中山大學
SUN YAT-SEN UNIVERSITY

《分布式计算》

Spring 数据访问

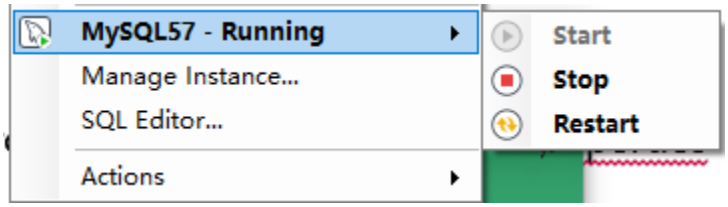
学 院 名 称 ： 数据科学与计算机学院

成　　员 ： 陈伟宸

时　　间 ： 2016 年 11 月 8 日

1. 代码见 gtvg-mvc-jdbc.rar

2. （1）安装并配置好 MySQL Server



（2）在 pom.xml 中添加 Hibernate、MySQL 等依赖

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>${hibernate.version}</version>
</dependency>
<dependency>
    <groupId>org.javassist</groupId>
    <artifactId>javassist</artifactId>
    <version>3.18.1-GA</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mysql-connector-java.version}</version>
    <scope>runtime</scope>
</dependency>
```

（3）在 resources 中添加 persistence-mysql.properties

```
# jdbc.X
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/gtvg?createDatabaseIfNotExist=true&useSSL=false
jdbc.user=root
jdbc.pass=CHENweichen

# hibernate.X
hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
hibernate.show_sql=false
hibernate.hbm2ddl.auto=create-drop
```

（4）新建 persistence 包，在其中再新建三个包，分别为：dao，model，services

（5）修改每个 model，让每个类对应数据库中的一个表，每个类的成员变量对应于相应数据库中的一个字段

Comment 类：

```
@Entity
public class Comment implements Serializable {

    @Id
    @Column(name = "COMMENT_ID")
    private Integer id = null;

    @Column(name = "TEXT")
    private String text = null;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "PRODUCT_ID")
    private Product product = null;
```

Customer 类：

```
@Entity
public class Customer implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "CUSTOMER_ID")
    private Integer id = null;

    @Column(name = "NAME")
    private String name = null;

    @Column(name = "CUSTOMERSINCE")
    private Calendar customerSince = null;
```

Order 类：

```java
@Entity
public class ProductOrder implements Serializable{

    @Id
    @Column(name = "ORDER_ID")
    private Integer id = null;

    @Column(name = "DATE")
    private Calendar date = null;

    @ManyToOne(cascade = CascadeType.MERGE)
    @JoinColumn(name = "CUSTOMER")
    private Customer customer = null;

    @Column(name = "ORDERLINES")
    @ElementCollection(targetClass=OrderLine.class)
    private List<OrderLine> orderLines = new ArrayList<>();
```

OrderLine 类：

```java
@Entity
public class OrderLine implements Serializable{

    @Id
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "PRODUCT")
    private Product product = null;

    @Id
    @Column(name = "AMOUNT")
    private Integer amount = null;

    @Id
    @Column(name = "PURCHASEPRICE")
    private BigDecimal purchasePrice = null;
```

Product 类：

```java
@Entity
public class Product implements Serializable{

    @Id
    @Column(name = "PRODUCT_ID")
    private Integer id = null;

    @Column(name = "NAME")
    private String name = null;

    @Column(name = "PRICE")
    private BigDecimal price = null;

    @Column(name = "INSTOCK")
    private boolean inStock = false;

    @OneToMany(mappedBy = "product", fetch = FetchType.EAGER)
    @Column(name = "COMMENTS")
    private List<Comment> comments = new ArrayList<>();
```

User 类：

```java
@Entity
public class User implements Serializable{

    @Id
    @Column(name = "FIRSTNAME")
    private String firstName = null;

    @Column(name = "LASTNAME")
    private String lastName = null;

    @Column(name = "NATIONALITY")
    private String nationality = null;

    @Column(name = "AGE")
    private Integer age = null;
```

（6）实现每个类的 dao

① 定义一个抽象类，AbstractJpaDAO 类，通过 EntityManager 完成对数据库的操作

```java
private Class<T> clazz;

@PersistenceContext
private EntityManager entityManager;

public EntityManager getEntityManager() { return entityManager; }

public final void setClazz(final Class<T> clazzToSet) { this.clazz = clazzToSet; }

public T findOne(final long id) { return entityManager.find(clazz, id); }

/unchecked/
public List<T> findAll() { return entityManager.createQuery("from " + clazz.getName()).getResultList(); }

public void create(final T entity) {
    entityManager.persist(entity);
}

public T update(final T entity) { return entityManager.merge(entity); }

public void delete(final T entity) { entityManager.remove(entity); }

public void deleteById(final long entityId) {
    final T entity = findOne(entityId);
    delete(entity);
}
```

② 对于每个 model 都申明一个 Dao 接口，然后实现该接口

CommentDao：

```java
public interface CommentDao {
    Comment findOne(long id);

    List<Comment> findAll();

    void create(Comment comment);

    Comment update(Comment comment);

    void delete(Comment comment);

    void deleteById(long commentId);
}
```

CommentDaoImpl：

```java
@Repository
public class CommentDaoImpl extends AbstractJpaDAO<Comment> implements CommentDao{

    public CommentDaoImpl() {
        super();

        setClazz(Comment.class);
    }

}
```

CustomerDao：

```java
public interface CustomerDao {
    Customer findOne(long id);

    List<Customer> findAll();

    void create(Customer customer);

    Customer update(Customer customer);

    void delete(Customer customer);

    void deleteById(long customerId);
}
```

CustomerDaoImpl：

```java
@Repository
public class CustomerDaoImpl extends AbstractJpaDAO<Customer> implements CustomerDao {
    public CustomerDaoImpl() {
        super();

        setClazz(Customer.class);
    }
}
```

OrderDao：

```java
public interface ProductDao {
    Product findOne(long id);

    List<Product> findAll();

    void create(Product product);

    Product update(Product product);

    void delete(Product product);

    void deleteById(long productId);
}
```

OrderDaoImpl :

```java
@Repository
public class ProductDaoImpl extends AbstractJpaDAO<Product> implements ProductDao{

    public ProductDaoImpl() {
        super();

        setClazz(Product.class);
    }
}
```

OrderLineDao :

```java
public interface OrderLineDao {
    OrderLine findOne(long id);

    List<OrderLine> findAll();

    void create(OrderLine orderLine);

    OrderLine update(OrderLine orderLine);

    void delete(OrderLine orderLine);
}
```

OrderLineDaoImpl :

```java
@Repository
public class OrderLineDaoImpl extends AbstractJpaDAO<OrderLine> implements OrderLineDao {

    public OrderLineDaoImpl() {
        super();

        setClazz(OrderLine.class);
    }

}
```

ProduceDao：

```java
public interface ProductDao {
    Product findOne(long id);

    List<Product> findAll();

    void create(Product product);

    Product update(Product product);

    void delete(Product product);

    void deleteById(long productId);
}
```

ProductDaoImpl：

```java
@Repository
public class ProductDaoImpl extends AbstractJpaDAO<Product> implements ProductDao{

    public ProductDaoImpl() {
        super();

        setClazz(Product.class);
    }
}
```

UserDao：

```java
public interface UserDao {
    User findOne(long id);

    List<User> findAll();

    void create(User user);

    User update(User user);

    void delete(User user);

    void deleteById(long userId);
}
```

UserDaoImpl：

```java
@Repository
public class UserDaoImpl extends AbstractJpaDAO<User> implements UserDao {
    public UserDaoImpl() {
        super();

        setClazz(User.class);
    }
}
```

（7）实现 Service

CustomerService：

```java
@Service
@Transactional
public class CustomerService {

    @Autowired
    private CustomerDao dao;

    public CustomerService() { super(); }

    public void create(Customer customer) {
        dao.create(customer);
    }

    public void update(Customer customer) {dao.update(customer); }

    public List<Customer> findAll() { return dao.findAll(); }

    public Customer findById(final Integer id) { return dao.findOne(id); }

}
```

OrderLineService：

```java
@Service
@Transactional
public class OrderLineService {
    @Autowired
    private OrderLineDao dao;

    public OrderLineService() { super(); }

    public void create(OrderLine orderLine) { dao.create(orderLine); }

    public List<OrderLine> findAll() { return dao.findAll(); }

}
```

OrderService：

```java
@Service
@Transactional
public class OrderService {

    @Autowired
    private OrderDao dao;

    public OrderService() { super(); }

    public void create(ProductOrder productOrder) {
        dao.create(productOrder);
    }

    public List<ProductOrder> findAll() { return dao.findAll(); }

    public ProductOrder findById(final Integer id) { return dao.findOne(id); }

    public BigDecimal getTotal() {...}

}
```

ProductService：

```java
@Service
@Transactional
public class ProductService {

    @Autowired
    private ProductDao dao;

    public ProductService() { super(); }

    public void create(Product product) {
        dao.create(product);
    }

    public List<Product> findAll() { return dao.findAll(); }

    public Product findById(final Integer id) { return dao.findOne(id); }

}
```

（8）配置 JPA。新建 config 包，实现 PersistenceJPAConfig

```java
@Configuration
@ComponentScan({ "thymeleafexamples.gtvg.persistence" })
@EnableTransactionManagement
@PropertySource({ "classpath:persistence-mysql.properties" })
//@EnableJpaRepositories(basePackages = "me.database.persistence.dao")
public class PersistenceJPAConfig {

    @Autowired
    private Environment env;

    public PersistenceJPAConfig() { super(); }

    // beans

    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
        final LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();
        em.setDataSource(dataSource());
        em.setPackagesToScan(new String[] { "thymeleafexamples.gtvg.persistence.model" });

        final HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
        em.setJpaVendorAdapter(vendorAdapter);
        em.setJpaProperties(additionalProperties());

        return em;
    }

    @Bean
    public DataSource dataSource() {
        final BasicDataSource dataSource = new BasicDataSource();
        dataSource.setDriverClassName(Preconditions.checkNotNull(env.getProperty("jdbc.driverClassName")));
        dataSource.setUrl(Preconditions.checkNotNull(env.getProperty("jdbc.url")));
        dataSource.setUsername(Preconditions.checkNotNull(env.getProperty("jdbc.user")));
        dataSource.setPassword(Preconditions.checkNotNull(env.getProperty("jdbc.pass")));

        return dataSource;
    }

    @Bean
    public PlatformTransactionManager transactionManager(final EntityManagerFactory emf) {
        final JpaTransactionManager transactionManager = new JpaTransactionManager();
        transactionManager.setEntityManagerFactory(emf);
        return transactionManager;
    }

    @Bean
    public PersistenceExceptionTranslationPostProcessor exceptionTranslation() {
        return new PersistenceExceptionTranslationPostProcessor();
    }

    final Properties additionalProperties() {
        final Properties hibernateProperties = new Properties();
        hibernateProperties.setProperty("hibernate.hbm2ddl.auto", env.getProperty("hibernate.hbm2ddl.auto"));
        hibernateProperties.setProperty("hibernate.dialect", env.getProperty("hibernate.dialect"));
        // hibernateProperties.setProperty("hibernate.globally_quoted_identifiers", "true");
        return hibernateProperties;
    }
```

（9）修改 Controller，将数据访问方式修改为使用 Dao

OrderDetailsController

```
@Controller
public class OrderDetailsController  {
    @RequestMapping("/order/details")
    public String orderDetails(@RequestParam(value="orderId") int orderId, Model model) {
        AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
        ctx.register(PersistenceJPAConfig.class);
        ctx.refresh();
        OrderService os = ctx.getBean(OrderService.class);
        final Order order = os.findById(orderId);
        model.addAttribute("order", order);
        return "order/details";
    }
}
```

OrderListController：

```
@Controller
public class OrderListController {
    @RequestMapping("order/list")
    public String orderList(Model model) {
        AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
        ctx.register(PersistenceJPAConfig.class);
        ctx.refresh();
        OrderService os = ctx.getBean(OrderService.class);
        final List<Order> allOrders = os.findAll();
        model.addAttribute("orders", allOrders);
        return "order/list";
    }
}
```

ProductCommentsController：

```java
@Controller
public class ProductCommentsController {
    @RequestMapping("product/comments")
    public String productComments(@RequestParam(value="prodId") int prodId, Model model) {
        AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
        ctx.register(PersistenceJPAConfig.class);
        ctx.refresh();
        ProductService ps = ctx.getBean(ProductService.class);
        final Product product = ps.findById(prodId);
        model.addAttribute("prod", product);
        return "product/comments";
    }
}
```

ProductListController：

```java
@Controller
public class ProductListController {
    @RequestMapping("product/list")
    public String productList(Model model) {
        AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
        ctx.register(PersistenceJPAConfig.class);
        ctx.refresh();
        ProductService ps = ctx.getBean(ProductService.class);
        final List<Product> allProducts = ps.findAll();
        model.addAttribute("prods", allProducts);
        return "product/list";
    }
}
```

CommentService：

```java
@Service
@Transactional
public class CommentService {

    @Autowired
    private CommentDao dao;

    public CommentService() { super(); }

    public void create(Comment comment) {
        dao.create(comment);
    }

    public void update(Comment comment) {dao.update(comment); }

    public List<Comment> findAll() { return dao.findAll(); }

    public Comment findById(final Integer id) { return dao.findOne(id); }

}
```

（10）在 GTVGApplication.java 中向数据库中写入信息，

```
AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
ctx.register(PersistenceJPAConfig.class);
ctx.refresh();
ProductService ps = ctx.getBean(ProductService.class);
CustomerService cs = ctx.getBean(CustomerService.class);
OrderService os = ctx.getBean(OrderService.class);
OrderLineService ols = ctx.getBean(OrderLineService.class);
CommentService cms = ctx.getBean(CommentService.class);

Customer customer1 = new Customer();
customer1.setId(1);
customer1.setName("David");
customer1.setCustomerSince(CalendarUtil.calendarFor(2002, 3, 2, 10, 23));
Customer customer2 = new Customer();
customer2.setId(2);
customer2.setName("Nancy");
customer2.setCustomerSince(CalendarUtil.calendarFor(2014, 3, 2, 1, 5));

cs.create(customer1);
cs.create(customer2);

Comment comment1 = new Comment();
comment1.setId(1);
comment1.setText("comment1");
Comment comment2 = new Comment();
comment2.setId(2);
comment2.setText("comment2");
```

```
cms.create(comment1);
cms.create(comment2);

Product product1 = new Product(1, "test1", false, new BigDecimal(100));
Product product2 = new Product(2, "test2", true, new BigDecimal(200));
product1.getComments().add(comment1);
product1.getComments().add(comment2);

ps.create(product1);
ps.create(product2);

OrderLine orderLine1 = new OrderLine();
orderLine1.setProduct(product1);
orderLine1.setAmount(5);
orderLine1.setPurchasePrice(new BigDecimal("10"));
OrderLine orderLine2 = new OrderLine();
orderLine2.setProduct(product2);
orderLine2.setAmount(10);
orderLine2.setPurchasePrice(new BigDecimal("20"));

ols.create(orderLine1);
ols.create(orderLine2);

ProductOrder order1 = new ProductOrder(1, CalendarUtil.calendarFor(2009, 1, 12, 10, 23), customer1);
order1.getOrderLines().add(orderLine1);

ProductOrder order2 = new ProductOrder(2, CalendarUtil.calendarFor(2016, 3, 2, 8, 3), customer2);
order2.getOrderLines().add(orderLine2);

os.create(order1);
os.create(order2);
```

并使用 mvn tomcat7:run 命令运行程序，查看数据库

```
mysql> use gtvg;
Database changed
mysql> show tables;
+-----------------------------+
| Tables_in_gtvg              |
+-----------------------------+
| comment                     |
| customer                    |
| orderline                   |
| product                     |
| product_comment             |
| productorder                |
| productorder_orderline      |
| user                        |
+-----------------------------+
8 rows in set (0.00 sec)

mysql>
```

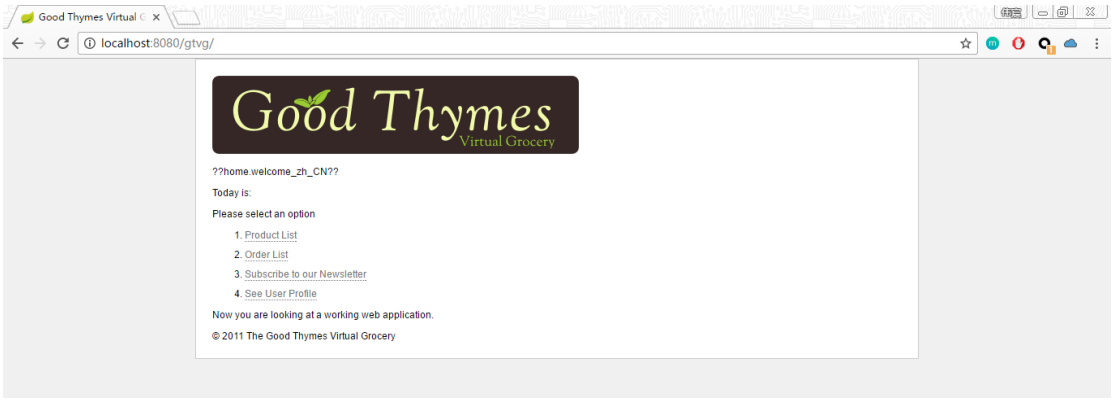
各表成功创建

（11）终止程序运行，将属性 hibernate.hbm2ddl.auto 修改为 update

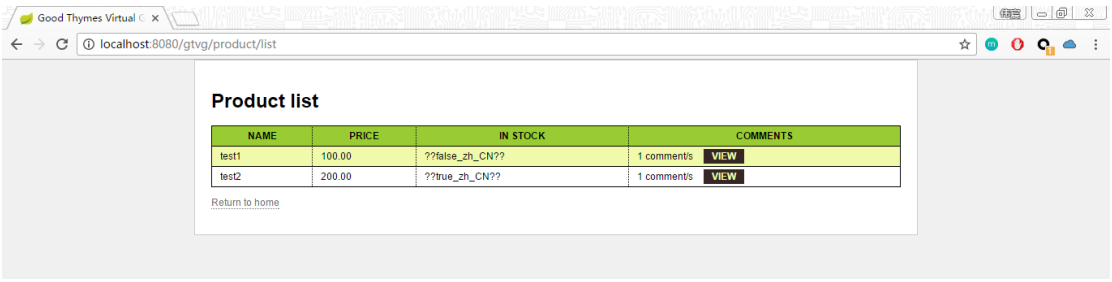
```
hibernate.hbm2ddl.auto=update
```

删除 GTVGApplication 中对数据库的写操作，再次运行程序

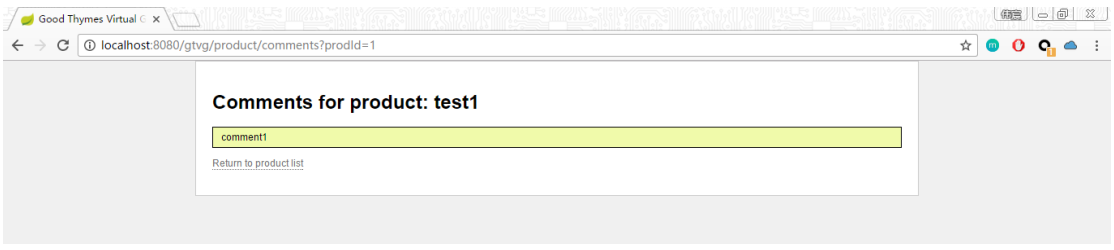
（12）访问 localhost:8080/gtvg

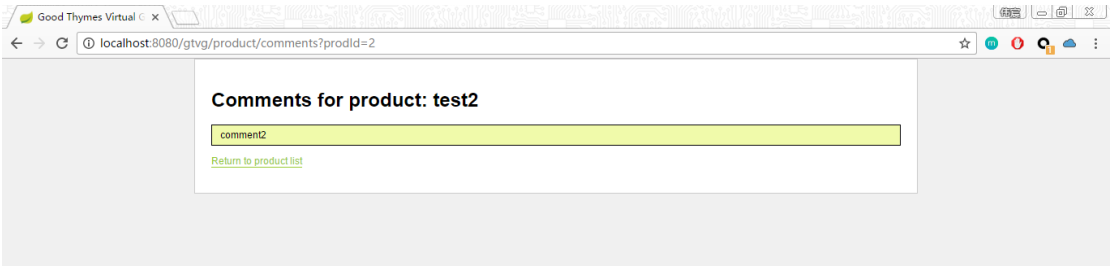


点击 Product List



点击 VIEW




Good Thymes Virtual
localhost:8080/gtvg/product/comments?prodId=2
Comments for product: test2
comment2
Return to product list

总结

1. MySQL 中 order 是关键字 ,所以必须修改 order 类的类名 ,order 表才能被正确地创建 ！！！！！！！！！

2. 由于在 ProductList 中需要加载 comments ,所以不能使用 LAZY 的加载方式，应该使用 EAGER

```
@OneToMany(mappedBy = "product", fetch = FetchType.EAGER)
@Column(name = "COMMENTS")
private List<Comment> comments = new ArrayList<>();
```

3. hibernate.hbm2ddl.auto 属性被设置为 create-drop 时 ,在程序运行结束后会清空表中的所有内容。在成功创建所有表后，应该将该属性改为 update，再向数据库中添加数据，数据才会保存下来