Spring Framework 基础培训

JavaEE 框架培训之二

郝金隆（ haojinlong@189.cn ）

2014 年 7 月

培训说明

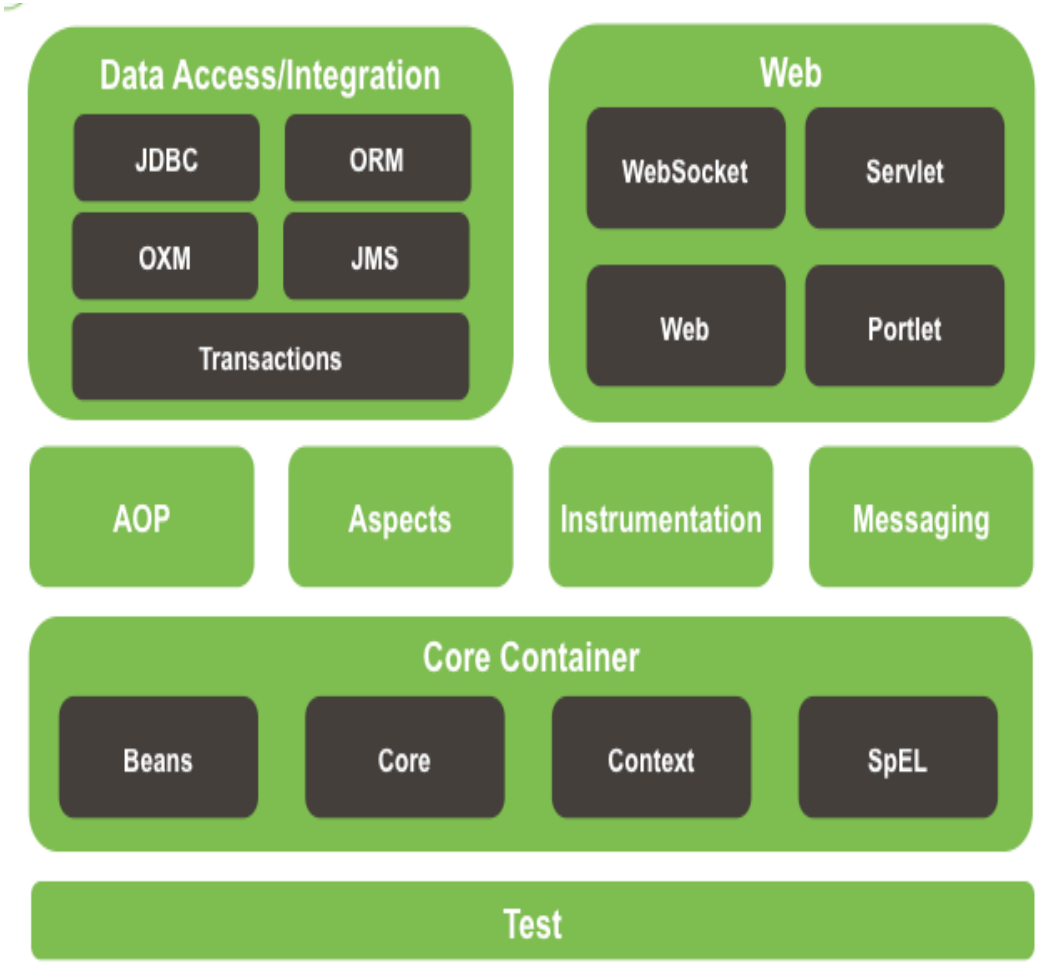
- 培训假设
  - 本培训材料认为阅读人员具备一定的 Java 开发基础，熟悉 Java 开发环境的配置和使用（如 JDK、 Eclipse 的安装等）
  - 前文阅读：《Java 日志框架培训》
- 本培训的内容
  - 包括 spring 框架的基本使用方法和技巧，以及配置文件说明
  - 培训包括 spring 与数据持久化框架的集成方式
  - 包括使用 spring 进行远程服务调用
  - 不包括 spring mvc 使用方法（将专题进行）
- 更多培训资料，请访问
  - https://github.com/haojinlong/trainning/tree/master/doc/pdf
  - www.haojinlong.net（建设中）

目录

Spring 简介

- Spring 简介
  - Spring 是一个开源框架， Spring 是于 2003 年兴起的一个轻量级的 Java 开发框架，由 Rod Johnson 在其著作 Expert One-On-One J2EE Development and Design 中阐述的部分理念和原型衍生而来。它是为了解决企业应用开发的复杂性而创建的。 Spring 使用基本的 JavaBean 来完成以前只可能由 EJB 完成的事情。然而， Spring 的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何 Java 应用都可以从 Spring 中受益。
  - 目的：解决企业应用开发的复杂性
  - 功能：使用基本的 JavaBean 代替 EJB ，并提供了更多的企业应用功能
- 下载地址
  - 官方地址： http://projects.spring.io/spring-framework/
  - 下载地址： http://repo.spring.io/release/org/springframework/spring/
- 所依赖的包（ Java 日志）
  - commons-logging（可以使用 jcl-over-slf4j 代替）、 slf4j
  - 关于 Java 日志的使用，可参见《 Java 日志框架培训》

Spring Framework 整体框架



Core Containner（核心容器）：

- spring-core、 spring-bean：框架的最基础部分
- spring-context：在 core 和 bean 的基础上，提供被管对象的封装
- spring-expression：表达式的语义解析和封装

Data Access/Integration（数据存取 / 封装）：

- spring-jdbc：提供一个 jdbc 的抽象层
- Spring-orm：提供 OR 映射的 API，包括与 JDO、 JPA、 Hibernate 的集成
- Spring-oxm：提供与 XML 数据的映射
- Spring-jms：包含了生产和消费消息的功能
- Spring-tx：事务管理

WEB：

- Spring-web： spring web 基础整合共同
- Spring-webmvc： spring 自己的 mvc 框架
- Spring-websocket： websocket 支持，主要用于与客户端交互

AOP：使用注解时必须引入， spring 的一种增强，实现声明式事务管理

Aspects：提供对 aspectj 的集成支撑

Messaging：

目录

第一个 Spring 实例 (1/4)：创建接口类

```
/**
 * # Person.java -- (2014 年 7 月 18 日 )
 * 作者：郝金隆
 * 联系方式： haojinlong@189.cn
 */
package com.github.haojinlong.trainning.spring.basic.inter;

public interface Person {
        public String getName();
        public int getAge();
}
```

```
/**
 * # SayHello.java -- (2014 年 7 月 18 日 )
 * 作者：郝金隆
 * 联系方式： haojinlong@189.cn
 */
package com.github.haojinlong.trainning.spring.basic.inter;

public interface SayHello {
        public String sayHelo();

}
```

基础 spring 应用所需的 jar 包括：

- Spring-core
- Spring-beans
- Spring-context
- Spring-expressions

外部依赖包括：

- commons-logging 或者 jcl-over-slf4j
- slf4j-api
- 日志实现框架（ log4j/log4j2/logback ），具体请参考 java 日志框架培训

第一个 spring 示例 (2/4)：创建实现类

```
package com.github.haojinlong.trainning.spring.basic.impl;
import com.github.haojinlong.trainning.spring.basic.inter.Person;
public class JamesPerson implements Person {
	@Override
	public String getName() {
		return "James";
	}
	@Override
	public int getAge() {
		return 40;
	}
}
```

```
package com.github.haojinlong.trainning.spring.basic.impl;
import com.github.haojinlong.trainning.spring.basic.inter.Person;
public class HaoJinlongPerson implements Person {
	@Override
	public String getName() {
		return "Hao JInlong";
	}
	@Override
	public int getAge() {
		return 31;
	}
}
```

```
Package
	com.github.haojinlong.trainning.spring.basic.impl;
Import
	com.github.haojinlong.trainning.spring.basic.inter.Person;
Import
	com.github.haojinlong.trainning.spring.basic.inter.SayHello;

public class MySayHello implements SayHello {
	private Person person;

	public Person getPerson() {
		return person;
	}

	public void setPerson(Person person) {
		this.person = person;
	}

	@Override
	public String sayHelo() {
		return "Hello, " + person.getName() + "!";
	}
}
```

第一个 Spring 示例 (3/4)：设置配置文件

src/BasicApplicationContext.xml（文件名和地址可自己指定）：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-
       <bean id="haoJinlongPerson"
             class="com.github.haojinlong.trainning.spring.basic.impl.HaoJinlongPerson">
       </bean>

       <bean id="jamesPerson"
             class="com.github.haojinlong.trainning.spring.basic.impl.JamesPerson">
       </bean>

       <bean id="mySayHello"
             class="com.github.haojinlong.trainning.spring.basic.impl.MySayHello">
             <property name="person" ref="haoJinlongPerson" />
       </bean>
</beans>
```

Bean ID 对应与下面的 ref 值

指定 mySayHello 中的 person 使用 haojinlongPerson，如修改为 jamesPerson，则调用另外的 Bean

第一个 Spring 示例 (4/4)：使用 spring 容器进行调用

调用代码：

```
ApplicationContext applicationContext = new ClassPathXmlApplicationContext(
        "BasicApplicationContext.xml");
Person hPerson = applicationContext.getBean("haoJinlongPerson",
        Person.class);
logger.debug("hPerson name: {}", hPerson.getName());
Person jPerson = applicationContext.getBean("jamesPerson", Person.class);
logger.debug("jPerson name: {}", jPerson.getName());
        SayHello sayHello = applicationContext.getBean(SayHello.class);
logger.debug("sayHello: {}", sayHello.sayHelo());
```

配置文件名称

配置文件中的 bean id

配置文件中的 bean id

配置文件中只有一个 SayHello 的实现，故不需要指定 bean id

输出：

```
2014-07-19 10:37:24,094 DEBUG c.g.h.t.spring.basic.main.BasicMain 28 - hPerson name: Hao JInlong
2014-07-19 10:37:24,099 DEBUG c.g.h.t.spring.basic.main.BasicMain 31 - jPerson name: James
2014-07-19 10:37:24,099 DEBUG c.g.h.t.spring.basic.main.BasicMain 34 - sayHello: Hello, Hao JInlong!
```

Spring 中使用构造函数 (1/2)： construct-arg 方式

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd">
       <bean id="basicPerson"
             class="com.github.haojinlong.trainning.spring.basic.impl.BasicPerson">
             <constructor-arg index="0" type="java.lang.String"
                   value="HaoJinlong" />
             <constructor-arg index="1" type="int" value="32" />
       </bean>

       <bean id="basicSayHello"
             class="com.github.haojinlong.trainning.spring.basic.impl.BasicSayHello"
             c:person-ref="basicPerson">
             <constructor-arg name="person" ref="basicPerson" />
       </bean>

</beans>
```

可以通过 index 设置来指定先后顺序

也可以使用 name 参数来直接指定参数名称

Spring 中使用构造函数 (2/2)： c: 参数方式

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:c="http://www.springframework.org/schema/c"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd">
       <bean id="basicPerson"
             class="com.github.haojinlong.trainning.spring.basic.impl.BasicPerson"
             c:name="HaoJinlong" c:age="32" />

       <bean id="basicSayHello"
             class="com.github.haojinlong.trainning.spring.basic.impl.BasicSayHello"
             c:person-ref="basicPerson" />

</beans>
```

必须有这一行，否则会报错

直接执行值

引用另外一个 Bean

Spring 中通过 setter 方法设置参数

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd">
       <bean id="hPerson"
             class="com.github.haojinlong.trainning.spring.basic.impl.DefaultPerson">
             <property name="name" value="Haojinlong" />
       </bean>

       <bean id="jPerson"
             class="com.github.haojinlong.trainning.spring.basic.impl.DefaultPerson" p:name="James">
       </bean>

       <bean id="basicSayHello"
             class="com.github.haojinlong.trainning.spring.basic.impl.DefaultSayHello">
             <property name="person" ref="hPerson" />
       </bean>

</beans>
```

p: 参数方式需要这一行

property 方式

p: 参数方式

可使用 **p:person-ref="hPerson"** 替换

使用注解 (1/4)： @Component

```
/**
 * # HaoJinlongPerson.java -- (2014 年 7 月 19 日 )
 * 作者：郝金隆
 * 联系方式： haojinlong@189.cn
 */
package com.github.haojinlong.trainning.spring.anno.basic.impl;

import org.springframework.stereotype.Component;
import com.github.haojinlong.trainning.spring.anno.basic.inter.Person;


@Component("hPerson")  // 可以替换为 Service/Controller/Repository，效果一样
public class HaoJinlongPerson implements Person {

        @Override
        public String getName() {
                return "Hao Jinlong";
        }

        @Override
        public int getAge() {
                return 30;
        }
}
```

就相当于配置文件中的
<bean id="hPerson" ...../>
如果不指定名称，则默认为类名的首字母小写

使用 @Service、@Controller、@Repository 能够取得同样的效果，一般而言 @Repository 用于 DAO 层，@Service 用于服务层，@Controller 专用于 WEB 层

注意：使用注解需要引入 spring-aop 的 jar 包

使用注解 (2/4)： @Autowired

```
package com.github.haojinlong.trainning.spring.anno.basic.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.github.haojinlong.trainning.spring.anno.basic.inter.Person;
import com.github.haojinlong.trainning.spring.anno.basic.inter.SayHello;

@Service
public class MySayHello implements SayHello {
        static Logger logger = LoggerFactory.getLogger(MySayHello.class);

        @Autowired
        private Person person;

        @Override
        public String sayHello() {
                return "Hello, " + person.getName() + "!";
        }
}
```

与 @Component、@Controller、@Repository 等效

通过 @Autowired 实现 person 的自动注入，要求容器中的 Person 接口实现职能有一个，否则容器将不知道注入哪一个
如果有 settter 方法，在 setter 方法前标注效果相同

使用注解 (3/4)： 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context.xsd ">

       <context:component-scan
            base-package="com.github.haojinlong.trainning.spring.anno.basic" />

</beans>
```

表示徐增加这几行内容

容器将自动扫描这个配置文件下的注解

使用注解 (4/4)：系统调用

程序调用：

```
ApplicationContext applicationContext = new ClassPathXmlApplicationContext(
                "BasicApplicationContext.xml");
SayHello sayHello = applicationContext.getBean("mySayHello",
                SayHello.class);
logger.debug("say hello: {}", sayHello.sayHello());

Person person = applicationContext.getBean("hPerson", Person.class);
logger.debug("person name: {}", person.getName());
```

结果输出：

```
2014-07-19 13:11:39,752 DEBUG c.g.h.t.spring.anno.basic.TestMain 31 - say hello: Hello, Hao Jinlong!
2014-07-19 13:11:39,757 DEBUG c.g.h.t.spring.anno.basic.TestMain 34 - person name: Hao Jinlong
```

替换要扫描的注解类 (1/3)：创建新类

```
package com.github.haojinlong.trainning.spring.anno.custom.impl;

import org.springframework.stereotype.Component;
import com.github.haojinlong.trainning.spring.anno.custom.inter.Person;

@Component
public class JamesPerson implements Person {

	@Override
	public String getName() {
		return "James";
	}

	@Override
	public int getAge() {
		return 40;
	}
}
```

替换要扫描的注解类 (2/3)：修改配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context.xsd">
       <context:component-scan
            base-package="com.github.haojinlong.trainning.spring.anno.custom">
            <context:exclude-filter type="assignable"
                 expression="com.github.haojinlong.trainning.spring.anno.custom.impl.HaoJinlongPerson" />
       </context:component-scan>
</beans>
```

不自动加载
HaoJinlongPerson

注意：若在配置文件中再通过 <bean id='' /> 加载 HaoJinlongPerson， autowire 仍会出错

替换要扫描的注解类 (3/3)：系统调用

程序调用：

```
ApplicationContext applicationContext = new ClassPathXmlApplicationContext(
                "CustomApplicationContext.xml");
SayHello sayHello = applicationContext.getBean("mySayHello",
                SayHello.class);
logger.debug("say hello: {}", sayHello.sayHello());

Person person = applicationContext.getBean(Person.class);
logger.debug("person name: {}", person.getName());
```

说明 autowired 的是 JamesPerson

结果输出：

```
2014-07-19 13:47:07,742 DEBUG c.g.h.t.spring.anno.custom.TestMain 32 - say hello: Hello, James!
2014-07-19 13:47:07,747 DEBUG c.g.h.t.spring.anno.custom.TestMain 35 - person name: James
```

多个接口实现并存，避免 Autowire 出错

- 接口类：
  - com.github.haojinlong.trainning.spring.anno.complex.inter.Person
  - com.github.haojinlong.trainning.spring.anno.complex.inter.SayHello
- 实现类
  - Person 实现
    - com.github.haojinlong.trainning.spring.anno.complex.impl.HaoJinlongPerson
    - com.github.haojinlong.trainning.spring.anno.complex.impl.JamesPerson
  - SayHello 实现
    - com.github.haojinlong.trainning.spring.anno.complex.impl.ChSayHello
    - com.github.haojinlong.trainning.spring.anno.complex.impl.EnSayHello
- 实现方式
  - @Autowired 后面增加 @Qualifier(实例名) 来指定引用哪个 Person ，其中实例名默认为类名的首字母小写
  - 具体参见 2-spring-anno 中示例

目录

Spring 与 Hibernate 集成 (1/7)：准备工作

- Spring jar 包
  - spring 基础包，包括：Spring-core, spring-beans, spring context, spring-aop, spring-expressions
  - spring 数据存取包，包括：spring-jdbc，spring-tx，spring-orm
  - 下载地址：http://repo.spring.io/release/org/springframework/spring/
  - 其他：aopalliance.jar（spring+hibernate 事务依赖，http://sourceforge.net/projects/aopalliance/files/）
- Hibernate jar 包
  - 包括：antlr, dom4j, hibernate-commons-annotations, hiberante-core, hibernate-jpa, javassist, jboss-logging, jboss-transaction-api
  - 下载地址：http://sourceforge.net/projects/hibernate/files/hibernate4/
- 数据库访问基础包：
  - mysql 数据库驱动：mysql-connector-java（mysql 官方下载）
  - c3p0 连接池：c3p0、mchange-commons-java （http://sourceforge.net/projects/c3p0/）
- Java 日志包（具体用法详见 《Java 日志培训》）
  - 接口：slf4j-api, jcl-over-slf4j
  - 实现：logback-core, logback-classic

Spring 与 Hibernate 集成 (2/7)：创建库表和测试数据

```
-- 创建库表
create table if not exists users(
        id integer primary key auto_increment comment '唯一标识',
        name varchar(20) comment '姓名',
        age int comment '年龄'
) comment '用户表';

-- 插入测试数据
insert into  users(name, age) values('haojinlong', 32);
insert into  users(name, age) values('james', 40);
```

Spring 与 Hibernate 集成 (3/7)：创建实体类

```
package com.github.haojinlong.trainning.spring.hibernate.entity;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import org.apache.commons.lang3.builder.ReflectionToStringBuilder;

@Entity
@Table(name = "users")
public class User implements Serializable {
        private static final long serialVersionUID = -7416211273151385627L;
        static Logger logger = LoggerFactory.getLogger(User.class);
        @Id
        @GeneratedValue
        @Column(name="id")
        private Integer id;
        @Column
        private String name;
        @Column
        private Integer age;

        // getters, setters and overided toString method
}
```

表示是实体映射类

指定表名

表示为主键，必须要有

自动生成字段

可指定列名，默认与属性名一致

Spring 与 Hibernate 集成 (4/7)：创建 DAO 接口

```
package com.github.haojinlong.trainning.spring.hibernate.dao;

import java.util.List;
import com.github.haojinlong.trainning.spring.hibernate.entity.User;

/**
 * @author  郝金隆
 *
 */
public interface UserDao {

	public List<User> listAll();

	public User get(int id);

	public void saveOrUpdate(User user);

	public void delete(int id);

	public void delete(User user);
}
```

Spring 与 Hibernate 集成 (5/7)：创建 DAO 实现

```
package com.github.haojinlong.trainning.spring.hibernate.daoimpl;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import com.github.haojinlong.trainning.spring.hibernate.dao.UserDao;
import com.github.haojinlong.trainning.spring.hibernate.entity.User;

@Repository
public class UserDaoImpl implements UserDao {

        @Autowired
        private SessionFactory sessionFactory;

        @Override
        @Transactional
        public List<User> listAll() {
                @SuppressWarnings("unchecked")
                List<User> userList = (List<User>) this.sessionFactory
                                .getCurrentSession().createQuery("from User").list();
                return userList;
        }

        // 其他接口实现
}
```

用于 spring 自动查找

自动注入 sessionFactory

无论是读操作还是写操作都需要 @Transactional 注解，否则会报错

Spring 与 Hibernate 集成 (6/7)：配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:tx="http://www.springframework.org/schema/tx"
        xmlns:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
                http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context-4.0.xsd
                http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

        <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
                p:driverClass="com.mysql.jdbc.Driver" p:jdbcUrl="jdbc:mysql://localhost:3306/spring_test"
                p:user="root" p:password="root" />

        <bean id="sessionFactory"
                class="org.springframework.orm.hibernate4.LocalSessionFactoryBean"
                p:dataSource-ref="dataSource"
                p:packagesToScan="com.github.haojinlong.trainning.spring.hibernate.entity" />

        <context:component-scan base-package="com.github.haojinlong.trainning.spring" />

        <tx:annotation-driven />
        <bean id="transactionManager"
                class="org.springframework.orm.hibernate4.HibernateTransactionManager"
                p:sessionFactory-ref="sessionFactory" />
</beans>
```

设置数据源

设置 sessionFactory

设置自动扫描加载的映射实体类所在的包

自动扫描加载 DAO 实现类

启动事务，并确定事务管理类（必须，否则报错）

注意：因版面关系，上述配置修改为 p: 参数方式，实际可读性不如传统方式，具体用法参见第二节

Spring 与 Hibernate 集成 (7/7)：请求调用

程序调用：

```
ApplicationContext applicationContext = new ClassPathXmlApplicationContext(
        "ApplicationContext.xml");
UserDao userDao = applicationContext.getBean(UserDao.class);
logger.debug("user list: {}", userDao.listAll());
logger.debug("user: {}", userDao.get(1));
```

结果输出：

```
2014-07-19 21:55:08,680 DEBUG c.g.h.t.spring.hibernate.TestMain 30 - user list:
[com.github.haojinlong.trainning.spring.hibernate.entity.User@302b3e2e[id=1,name=haojinlong,age=32],
com.github.haojinlong.trainning.spring.hibernate.entity.User@34beef49[id=2,name=james,age=40]]
2014-07-19 21:55:08,704 DEBUG c.g.h.t.spring.hibernate.TestMain 31 - user:
com.github.haojinlong.trainning.spring.hibernate.entity.User@7be07643[id=1,name=haojinlong,age=32]
```

更多 Hibernate 使用技巧请参见《 Hibernate 培训材料》

Spring 与 JPA 集成 (1/5)：前期准备

- Spring jar 包
  - spring 基础包，包括：Spring-core, spring-beans, spring context, spring-aop, spring-expressions
  - spring 数据存取包，包括：spring-jdbc，spring-tx，spring-orm
  - 下载地址：http://repo.spring.io/release/org/springframework/spring/
  - 其他：aopalliance.jar（spring+hibernate 事务依赖，http://sourceforge.net/projects/aopalliance/files/）
- Hibernate 的 jpa 实现包
  - 包括：antlr, dom4j, hibernate-commons-annotations, hiberante-core, hibernate-jpa, javassist, jboss-logging, jboss-transaction-api,hibernate-entitymanager（相对于 Hibernate 需增加）
  - 下载地址：http://sourceforge.net/projects/hibernate/files/hibernate4/
- 数据库访问基础包：
  - mysql 数据库驱动：mysql-connector-java（mysql 官方下载）
  - c3p0 连接池：c3p0、mchange-commons-java（http://sourceforge.net/projects/c3p0/）
- Java 日志包（具体用法详见《Java 日志培训》）
  - 接口：slf4j-api, jcl-over-slf4j
  - 实现：logback-core, logback-classic

Spring 与 JPA 集成 (2/5)：创建实体类与 DAO 接口

- 创建库表和测试数据
  - 同上例
- 创建实体类
  - 同上例
- 创建接口类
  - 同上例
- 说明
  - Hibernate 4.0 开始，实体类的注解已经全面采用 JPA 的注解，原有 Hibernate 自己的注解已经全部不再提供，故此实体类与 JPA 完全相同

Spring 与 JPA 集成 (3/5)：创建 DAO 实现类

```
package com.github.haojinlong.trainning.spring.jpa.daoimpl;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.PersistenceContext;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import com.github.haojinlong.trainning.spring.jpa.dao.UserDao;
import com.github.haojinlong.trainning.spring.jpa.entity.User;

@Repository
public class UserDaoImpl implements UserDao {
        static Logger logger = LoggerFactory.getLogger(UserDaoImpl.class);

        @PersistenceContext
        private EntityManager em;

        @Override
        public List<User> listAll() {
                return em.createQuery("from User")
                                .getResultList();
        }

        // 其他方法的实现
}
```

使用 PersistenceContext 可以保证 entityManager 在线程内共享，从而确保事务完整性，且可以自主注入到 DaoImpl 中

读操作无需 @Transactional 声明

Spring 与 JPA 集成 (4/5)：配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"   xmlns:tx="http://www.springframework.org/schema/tx"
        xmlns:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
                http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context-4.0.xsd
                http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

        <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
                p:driverClass="com.mysql.jdbc.Driver" p:jdbcUrl="jdbc:mysql://localhost:3306/spring_test"
                p:user="root" p:password="root" />

        <bean id="myProvider" class="org.hibernate.ejb.HibernatePersistence" />

        <bean id="myEmf"
                class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
                p:packagesToScan="com.github.haojinlong.trainning.spring.jpa.entity"
                p:persistenceProvider-ref="myProvider" p:dataSource-ref="dataSource" />

        <context:component-scan base-package="com.github.haojinlong.trainning.spring" />

        <tx:annotation-driven />
        <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager"
                p:entityManagerFactory-ref="myEmf" />
</beans>
```

创建持久化的 Provider

创建 EntityManagerFactoryBean，用于 @PersistConext EntityManager 的注入

声明启动事务

注意：因版面关系，上述配置修改为 p: 参数方式，实际可读性不如传统方式，具体用法参见第二节

Spring 与 JPA 集成 (5/5)：请求调用

程序调用：

```
ApplicationContext applicationContext = new ClassPathXmlApplicationContext(
        "ApplicationContext.xml");
UserDao userDao = applicationContext.getBean(UserDao.class);
logger.debug("user list: {}", userDao.listAll());
logger.debug("user: {}", userDao.get(1));
```

结果输出：

```
2014-07-19 21:55:08,680 DEBUG c.g.h.t.spring.hibernate.TestMain 30 - user list:
[com.github.haojinlong.trainning.spring.hibernate.entity.User@302b3e2e[id=1,name=haojinlong,age=32],
com.github.haojinlong.trainning.spring.hibernate.entity.User@34beef49[id=2,name=james,age=40]]
2014-07-19 21:55:08,704 DEBUG c.g.h.t.spring.hibernate.TestMain 31 - user:
com.github.haojinlong.trainning.spring.hibernate.entity.User@7be07643[id=1,name=haojinlong,age=32]
```

更多 jpa 使用技巧请参见《 jpa 培训材料》

Spring 与 Mybatis 集成 (1/6)：前期准备

- Spring jar 包
  - spring 基础包，包括： Spring-core, spring-beans, spring context, spring-aop
  - spring 数据存取包，包括： spring-jdbc ， spring-tx（无需 spring-orm)
  - 下载地址： http://repo.spring.io/release/org/springframework/spring/
  - 其他： aopalliance.jar（ spring 事务依赖， http://sourceforge.net/projects/aopalliance/files/ ）
- mybatis jar 包
  - 包括： mybatis, mybatis-spring（作用与 spring-orm 类似）
  - 下载地址： http://blog.mybatis.org/ http://mybatis.github.io/
- 数据库访问基础包：
  - mysql 数据库驱动： mysql-connector-java（ mysql 官方下载）
  - c3p0 连接池： c3p0、 mchange-commons-java （ http://sourceforge.net/projects/c3p0/ ）
- Java 日志包（具体用法详见 《Java 日志培训》 ）
  - 接口： slf4j-api, jcl-over-slf4j
  - 实现： logback-core, logback-classic

Spring 与 Mybatis 集成 (2/6)：初始化数据库及实体类

- 创建库表
  - 同上例
- 创建实体类（见右面示例）
  - 映射类的属性和方法与 Hibernate、Jpa 相同，但不需要使用 @Entity、@Id、@Column 等注解，所有的配置均通过 Mapper 配置文件完成

```
package com.github.haojinlong.trainning.spring.mybatis.entity;

import java.io.Serializable;
import org.apache.commons.lang3.builder.ReflectionToStringBuilder;

/**
 * @author 郝金隆
 *
 */
public class User implements Serializable {
	private static final long serialVersionUID = 406473712340282168 4L;

	private Integer id;
	private String name;
	private Integer age;

	// getter and setters
	// override toString method
}
```

Spring 与 Mybatis 集成 (3/6)：创建 Mapper 接口

```
package com.github.haojinlong.trainning.spring.mybatis.mapper;

import java.util.List;
import com.github.haojinlong.trainning.spring.mybatis.entity.User;

/**
 * @author  郝金隆
 */
public interface UserMapper {

        public User get(int id);

        public List<User> listAll();

        public int insert(User user);

        public int update(User user);

        public int delete(User user);
}
```

Spring 与 Mybatis 集成 (4/6)：创建 Mapper 配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper  namespace="com.github.haojinlong.trainning.spring.mybatis.mapper.UserMapper">
        <resultMap id="userMap"
                type="com.github.haojinlong.trainning.spring.mybatis.entity.User">
                <id property="id" column="id" />
                <result property="name" column="name" />
                <result property="age" column="age" />
        </resultMap>

        <select id="get" parameterType="int" resultMap="userMap">
                select * from
                users where id=#{id}
        </select>

        <select id="listAll" resultMap="userMap">
                select * from users
        </select>

        <!--  其他的映射配置略  -->

</mapper>
```

namespace 与 Mapper 接口保持一致

id 与 Mapper 接口方法保持一致

注： Mapper 配置文件需要与 Mapper 接口文件位于同一包下，且名称相同

Spring 与 Mybatis 集成 (5/6)： Spring 中的配置

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tx="http://www.springframework.org/schema/tx"
        xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="
      http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd
      http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
      http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

        <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
              p:driverClass="com.mysql.jdbc.Driver" p:jdbcUrl="jdbc:mysql://localhost:3306/spring_test"
              p:user="root" p:password="root" />

        <tx:annotation-driven />
        <bean id="transactionManager"
              class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
              p:dataSource-ref="dataSource" />

        <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"
              p:dataSource-ref="dataSource" />

        <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"
              p:basePackage="com.github.haojinlong" />
</beans>
```

引入 Mybatis 的 SqlSessionFactoryBean

确定自动扫描 Mapper 配置文件路径

注意：因版面关系，上述配置修改为 p: 参数方式，实际可读性不如传统方式，具体用法参见第二节

Spring 与 Mybatis 集成 (6/6)：接口调用

程序调用：

```
ApplicationContext applicationContext = new ClassPathXmlApplicationContext(
        "ApplicationContext.xml");
UserMapper userMapper = applicationContext.getBean(UserMapper.class);
logger.debug("user list: {}", userMapper.listAll());
```

结果输出：

```
2014-07-20 14:44:02,389 DEBUG c.g.h.t.s.m.m.UserMapper.listAll 132 - ooo Using Connection
[com.mchange.v2.c3p0.impl.NewProxyConnection@7c4e758a]
2014-07-20 14:44:02,405 DEBUG c.g.h.t.s.m.m.UserMapper.listAll 132 - ==>  Preparing: select * from users
2014-07-20 14:44:02,486 DEBUG c.g.h.t.s.m.m.UserMapper.listAll 132 - ==> Parameters:
2014-07-20 14:44:02,514 DEBUG c.g.h.t.spring.mybatis.TestMain 30 - user list:
[com.github.haojinlong.trainning.spring.mybatis.entity.User@178fd0ca[id=1,name=haojinlong,age=32],
com.github.haojinlong.trainning.spring.mybatis.entity.User@62d489db[id=2,name=james,age=40]]
```

更多 Mybatis 使用技巧请参见《 Mybatis 培训》

注：当 Mapper 接口的日志级别设置卫 DEBUG 时， mybatis 将自动输出响应的 SQL 语句，供调试使用

Spring 事务管理 (1/3)： Spring 配置

Spring 中 Hibernate 事务设置：

启动事务注解

设定事务管理器

```
<tx:annotation-driven />
<bean id="transactionManager"
      class="org.springframework.orm.hibernate4.HibernateTransactionManager"
      p:sessionFactory-ref="sessionFactory" />
```

Spring 中 Jpa 事务设置：

若事务管理器的 Bean ID 不是 transactionManager 时，需要手工指定事务管理器的 Bean ID

```
<tx:annotation-driven transaction-manager="transactionManager" />
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager"
      p:entityManagerFactory-ref="entityManagerFactory" />
```

Spring 中 Mybatis 事务设置：

```
<tx:annotation-driven />
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
      p:dataSource-ref="dataSource" />
```

Spring 事务管理 (2/3)： 使用 @Trasactional 注解

```
package com.github.haojinlong.trainning.spring.jpa.impl;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.github.haojinlong.trainning.spring.jpa.dao.UserDao;
import com.github.haojinlong.trainning.spring.jpa.entity.User;
import com.github.haojinlong.trainning.spring.jpa.inter.UserService;

@Service
public class UserServiceImpl implements UserService {
        static Logger logger = LoggerFactory.getLogger(UserServiceImpl.class);

        @Autowired
        private UserDao userDao;

        @Override
        @Transactional
        public void insert(List<User> userList) {
                if (userList != null){
                        for (User user: userList){
                                userDao.saveOrUpdate(user);
                        }
                }
        }
}
```

表示此方法中多次调用的数据存储操作为同一个事务

Spring 事务管理 (3/3)： 测试程序

```
ApplicationContext applicationContext = new ClassPathXmlApplicationContext(
        "MybatisApplicationContext.xml");
UserService userService = applicationContext.getBean(UserService.class);
User user = new User();
user.setName("haoj");
user.setAge(0);
User user2 = new User();
user2.setName("aslfjlasfjdlifeljafijalejfliajalefjlajefliajefljaelfjalefjalejflajelfjalejfliajfliajefljef");
user2.setAge(1);
List<User> userList = new ArrayList<>();
userList.add(user);
userList.add(user2);
userService.insert(userList);
```

字符串长度超过列长度限制导致保存失败；
因为在同一个事务中，故此 user 同样无法插入到数据库中

Tips：

- mysql 数据库 5.5 以前默认的存储引擎为 MyISAM ，不支持事务处理，需手工设置卫 InnoDB 才可支持事务
- 对于 Jpa 和 Hibernate ， DAO 中也需要使用 @Transactional 注解

目录

RMI 服务注册与调用：创建服务接口

接口：

```
package com.github.haojinlong.trainning.spring.remote.rmi.inter;
import com.github.haojinlong.trainning.spring.remote.rmi.entity.Person;

public interface SayHelloService {
        public String sayHello(Person person);
}
```

注意：接口中的参数和返回值均需要实现 Serializable 接口，以确保可序列化

```
package com.github.haojinlong.trainning.spring.remote.rmi.entity;
import java.io.Serializable;
public class Person implements Serializable {
        private static final long serialVersionUID = -2627303070179969437L;
        private String name;
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
}
```

RMI 服务注册与调用：创建接口实现类

```
package com.github.haojinlong.trainning.spring.remote.rmi.impl;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;
import com.github.haojinlong.trainning.spring.remote.rmi.entity.Person;
import com.github.haojinlong.trainning.spring.remote.rmi.inter.SayHelloService;

@Service
public class MySayHelloService implements SayHelloService {
	static Logger logger = LoggerFactory.getLogger(MySayHelloService.class);

	@Override
	public String sayHello(Person person) {
		if (person != null && person.getName() != null) {
			return "Hello, " + person.getName() + "!";
		}
		return "Hello!";
	}
}
```

RMI 服务注册与调用：创建接口发布配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-4.0.xsd">

       <context:component-scan
            base-package="com.github.haojinlong.trainning.spring.remote.rmi.impl" />

       <bean class="org.springframework.remoting.rmi.RmiServiceExporter">
            <property name="registryPort" value="1099" />
            <property name="servicePort" value="1199" />
            <property name="serviceName" value="mySayHelloService" />
            <property name="service" ref="mySayHelloService" />
            <property name="interface"
                 value="com.github.haojinlong.trainning.spring.remote.rmi.inter.SayHelloService" />
       </bean>

</beans>
```

服务注册端口默认 1099

服务端口默认随机生成

RMI 服务注册与调用：启动 RMI 服务

```
public class ExploreMain {
        static Logger logger = LoggerFactory.getLogger(ExploreMain.class);

        /**
         * @param args
         */
        public static void main(String[] args) {
                ApplicationContext applicationContext = new ClassPathXmlApplicationContext(
                                "RmiExploreApplicationContext.xml");
        }
}
```

RMI 服务注册与调用：通过配置引用 RMI 服务

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-4.0.xsd">


       <bean id="mySayHelloService" class="org.springframework.remoting.rmi.RmiProxyFactoryBean">
            <property name="serviceUrl" value="rmi://localhost:1099/mySayHelloService" />
            <property name="serviceInterface"
                 value="com.github.haojinlong.trainning.spring.remote.rmi.inter.SayHelloService" />
       </bean>

</beans>
```

注：serviceUrl 格式为： rmi://[ip/hostname]:[registryPort]/[serviceName]
其中 registryPort 和 serviceName 均由 RMI 服务发布的配置文件中定义

RMI 服务注册与调用：具体调用

调用程序：

```
Person person = new Person();
person.setName("Haojinlong");
ApplicationContext applicationContext = new ClassPathXmlApplicationContext(
            "RmiImportApplicationContext.xml");
SayHelloService sayHelloService = applicationContext
            .getBean(SayHelloService.class);
logger.debug("say hello: {}", sayHelloService.sayHello(person));
```

输出结果：

```
2014-07-20 22:51:39,334 DEBUG c.g.h.t.s.remote.rmi.ImportMain 33 - say hello: Hello, Haojinlong!
```

注：启动服务调用前，需要首先启动服务发布程序，以完成服务发布

示例代码下载地址

- https://github.com/haojinlong/trainning/tree/master/src/3
  - 1-spring-basic： Spring 基于配置文件的基础使用
  - 2-spring-anno： spring 基于注解的使用
  - 3-spring-hibernate： spring 与 hibernate 的集成示例
  - 4-spring-jpa： spring 与 jpa 的集成示例
  - 5-spring-mybatis： spring 与 mybatis 的集成示例
  - 6-spring-tx：使用 pring 进行事务管理
  - 7-spring-remote：使用 spring 进行远程服务注册与调用
-