# 数据库

DAO（Data Access Object）= Data Accessor+ Active Domain Object。Data Accessor实现数据访问和业务逻辑的分离，Active Domain Object实现业务数据的对象化封装。

POJO：Plain Ordinary Java Object，无格式普通Java对象

## Connection Pool

|  |  |  |
| --- | --- | --- |
|  | 简介 | 使用评价 |
| DBCP | DBCP是一个依赖Jakarta commons-pool对象池机制的数据库连接池。是Tomcat数据连接的组件。（注：由于bug太多，Hibernate已经不再支持） | spring推荐  无法自动重连 |
| C3P0 | C3P0是一个开放源代码的JDBC连接池，Hibernate3.0默认自带的数据连接池，性能比较稳定。 | 性能稳定  hibernate推荐  自动重连 |
| Proxool | 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的功能，方便易用，便于发现连接泄漏的情况。 | 连接池监控 |
| BoneCP | BoneCP是一个快速，开源的数据库连接池。帮你管理数据连接让你的应用程序能更快速地访问数据库。比C3P0/DBCP连接池快25倍 | 效率高，速度快 |

## JDBC vs JNDI

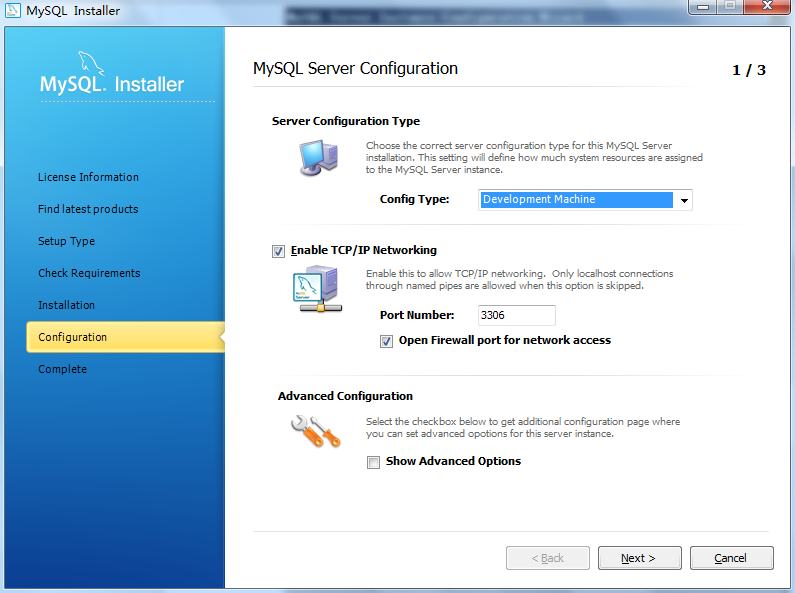
JDBC（Java Data Base Connectivity，java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。

JNDI （Java Naming and Directory Interface）被用于执行名字和目录服务。它提供了一致的模型来存取和操作企业级的资源如DNS和LDAP，本地文件系统，后者在应用服务器中的对象。在JNDI中，在目录结构中的每一个结点称为context。每一个JNDI名字都是相对于context的。应用可以通过这个初始化的context经有这个目录树来定位它所需要的资源或对象。

# MySQL

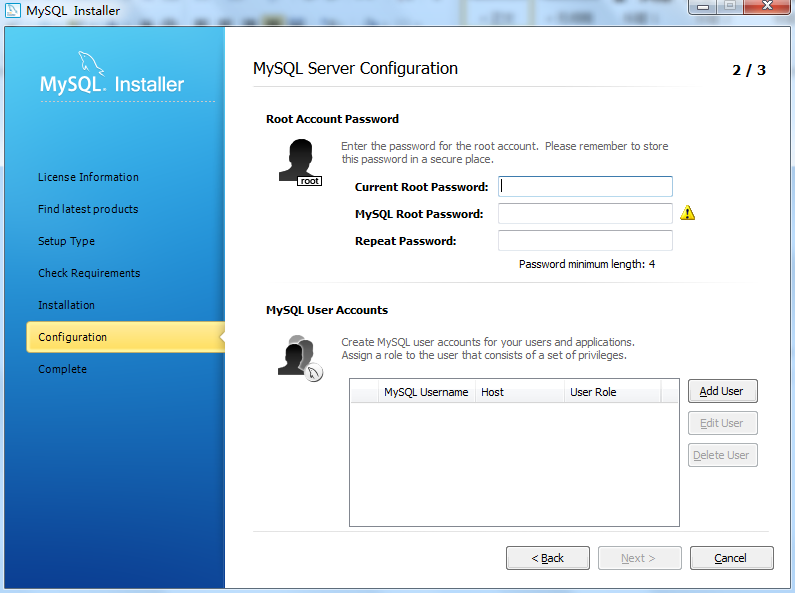
## 安装

### Window安装

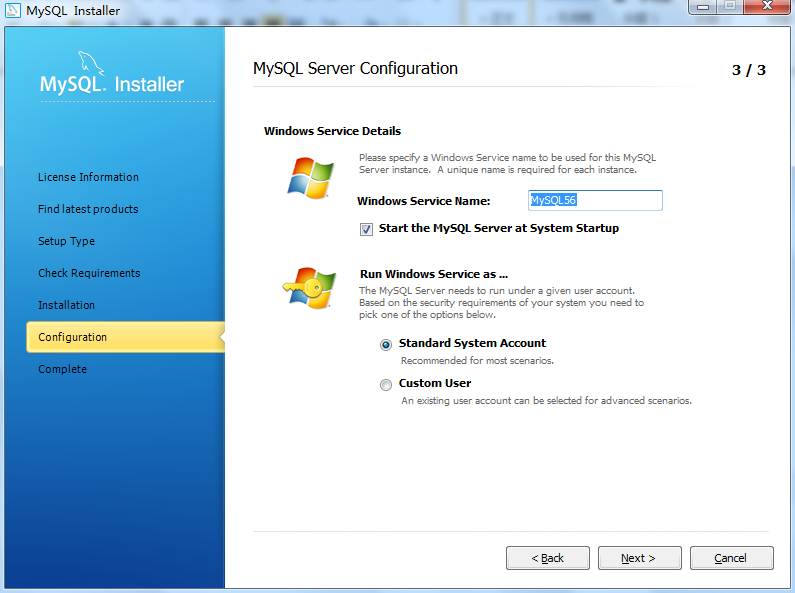


选择服务器类型，“Developer Machine（开发测试类，mysql占用很少资源）”、“Server Machine（服务器类型，mysql占用较多资源）”、“Dedicated MySQL Server Machine（专门的数据库服务器，mysql占用所有可用资源）”

是否启用TCP/IP连接，设定端口，如果不启用，就只能在自己的机器上访问mysql数据库了，我这里启用，把前面的勾打上，Port Number：3306



配置root密码和用户帐号



### CentOS 7安装

yum -y install mariadb-server mariadb

systemctl start mariadb

systemctl enable mariadb

systemctl status mariadb

systemctl stop mariadb

### Ubuntu安装

sudo apt-get install mysql-server

sudo netstat -tap | grep mysql

tcp 0 0 localhost:mysql \*:\* LISTEN 2556/mysqld

sudo service mysql restart

MySql的时间类型有 Java中与之对应的时间类型

date java.sql.Date

Datetime java.sql.Timestamp

Timestamp java.sql.Timestamp

Time java.sql.Time

Year java.sql.Date

**Timestamp转化为String:**

SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");//定义格式，不显示毫秒

Timestamp now = new Timestamp(System.currentTimeMillis());//获取系统当前时间

String str = df.format(now);

**String转化为Timestamp:**

SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
String time = df.format(new Date());

Timestamp ts = Timestamp.valueOf(time);

## 问题

**问题：**

导入工程后，修改lib，.classpath拒绝访问

原因：.classpath，.project成为隐藏文件

解决方法：右击文件属性，去掉隐藏选项

**问题：**

程序在Eclipse中运行没问题，打包后运行报错：

Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/commons

/logging/LogFactory

private static Log logger = LogFactory.getLog(Module.class);

原因：LogFactory找不到log4j.properties，因为PropertyConfiguratorr.configure(configFilename)位置不对，Module未获得。

**问题：**

Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/log4j

/PropertyConfigurator

原因：路径配置错误，找不到log4j.properties的位置。

字段值为NULL，返回String为NULL

字段值为空，返回String为””

DateTime为空时，插入必须为NULL

**问题：**

如果报错ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2),

原因是MySQL服务未启动。

**问题：**

ERROR 1044 (42000): Access denied for user '' @'localhost' to database 'menagerie'

原因是mysql数据库的user表里，存在用户名为空的账户即匿名账户，导致登录的时候是虽然用的是root，但实际是匿名登录的。

mysql> delete from user where USER='';

**问题：**

**无法连接MySQL， 抛异常Cannot create PoolableConnectionFactory**

**无法登陆，报错ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)**

解决方法：

1、检查配置，用户是否具有权限

mysql> CREATE USER 'dwt'@'%' IDENTIFIED BY '123456';

mysql> GRANT ALL PRIVILEGES ON \*.\* TO 'dwt'@'%'

-> WITH GRANT OPTION;

# mysql –u dwt -p

# mysql -u dwt -p localhost

# mysql –u dwt –h 192.168.214.128 -p

2、关闭防火墙

service iptables stop

chkconfig iptables off

service iptables status

3、修改/etc/mysql/my.cnf

注释掉bind-address=127.0.0.1

然后重启机器

导入SQL文件：

source /home/dwt/workspace/database.sql

## 数据类型

### 时间类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MySQL | 字节 | 最小值 | 最大值 | Java |
| DATE | 4 | 1000-01-01 | 9999-12-31 | java.sql.Date |
| DATETIME | 8 | 1000-01-01 00:00:00 | 9999-12-31 23:59:59 | java.sql.Timestamp |
| TIMESTAMP | 4 | 19700101080001 | 2038年的某个时刻 | java.sql.Timestamp |
| TIME | 3 | -838:59:59 | 838:59:59 | java.sql.Time |
| YEAR | 1 | 1901 | 2155 | java.sql.Date |

java.sql继承自java.util.Date

java.sql.Date对应于SQL DATE类型，格式为yyyy-mm-dd HH:MM:ss，没有毫秒

java.sql.Time对应于SQL TIME，格式为HH:MM:ss.ffffff，包括毫秒

java.sql.Timestamp对应于SQL TIMESTAMP，支持到纳秒，util.Date只支持到毫秒

date

1. 只记录日期信息，表示范围为1000-01-01 至 9999-12-31。
2. MySql 按照YYYY-MM-DD 的方式进行该类字段的显示。添加该类字段数据，即可以使用字符串类型，也可以使用数字类型
3. 由于Date类型的字段只记录日期信息，所以如果添加的数据中包含了时间信息，该时间信息将会自动被截断。
4. 如果要保存时间信息，可以考虑使用DateTime类型。

timestamp

1. **4字节**
2. **时区相关**，将当前时区转换为UTC保存，检索时再从UTC转为当前时区
3. **范围小**，'1970-01-01 00:00:01' UTC 到'2038-01-09 03:14:07' UTC
4. **用于跟踪记录变化，每次插入或更新时会同步第一个timestamp类型的字段**
5. 支持索引

datetime

1. **8字节**
2. 保存日期和时间，格式为'YYYY-MM-DD HH:MM:SS'
3. **与时区无关**
4. **范围大**，是'1000-01-01 00:00:00'到'9999-12-31 23:59:59'
5. 不支持索引。

### 字符类型

|  |  |
| --- | --- |
| MySQL | 应用 |
| CHAR(M) | 定长字符 |
| VARCHAR(M) | 变长字符 |
| TINYBLOB | 二进制数据 |
| BLOB |
| MEDIUMBLOB |
| LONGBLOB |
| TINYTEXT | 字符数据 |
| TEXT |
| MEDIUMTEXT |
| LONGTEXT |
| VARBINARY(M) | 二进制变长 |
| BINARY(M) | 二进制定长 |

1. **CHAR vs VARCHAR**

CHAR：长度固定，0-255范围；删除尾部空格；

VARCHAR：可变长字符串，0-65535范围；保留尾部空格；

MyISAM：CHAR

MEMORY：CHAR

InnoDB：VARCHAR

1. **TEXT vs BLOB**

BLOB和TEXT删除时会留空洞，OPTIMIZE TABLE优化碎片

合成索引提高查询性能

BLOB和TEXT列保存在单独表中，减少传输数据

1. **浮点数 vs 定点数**

浮点数存在误差

货币采用定点数

避免浮点数比较

## 存储引擎

### MyISAM

MySQL默认存储引擎

结构：表名.frm(存储表定义)，表名.MYF(MYData，存储数据)，表名.MYI(MYIndex，存储索引)

优点：**访问速度快**

缺点：**不支持事务，不支持外键**

存储格式：静态表（固定字长，存储快，故障易修复，占用空间大，末尾空格去掉），动态表，压缩表

存储引擎：表级锁

### InnoDB

优点：支持事务，支持外键，自动增长列

缺点：占用空间大些

存储方式：共享表空间存储，多表空间存储

存储引擎：行级锁（默认），表级锁

### MEMORY

数据保存在RAM中，.frm，默认HASH索引

优点：访问速度非常快

缺点：对表大小有限制，关闭服务后，数据丢失

存储引擎：表级锁

### MERGE

一组MyISAM表的组合，这些MyISAM表必须结构完全相同，对MERGE表的CRUD实际是对MyISAM表，INSERT\_METHOD=LAST,FIRST

优点：突破对单个MyISAM表大小的限制

### 修改表引擎

create table innodb\_table like myisam\_table;

alter table innodb\_table ENGINE=InnoDB;

insert into innodb\_table select \* from myisam\_table;

## 常用命令

|  |  |  |
| --- | --- | --- |
| show warnings | 查看警告信息 |  |
| show table status like ‘表名’ | 查看表状态 |  |
| show create table 表名 | 查看建表语句 |  |
| show character set | 查看可用字符集 | desc information\_schema.character\_sets |
| show collation like ‘gbk%’ | 查看字符集的校验规则 | desc information\_schema.COLLATIONS |
| show index from 表名 | 查看索引 |  |
| show variables like “table\_type” | 查看默认存储引擎 |  |
| show variables like “have%” | 查看支持存储引擎 | show engines |
| show variable like “character\_%” | 查看字符集 |  |
| show variable like “collation\_%” | 查看校对规则 |  |
| show status | 查看服务器状态 | mysqladmin extended-status |
| show status like “Com\_%” | 查看CRUD次数 |  |
| show status like “Handler\_read%” | 查看索引使用次数 |  |

## 字符集

字符集定义MySQL存储字符串的方式，校对规则定义了比较字符串的方式。

校对规则命名：字符集名称\_语言\_ci(或cs或bin)，如gbk\_chinese\_ci

ci大小写不敏感，cs大小写敏感，bin二元基于字符编码值

MySQL字符集4个级别：服务器、数据库级、表级、字段级

### 设置字符集

# vi /etc/my.cnf

[client]

default-character-set=utf8

[mysql]

default-character-set=utf8

[mysqld]

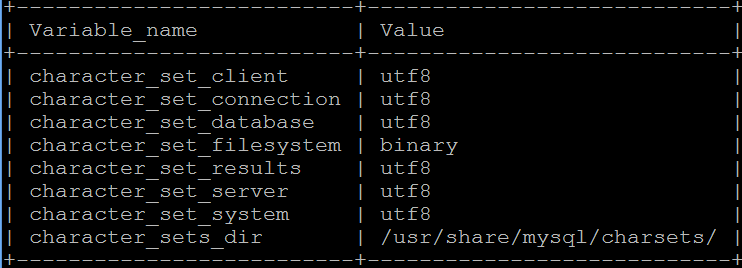
collation-server=utf8\_unicode\_ci

init-connect='SET NAMES utf8'

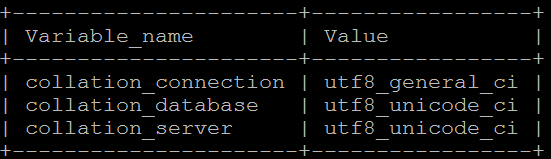
character-set-server=utf8

# service mysqld restart

mysql> show variables like 'char%';



mysql> show variables like 'collation%';



设置数据库字符集

ALTER DATABASE nagios DEFAULT CHARACTER SET utf8;

ALTER TABLE nagios\_instances DEFAULT CHARACTER SET utf8;

### 修改字符集

mysqldump -uroot -p --default-character-set=gbk -d databasename > createtab.sql

将createtab.sql的字符集改为新字符集

mysqldump -uroot -p --quick --no-create-info --extended-insert --default-character-set=latin1 databasename > data.sql

将data.sql的SET NAMES latin1 改为SET NAMES gbk

mysql -uroot -p databasename < createtab.sql

mysql -uroot -p databasename < data.sql

## 索引

1. **HASH**

MEMORY默认

只用于=、>、<比较

不用于ORDER BY

1. **BTREE**

MyISAM，InnoDB默认

支持=、>、<、BETWEEN、!=、LIKE

1. **前缀索引**

对字段前N个字符创建索引，MyISAM1000字符，InnoDB767字符

1. **全文本索引**

仅MyISAM支持，限于CHAR、VARCHAR、TEXT列，对整列索引不支持前缀索引

1. **空间索引**

MyISAM支持

**索引使用原则**

1. 多列索引，查询条件用到最左边的列；
2. like查询，%不在第一个字符；
3. where条件的字符串用引号引起

## SQL Mode

ANSI：保证SQL符合标准SQL语法，便于移植；

STRICT\_TRANS\_TABLES：严格模式，不允许截断长度；

TRANDITIONAL：严格模式，不可插入非法日期，MOD(x,0)等操作；

NO\_BACKSLASH\_ESCAPES：反斜杠变为普通字符；

PIPES\_AS\_CONCAT：||连接操作

select @@sql\_mode

set [session|global] session sql\_mode=’modes’

## SQL优化

show status like ‘Com\_%’

Com\_select，Com\_insert，Com\_update，Com\_delete查看各类SQL执行比例

Com\_commit，Com\_rollback查看事务回滚情况

explain SQL分析索引

analyze table tablename：分析表的关键字分布

check table tablename：检查表是否出错

optimize table tablename：优化表

select \* from tablename procedure analyse()

## 外键

### 设置外键

不建议使用外键，有业务层控制维护关系，外键容易造成死锁，复杂逻辑。

**被引用外键必须为UNIQUE属性，两个外键的属性必须一致**

#### Key

KEY forum (status,type,displayorder)  # 是多列索引（键）   
KEY tid (tid)                         # 是单列索引（键）。

Key建索引，加快查询速度

#### Unique Key

防止数据插入时重复

### 关闭外键

SET FOREIGN\_KEY\_CHECKS=0;

## 其他

### count(1),count(\*),count(col)区别

count(1)和count(\*)没有任何区别，TABLE ACCESS FULL

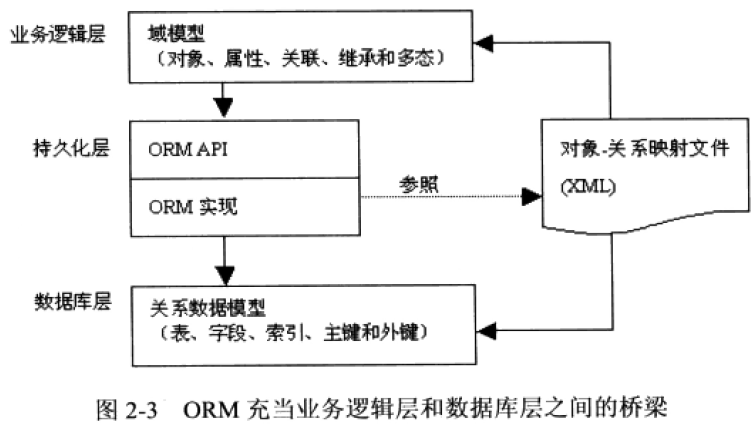
count(col)统计col列不为空的记录，如果有索引，不管col是否为空，都能走索引，没有索引就无法走

### Rand()随机提取行

select \* from tablename order by rand() limit 10;

### group by + with rollup

# hibernate

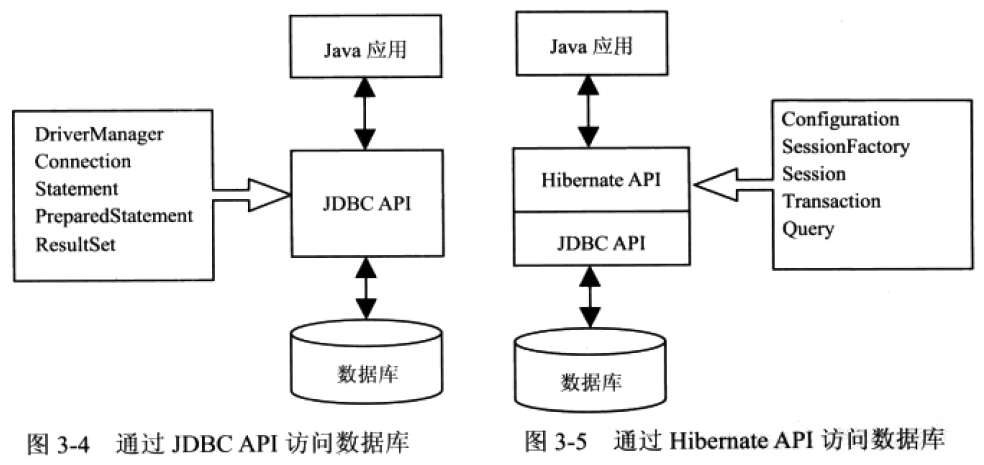


应用层

业务层

持久层

数据库层



hibernate：位于持久层，是域模型和关系数据模型之间的桥梁。ORM

Configuration：配置Hibernate，创建SessionFactory。

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

hibernate自动在当前CLASSPATH中寻找hibernate.cfg.xml并载入内存。

SessionFactory：初始化Hibernate，创建Session，线程安全，重量级，二级缓存。如果要同时访问多个数据库，要为每个数据库创建一个单独的SessionFactory实例。

SessionFactory sessionFactory = config.buildSessionFactory();

Configuration根据当前的数据库配置信息，构造SessionFactory实例返回。

Session：负责CRUD对象，非线程安全，轻量级，一级缓存。

Session session = sessionFactory.openSession();

session.save(user);

session.update(user);

session.delete(user);

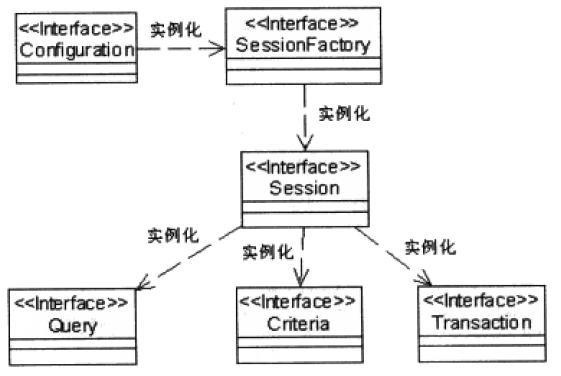
session.load(User.class, id);

session.get()

Transaction：管理事务。

Query，Criteria：执行数据库查询。

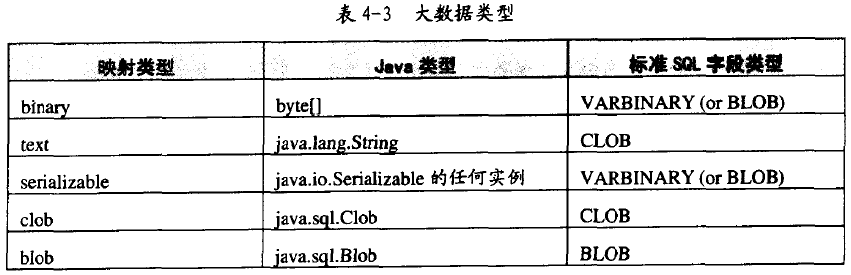
HQL：面向对象，SQL：面向数据库。

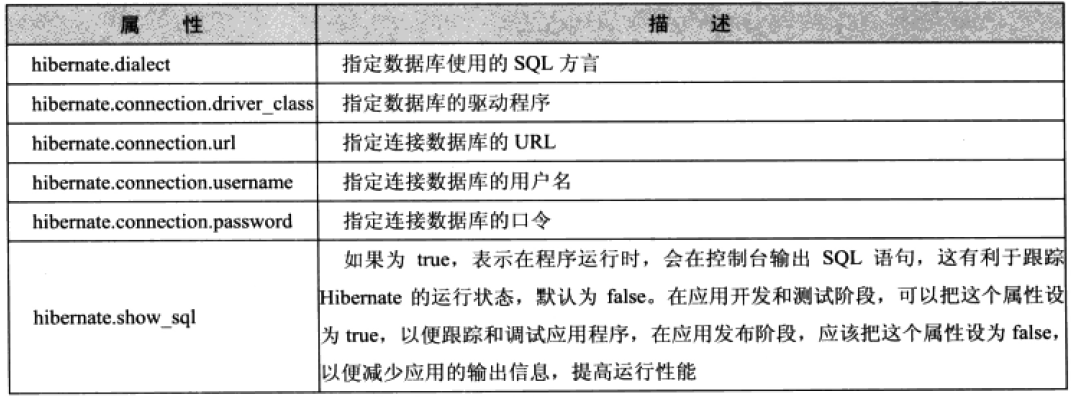


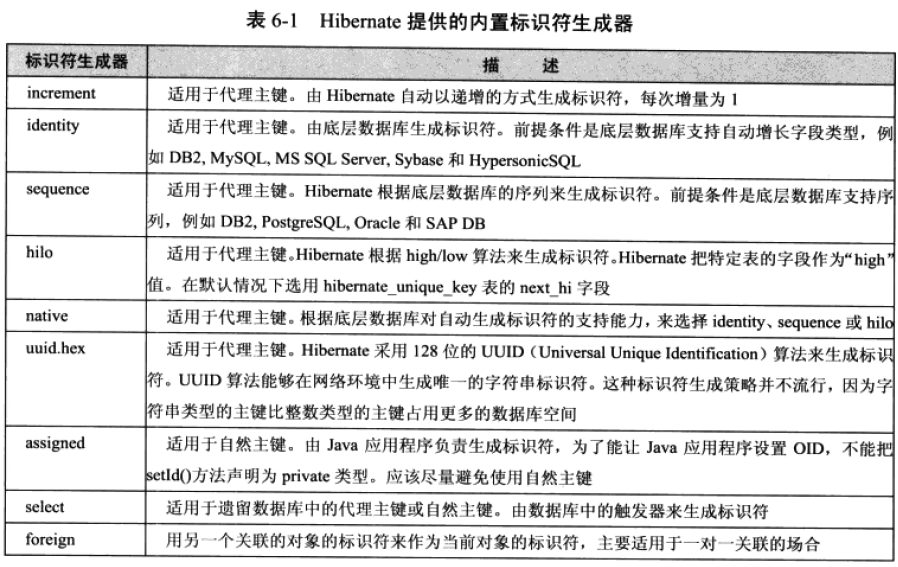
配置Hibernate的步骤：

1. 创建Hibernate配置文件；
2. 创建持久化类；
3. 创建对象-关系映射文件；
4. 通过Hibernate API访问数据库。









hibernate对象的四种状态：

transient：临时状态；

persistent：持久化状态；

removed：删除状态；

detached：游离状态

## Id& GeneratedValue

需要指明GeneratedValue和XXXGenerator

@GeneratedValue 生成ID的值

其属性包括：

* generator：指明生成器名称，在strategy=sequence或table时有效
* strategy：ID生成策略，
  + GenerationType.AUTO：由程序控制，默认值
  + GenerationType.IDENTITY：数据库自动生成，自动增长，适用于MySQL，SQL Server
  + GenerationType.SEQUENCE：序列生成ID，适用于oracle，name该生成器名称，allocationSize步长，initialValue初始值
  + GenerationType.Table：使用特定的数据库表来保存主键

## Cache缓存策略

* + CacheConcurrencyStrategy.NONE：不使用缓存；
  + CacheConcurrencyStrategy.READ\_ONLY：只读，如果对数据进行更新，则抛异常；
  + CacheConcurrencyStrategy.READ\_WRITE：读写模式，在更新缓存时会对缓存里的数据加锁，其他事务如果取相同的缓存数据发现被锁了，就直接去数据库查询。
  + CacheConcurrencyStrategy.NONSTRICT\_READ\_WRITE：不严格的读写模式则不会对缓存数据加锁。
  + CacheConcurrencyStrategy.TRANSACTIONAL：事务模式指缓存支持事务，当事务回滚时，缓存也能回滚，只支持JTA环境。

## Inheritance

## DiscriminatorColumn

## Session方法

flush()：执行清理缓存操作，但不提交事务

commit()：先调用flush方法，然后提交事务

persist()：hibernate 3新增，实现EJB3规范中定义的持久化语义。

save()：

load()：当数据库不存在OID对应的记录时，抛出org.hibernate.ObjectNotFoundException;加载对象以使用其OID时。

get()：不存在则返回null；忽略延迟检索策略（lazy属性）；加载对象以访问其属性时。

# mybatis

