高级补充

Spring 注解

初始化

@Configuration
注解类：id 是方法名，方法上@Bean

使用：

```
new AnnotationConfigApplicationContext(class)
```

包扫描

```
@ComponentScan(value = "annotest",excludeFilters = {
        @ComponentScan.Filter(type = FilterType.ANNOTATION,classes
={controller.class})}
        /includeFilters = {@ComponentScan.Filter(type =
FilterType.ANNOTATION,classes = controller.class)}
        ,useDefaultFilters = false)
```

跳过扫描有好几种，此处为 class 类型。

类型过滤

```
@ComponentScan.Filter(type = FilterType.CUSTOM,classes = MyTypeFilter.class)
```

```
public class MyTypeFilter implements TypeFilter {
    @Override
    public boolean match(MetadataReader metadataReader, MetadataReaderFactory
```

```
metadataReaderFactory) throws IOException {
        AnnotationMetadata annotationMetadata =
metadataReader.getAnnotationMetadata();
        ClassMetadata classMetadata = metadataReader.getClassMetadata();
        Resource resource = metadataReader.getResource();
        return false;
    }
}
```

单多实例

```
@see ConfigurableBeanFactory#SCOPE_PROTOTYPE
* @see ConfigurableBeanFactory#SCOPE_SINGLETON
* @see org.springframework.web.context.WebApplicationContext#SCOPE_REQUEST
* @see org.springframework.web.context.WebApplicationContext#SCOPE_SESSION


@Scope(value = "SCOPE_PROTOTYPE")
@Bean
```

懒加载

```
@Lazy
@Bean
```

条件

```
@Configuration
@Conditional(WinCondition.class)
public class PersonConfiguration {

    @Bean
    //@Conditional(WinCondition.class)
    public Person person(){
        return new Person("刘彪",23);
    }
}
```

```
public class WinCondition implements Condition {
    @Override
    public boolean matches(ConditionContext context, AnnotatedTypeMetadata
metadata) {
        Environment environment = context.getEnvironment();
        String os_name=environment.getProperty("os.name");
        if(os_name.contains("Win")){
            return true;
        }else {
            return false;
        }
    }
}
```

导入组件

```
@Configuration
@ComponentScan(value = "annotest")
@Import({Person.class})
public class PersonConfiguration {
```

选择导入

```
public class MySelectImport implements ImportSelector {
    @Override
    public String[] selectImports(AnnotationMetadata importingClassMetadata) {
        MultiValueMap<String, Object> controller =
                importingClassMetadata.getAllAnnotationAttributes("controller");
        return new String[]{};
    }
}


@Import({Person.class,----MySelectImport.class---})
```

注册导入

```
public class MyRes implements ImportBeanDefinitionRegistrar {
    @Override
    public void registerBeanDefinitions(AnnotationMetadata
importingClassMetadata, BeanDefinitionRegistry registry) {
        boolean HASDOG = registry.containsBeanDefinition("dog");
        boolean HASCAT = registry.containsBeanDefinition("cat");
        if(HASCAT&&HASDOG){
            RootBeanDefinition definition=new RootBeanDefinition(Person.class);
            registry.registerBeanDefinition("person",definition);
        }
    }
}
```

```
@Configuration
@ComponentScan(value = "annotest")
@Import({Person.class,======MyRes.class=====})
public class PersonConfiguration {

    @Bean
    @Conditional(WinCondition.class)
    public Person person(){
        return new Person("刘彪",23);
    }
}
```

生命周期

```
@Bean(value = "mybean",initMethod = "myInit",destroyMethod = "myDes")

@PostConstruct
public void myInit(){}

@PreDestroy
public void myDes(){}
```

属性赋值

```
@Value("张三")
//@Value("${name}")
```

首选装配

```
@Primary
```

Java 规范装配

```
@Resource
```

```
@Resource(name = @Resource("bookdao2"))
```

Springboot

Yml_list

```
pets:
  - cat
  - dog
  - pig
```

```
pets: [cat,dog,pig]
```

配置文件值注入

```
person:
    lastName: hello
    age: 18
    boss: false
    birth: 2017/12/12
    maps: {k1: v1,k2: 12}
    lists:
      - lisi
```

```
    - zhaoliu
  dog:
    name: 小狗
    age: 12
```

```
@Component
@ConfigurationProperties(prefix = "person")
public class Person {

    private String lastName;
    private Integer age;
    private Boolean boss;
    private Date birth;

    private Map<String,Object> maps;
    private List<Object> lists;
    private Dog dog;
```

数值配置校验

```
@Component
@ConfigurationProperties(prefix = "person")
@Validated
public class Person {
```

PropertySource

```
/**
 * 将配置文件中配置的每一个属性的值，映射到这个组件中
 *@ConfigurationProperties：告诉 SpringBoot 将本类中的所有属性和配置文件中相关的配置进行绑定；
 *          prefix = "person"：配置文件中哪个下面的所有属性进行一一映射
 *
 * 只有这个组件是容器中的组件，才能容器提供的@ConfigurationProperties 功能；
 *  @ConfigurationProperties(prefix = "person")默认从全局配置文件中获取值；
 *
 */
@PropertySource(value = {"classpath:person.properties"})
```

```
@Component
@ConfigurationProperties(prefix = "person")
//@Validated
public class Person {

    /**
     * <bean class="Person">
     *         <property name="lastName" value="字面量/${key}从环境变量、配置文件中获取值/#{SpEL}"></property>
     * <bean/>
     */

    //lastName 必须是邮箱格式
    // @Email
     //@Value("${person.last-name}")
     private String lastName;
     //@Value("#{11*2}")
     private Integer age;
     //@Value("true")
     private Boolean boss;
```

ConfigurationProperties

```
/**
 * 将配置文件中配置的每一个属性的值，映射到这个组件中
 *@ConfigurationProperties：告诉 SpringBoot 将本类中的所有属性和配置文件中相关的配置进行绑定；
 *        prefix = "person"：配置文件中哪个下面的所有属性进行一一映射
 *
 * 只有这个组件是容器中的组件，才能容器提供的@ConfigurationProperties 功能；
 *  @ConfigurationProperties(prefix = "person")默认从全局配置文件中获取值；
 *
 */
@PropertySource(value = {"classpath:person.properties"})
@Component
@ConfigurationProperties(prefix = "person")
//@Validated
public class Person {

    /**
     * <bean class="Person">
     *         <property name="lastName" value="字面量/${key}从环境变量、配置文件中获
```

```
取值/#{SpEL}"></property>
        * <bean/>
        */

    //lastName 必须是邮箱格式
    // @Email
      //@Value("${person.last-name}")
      private String lastName;
      //@Value("#{11*2}")
      private Integer age;
      //@Value("true")
      private Boolean boss;
```

ImportResource

```
@ImportResource(locations = {"classpath:beans.xml"})
导入 Spring 的配置文件让其生效
```

Configuration

```
@Configuration
public class MyAppConfig {

    //将方法的返回值添加到容器中；容器中这个组件默认的 id 就是方法名
    @Bean
    public HelloService helloService02(){
        System.out.println("配置类@Bean 给容器中添加组件了...");
        return new HelloService();
    }
}
```

多 **Profile** 文件

```
server:
  port: 8081
spring:
  profiles:
    active: prod
```

```
---
server:
  port: 8083
spring:
  profiles: dev


---

server:
  port: 8084
spring:
  profiles: prod    #指定属于哪个环境
```

指定激活那种 profile

spring.profiles.active=dev

thymeleaf

Message Expressions: #{...}：获取国际化内容
Link URL Expressions: @{...}：定义 URL；
@{/order/process(execId=${execId},execType='FAST')}
Fragment Expressions: ~{...}：片段引用表达式
...

Thymeleaf 内置类型

1)、获取对象的属性、调用方法

2)、使用内置的基本对象：

#ctx : the context object.
#vars: the context variables.
#locale : the context locale.
#request : (only in Web Contexts) the HttpServletRequest object.
#response : (only in Web Contexts) the HttpServletResponse object.
#session : (only in Web Contexts) the HttpSession object.
#servletContext : (only in Web Contexts) the ServletContext object.

Thymeleaf 工具对象

#execInfo : information about the template being processed.
#messages : methods for obtaining externalized messages inside variables expressions, in the same way as they would be obtained using #{…} syntax.
#uris : methods for escaping parts of URLs/URIs
#conversions : methods for executing the configured conversion service (if any).
#dates : methods for java.util.Date objects: formatting, component extraction, etc.
#calendars : analogous to #dates , but for java.util.Calendar objects.
#numbers : methods for formatting numeric objects.
#strings : methods for String objects: contains, startsWith, prepending/appending, etc.
#objects : methods for objects in general.
#bools : methods for boolean evaluation.
#arrays : methods for arrays.
#lists : methods for lists.
#sets : methods for sets.
#maps : methods for maps.
#aggregates : methods for creating aggregates on arrays or collections.
#ids : methods for dealing with id attributes that might be repeated (for example, as a result of an iteration).

Thymeleaf 公共元素

```
<footer th:fragment="copy">
&copy; 2011 The Good Thymes Virtual Grocery
</footer>
```

引入方式

```
<div th:insert="footer :: copy"></div>
<div th:replace="footer :: copy"></div>
<div th:include="footer :: copy"></div>
```

引入片段的时候传入参数

```
<nav class="col-md-2 d-none d-md-block bg-light sidebar" id="sidebar">
    <div class="sidebar-sticky">
        <ul class="nav flex-column">
            <li class="nav-item">
                <a class="nav-link active"
                    th:class="${activeUri=='main.html'?'nav-link active':'nav-link'}"
                    href="#" th:href="@{/main.html}">
                    <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24"
viewBox="0 0 24 24" fill="none" stroke="currentColor" stroke-width="2" stroke-
linecap="round" stroke-linejoin="round" class="feather feather-home">
                        <path d="M3 9l9-7 9 7v11a2 2 0 0 1-2 2H5a2 2 0 0 1-2-
2z"></path>
                        <polyline points="9 22 9 12 15 12 15 22"></polyline>
                    </svg>
                    Dashboard <span class="sr-only">(current)</span>
                </a>
            </li>

<!--引入侧边栏;传入参数-->
<div th:replace="commons/bar::#sidebar(activeUri='emps')"></div>
```

错误页面

有模板引擎的情况下；error/状态码；【将错误页面命名为 错误状态码.html 放在模板引擎文件夹里面的 error 文件夹下】，发生此状态码的错误就会来到对应的页面；

我们可以使用 4xx 和 5xx 作为错误页面的文件名来匹配这种类型的所有错误，精确优先（优先寻找精确的状态码.html）；

页面能获取的信息；

timestamp：时间戳

status：状态码

error：错误提示

exception：异常对象

message：异常消息

errors：JSR303 数据校验的错误都在这里

2）、没有模板引擎（模板引擎找不到这个错误页面），静态资源文件夹下找；

3）、以上都没有错误页面，就是默认来到 SpringBoot 默认的错误提示页面；

注册 servlet

```
//注册三大组件
@Bean
public ServletRegistrationBean myServlet(){
    ServletRegistrationBean registrationBean = new ServletRegistrationBean(new MyServlet(),"/myServlet");
    return registrationBean;
}
```

注册 Filter

```
@Bean
public FilterRegistrationBean myFilter(){
    FilterRegistrationBean registrationBean = new FilterRegistrationBean();
    registrationBean.setFilter(new MyFilter());
    registrationBean.setUrlPatterns(Arrays.asList("/hello","/myServlet"));
    return registrationBean;
}
```

注册 ServletListenerRegistrationBean

```
@Bean
public ServletListenerRegistrationBean myListener(){
    ServletListenerRegistrationBean<MyListener> registrationBean = new ServletListenerRegistrationBean<>(new MyListener());
    return registrationBean;
}
```

Docker

1、检查内核版本，必须是 3.10 及以上
uname -r
2、安装 docker
yum install docker
3、输入 y 确认安装
4、启动 docker
[root@localhost ~]# systemctl start docker
[root@localhost ~]# docker -v
Docker version 1.12.6, build 3e8e77d/1.12.6
5、开机启动 docker
[root@localhost ~]# systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.
6、停止 docker
systemctl stop docker
1、搜索镜像
[root@localhost ~]# docker search tomcat
2、拉取镜像
[root@localhost ~]# docker pull tomcat
3、根据镜像启动容器
docker run --name mytomcat -d tomcat:latest
4、docker ps
查看运行中的容器
5、 停止运行中的容器
docker stop 容器的 id
6、查看所有的容器
docker ps -a
7、启动容器
docker start 容器 id
8、删除一个容器
 docker rm 容器 id

9、启动一个做了端口映射的 tomcat
[root@localhost ~]# docker run -d -p 8888:8080 tomcat
-d：后台运行
-p: 将主机的端口映射到容器的一个端口　　主机端口:容器内部的端口

整合德鲁伊

initialSize: 5
minIdle: 5
maxActive: 20
maxWait: 60000
timeBetweenEvictionRunsMillis: 60000
minEvictableIdleTimeMillis: 300000
validationQuery: SELECT 1 FROM DUAL
testWhileIdle: true
testOnBorrow: false
testOnReturn: false
poolPreparedStatements: true

```
@Configuration
public class DruidConfig {

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

    //配置 Druid 的监控
    //1、配置一个管理后台的 Servlet
    @Bean
    public ServletRegistrationBean statViewServlet(){
        ServletRegistrationBean bean = new ServletRegistrationBean(new StatViewServlet(),
"/druid/*");
        Map<String,String> initParams = new HashMap<>();

        initParams.put("loginUsername","admin");
        initParams.put("loginPassword","123456");
        initParams.put("allow","");//默认就是允许所有访问
        initParams.put("deny","192.168.15.21");
```

```
        bean.setInitParameters(initParams);
        return bean;
    }


    //2、配置一个 web 监控的 filter
    @Bean
    public FilterRegistrationBean webStatFilter(){
        FilterRegistrationBean bean = new FilterRegistrationBean();
        bean.setFilter(new WebStatFilter());

        Map<String,String> initParams = new HashMap<>();
        initParams.put("exclusions","*.js,*.css,/druid/*");

        bean.setInitParameters(initParams);

        bean.setUrlPatterns(Arrays.asList("/*"));

        return  bean;
    }
}
```

整合 mybatis

注解版
使用 MapperScan 批量扫描所有的 Mapper 接口；

```
@MapperScan(value = "com.atguigu.springboot.mapper")
@SpringBootApplication
public class SpringBoot06DataMybatisApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBoot06DataMybatisApplication.class, args);
    }
}
```

Xml 版

```
mybatis:
  config-location: classpath:mybatis/mybatis-config.xml  指定全局配置文件的位置
  mapper-locations: classpath:mybatis/mapper/*.xml    指定 sql 映射文件的位置
```

整合 jpa

```
//继承 JpaRepository 来完成对数据库的操作
public interface UserRepository extends JpaRepository<User,Integer> {
}
```

WebMvcConfigurerAdapter

```
@Bean //将组件注册在容器
    public WebMvcConfigurerAdapter webMvcConfigurerAdapter(){
        WebMvcConfigurerAdapter adapter = new WebMvcConfigurerAdapter() {
            @Override
            public void addViewControllers(ViewControllerRegistry registry) {
                registry.addViewController("/").setViewName("login");
                registry.addViewController("/index.html").setViewName("login");
                registry.addViewController("/main.html").setViewName("dashboard");
            }

            //注册拦截器
            @Override
            public void addInterceptors(InterceptorRegistry registry) {
                //super.addInterceptors(registry);
                //静态资源；  *.css , *.js
                //SpringBoot 已经做好了静态资源映射
                registry.addInterceptor(new
LoginHandlerInterceptor()).addPathPatterns("/**")
                        .excludePathPatterns("/index.html","/","/user/login");
            }
        };
        return adapter;
    }
```

Shiro

```
Subject subject=SecurityUtils.getSubject();
subject.isAuthenticated();
Session session=subject.getSession();
session.setAttribute("a","b");
session.getAttribute("a");
subject.hasRole("管理员");
List<String> stringList=new ArrayList<>();
subject.hasAllRoles(stringList);
//具备的行为--保安：看大门
subject.isPermitted("用户：行为");
subject.isPermitted("用户：删除：张三");
```

实现 realm

```
public class UserRealm extends AuthorizingRealm {
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principalCollection) {

        Object object=principalCollection.getPrimaryPrincipal();
        Set<String> roles=new HashSet<>();
        if("admin".equals(object)){
        roles.add("admin");
   }
   SimpleAuthorizationInfo simpleAuthorizationInfo=
        new SimpleAuthorizationInfo(roles);
        return simpleAuthorizationInfo;

    }

    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken
authenticationToken) throws AuthenticationException {
        UsernamePasswordToken usernamePasswordToken=
                (UsernamePasswordToken)authenticationToken;
        Object user=usernamePasswordToken.getUsername();
        char[] password_char=usernamePasswordToken.getPassword();
        Object password=password_char.toString();
```

```
        String name = getName();
        ByteSource credentialsSalt=ByteSource.Util.bytes("solt");
        SimpleAuthenticationInfo simpleAuthenticationInfo=
                new
SimpleAuthenticationInfo(user,password,credentialsSalt,getName());
        return simpleAuthenticationInfo;
    }
}
```

MD5

```
SimpleHash simpleHash=
        new SimpleHash("MD5","1234569","刘彪",66);
String s=simpleHash.toString();
String s2=simpleHash.getAlgorithmName();
System.out.println(s+"--"+s2+"--"+simpleHash.getIterations());
```

权限

/index.jsp* = anon
/home* = anon
/sysadmin/login/login.jsp* = anon
/sysadmin/login/logout.jsp* = anon
/login* = anon
/logout* = anon

/login=roles[user1]

/*.* = authc
/resource/** = anon

```
1. [users]
2. zhang=123,role1,role2
3. wang=123,role1
4. [roles]
5. role1=user:create,user:update
6. role2=user:create,user:delete
```

Sesson

后台直接获取

```
Session session = subject.getSession();
session.setTimeout(毫秒);
```

记住我

要配合 athoc 和角色

```
usernamePasswordToken.setRememberMe(true);
```

Jsp 标签

```
<%@taglib prefix="shiro" uri="http://shiro.apache.org/tags" %>
```

权限注解

从数据表中获取权限

```
<!-- 配置一个 bean, 该 bean 实际上是一个 Map. 通过实例工厂方法的方式 -->
<bean id="filterChainDefinitionMap"
    factory-bean="filterChainDefinitionMapBuilder" factory-method="buildFilterChainDefinitionMap"></bean

<bean id="filterChainDefinitionMapBuilder"
    class="com.atguigu.shiro.factory.FilterChainDefinitionMapBuilder"></bean>
```

```
ct.xml  ShiroFilterFactoryBean.class
    <property name="successUrl" value="/list.jsp"/>
    <property name="unauthorizedUrl" value="/unauthorized.jsp"/>

    <property name="filterChainDefinitionMap" ref="filterChainDefinitionMap"></property>

    <!--
        配置哪些页面需要受保护.
        以及访问这些页面需要的权限.
        1). anon 可以被匿名访问
        2). authc 必须认证(即登录)后才可能访问的页面.
        3). logout 登出.
        4). roles 角色过滤器
    -->
    --
2
3   import java.util.LinkedHashMap;
4
5   public class FilterChainDefinitionMapBuilder {
6
7       public LinkedHashMap<String, String> buildFilterChainDefinitionMap(){
8           LinkedHashMap<String, String> map = new LinkedHashMap<>();
9
10          map.put("/login.jsp", "anon");
11          map.put("/shiro/login = anon", "anon");
12          map.put("/**", "authc");
13
14          return map;
15      }
16
17  }
18
```

Springboot 和数据库权限等详见

Springboot: https://blog.csdn.net/ityouknow/article/details/73836159
数据库: https://blog.csdn.net/hzw2312/article/details/54612962
综合： https://www.sojson.com/shiro#_lib

Springboot webservice

```
import javax.xml.ws.Endpoint;

import application.webservice.LiubiaoService;
import org.apache.cxf.Bus;
import org.apache.cxf.jaxws.EndpointImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;


@Configuration
public class CxfConfiguration {
```

```
    @Autowired
    private Bus bus;


    @Autowired
    private LiubiaoService liubiaoService;

    /** JAX-WS **/
    @Bean
    public Endpoint endpoint() {
        EndpointImpl endpoint = new EndpointImpl(bus, liubiaoService);
        endpoint.publish("/ws");
        return endpoint;
    }
}
```

```
package application.webservice.impl;

import application.webservice.LiubiaoService;
import org.springframework.stereotype.Component;

import javax.jws.WebService;

@WebService(name = "LiubiaoService",targetNamespace = "http://application",
        endpointInterface = "application.webservice.LiubiaoService")
@Component
public class LiubiaoServiceImpl implements LiubiaoService {
    @Override
    public String sayHello(String name) {

        return "Hello ," + name;
    }
}
```

```
package application.webservice;


import javax.jws.WebMethod;
```

```
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;

@WebService(name = "LiubiaoService",targetNamespace = "http://application")
public interface LiubiaoService {
    @WebMethod
    @WebResult(name = "String", targetNamespace = "")
    public String sayHello(@WebParam(name = "userName") String name);
}
```

AspectJ 注解

```
<?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:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd">
    <!-- 开启 AspectJ 自动代理-->
    <aop:aspectj-autoproxy />
     <!-- 配置切面 -->
    <bean id="myAspectAnno" class="cn.augmentum.aspectj.demo1.MyAspectAnno"/>

</beans>
```

```
/**
 * 自定义切面类:
 *
 */
@Aspect
public class MyAspectAnno {

    @Before(value="execution(* cn.augmentum.aspectj.demo1.CustomerService+.save(..))")
    public void before(JoinPoint joinPoint){
        System.out.println("前置通知==========="+joinPoint);
    }

    @AfterReturning(value="execution(*
cn.augmentum.aspectj.demo1.CustomerService+.update(..))",returning="result")
    public void afterReturing(Object result){
```

```
15          System.out.println("后置通知============"+result);
16      }
17
18      @Around(value="execution(* cn.augmentum.aspectj.demo1.CustomerService+.delete(..))")
19      public Object around(ProceedingJoinPoint joinPoint) throws Throwable{
20          System.out.println("环绕前通知==========");
21          Object obj = joinPoint.proceed();
22          System.out.println("环绕后通知==========");
23          return obj;
24      }
25
26      @AfterThrowing(value="MyAspectAnno.myPointcut1()",throwing="e")
27      public void afterThrowing(Throwable e){
28          System.out.println("异常抛出通知=========="+e.getMessage());
29      }
30
31      @After(value="MyAspectAnno.myPointcut1()")
32      public void after(){
33          System.out.println("最终通知==========");
34      }
35
36      @Pointcut(value="execution(* cn.augmentum.aspectj.demo1.CustomerService+.find(..))")
37      private void myPointcut1(){}
38  }
```

AspectJ_XML

```
<!-- 配置目标类 -->
10      <bean id="orderService" class="cn.augmentum.aspectj.demo2.OrderService"></bean>
11
12      <!-- 配置切面 -->
13      <bean id="myAspectXml" class="cn.augmentum.aspectj.demo2.MyAspectXml"></bean>
14
15      <!-- AOP 的配置 -->
16      <aop:config>
17          <aop:pointcut expression="execution(*
cn.augmentum.aspectj.demo2.OrderService.save(..))" id="pointcut1"/>
18          <aop:pointcut expression="execution(*
cn.augmentum.aspectj.demo2.OrderService.update(..))" id="pointcut2"/>
19          <aop:pointcut expression="execution(*
cn.augmentum.aspectj.demo2.OrderService.delete(..))" id="pointcut3"/>
```

```
20          <aop:pointcut expression="execution(*
cn.augmentum.aspectj.demo2.OrderService.find(..))" id="pointcut4"/>
21          <aop:aspect ref="myAspectXml">
22              <aop:before method="before" pointcut-ref="pointcut1"/>
23              <aop:after-returning method="afterReturing" pointcut-ref="pointcut2"
returning="result"/>
24              <aop:around method="around" pointcut-ref="pointcut3"/>
25              <aop:after-throwing method="afterThrowing" pointcut-ref="pointcut4"
throwing="e"/>
26              <aop:after method="after" pointcut-ref="pointcut4"/>
27          </aop:aspect>
28      </aop:config>
```

FastDFS

```
@Test
    public void testUpload() throws Exception {
        // 1、把 FastDFS 提供的 jar 包添加到工程中
        // 2、初始化全局配置。加载一个配置文件。
        ClientGlobal.init("D:\\workspaces-itcast\\JaveEE18\\taotao-manager\\taotao-manager-
web\\src\\main\\resources\\properties\\client.conf");
        // 3、创建一个 TrackerClient 对象。
        TrackerClient trackerClient = new TrackerClient();
        // 4、创建一个 TrackerServer 对象。
        TrackerServer trackerServer = trackerClient.getConnection();
        // 5、声明一个 StorageServer 对象，null。
        StorageServer storageServer = null;
        // 6、获得 StorageClient 对象。
        StorageClient storageClient = new StorageClient(trackerServer, storageServer);
        // 7、直接调用 StorageClient 对象方法上传文件即可。
        String[]                                    strings                                    =
storageClient.upload_file("D:\\Documents\\Pictures\\images\\2f2eb938943d.jpg", "jpg", null);
        for (String string : strings) {
            System.out.println(string);
        }
    }
```

SSH

POM

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>cn.liubiao</groupId>
    <artifactId>SSH-FREAMWORK</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <!-- 统一源码的编码方式 -->
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <!-- 统一各个框架版本 -->
        <struts.version>2.5.10</struts.version>
        <spring.version>4.3.8.RELEASE</spring.version>
        <hibernate.version>5.1.7.Final</hibernate.version>
    </properties>


    <dependencies>
        <!-- Junit 依赖 -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <!-- Spring 核心依赖 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <!-- Spring web 依赖 -->
        <dependency>
```

```
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <!-- Spring 整合 ORM 框架依赖 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <!-- Struts2 核心依赖 -->
    <dependency>
        <groupId>org.apache.struts</groupId>
        <artifactId>struts2-core</artifactId>
        <version>${struts.version}</version>
    </dependency>
    <!-- Struts2 和 Spring 整合依赖 -->
    <dependency>
        <groupId>org.apache.struts</groupId>
        <artifactId>struts2-spring-plugin</artifactId>
        <version>${struts.version}</version>
    </dependency>
    <!-- Hibernate 核心依赖 -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>${hibernate.version}</version>
    </dependency>
    <!-- MySQL 依赖 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.42</version>
    </dependency>
    <!-- C3P0 依赖 -->
    <dependency>
        <groupId>com.mchange</groupId>
        <artifactId>c3p0</artifactId>
        <version>0.9.5</version>
    </dependency>
    <!-- AspectJ 依赖 -->
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
```

```
            <version>1.8.10</version>
        </dependency>
        <!-- SLF4J 依赖 -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.7.25</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- 统一源代码编译输出的 JDK 版本 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <!-- 打包时跳过单元测试 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.19.1</version>
                <configuration>
                    <skipTests>true</skipTests>
                </configuration>
            </plugin>

            <!-- 集成 Tomcat 插件 -->
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <version>2.2</version>
                <configuration>
                    <path>/${project.artifactId}</path>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

ACTION

```
@Controller
@Scope("prototype")
public class PageAction extends ActionSupport {

    public String login() throws Exception {
        return "login";
    }
}
```

DAO

```
import cn.liubiao.pojo.Product;

public interface ProductDao {
    void saveProduct(Product product);
}
```

DAOIMPL

```
import cn.liubiao.dao.ProductDao;
import cn.liubiao.pojo.Product;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.stereotype.Repository;

@Repository
public class ProductDaoImpl implements ProductDao {

    private HibernateTemplate template;

    @Autowired
    public ProductDaoImpl(SessionFactory sessionFactory) {
```

```
        this.template = new HibernateTemplate(sessionFactory);

    }

    @Override
    public void saveProduct(Product product) {

        template.save(product);

    }

}
```

POJO

```
import org.hibernate.annotations.GenericGenerator;

import javax.persistence.*;

@Entity
@Table(name = "product")
public class Product {
    @Id
    @GeneratedValue(generator = "pid")
    @GenericGenerator(name = "pid", strategy = "native")
    private int pid;// 商品 ID
    @Column(length = 100)
    private String pname;// 商品名称
    private double price;// 商品价格

    public Product() {

    }

    public Product(String pname, double price) {

        this.pname = pname;
        this.price = price;

    }
```

```
    public int getPid() {
        return pid;
    }

    public void setPid(int pid) {
        this.pid = pid;
    }

    public String getPname() {
        return pname;
    }

    public void setPname(String pname) {
        this.pname = pname;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }
}
```

SERVICE

```
import cn.liubiao.pojo.Product;

public interface ProductService {
    void saveProduct(Product product);
}
```

SERVICEIMPL

```
import cn.liubiao.dao.ProductDao;
import cn.liubiao.pojo.Product;
```

```java
import cn.liubiao.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class ProductServiceImpl implements ProductService {

    @Autowired
    private ProductDao productDao;

    @Override
    public void saveProduct(Product product) {

        productDao.saveProduct(product);

    }

}
```

APPLICATION.XML

Resource 下

```xml
<?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:aop="http://www.springframework.org/schema/aop"
       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
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- 开启包扫描，并注册注解 -->
    <context:component-scan base-package="cn.liubiao.*"/>

    <!-- 引入属性文件 -->
```

```
    <context:property-placeholder location="classpath:jdbc.properties"/>

    <!-- 配置C3P0连接池 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <!-- 数据库连接相关信息 -->
        <property name="jdbcUrl" value="${jdbc.url}"/>
        <property name="driverClass" value="${jdbc.driverClass}"/>
        <property name="user" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>

    <!-- 配置Hibernate的SessionFactory -->
    <bean id="sessionFactory"
class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
        <!-- 注入连接池 -->
        <property name="dataSource" ref="dataSource"/>
        <!-- 配置Hibernate属性 -->
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.show_sql">true</prop><!-- 是否展示SQL -->
                <prop key="hibernate.hbm2ddl.auto">update</prop><!-- 是否自动创建
表结构 -->
                <prop
key="hibernate.dialect">org.hibernate.dialect.MySQL5InnoDBDialect</prop>
            </props>
        </property>
        <!-- 扫描并加载注解过的实体类 -->
        <property name="packagesToScan" value="cn.liubiao.pojo"/>
    </bean>

    <!-- 配置事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.orm.hibernate5.HibernateTransactionManager">
        <!-- 注入SessionFactory -->
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <!-- 配置事务增强 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <!-- 配置需要进行事务管理的方法，和事务传播行为 -->
            <tx:method name="save*" propagation="REQUIRED"/>
            <tx:method name="update*" propagation="REQUIRED"/>
            <tx:method name="delete*" propagation="REQUIRED"/>
```

```
        </tx:attributes>
    </tx:advice>

    <!-- 配置切面 -->
    <aop:config>
        <!-- 配置切入点
            * org.ssh.service.*+.*(..)
                *：表示方法的作用域，*表示所有
                org.ssh.service.*：表示org.ssh.service 下的任何包
                org.ssh.service.*+：表示org.ssh.service 下的任何包及其子包
                *(..)：*表示任何方法，(..)表示方法的任何参数
         -->
        <aop:pointcut id="pointcut" expression="execution(*
cn.liubiao.service.*+.*(..))"/>
        <!-- 适配切入点和事务增强 -->
        <aop:advisor advice-ref="txAdvice" pointcut-ref="pointcut"/>
    </aop:config>

</beans>
```

JDBC.PS

```
jdbc.url=jdbc:mysql:///jpa?characterEncoding=UTF-8
jdbc.driverClass=com.mysql.jdbc.Driver
jdbc.username=root
jdbc.password=admin
```

国际化

messageResource.properties

```
invalid.fieldvalue.price =
\u5546\u54c1\u4ef7\u683c\u8f93\u5165\u683c\u5f0f\u6709\u8bef
```

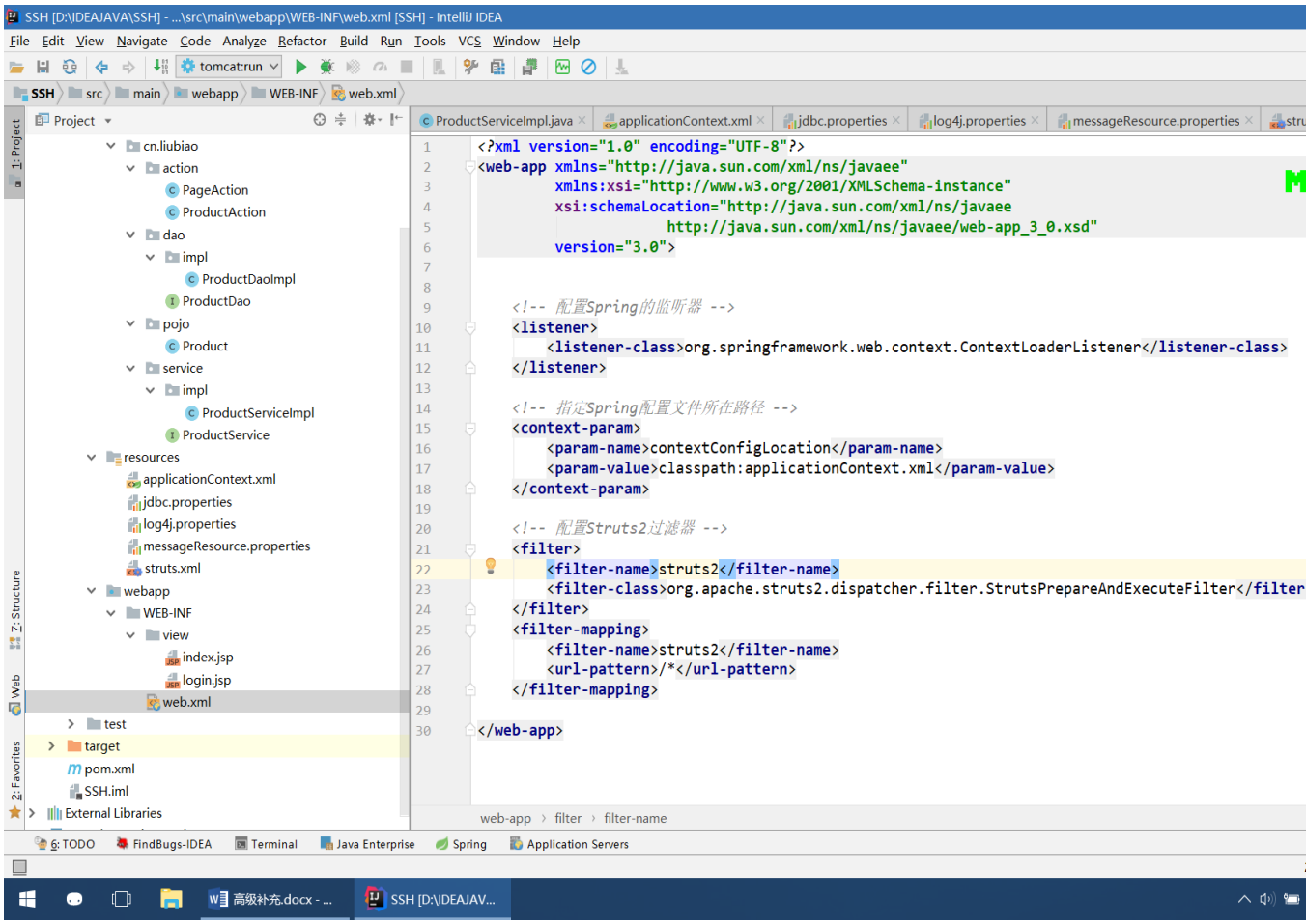
struts

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"
        "http://struts.apache.org/dtds/struts-2.5.dtd">
<struts>
    <!-- 引入资源文件 -->
    <constant name="struts.custom.i18n.resources"
value="messageResource"></constant>


    <!-- 默认访问页面 -->
    <package name="default" extends="struts-default" namespace="/">
        <default-action-ref name="default"/>
        <action name="default">
            <result>/WEB-INF/view/index.jsp</result>
        </action>
    </package>

    <!-- 商品相关请求转发 -->
    <!-- Struts2 在 2.5 版本后添加 strict-method-invocation(严格方法访问)，默认为
true，不能使用动态方法调用功能，故需设为 false -->
    <package name="product" extends="struts-default" namespace="/" strict-
method-invocation="false">
        <!-- 保存商品 -->
        <action name="product_*" class="productAction" method="{1}Product">
            <result>WEB-INF/view/index.jsp</result>
            <result name="input">WEB-INF/view/index.jsp</result>
        </action>

        <action name="login" class="cn.liubiao.action.PageAction"
method="login">
            <result name="login">WEB-INF/view/login.jsp</result>
        </action>
    </package>

</struts>
```

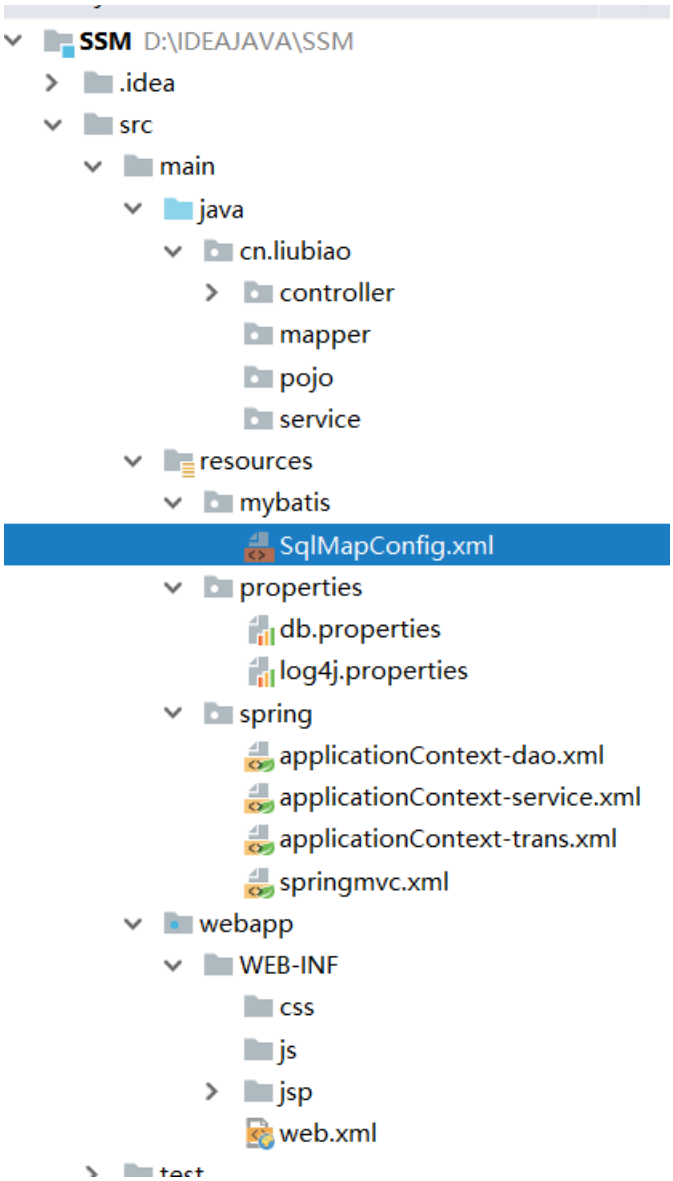
Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">


    <!-- 配置 Spring 的监听器 -->
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <!-- 指定 Spring 配置文件所在路径 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>

    <!-- 配置 Struts2 过滤器 -->
    <filter>
        <filter-name>struts2</filter-name>
        <filter-
class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</filte
r-class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```

目录


SSH [D:\IDEAJAVA\SSH] - ...\src\main\webapp\WEB-INF\web.xml [SSH] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
tomcat:run
SSH src main webapp WEB-INF web.xml
Project
cn.liubiao
action
PageAction
ProductAction
dao
impl
ProductDaoImpl
ProductDao
pojo
Product
service
impl
ProductServiceImpl
ProductService
resources
applicationContext.xml
jdbc.properties
log4j.properties
messageResource.properties
struts.xml
webapp
WEB-INF
view
index.jsp
login.jsp
web.xml
test
target
pom.xml
SSH.iml
External Libraries
ProductServiceImpl.java
applicationContext.xml
jdbc.properties
log4j.properties
messageResource.properties
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">
<!-- 配置Spring的监听器 -->
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<!-- 指定Spring配置文件所在路径 -->
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>classpath:applicationContext.xml</param-value>
</context-param>
<!-- 配置Struts2过滤器 -->
<filter>
<filter-name>struts2</filter-name>
<filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</filter
</filter>
<filter-mapping>
<filter-name>struts2</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>
web-app › filter › filter-name
6: TODO FindBugs-IDEA Terminal Java Enterprise Spring Application Servers
高级补充.docx - ...
SSH [D:\IDEAJAV...

SSM




```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

</configuration>
```

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/jpa?characterEncoding=utf-8
jdbc.username=root
jdbc.password=admin


log4j.rootLogger=debug, stdout, R

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n

log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=example.log

log4j.appender.R.MaxFileSize=100KB
# Keep one backup file
log4j.appender.R.MaxBackupIndex=5

log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
```

dao

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
       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-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/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
   http://www.springframework.org/schema/util
```

```
http://www.springframework.org/schema/util/spring-util-4.0.xsd">

    <!-- 数据库连接池 -->
    <!-- 加载配置文件 -->
    <context:property-placeholder location="classpath:properties/*.properties"
/>
    <!-- 数据库连接池 -->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
          destroy-method="close">
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
        <property name="driverClassName" value="${jdbc.driver}" />
        <property name="maxActive" value="10" />
        <property name="minIdle" value="5" />
    </bean>
    <!-- 让 spring 管理 sqlsessionfactory 使用 mybatis 和 spring 整合包中的 -->
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <!-- 数据库连接池 -->
        <property name="dataSource" ref="dataSource" />
        <!-- 加载 mybatis 的全局配置文件 -->
        <property name="configLocation"
value="classpath:mybatis/SqlMapConfig.xml" />
    </bean>
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="cn.liubiao.mapper" />
    </bean>
</beans>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
       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-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/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
   http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.0.xsd">

    <!-- 包扫描器，扫描带@Service 注解的类 -->
    <context:component-scan base-
package="cn.liubiao.service"></context:component-scan>

</beans>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
       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-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/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
   http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.0.xsd">
    <!-- 事务管理器 -->
    <bean id="transactionManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <!-- 数据源 -->
        <property name="dataSource" ref="dataSource" />
    </bean>
    <!-- 通知 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <!-- 传播行为 -->
```

```
            <tx:method name="save*" propagation="REQUIRED" />
            <tx:method name="insert*" propagation="REQUIRED" />
            <tx:method name="add*" propagation="REQUIRED" />
            <tx:method name="create*" propagation="REQUIRED" />
            <tx:method name="delete*" propagation="REQUIRED" />
            <tx:method name="update*" propagation="REQUIRED" />
            <tx:method name="find*" propagation="SUPPORTS" read-only="true" />
            <tx:method name="select*" propagation="SUPPORTS" read-only="true" />
            <tx:method name="get*" propagation="SUPPORTS" read-only="true" />
        </tx:attributes>
    </tx:advice>
    <!-- 切面 -->
    <aop:config>
        <aop:advisor advice-ref="txAdvice"
                     pointcut="execution(* cn.liubiao.service.*.*(..))" />
    </aop:config>
</beans>
```

```
<?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"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 配置包扫描器 -->
    <context:component-scan base-
package="cn.liubiao.controller"></context:component-scan>
    <!-- 配置注解驱动 -->
    <mvc:annotation-driven/>
    <!-- 视图解析器 -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>
```

```
    <mvc:resources mapping="/js/**" location="/WEB-INF/js/"/>
    <mvc:resources mapping="/css/**" location="/WEB-INF/css/"/>
</beans>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         id="taotao" version="2.5">
    <display-name>taotao-manager</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
    <!-- 初始化 spring 容器 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring/applicationContext-*.xml</param-value>
    </context-param>
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <!-- 解决 post 乱码 -->
    <filter>
        <filter-name>CharacterEncodingFilter</filter-name>
        <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>utf-8</param-value>
        </init-param>
    </filter>
    <filter-mapping>
```

```
        <filter-name>CharacterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>


    <!-- springmvc 的前端控制器 -->
    <servlet>
        <servlet-name>ssm-manager</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!-- contextConfigLocation 不是必须的， 如果不配置 contextConfigLocation，
springmvc 的配置文件默认在：WEB-INF/servlet 的 name+"-servlet.xml" -->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:spring/springmvc.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>ssm-manager</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```