第三章 Hibernate的配置

Hibernate是被设计提供广泛的参数配置来适应不同的数据库环境。幸运的是，多数的默认设置值的含义是简而易见的。你同样可以在Hibernet根目录下的etc/文件夹下的一份使用示例文件hibernate.properties中看到具体的参数配置。你可以将这一个示例文件直接拷贝到你的类路径下，然后根据你的个人需求来适当的修改它。

3.1 编程式配置

一个org.hibernate.cfg.Configuration的实例代表了一个Java应用中全部的数据库中的数据表在程序中的映射文件。org.hibernate.cfg.Configuration被用于建立一个持久的(不可变的)org.hibernate.SessinFactory对象实例。这些关系映射则被Hibernate编译为各种不同的XML映射文件。

你可以得到一个org.hibernate.cfg.Configuration实例，通过实例化一系列具体指定的XML映射文档 。如果这个映射文件在类路径下，你必须使用addResource()方法来指定这个文件。例如

```
Configuration cfg = new Configuration()
                .addRexource("Item.hbm.xml")
                .addResource("Bid.hbm.xml");
```

另外一种可以选择的方法是指定被映射的类并允许Hibernate发现这些文件

```
Configuration cfg = new Configuration()
                .addClass("org.hibernate.auction.Item.class")
                .addClass("org.hibernate.auction.Bid.class");
```

Hibernate会搜索在类路径下的映射文件名为：/org/hibernate/auction/Item.hbm.xml和/org/hibernate/auction/Bid.hbm.xml的文件。这个方法可以消除任何硬编码的文件名。

一个org.hibernate.cfg.Configuration对象同样允许你可以指定多个配置属性值。例如

```
Configuration cfg = new Configuration()
        .addClass(org.hibernate.auction.Item.class)
        .addClass(org.hibernate.auction.Bid.class)
        .setPropety("hibernate.dialect","org.hibernate.dialect.MyAQLInnoDBDialect")
        .setPropety("hibernate.connection.datasource","java:comp/env/jdbc/test")
        .setPropety("hibernate.order_updates","true");
```

这个并不是唯一一种配置Hibernate的方法。下面还有几种可以选择用于配置Hibernate的方法

1 - 通过实例化一个java.util.properties来读取配置文件，并将这些个读取到的属性值注入到Configuration.setPropetys()中。
2 -在类路径的根目录下新建一个名为hibernate.properties的文件。

3 -设置系统属性通过使用java -Dproperty = value的方式 4 - 在hibernate.cfg.xml文件下包括

元素节点。(稍后我们会讲到)
如果你想要快速开始一个项目，那么建立hibernate.properties时最为简单的方法.org.hibernate.cfg.Configuration只有在程序刚开始有用，一旦一个SessinFactory被建立，那么这个对象就会被丢弃。

3.2 得到一个SessionFactory对象 当所有的映射文件被org.hibernate.cfg.Configuration解析后，这个应用程序必须要为org.hibernate.cfg.Session实例提供一个工厂函数。这个工厂会被这个应用程序的所有线程所共享：

```
SessionFactory sessions = cfg.buildSessionFactory();
```

Hibernate允许你的应用程序得到多个org.hibernate.SessionFactory实例，这对于你在一个程序中使用了多个的数据库有非常大的好处。

3.3 JDBC 连接

这是被官方推荐的做法是用一个org.hibernate.SessionFactory来创造一个JDBC连接池。如果你采用的是这一种方法，那么应按照一下来打开一个org.hibernate.Session：

```
Session session = sessions.openSession();//穿件一个新的session
```

一旦你的开始一个需要连接数据库的任务，那么你就可以从数据连接池中得到一个JDBC连接。

在此之前，你首先需要通过Hibernate的配置来通过JDBC的连接。所有的Hibernate属性名称和语义都被定义在org.hibernate.cfg.Environment.这个类之中。对JDBC最重要的连接配置如下文的概述。如果你设置了以下的属性，Hibernate就可以通过使用java.sql.DriverManager来得到数据库连接池。

表 3.1 Hibernate JDBC 属性

属性名称	属性值
hibernate.connection.driver_class	JDBC驱动类
hibernate.connection.url	JDBC的URL
hibernate.connection.username	数据库用户名
hibernate.connection.password	数据库用户密码
hibernate.connection.pool_size maximum	连接池最大的连接数目

Hibernate拥有自己最基本的连接池算法。但是这个连接池只是为了减轻初学者的学习压力，他并不适用于实际的生产环境，甚至不适用于性能测试。你应该使用第三方的连接池，来获得更好的性能和稳定性。使用第三方连接池，你只需要为hibernate.connection.pool_size指定合适的属性值，这样也可以同时关闭掉Hibernate的内部连接池。举个例子，你可以使用C3P0。

C3P0是一个开源的JDBC连接池，它被和Hibernate打包在一起放在lib目录下。如果你设置了hibernate.c3p0.* 的属性， Hibernate会使用它的org.hibernate.connection.C3P0ConnectionProvider来作为连接池。如果你想使用Proxool（另外一种Java数据库连接池技术）相关的hibernate.properties，并上Hibernate的网站了解更多的信息。
以下是使用C3P0作为数据库连接池的hibernate.properties的配置的例子

```
hibernate.connection.driver_class = org.postgresql.Driver
hibernate.connection.url = jdbc:postgresql://localhost/mydatabase
hibernate.connection.username = myuser
hibernate.connection.password = secret
hibernate.c3p0.min_size=5
hibernate.c3p0.max_size=20
hibernate.c3p0.timeout=1800
hibernate.c3p0.max_statements=50
hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
```

为了使用内置的应用程序服务器，你必须配置Hibernate即通过在JNDI进行注册的应用程序服务javax.sql.Datasource来获得数据连接。因而，你需要设置以下至少一个属性：

属性名称	属性值
hibernate.connection.datasource	数据库JNDI的名称
hibernate.jndi.url	提供JNDI的url（可选）
hibernate.jndi.class	JNDI初始化上下文工厂类（可选）
hibernate.connection.username	数据库用户名
hibernate.connection.password	数据库用户密码

这里有一个应用程序服务其提供JNDI数据库的hibernate.properties的例子的文档：

```
hibernate.connection.datasource = java:/comp/env/jdbc/test
hibernate.transaction.factory_class = \
org.hibernate.transaction.JTATransactionFactory
hibernate.transaction.manager_lookup_class = \
org.hibernate.transaction.JBossTransactionManagerLookup
hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
```

从一个JNDI数据源中得到JDBC连接将会自动的接受程序服务器的容器管理。

你可以在任意的连接属性名之前添加"hibernate。connection"前缀。比如你要指定一个charSet连接属性，你可以使用属性名hibernate.connection.charSet。
你可以通过实现org.hibernate.connection.ConnectionProvider接口，来自定义得到JDBC连接的插件策略并在hibernate.connection.provider_class属性制定你的具体实现类。

3.4 可选的配置属性

Hibernate中有许多有关控制Hibernate运行时的属性。它们都是可选的并且都有相对合理的默认值。

> 警告：
> 有一些属性只存在“系统级别”。系统级别的属性只能通过`java -Dproperty=value` 或者是 `hibernate.properties`这两种方式来设置。他们不可能再有 其他的设置方式了。

表 3.3 Hibernate 配置属性表

属性名	属性值
hibernate.dialect	对于Hibernate的类名是org.hibernate.dialect.Dialect ，这个属性允许Hibernate可以为特定的关系型数据库生成最优化的SQl语句.例子：full.classname.of.dialect 。多数情况下Hibernate实际能够根据JDBC驱动返回的JDBC元数据来选择正确的org.hibernate.dialect.Dialect实现
hibernate.show_sql	将SQL语句的执行过程显示到命令行窗口中。这是一个可选的操作你可以设置在日志目录下属性org.hibernate.SQL to debug 。例子： true /false
hibernate.format_sql	格式化在日志或是命令行窗口输出SQL语句。例子：true/false
hibernate.default_schema	在生成SQL语句时，通过给定模式或表空间来限制不合格的表名。例子：SCHEMA_NAME
hibernate.default_catalog	在生成SQL语句时，通过给定目录来限制不合格的表名。例子：CATALOG_NAME
hibernate.session_factory_name	在使用JNDI情况下，当一个 org.hibernate.SessionFactory被创建之后，它会被自动的绑定到这个名字下。例子：jndi/composite/name
hibernate.max_fetch_depth	为外连接的单端型连接（一对一，多对一）设置最大的"深度"默认值是0，表示禁止使用外部连接。推荐值在0-3之间
hibernate.default_batch_fetch_size	为多连接时，Hibernate批量取值定一个默认的大小。推荐的值是：4，8,16
hibernate.default_entity_mode	为所有session会话设置一个实体所代表的默认模式。dynamic-map,dom4j,pojo
hibernate.order_updates	当数据库中的项目被更新时，强制Hibernate为SQL命令更新主键值。这个会导致在高并发系统思索是更少的事务处理。例子：true/false
hibernate.generate_statistics	如果设置为有效，Hibernate将会收集统计数据用于性能优化。例子：true/false
hibernate.use_identifier_rollback	如果设置为有效，当对象被删除是，生成的标识符的属性将会被重置为默认值。。例子：true/false
hibernate.use_sql_comments	如果打开这个，Hibernate将会生成注释在SQL语句中，为了更方便的debug，默认值设为false。例子：true/false

Table 3.4 Hibernate JDBC 和连接属性

属性名	属性值
hibernate.jdbc.fetch_size	为JDBC取值的个数设置一个非零值（称为：Statement.setFetchSize()）
hibernate.jdbc.batch_size	非零值允许Hibernate使用JDBC2批量更新。例子：推荐值为5-30
hibernate.jdbc.batch_versioned_data	将此值设置为true，当你的JDBC驱动通过执行executeBatch( )会返回正确的行数。打开这个操作，通常都是安全的，Hibernate 将会为自动的版本化数据调用批量DML控制语句。默认值设置为false。例子：true/false.
hibernate.jdbc.factory_class	选择一个经过自定义的org.hibernate.jdbc.Batcher类。多数应用程序都不需要这个配置这个属性。例子：classname.og.BatcherFactory
hibernate.jdbc.use_scrollable_resultset	允许Hibernate使用JDBC2滚动数据结果集。这个属性只有当使用用户提供的JDBC连接的时候才需要设置。否则Hibernate将会使用连接元数据。例子true/false
hibernate.jdbc.use_stream_for_binary	当从JDBC中写/读二进制或是经过序列化的类型使用流处理方式。这是一个系统级别的属性。例子：true/false
hibernate.jdbc.use_get_generated_keys	允许使用JDBC3PrepareStatement.getGeneratedKeys（ ）在插入操作后，取回本机自增的键。需要的版本为JDBC3+驱动和JRE1.4+，如果你的驱动不支持Hibernate标识符发生器，你应该把这个属性设置为false。系统默认使用连接元数据驱动的能力。例子：true/false。
hibernate.connection.provider_class	一个自定义的org.hibernate.connection.ConnectionProvider的类型用于体统JDBC连接到Hibernate框架上。例子：classname.of.ConnectionProvider
hibernate.connection.isolation	设置JDBC事务处理隔离级别。应当为富有含义的值进行检查java.sql.Connection，但是注意多数数据库并不支持所有的隔离级别，同时有一些定义是额外添加的，这说明并没有标准的隔离级别。例子1,2,4,8
hibernate.connection.autocommit	允许JDBC连接池自动的提交数据。这是不被推荐的。例子：true/false
hibernate.connection.release_mode	当Hibernate要释放JDBC连接时，这个值需要被指定。默认的，当一个Hibernate会话在显示的关闭或者是断开的一个JDBC连接仍然会被保持。对于一个应用程序服务器的JTA数据源，在每一次JDBC调用后，使用after_statement主动的释放连接。对于一个非JTA的连接，在每一次事务处理过后才释放连接是有意义的。通过使用after_transaction.auto将会选择after_statement作为JTA和CMT事务策略。使用after_transaction为JDBC事务策略。例子：auto（默认）/on_close/after_transaction/after_statement.这个

	设置只影响从SessionFactory.openSession放回的Sessions。对于Sessions的获得可以通过SeeionFactory.getCurrentSession，或是配置为使用currentSessionContext实现控制这些Sessions连接释放模式。详解请看第2.5部分的"Contextual sessions"
hibernate.connection.< propertyname>	通过JDBC的属性< propertyname> 来设置DriverManager.getConnection( )
hibernate.jndi.< propertyname>	通过设置< propertyname> 属性来设置JNDI的InitialContextFactory

表 3.5 Hibernate 缓存属性

属性名	属性值
hibernate.cache.provideer_class	设置自定义的CacheProvider类。例子：classname.of.CacheProvider
hibernate.cache.use_minimal_puts	优化二级缓存操作为最小的写入，在
hibernate.cache.use_query_cache	打开查询缓存。个别查询茬苒需要被设置为可缓存。例子：true/false
hibernate.cacha.use_seconde_level_cache	能够使二级缓存完全的失效，这也就是可以默认的为特定的类指定cache映射。例子：true/false
hibernate.cachequery_cache_factory	为QueryCache接口自定义一个类名，默认是内建的StandardQueryCache。 例子：classname.of.QueryCache。
hibernate.cache.region_prefix	为二级缓存地区名称添加前缀名称。 例子：prefix。
hibernate.cache.use_structured_entries	强制Hibernate在二级缓存是用一种更为人性化的格式来存储数据。 例子：true/false.

表 3.6 Hibernate 事务处理属性

属性名	属性值
hibernate.transaction.factory_class	指定Hibernate Transaction API 中的TransactionFactory的类名。（默认使用的是JDBCTransactionFactory）例子：classname.of.TransactionFactory
jta.UserTransaction	使用JTATransactionFactory的名字来从应用程序服务器获得JTA UserTransaction 例子：jndi/composite/name
hibernate.transaction.manager_lookup_class	TransactionManagerLookup的类名。当使用到JVM级别的缓存或者是在JTA环境中使用到了hilo增长的时候，这个参数是必须的。例子：classname.of.TransactionManagerLookup
	如果设置为可用，一个会话在事务处理完成阶段之前会被自动的填充。内建和自动的会话上

hibernate.transaction.flush_before_completion	下文管理是非常的优秀的。可以看部分 2.5，"Contextual sessions" 例子：true/false
hibernate.transaction.auto_close_session	如果设置为可用，一个会话在事务处理完成阶段之前会被自动的关闭。内建和自动的会话上下文管理是非常的优秀的。可以看部分 2.5，"Contextual sessions"。例子：true/false

表3.7 其他参数选项

属性名	属性值
hibernate.current_session_context_class	支持为当前的Session会话自定义一个有效范围。查看 Section 2.5,"Contextual sessions"可以获得更多有关于内建策略的信息。例子：jta
hibernate.query.factory_class	选择HQL解析器的实现。例子 org.hibernate.hql.ast.ASTQueryTranslatorFactory或 org.hibernate.hql.classic.ClassicQueryTranslatorFactory
hibernate.query.substitutions	该属性被用于将Hibernate中的查询符号映射到SQL查询符号。（符号可以是函数或者是字符串名字。）例子：hqlLiteral=SQL_LITERAL,hqlFunction=SQLFUNC
hibernate.hbm2ddl.auto	当SessionFactory被创建后，可以自动的验证和输出数据库的DDL语句。如果使用create-drop，SessionFactory被显式的关闭的时候，数据架构将会被丢弃。例子：validate
hibernate.cglib.use_reflection_optimizer	设置为可用的时候，可以用CGLIB代替运行时反射（系统级别的属性）。反射很有利于某些时候解决问题。即使你关闭了优化，但是Hibernate仍然会需要CGLIB。你无法在hibernate.cfg.xml文件中设置这一个属性。例子：true/false。

3.4.1 SQL方言

我们总是会为你的数据库设置hibernate.dialect属性来调整org.hibernate.dialect.Dialect的子类。如果你指定了方言，Hibernate将会为其他一系列的属性列表设置合理的默认值。这也就是意味着你不再需要人为的指定其他的属性值了。

表3.8Hibernate SQL 方言表(Hibernate.dialect)

关系型数据库管理系统	方言
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 (any version)	org.hibernate.dialect.OracleDialect
Oracle 9i	org.hibernate.dialect.Oracle9iDialect
Oracle 10g	org.hibernate.dialect.Oracle10gDialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Informix	org.hibernate.dialect.InformixDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Ingres	org.hibernate.dialect.IngresDialect
Progress	org.hibernate.dialect.ProgressDialect
Mckoi SQL	org.hibernate.dialect.MckoiDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Pointbase	org.hibernate.dialect.PointbaseDialect
FrontBase	org.hibernate.dialect.FrontbaseDialect
Firebird	org.hibernate.dialect.FirebirdDialect

3.4.2 外连接检索

如果你的数据库支持ANSI标准。Oracle或者是Sybase风格的外连接，外连接检索经常会通过限制和数据库的往返次数来提高性能。但是，通过数据库本身的的进行处理会造成更大的工作性能的消耗。外连接抓取允许将所有的对象的连接（一对一，一对多，多对多，多对一）形成一整张表，然后再单条SQL语句中进行关联检索。

外连接检索可以通过设置hibernate.max_fetch_depth的属性值为0来全局的关闭这个功能。如果设置1或者更高的属性可以启动一对一，和多对一的外连接检索，这时候在映射中将是fetch ="join"。

要了解更多的详情，请参考Section 19.1, "Fetching strategies"

3.4.3 二进制流

Oracle可以通过来自它自己的JDBC驱动的与/或逻辑来限制byte数组的大小。如果你希望使用大量的binary或者serializable类型的实例，你应当将hibernate.jdbc.use_stream_for_binary设置为可用。

3.4.4 二级查询缓存

Hibernate通过设置hibernate.cache的属性前缀，从而允许你来使用一个进程或是一系列范围是二级缓存系统。更多的信息，请查看 Section 19.2, "The Second Level Cache"。

3.4.5 替换查询语言

你可以通过使用hibernate.query.substitutions来定义一个新的hibernate查询标识。例如下面的例子：

```
hibernate.query.substitutions true=1, false=0
```

这个将会导致在生成SQL语言时令牌值true和false会被转化为整型字面值。

```
hibernate.query.substitutions toLowercase=LOWER
```

这个将会允许你通过LoWER函数来重新命名SQL语句

3.4.6 Hibernate 统计
如果你将 hibernate.generate_statistics设置为可用，当Hibernate通过SessionFactory.getStatistics()方法会协调一个正在运行中的系统，让Hibernate自身暴露大量的有益的指标。Hibernate甚至可以设置JMX来暴露这些统计。你可以通过阅读在javadoc在org.hibernate.stats的接口来获得更多的消息。
3.5 日志 Hibernate

利用Simple Logging Facade for Java（SLF4J）来记录系统各种各样的事件。SLF4J可以指定你的日志输出到一些你所绑定的日志框架（NOP，Simple，log4J 版本1.2，JDK1.4 logging，JCL 或者logback）中。为了可以启动日志，如果你是采用Log4J日志框架，你需要在你的类路径下添加slf4j-api.jar文件和slf4j-log4j12.更具体的SLF4J的详情，请参考SLF4J documentation。为了使用Log4J，你同样需要在类路径下放置一份log4j.properites文件。你可以在hibernate的src/directory目录下找到一份log4j.properites示例文件。

在你的Hibernate框架下使用你所熟悉的日志记录这是被官方所推荐的。大量的工作已经被投入到使hibernate 日志尽可能详细且可读。日志是基本的解决问题的设备。最令人感兴趣的日志分类如下：

日志类别	功能
org.hibernate.SQL	记录所有的SQL DML执行时的状态
org.hibernate.type	记录所有的JDBC参数
org.hibernate.tool.hbm2ddl	记录所有的SQL DDL 执行时的状态
org.hibernate.pretty	记录所有在会话缓冲时间的实体得状态(最多20个实体)
org.hibernate.cache	记录所有的二级缓存的活动
org.hibernate.transaction	记录事务关联的活动
org.hibernate.jdbc	记录所有需要的JDBC资源
org.hibernate.hql.ast.AST	记录所有的HQL 和SQL AST在执行查询解析期间
org.hibernate.secure	记录所有的JAAS请求的授权
org.hibernate	记录一切。这个将会长生很多的信息，但是对于解决问题也非常的有帮助。

当使用Hibernate进行应用程序的开发的时候，你应当总是使用debug模式，并且是

org.hibernate.SQL可用，或者你应当使用hibernate.show_sql的属性为可用。

3.6 实现一个命名策略

org.hibernate.cfg.NamingStrategy这个接口允许你来为数据库实体和架构元素指定一个"命名标准"。你可以提供规则使拥有自增长的数据库标识符映射给相关的Java标识符，或者将数据库的"物理"表名和列名映射给映射文件中的"逻辑"表名和列名。这个特性可以帮助减少冗长的映射文件，消除重复的噪音（比如说TBL_profixes）。使用Hibernate的默认的策略可以得到最低限度的文件。

你可以在增加映射文件之前，通过调用Configuration.setNamingStrategy()来指定一种不同的策略。如下所示：

```
SessionFactory sf = new Configuration()
                .setNamingStrategy(ImprovedNamingStrategy.INSTANCE)
                .addFile("Item.hbm.xml")
                .addFile("Bid.hbm.xml")
                .buildSessionFactory();
```

org.hibernate.cfg.ImprovedNamingStrategy是一个内建的策略，这个策略可能对于开始某些应用程序是非常有利的。

3.7 XML配置文件

一个可选的配置方法是指定一份叫做hibernate.cfg.xml配置文件，它拥有全部的配置属性。这份文件可以被一份命名为hibernate.properties的文件所替代。但是如果两份文件都存在，可以使用hibernate.properties文件重写hibernate.cfg.xml的属性。

这一份XML配置文件默认的被希望放在类路径的根目录下，下面是一个例子样板：

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <!-- a SessionFactory instance listed as /jndi/name -->
  <session-factory
  name="java:hibernate/SessionFactory">
  <!-- properties -->
  <property name="connection.datasource">java:/comp/env/jdbc/MyDB</property>
  <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
  <property name="show_sql">false</property>
  <property name="transaction.factory_class">
  org.hibernate.transaction.JTATransactionFactory
  </property>
  <property name="jta.UserTransaction">java:comp/UserTransaction</property>
  <!-- mapping files -->
  <mapping resource="org/hibernate/auction/Item.hbm.xml"/>
  <mapping resource="org/hibernate/auction/Bid.hbm.xml"/>
  <!-- cache settings -->
  <class-cache class="org.hibernate.auction.Item" usage="read-write"/>
```

```
  <class-cache class="org.hibernate.auction.Bid" usage="read-only"/>
  <collection-cache collection="org.hibernate.auction.Item.bids" usage="read-write"/>
  </session-factory>
</hibernate-configuration>
```

这一种方法的好处是可以直观的通过映射文件名来进行配置。一旦你不得不调整Hibernate的缓存，使用hibernate.cfg.xml配置也更方便。你可以选择配置hibernate.cfg.xml或者是hibernate.properties两种中的任意一种。这两者都是等价的，除了说使用xml的语法拥有更为直观的配置提醒。

如果是使用XML配置方式，启动Hibernate应当是这样的子：

```
SessionFactory sf = new Configuration().configure().buildSessionFactory();
```

你同样可以选择使用另外一种不同命的XML配置文件，之后你将这样使用：

```
SessionFactory sf = new Configuration()
                .configure("catdb.cfg.xml")
                .buildSessionFactory();
```

3.8 J2EE 应用程序服务器整合
Hibernate对J2EE应用程序构建有以下几个整合点：

- 容器管理的数据源：Hibernate可以通过容器进行管理，同时也可以通过由JNDI提供的JDBC连接进行管理。通常的JTA是兼容TransactionManager和关注事务管理的ResourceManager。特别是支持通过多个数据源的分布式事务处理。你同样可以划分事务边界编程，或者你可能会想要使用可选的Hibernate Transaction API 来保证代码的移植性。
- 自动的JNDI绑定：Hibernate可以在启动后将SessionFactory绑定到JNDI上。
- JTA会话绑定：Hibernate会话可以自动的绑定到JTA的事务处理范围之内。它仅仅是从JNDI查找到SessiongFactory，并且得到当前的session会话。当你的JTA事务完成之后，请让Hibernate管理 刷新缓冲区，并关闭当前的Session对象。事务划分是另外一种声明性的（CMT）或者是可编程的（BMT/UserTransaction）。
- JMX部署：如果你拥有一个可以应用程序服务的JMX（类似于JBoss AS），你可以选择将Hibernate作为一个管理的MBean来部署。这个可以节省你一行启动代码来从Configuration配置建立一个SessionFactory。容器会启动你的HibernateService服务，同时也可以关注你的服务依赖（在Hibernate启动之前，数据源应当是可得的）。

你可能需要设置你的配置选项hibernate.connection.aggressive_release为true，如果你的应用服务器显示的是"sonnection containment"异常，这个完全取决于你的环境。

3.8.1 事务策略配置

Hibernate Session API 在你的架构中是独立于任何事务划分系统的。如果你让Hibernate直接通过一个连接池来使用JDBC，你可以通过调用JDBC API来开始或者关闭你的事务。如果你的项目跑在一

个J2EE的应用程序服务器上，当你需要的时候，你可能会想通过调用JTA API 和UserTransaction 来使用bean管理事务。

为了让你的在两个或者更多的环境下代码保证移植性，我们建议用户选择Hibernate的Transaction API，这个API被封装和隐藏了底层系统的实现。你同样也可以通过设置hibernate.transaction.factory_class来指定一个Transaction 的工厂类实例。
这里有三个标准或是内建的API可以供用户选择：

```
org.hibernate.transaction.JDBCTransactionFactory
```

默认委托数据库（JDBC）事务处理

```
org.hibernate.transaction.JTATransactionFactory
```

如果存在事务在这个文中是正在进行的，就委托给一个容器管理事务（比如说，EJB会话bean方法）。否则，将启动一个新的事务，并使用Bean管理的事务.

```
org.hibernate.transaction.CMTTransactionFactory
```

委托给JTA容器进行管理事务。

你也可以定义你的字的事务处理策略（比如说，一个CORBA事务服务）
一些特性在Hibernate中（比如说，二级缓存，或是与上下文相关的JTA）都需要在一个管理环境中得到JTA的TransactionManager。在一个应用服务器， 因为J2EE中不闺房单一的机制，因此你不得不指定Hibernate应该如何获得一个服务器相关的TransactionManager对象。

表 3.10 JTA 事务管理器

事务管理工厂	应用服务器名称
org.hibernate.transaction.JBossTransactionManagerLookup	JBoss
org.hibernate.transaction.WeblogicTransactionManagerLookup	Weblogic
org.hibernate.transaction.WebSphereTransactionManagerLookup	WebSphere
org.hibernate.transaction.WebSphereExtendedJTATransactionLookup	WebSphere 6
org.hibernate.transaction.OrionTransactionManagerLookup	Orion
org.hibernate.transaction.ResinTransactionManagerLookup	Resin
org.hibernate.transaction.JOTMTransactionManagerLookup	JOTM
org.hibernate.transaction.JOnASTransactionManagerLookup	JOnAS
org.hibernate.transaction.JRun4TransactionManagerLookup	JRun4
org.hibernate.transaction.BESTransactionManagerLookup	Borland ES

3.8.2 JNDI绑定 SessionFactory

一个JNDI绑定的HibernateSessionFactory可以简化工厂的查询，简化创建新的Session对象，需要注意的是这届JNDI绑定的Datasource没有关系，他们只是恰巧的进行了相同的注册。

如果你希望一个SessionFactory可以绑定到一个JNDI的命名空间中，可以使用属性hibernate.session_factory_name来指定一个名字（比如说，java：hibernate/SessionFactory）。如果这个属性并没进行指定，那么这个SessionFactory就不会被绑定到JNDI上。这个是非常有利的在一个只读的JNDI的环境使用默认的实现（比如说，Tomcat中）。

当绑定一个SessionFactory到JNDI上，Hibernate将会使用hibernate.jndi.url ，hibernate.jndi.class的实例的值来初始化上下文。如果，他们没有被指定具体值，那么默认的InitialContext将会被应用。

Hibernate会在你调用cfg.buildSessionFactory()方法之后然后自动的将SessionFactory添加到JNDI中。这也就是意味你需要调用一些启动代码，或者是在你的应用中调用某些使用的工具类，除非你使用使用了JMX配合HibernateService来部署的话（这个在线面的章节会提到）。

如果你使用一个绑定到JNDI的SessionFactory，一个EJB或者是任何其他的类，你都可以通过查询JNDI来获得一个SessionFactory。

这是一种被推荐的做法：在一个管理环境中，你把一个SessionFactory绑定到JNDI中，或者是使用一个静态的单例。为了封装你具体的应用代码，我们同样推荐隐藏实际在帮助类中查找SessionFactory的代码，比如说HibernateUtil.getSessionFactory()的方法。在第一章中同样有遍历的方式来启动Hibernate的类。

3.8.3 使用JTA来管理当前的Session会话上下文

Hibernate的自动的管理"当前"会话是最简单的处理Session会话和事务的方法。你可以查阅 Section 2.5, "Contextual sessions".来得到更多有关于上下文会话的谈论。如果Hibernate没有和当前的JTA事务建立连接，那么你可以使用"JTA"会话上下文。如果你是第一次启动关联了JTA的事务，你需要先调用sessionFactory.getCurrentSession()方法。在"jta"上下文中，Session会被通过调用getCurrentSession（）的方法重新取回，并且被自动的设置为事务完成之前进行刷新缓冲区，事务结束之后关闭会话，在每一个声明之后会主动的释放JDBC连接。如果Session会话被关联到JTA事务上，那么JTA事务就会管理Session的整个生命周期，这样的管理概念可以保持用户代码的清洁性。你的代码可以通过使用UserTransaction来进行编程式JTA管理，或者是使用Hibernate的Transaction API来进行事务管理（为了代码的移植性，这个方法更被官方所推荐使用）。如果你的应用运行在一个EJB的容器上，配合CMT的声明式事务管理更为完美。

3.8.4 JMX 部署

为了可以从JNDI中得到SessionFactory，我们仍然不得不在程序的某一处执行cfg.buildSessionFactory()方法。因此你也可以在一个静态初始块中执行这个方法，就像是HibernateUtil那样做的一样，或者你也可以就像是一个管理服务那样部署Hibernate。

如果Hibernate部署在一个类似于JBoss AS这样的拥有JMX能力的应用服务器上，那么Hibernate 分发org.hibernate.jmx.HibernateService服务。实际部署和配置都是取决于提供者的特性。下面是一份

JBoss 4.0.x的jboss-service.xml的示例文件：

```
<?xml version="1.0"?>
<server>
<mbean code="org.hibernate.jmx.HibernateService"
name="jboss.jca:service=HibernateFactory,name=HibernateFactory">

<!-- Required services -->
<depends>jboss.jca:service=RARDeployer</depends>
<depends>jboss.jca:service=LocalTxCM,name=HsqlDS</depends>

<!-- Bind the Hibernate service to JNDI -->
<attribute name="JndiName">java:/hibernate/SessionFactory</attribute>

<!-- Datasource settings -->
<attribute name="Datasource">java:HsqlDS</attribute>
<attribute name="Dialect">org.hibernate.dialect.HSQLDialect</attribute>

<!-- Transaction integration -->
<attribute name="TransactionStrategy">
org.hibernate.transaction.JTATransactionFactory</attribute>
<attribute name="TransactionManagerLookupStrategy">
org.hibernate.transaction.JBossTransactionManagerLookup</attribute>
<attribute name="FlushBeforeCompletionEnabled">true</attribute>
<attribute name="AutoCloseSessionEnabled">true</attribute>

<!-- Fetching options -->
<attribute name="MaximumFetchDepth">5</attribute>

<!-- Second-level caching -->
<attribute name="SecondLevelCacheEnabled">true</attribute>
<attribute name="CacheProviderClass">org.hibernate.cache.EhCacheProvider</attribute>
<attribute name="QueryCacheEnabled">true</attribute>

<!-- Logging -->
<attribute name="ShowSqlEnabled">true</attribute>

<!-- Mapping files -->
  <attribute  name="MapResources">auction/Item.hbm.xml,auction/Category.hbm.xml</
attribute>
</mbean>
</server>
```

这一份文件被放置在一个被打包的JAR文件中一个叫做META-INF的目录下，这个JAR文件是一个extension.jar（服务档案）。你也需要将hibernate和它所需要的第三方库打包在一起。你编译的持久类和你的映射文件也要放在同一个文件夹下。你的企业级的beans（常常是会话bean）可以保持在他们自己所拥有的JAR文件中，但是你应该包含EJB JAR文件在主要的服务文档中为了可以得到一个单一（热）部署的单元。查阅JBoss AS 的文档可以获取更多有官方JMX服务和EJB部署的信息。