Day3 3月1日

软件51 庞建业 2151601012

hibernate

冬眠

ORM框架

> 对象关系映射关系 ， 面向对象的对象模型和关系型数据之间的相互转换。基于关系型数据库的数据存储，实现一个虚拟的面向对象的数据访问接口。理想状态下，基于一个这样一个面向对象的接口，持久化一个oo对象应该不需要了解任何关系型数据库存储的数据实现细节。

简化数据库操作

数据库方言 数据库语言 数据库迁移

数据持久层 Hibernate/Mybatis

object/relation mapping对象/关系 映射 对象与关系数据库的映射

解决 `面向对象编程` 与 `关系数据库` 不匹配的问题

（1）建数据库：解决不匹配（子表 父表）

（2）封装不匹配 （地址 粒度 粗粒度/细粒度 省市县区->省 市 县 区 **粒度不同**）

地址做几个字段的设计 （各有各的好处）

对象继承关系 表与表之间只有主表子表

Key问题

数据库 对象不匹配

ORM框架解决 对象送到数据库中形同冬眠操作

java类（pojo/domain object）映射关系型数据库中的表

最好不要取name 数据库本身有name关键字

CRUD

增删改查找 **业务**

数据库的业务CRUD一定是放到**DAO**中 数据访问对象

控制器先调用service再调DAO（CRUD）

hibernate也有配置文件 .xml

```
ORM:
面向对象的java语言和关系型数据库之间的对应关系
类----表
类属性----表中的字段
对象----数据库的记录
```

使用hibernate

ORMapping：对象关系映射，通过在类与表之间建立关系，使程序操作类能自动影响到表中的数据。

	优点	**缺点**
JDBC	好学，执行速度快	重复代码比较多，开发速度慢
EJB1,2	提出了ORMapping	除了有概念，什么都不行。
JDO	简单	连接控制有问题
Apache OJB	无	太多
Hibernate	很多	执行速度慢
MyBATIS	比Hibernate执行速度快比JDBC代码简单	比Hibernate代码多比JDBC执行速度慢
EJB3	使用了Hibernate的源代码	架构还是EJB的原始架构

```
public interface IDAO<K, V> {
    public void doCreate(V vo) throws Exception;
    public void doUpdate(V vo) throws Exception;
    public void doRemove(K id) throws Exception;
    public List<V> findAll() throws Exception;
    public V findById(K id) throws Exception;
    //分页查询
    public List<V> findAll(int pageNo, int pageSize, String keyword,
            String column) throws Exception;
```

```
Connection    ->   Session
PreparedStatement     ->  Query
ResultSet    ->   List
DataBaseConnection     ->   HibernateSessionFactory
```

```
建立工厂类
public class DAOFactory {
    public static INewsDAO getINewsDAOInstance() {
        return new NewsDAOImpl();
    }
}
```

```
public interface INewsService {
    public void insert(News news) throws Exception;
    public void delete(int id) throws Exception;
    public News findById(int id) throws Exception;
    // 如果要一次性返回多种类型的数据,可以使用Map集合,这样方便区分.
    public Map<String, Object> list(int pageNo, int pageSize, String keyword,
            String column) throws Exception;
}
```

```
//建立实现类
public News findById(int id) throws Exception {
        News news = null;
        try {
            news = DAOFactory.getINewsDAOInstance().findById(id);
        } catch (Exception e) {
            e.printStackTrace();
            throw e;
        } finally {
            HibernateSessionFactory.closeSession();
        }
        return news;
    }
```

```
public void insert(News news) throws Exception {
        // 加入事务处理功能
        Transaction tx = HibernateSessionFactory.getSession()
                .beginTransaction();
        try {
            DAOFactory.getINewsDAOInstance().doCreate(news);
            tx.commit();
        } catch (Exception e) {
            e.printStackTrace();
            tx.rollback();
            throw e;
        } finally {
            HibernateSessionFactory.closeSession();
        }
    }
```

POJO

没有/不需要实现特定接口的类：与EJB相对

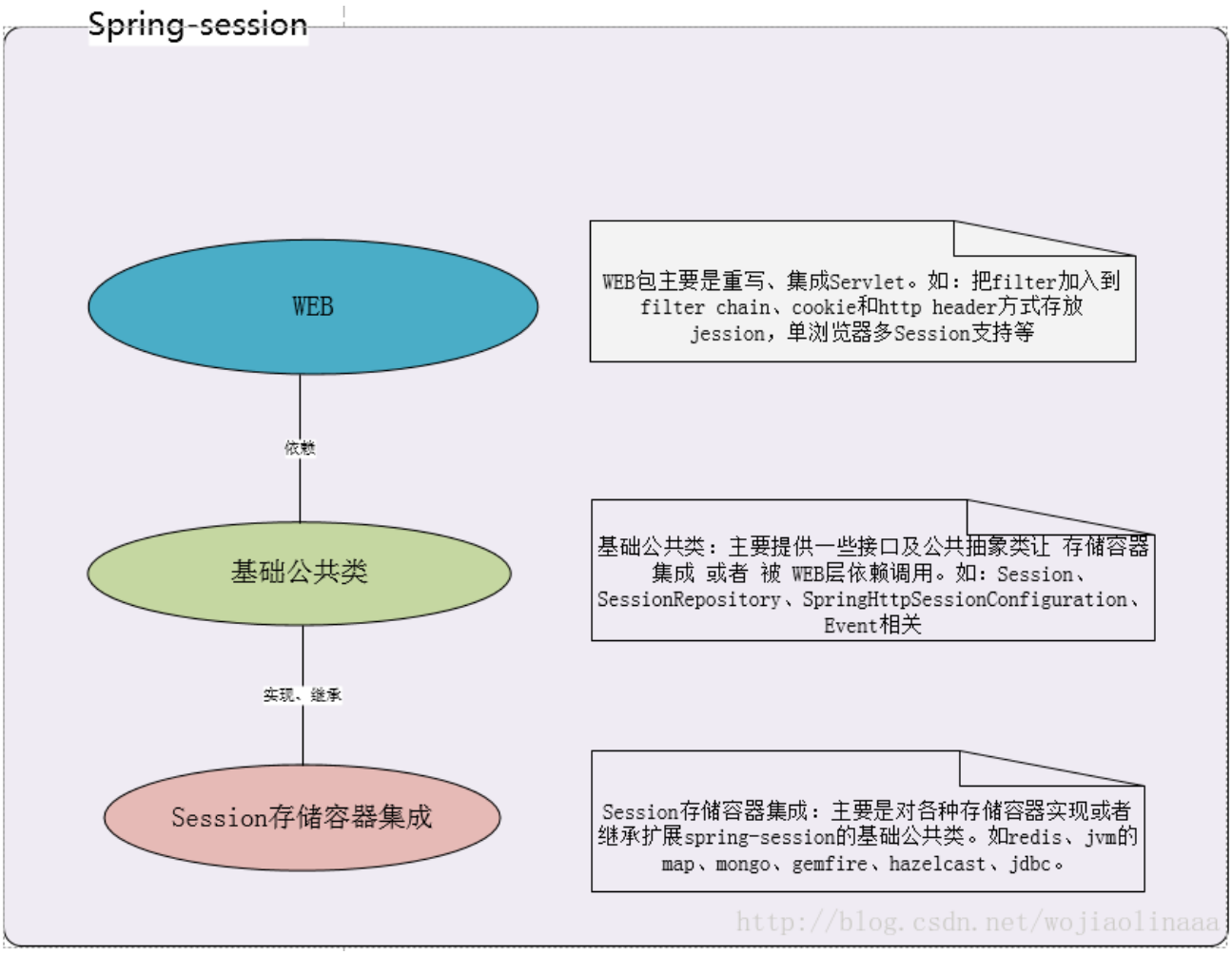
普通的JAVA对象

（1）每个hibernate有个主键 哈希生成

myeclipse中集成好了 hibernate

> hibernate session工厂： 做增删改查找的对象
>
> Spring Session provides an API and implementations for managing a user’s session information. It also provides transparent integration with:

> - HttpSession
>
>   - allows replacing the HttpSession in an application container (i.e. Tomcat) neutral way. Additional features include:
>
>   - **Clustered Sessions** - Spring Session makes it trivial to support clustered sessions without being tied to an application container specific solution.
>   - **Multiple Browser Sessions** - Spring Session supports managing multiple users' sessions in a single browser instance (i.e. multiple authenticated accounts similar to Google).
>   - **RESTful APIs** - Spring Session allows providing session ids in headers to work with RESTful APIs
> - WebSocket - provides the ability to keep the `HttpSession` alive when receiving WebSocket messages



对象与关系库间建立联系的会话

三种做数据库方式 JPA EJB hibernate

（1）Database explorar

配置数据库链接

（2）建项目：web项目

账号密码 unicode

```
<session-factory>
sql
```

（3）反向工程 模型类 注解方式 更新注解文件

映射类型 主键生成策略

```
save delect findby..
```

```
Session s=getSession();
Transaction ts=s.beginTransaction();
//事务
```

JAVA反射自动把表单反射回new对象中

```
Java的反射机制的实现要借助于4个类：class，Constructor，Field，Method;
```

其中class代表的时类对 象，Constructor - 类的构造器对象，Field - 类的属性对象，Method - 类的方法对象。通过这四个对象我们可以粗略的看到一个类的各个组 成部分。

Java反射的作用：

在Java运行时环境中，对于任意一个类，可以知道这个类有哪些属性和方法。对于任意一个对象，可以调用它的任意一个方法。这种动态获取类的信息以及动态调用对象的方法的功能来自于Java 语言的反射（Reflection）机制。

Java 反射机制主要提供了以下功能

```
在运行时判断任意一个对象所属的类。
在运行时构造任意一个类的对象。
在运行时判断任意一个类所具有的成员变量和方法。
在运行时调用任意一个对象的方法
```

```
得到构造器的方法
Constructor getConstructor(Class[] params) -- 获得使用特殊的参数类型的公共构造函数，
Constructor[] getConstructors() -- 获得类的所有公共构造函数
Constructor getDeclaredConstructor(Class[] params) -- 获得使用特定参数类型的构造函数(与接入级别无关)
Constructor[] getDeclaredConstructors() -- 获得类的所有构造函数(与接入级别无关)
```

```
获得字段信息的方法Field getField(String name) -- 获得命名的公共字段
Field[] getFields() -- 获得类的所有公共字段
Field getDeclaredField(String name) -- 获得类声明的命名的字段
Field[] getDeclaredFields() -- 获得类声明的所有字段
```

```
获得方法信息的方法Method getMethod(String name, Class[] params) -- 使用特定的参数类型，获得命名的公共方法

Method[] getMethods() -- 获得类的所有公共方法

Method getDeclaredMethod(String name, Class[] params) -- 使用特写的参数类型，获得类声明的命名的方法

Method[] getDeclaredMethods() -- 获得类声明的所有方法
```

hibernate有三个对象

游离对象 跟数据库无关的对象

持久化对象/托管状态对象 连接后或者关闭session变成持久对象

有数据库能不能变表结构：

默认加到get方法上

建议全部加到属性上

ddl 数据库定义语言 mdl 数据库操作语言

correct 默认创建新的 后面用update来更新

原理

session工厂自动找出文件

将session放到本地线程中 默认定位hibernate.xml

session对象有自身的增删改查找 所以用它做数据库操作

```
List<Test> list=dao.findAll();
for(Test test:list){
    System.out.println(test.getId());
}
//把所有对象打印出来
```

Spring Boot & Spring Cloud & Spring Cloud Dataflow

```
1、开发简单：简化Maven及Gradle配置，尽可能的自动化配置Spring，减少大量的xml配置
2、部署简单：嵌入式Tomcat，Jetty容器，无需部署WAR包，创建独立Spring应用程序
3、监控简单：提供完善的监控服务actuator
```

五个模块

```
issue-entity,
issue-dao,
issue-service,
issue-controller,
issue-application
```

```
<properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>
    //pom.xml中加入utf-8 不然会产生乱码
```

创建User实体

```
public class User{
    private Integer id;
    private String username;
    private String password;
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    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;
    }

    @Override
    public String toString() {
        return "User [id=" + id + ", username=" + username + ", password=" + password + "]";
    }
}
```

创建User-Dao

```
    @Bean
    @ConfigurationProperties(prefix = "spring.datasource")
    public DataSource dataSource(){
        return new com.alibaba.druid.pool.DruidDataSource();
    }
```

```
    @Bean(name="sqlSessionFactory")
    public SqlSessionFactory sqlSessionFactory() throws  Exception{
        SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();
        sqlSessionFactoryBean.setDataSource(dataSource());
        PathMatchingResourcePatternResolver resolver = new
PathMatchingResourcePatternResolver();
        sqlSessionFactoryBean.setMapperLocations(resolver.getResources(mapperLocationPattern));
        return sqlSessionFactoryBean.getObject();
    }
```

mysql加入pom.xml文件

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
 <version>5.1.45</version>
```

service文件

```
@Service
public class UserService{

    @Autowired
    UserMapper userMapper;

    @Override
    public void add(User user) {
        userMapper.add(user);
    }
}
```

并且完善pom文件

controller模块

```
@RestController
@RequestMapping("user")
public class UserController {
    @Autowired
    private UserService userService;

    @RequestMapping("add")
    public User add() {
        User user = new User();
        user.setPassword("123");
        user.setUsername("hello");
        userService.add(user);
        return user;
    }
}
```

对Spring boot demo进行测试

```
public class WebApplicationTest {

    @Autowired
    private UserMapper userMapper;
    @Test
    public void test(){
        User user = new User();
        user.setPassword("a");

        userMapper.add(user);
    }
}
```