Hibernate课堂笔记

基于Maven的自动构建

1. 在命令行中使用Maven
    - 下载Maven
        - http://maven.apache.org/download.cgi
    - 配置环境变量
        - JAVA_HOME
        - M2_HOME
        - PATH
    - 使用Maven
        - 创建项目 - mvn archetype:generate
        - 编译项目 - mvn compile
        - 测试项目 - mvn test
        - 打包项目 - mvn install
        - 清理项目 - mvn clean
2. 在Eclipse中使用Maven
    - 可以在Window -> Preference菜单中找到Maven项对Maven进行配置，其中主要是配置使用内置的Maven还是自己下载的Maven，另外就是Maven的设置（镜像服务器、本地仓库路径、JDK版本）
    - 创建Maven项目，设定groupId、artifactId、version、packaging
    - 修改pom.xml文件，主要是通过<dependencies>标签及其子标签<dependency>配置依赖项，依赖项可以通过http://mvnrepository.com网站查找

Hibernate五分钟上手

概述

Hibernate是一个ORM框架，ORM指的是对象关系映射，因为我们的Java程序中使用的是面向对象模型，而关系型数据库使用的是关系模型，这两种模型是不匹配的，所以在使用JDBC操作数据库时需要手动的进行两种模型的转换，有了ORM框架之后可以让ORM框架来完成两种模型的转换，我们只需要给出对应的映射就可以了，映射可以是XML或者注解。

> 说明：除了Hibernate还有很多其他的ORM框架，很多ORM框架都是JPA的实现，包括OpenJPA、EclipseLink，还有的不是JPA的实现但也能实现ORM，比如：MyBatis、jOOQ。

POJO + XML / Annotation ==> PO

在pom.xml文件中添加Hibernate依赖项

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.2.10.Final</version>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.38</version>
</dependency>
```

在类路径下添加Hibernate配置文件，默认的名字叫hibernate.cfg.xml，建议不要自己写配置，所有的配置文件都是"拷贝+修改"。

```
<?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>
        <!-- Database connection settings -->
        <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql:///hibernate?
useUnicode=true&amp;characterEncoding=utf8</property>
        <property name="connection.username">root</property>
        <property name="connection.password">123456</property>

        <!-- SQL dialect -->
        <property
name="dialect">org.hibernate.dialect.MySQL57Dialect</property>

        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>

        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">update</property>

        <mapping class="com.qfedu.hib1706.domain.User"/>

    </session-factory>
</hibernate-configuration>
```

映射实体类

```java
package com.qfedu.hib1706.domain;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "tb_user")
public class User implements Serializable {
    @Id
    @Column(length = 20)
    private String username;
    @Column(name = "userpass", length = 20)
    private String password;
    @Column(unique = true)
    private String email;

    public User() {
    }

    public User(String username, String password, String email) {
        this.username = username;
        this.password = password;
        this.email = email;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }
```

```
    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

实体类最好满足以下五个要求：

- 实现Serializable接口
- 属性不要使用基本数据类型
- 保留无参构造器
- 不能是final类
- 必须要有ID属性

通过Hibernate来实现增删改查的操作

```java
package com.qfedu.hib1706;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

import com.qfedu.hib1706.domain.User;

public class App {

    public static void main(String[] args) {
        User user = new User("admin", "123123", "admin@qq.com");
        SessionFactory factory = new Configuration().configure()
                .buildSessionFactory();
        Session session = factory.openSession();
        session.beginTransaction();
        session.save(user);
        session.getTransaction().commit();
        session.close();
        factory.close();
    }
}
```