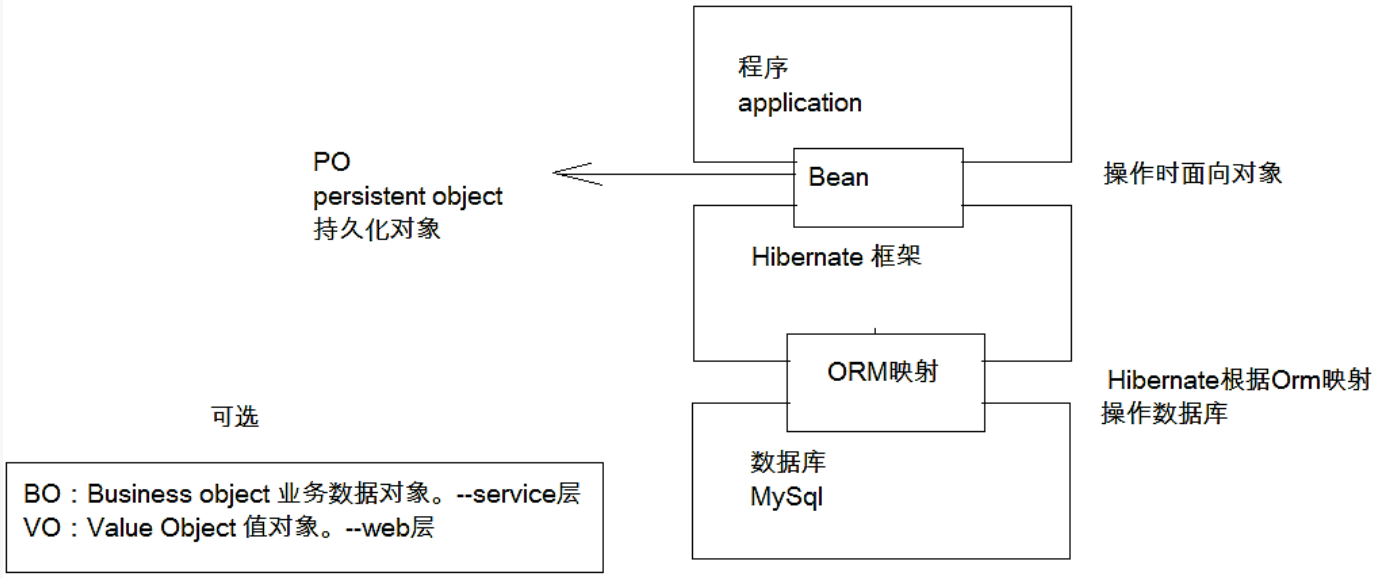
## 1、体系结构



## 2、API

### Configuration类

org.hibernate.cfg.Configuration接口的作用是加载主配置文件及映射文件，以实现对Hibernate的启动。

Configuration实例的获取方式：

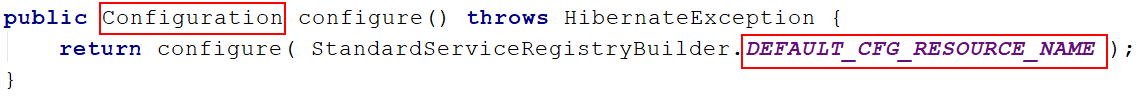
Configuration cfg = new Configuration().configure();

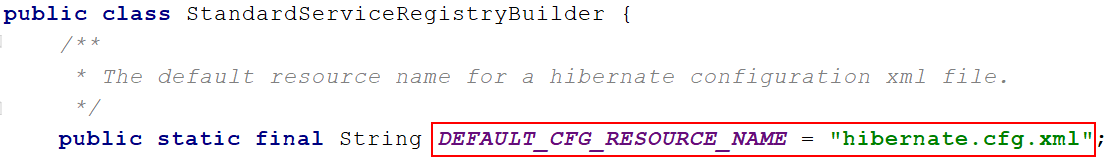
1）new Configuration()

new Configuration()会加载一个属性文件hibernate.properties。该属性文件中存放着连接配置、Hibernate配置等。（一般情况下不用设置该属性文件，其配置放到主配置文件中）

2）无参configure()方法

configure()方法，默认用于加载和解析名称为hibernate.cfg.xml的配置文件，并通过该配置文件找到并解析映射文件Xx.hbm,xml。该方法返回一个Configuration对象。所以cfg是一个包含配置信息及映射信息的Configuration对象。





3）带参configure()方法

Hibernate主配置文件默认名称为hibernate.cfg.xml，应存放在src类路径下。但也可以更换路径与文件名。此时，需要使用带参的configure(String resource)方法。

### SessionFactory接口

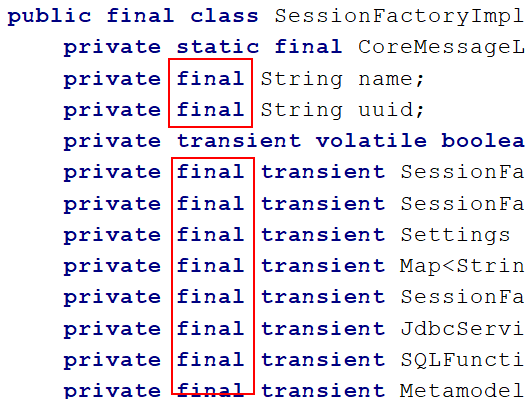
org.hibernate.SessionFactory接口对象是由Configuration对象通过buildSessionFactory()方法创建的。创建该对象的目的是，用于开启Session对象。

SessionFactory sessionFactory = cfg.buildSessionFactory();

1）SessionFactory对象特点

重量级对象（系统开销大）、单例的、线程安全的。

一般来说，单例对象一定是被共享的，是线程不安全的。查看SessionFactory接口的实现类SessionFactoryImpl源码，可以看到其大多数成员变量是final的，所以是线程安全的。



2）SessionFactory对象的使用原则

基于其实线程安全的重量级对象，其创建与销毁时系统开销大，又是单例的特点，SessionFactory对象一般不手工关闭，而是在应用结束时自动将其销毁。因此，SessionFactory不用进行close()关闭。

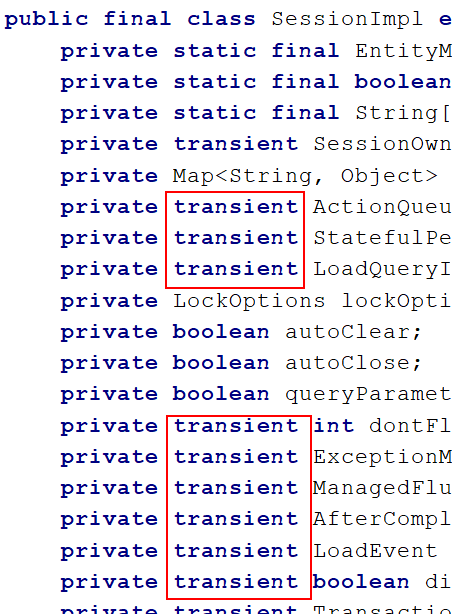
### Session接口

org.hibernate.classic.Session接口是应用程序与Hibernate连接的核心API，是Hibernate向应用程序提供的操纵DB的最主要接口。它提供了基本的保存、更新、删除与查询方法。

1）Session对象特点

一个轻量级对象、线程不安全的、多例的。

在web应用中，多个用户对同一应用访问，Hibernate会为每个用户创建一个Session对象。所以是多例的。Session中包含大量非final成员变量，对于同一个用户的操作，可能会产生多个事务，这多个事务若同时对同一个Session的同一个成员变量进行访问，就会引起并发问题。所以session是线程不安全的。



2）Session对象的使用原则

基于Session的特点，Session在使用时要做到一个线程一个Session，即一个事务一个Session。使用完毕，立即关闭。Session不要作为某个类的成员变量出现，因为这样会出现多个实例对象对同一个Session的共享，使其不安全。

3）Session对象的获取

SessionFactory对应Session对象的获取，提供了两种方式：

* sessionFactory.openSession(); 创建一个新的Session对象
* sessionFactory.getCurrentSession(); 获取当前线程中的Session对象

openSession()方式：

每执行一次，均会创建一个新的Session对象。

getCurrentSession()方式：

每次获取到的都是当前线程中的Session对象，都是同一个Session对象。

为什么getCurrentSession()获取到的是同一个对象？

getCurrentSession()首先会获取到当前线程Thread.currentThread()，然后从ThreadLocal中读取key为当前线程Thread，而value则为要共享的数据。当在Hibernate主配置文件中设置了hibernate.current\_session\_context\_class属性的值为thread时，就表明每个用户获取到的Session对象都是自己当前线程中的Session，是同一个Session。

当前Session上下文：

为了保证一个线程一个Session，即一个线程中使用的Session是同一个对象，一般在获取Session对象时，使用SessionFactory的getCurrentSession方法。不过，使用该方法获取Session对象，需要在主配置文件中对Session所处的上下文环境，即事务环境进行注册。

*<!--指定当前session上下文为线程-->*

<**property name="hibernate.current\_session\_context\_class"**>

thread

</**property**>

hibernate.current\_session\_context\_class取值有3种：

|  |  |
| --- | --- |
| 取值 | 意义 |
| thread | 表示当前Session所处的环境为本地事务环境，Session会与当前线程绑定。 |
| jta | 表示当前Session所处的环境为分布式事务环境。 |
| SpringSessionContext类 | SSH整合时使用。表示将当前Session的管理权交由Spring容器，而Spring容器中会有事务管理。即当前Session所处的环境为Spring事务环境。 |

Session的关闭：

使用getCurrentSession()方法获取的Session，在进行事务提交或回滚后，会自动关闭，无需手动close()。当然，也不能在最后的finally语句块中对Session进行手工关闭。因为，无论是执行事务的commit()。还是执行rollback()，均会在finally之前执行。也就是说，在执行finally之前，session已经关闭。若再手工关闭，将抛出异常。

两种获取Session对象方式的区别：

获取Session对象，可以使用openSession()方式，也可以使用getCurrentSession()方式。这两种方式在本质上有很大的区别。

|  |  |  |
| --- | --- | --- |
|  | getCurrentSession()方式 | openSession()方式 |
| 获取的对象 | 无论执行多少次该方法，只要是在同一线程中，获取的都是同一个Session对象。 | 每执行一次该方法，获取到的都是一个新的Session对象。 |
| 对象的关闭 | 自动关闭Session，无需手工关闭 | 必须手工关闭Session对象 |
| 环境的注册 | 需要注册Session的运行环境 | 无需注册 |
| 查询对事物的支持 | 查询必须在事务内执行 | 查询可以不再事务内执行 |

4）Session中的常用方法

save()、persist()：添加对象

update()：修改对象

saveOrUpdate()：添加或修改对象

delete()：根据主键删除对象

get()、load()：根据主键查询

### Transaction接口

通过该接口，可以将事务从持久层，提升到业务层。由Session对象创建。

session.getTransaction();

事务的开启：session.beginTransaction();或session.getTransaction().begin();

事务的提交：session.getTransaction().commit();

事务的回滚：session.getTransaction().rollback();

### 单例、多例与线程安全问题

单例与多例问题：当多个用户访问某个类时，系统是为每个用户创建一个该类实例，还是整个系统无论多少用户访问，只创建一个实例。

线程安全问题：多个用户同时在访问同一个程序时，其对于某一数据的修改，会不会影响到其他用户。若没有影响，则是线程安全的；若有可能影响，则线程不会安全。

1）HttpServlet

单例。无论多少用户访问同一个业务，如LoginServlet，Web容器只会创建一个该Servlet实例。而该实例允许多个用户访问的。若Servlet中包含成员变量，则每个用户对于成员变量的修改，均会影响到其他用户所看到的该变量的值，所以这时是线程不安全的。若不包含成员变量，则线程是安全的。

2）HttpSession

多例。Web容器会为每个用户开辟一个Session，多个用户会有多个Session。而每个用户只能访问自己的Session。所以，对于Session来说，就不存在并发访问的情况，是线程安全的。

3）Struts2的Action

多例。对于同一个业务，例如LoginAction，系统会为每一个用户创建一个LoginAction的实例，并使其成员变量username与password接收用户提交的数据。同一用户只能访问自己的Action，所有是线程安全的。

4）Hibernate的SessionFactory

单例。无论多少用户访问该项目，系统只会创建一个SessionFactory对象，即这个对象是可以被所有用户访问的。SessionFactory实现类中所包含的成员变量基本都是final常量，即任何用户均不能修改。所有是线程安全的。

5）Hibernate的Session

多例。系统会为每个用户创建一个Session。Session的实现类中定义了很多非final成员变量，一个事务对成员变量所做的修改，会影响到另一个事物对同一数据的访问结果，所有是线程不安全的。