# Reference study

## 1.1第一个 Hibernate 应用程序

1. 我们通常不会操作对象的标识（identity），因此它的 setter 方法的访问级别应该声明 private。这样当对象被保存的时候，只有 Hibernate 可以为它分配标识符值。

2. 所有的持久化类（persistent classes）都要求有无参的构造器，因为 Hibernate 必须使用 Java 反射机制来为你创建对象。构造器（constructor）的访问级别可以是 private，然而当生成运行时代理（runtime proxy）的时候则要求使用至少是 package 级别的访问控制，这样在没有字节码指令（bytecode instrumentation）的情况下，从持久化类里获取数据会更有效率。

3. 当处理映射文件时，Hibernate 用反射（reflection）来决定这个映射类型(string,timestamp..)。这需要时间和资源，所以如果你注重启动性能，你应该考虑显式地定义所用的类型。

4. 内置的 Hibernate 连接池不适用于产品环境。它缺乏连接池里的几个功能

## 3.1 可编程的配置方式

org.hibernate.cfg.Configuration 实例代表了一个应用程序中 Java 类型到SQL数据库映射的完整集合。被用来构建一个（不可变的（immutable））org.hibernate.SessionFactory。映射定义则由不同的 XML 映射定义文件编译而来。

org.hibernate.cfg.Configuration 实例被设计成启动期间（startup-time）对象，一旦 SessionFactory 创建完成它就被丢弃了。

当所有映射定义被 org.hibernate.cfg.Configuration 解析后，应用程序必须获得一个用于构造 org.hibernate.Session 实例的工厂。这个工厂将被应用程序的所有线程共享：

SessionFactory sessions = cfg.buildSessionFactory();

Hibernate 允许你的应用程序创建多个 org.hibernate.SessionFactory 实例。这对 使用多个数据库的应用来说很有用。

通常你希望 org.hibernate.SessionFactory 来为你创建和缓存（pool）JDBC 连接。如果你采用这种方式，只需要如下例所示那样，打开一个 org.hibernate.Session：

Session session = sessions.openSession(); // open a **new** Session

## 4.1 Persistent Classes

### Define a no-argument constructor

Cat has a no-argument constructor. All persistent classes must have a default constructor (which can be non-public) so that Hibernate can instantiate them using java.lang.reflect.Constructor.newInstance(). It is recommended that this constructor be defined with at least *package* visibility in order for runtime proxy generation to work properly.

## Provide an identifier property

Identifiers do not necessarily need to identify column(s) in the database physically defined as a primary key. They should just identify columns that can be used to uniquely identify rows in the underlying table.

我们建议你对持久化类声明命名一致的标识属性。我们还建议你使用一个可以为空（也就是说，不是原始类型）的类型。

## Disadvantage of Hibernate

Hibernate may not be the best solution for data-centric(中央的，中心的) applications that only use stored-procedures to implement the business logic in the database

## Setter method for id should be private

However, we usually do not manipulate the identity of an object, hence the setter method should be private. Only Hibernate will assign identifiers when an object is saved.

## The no-argument constructor is a requirement for all persistent classes

## Path of DTD declaration file of mapping XML

The DTD file is included in **hibernate-core.jar** (it is also included in the **hibernate3.jar**, if using the distribution bundle)

## Define the type in hbm files for performance tuning

Hibernate makes this mapping type determination using reflection when the mapping files are processed. This can take time and resources, so if startup performance is important you should consider explicitly defining the type to use.

## What does sessionFactory.getCurrentSession() do?

 First, you can call it as many times and anywhere you like once you get hold of your org.hibernate.SessionFactory. The getCurrentSession() method always returns the "current" unit of work. Remember that we switched the configuration option for this mechanism to "thread" in our **src/main/resources/hibernate.cfg.xml**? Due to that setting, the context of a current unit of work is bound to the current Java thread that executes the application

**Important**

Hibernate offers three methods of current session tracking. The "thread" based method is not intended for production use; it is merely useful for prototyping and tutorials such as this one. Current session tracking is discussed in more detail later on

A org.hibernate.Session begins when the first call to getCurrentSession() is made for the current thread. It is then bound by Hibernate to the current thread. When the transaction ends, either through commit or rollback, Hibernate automatically unbinds the org.hibernate.Session from the thread and closes it for you. If you call getCurrentSession() again, you get a new org.hibernate.Session and can start a new unit of work

The scope of a Hibernate org.hibernate.Session is flexible but you should never design your application to use a new Hibernate org.hibernate.Session for *every* database operation

## The build-in hibernate connection pool should not be used in production environment

The built-in Hibernate connection pool is in no way intended for production use. It

lacks several features found on any decent connection pool.

## Current session context configuration

Hibernate offers three methods of current session tracking. The "thread" based method is not intended for production use; it is merely useful for prototyping and tutorials such as this one.

## Hibernate support more than one session factory

SessionFactory is a global factory responsible for a particular database. If you have several databases, for easier startup you should use several <session-factory> configurations in several configuration files.

## Session factory is thread-safe

A org.hibernate.Session represents a single-threaded unit of work. The

org.hibernate.SessionFactory is a thread-safe global object that is instantiated once

## No need to configure dialect

In most cases, Hibernate is able to properly determine which dialect to use

Private the setter method for id

Hibernate建议由它自己来生成ID的值来给对象，所以我们必须提供setter方法给id，但hibernate认为这个方法不应当暴露给外围的程序使用，所以建议把它的setter方法设为private的。

## Automatic dirty checking

|  |
| --- |
| private void addPersonToEvent(Long personId, Long eventId) {  Session session = HibernateUtil.getSessionFactory().getCurrentSession();  session.beginTransaction();  Person aPerson = (Person) session.load(Person.class, personId);  Event anEvent = (Event) session.load(Event.class, eventId);  aPerson.getEvents().add(anEvent);  session.getTransaction().commit();  } |

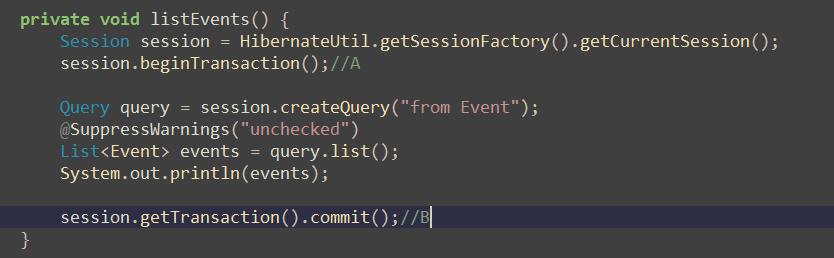
After loading a Person and an Event, simply modify the collection using the normal collection

methods. There is no explicit call to update() or save(); Hibernate automatically detects that the collection has been modified and needs to be updated. This is called ***automatic dirty checking***. You can also try it by modifying the name or the date property of any of your objects. As long as they are in ***persistent* state**, that is, bound to a particular Hibernate org.hibernate.Session, Hibernate monitors any changes and executes SQL in a write-behind fashion. The process of synchronizing the memory state with the database, usually only at the end of a unit of work, is called ***flushing***. In our code, the unit of work ends with a commit, or rollback, of the database transaction.

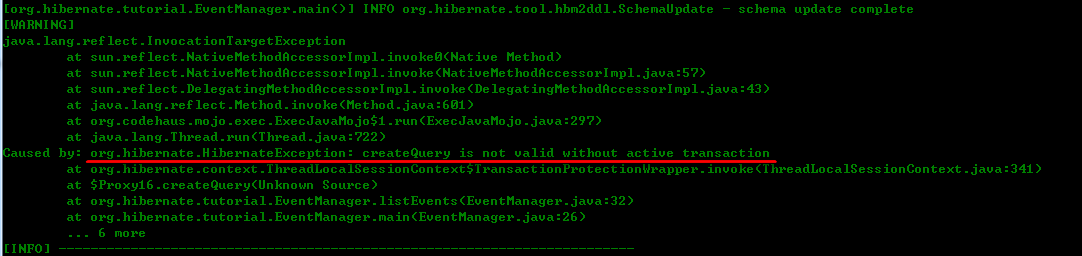
## All JDK classes are considered value types

In fact, in a Hibernate application all JDK classes are considered value types

## CreateQuery has to in a transaction



If above source remove statement A and B, then it will throw exception



Even the method are only retrieve data from DB.