名服务注册的工作，不再需要编程。

等使用 Spring 框架后，一般都使用 Spring 管理 SessionFactory，将此对象的管理交由 Spring IoC 容器完成，就不需要 Hibernate 的这种机制了，这也是实际项目开发最常用的管理 SessionFactory 的方式。

本章小结

本章中详细讲述了 Hibernate 框架的各种配置方式，包括 XML 配置文件方式、属性文件方式和手动编程方式，在实际应用开发中，最常使用的 XML 文件方式，该方式可以直接进行映射文件或映射类的引入。在使用 Hibernate 开发持久化应用中，各种参数的配置是重点，通过使用 Hibernate 提供的各种配置参数，可以对 Hibernate 进行精确的配置和设定，最常用的配置是有关数据库的配置，以及各种检索策略参数的设定。

在开始学习 Hibernate 时，需要重点掌握的是数据库的连接配置，以及使用各种数据库连接框架的配置，尤其是 C3P0 的配置，待逐渐对 Hibernate 的使用熟练以后，可以尝试使用各种与性能调整有关的配置参数，逐步提高系统的处理速度和性能。

在掌握了 Hibernate 的配置后，就需要熟练掌握映射配置，它是 Hibernate 的核心，是实现 Hibernate 基本功能的基础，以下各章从简单映射开始，逐渐深入讲述复杂的关联映射，最终完成复杂的数据模型的应用的设计和编程。