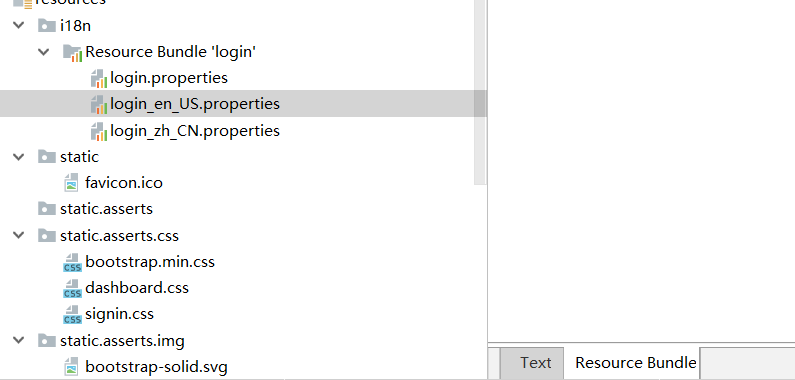
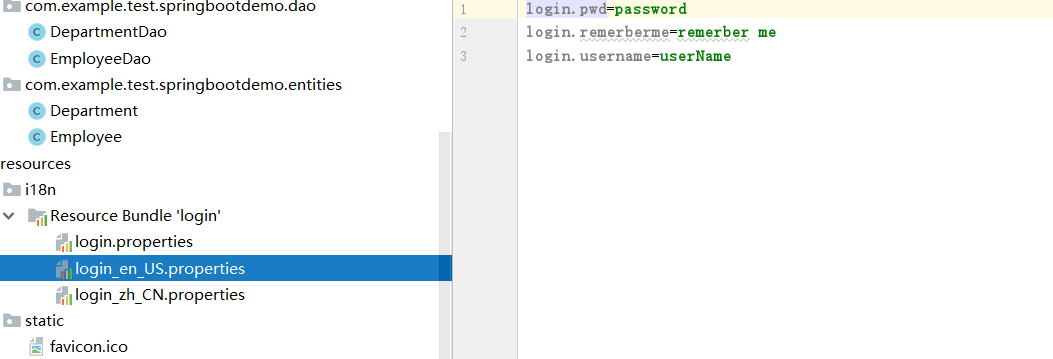
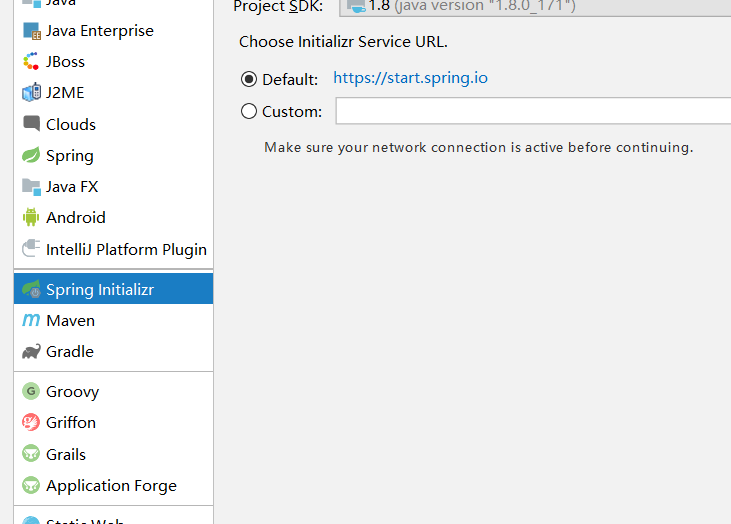
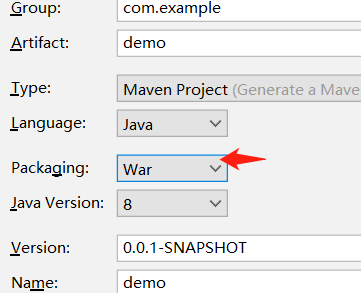
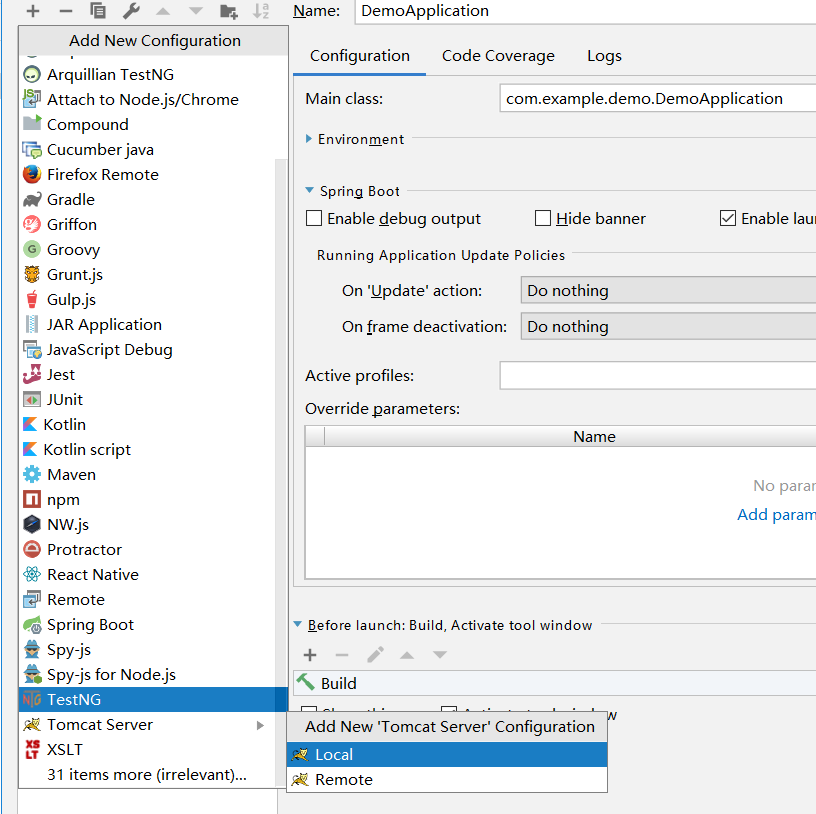
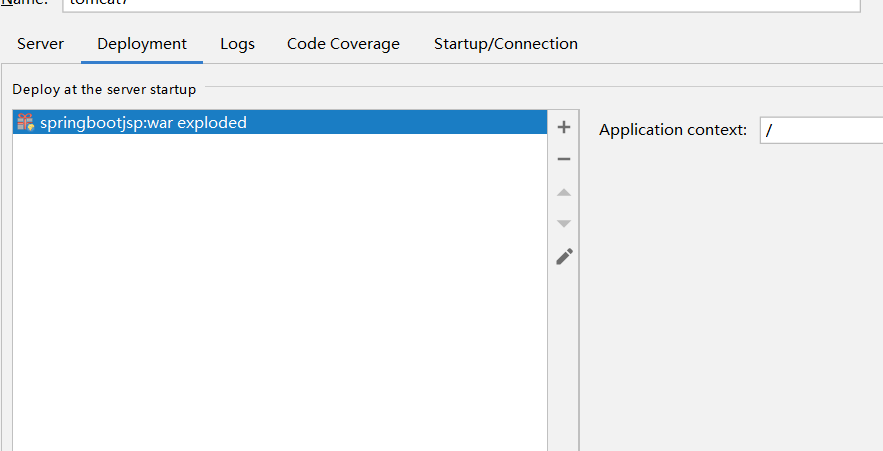
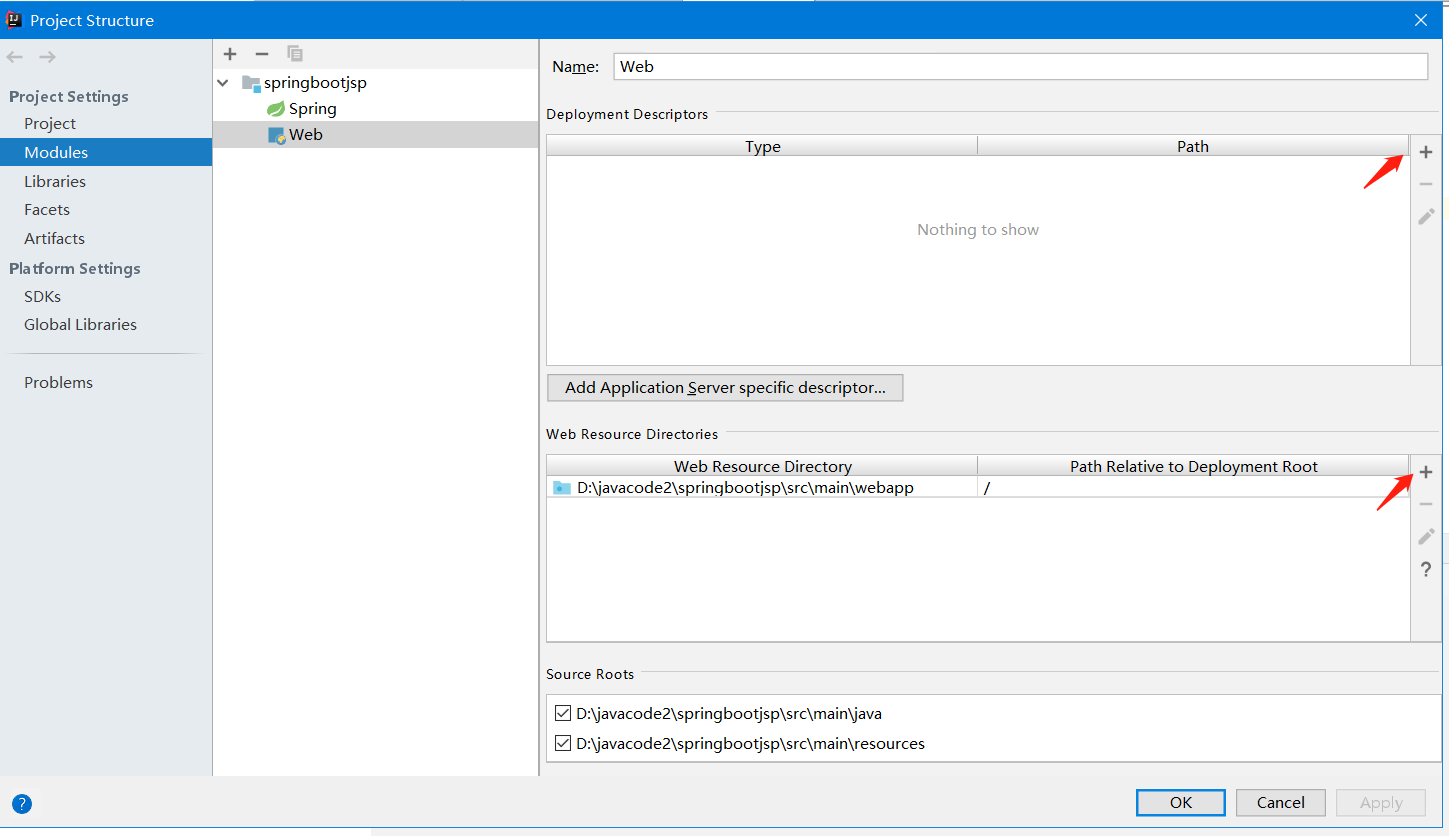
配置国际化步骤

1. 在resources下新建国际化配置文件
2. 
3. 配置中英文名称，加号直接写就行
4. 
5. 配置文件中指定基础名,包括包名

Spring.message.basename=i18n.login

Springboot整合jsp

1. 新建springboot项目，war包。
2. 2. 
3. 选择配置，添加tomcat服务器，并把该项目添加到服务器中
4. 
5. 
6. 配置MEA-INF的jsp文件
7. 
8. 添加webapp目录，并且添加mea-inf相关目录
9. 

**缓存**

/\*\*

\* 将方法的运行结果进行缓存；以后再要相同的数据，直接从缓存中获取，不用调用方法；

\* CacheManager管理多个Cache组件的，对缓存的真正CRUD操作在Cache组件中，每一个缓存组件有自己唯一一个名字；

\*

\*

\* 原理：

\* 1、自动配置类；CacheAutoConfiguration

\* 2、缓存的配置类

\* org.springframework.boot.autoconfigure.cache.GenericCacheConfiguration

\* org.springframework.boot.autoconfigure.cache.JCacheCacheConfiguration

\* org.springframework.boot.autoconfigure.cache.EhCacheCacheConfiguration

\* org.springframework.boot.autoconfigure.cache.HazelcastCacheConfiguration

\* org.springframework.boot.autoconfigure.cache.InfinispanCacheConfiguration

\* org.springframework.boot.autoconfigure.cache.CouchbaseCacheConfiguration

\* org.springframework.boot.autoconfigure.cache.RedisCacheConfiguration

\* org.springframework.boot.autoconfigure.cache.CaffeineCacheConfiguration

\* org.springframework.boot.autoconfigure.cache.GuavaCacheConfiguration

\* org.springframework.boot.autoconfigure.cache.SimpleCacheConfiguration【默认】

\* org.springframework.boot.autoconfigure.cache.NoOpCacheConfiguration

\* 3、哪个配置类默认生效：SimpleCacheConfiguration；

\*

\* 4、给容器中注册了一个CacheManager：ConcurrentMapCacheManager

\* 5、可以获取和创建ConcurrentMapCache类型的缓存组件；他的作用将数据保存在ConcurrentMap中；

\*

\* 运行流程：

\* @Cacheable：

\* 1、方法运行之前，先去查询Cache（缓存组件），按照cacheNames指定的名字获取；

\* （CacheManager先获取相应的缓存），第一次获取缓存如果没有Cache组件会自动创建。

\* 2、去Cache中查找缓存的内容，使用一个key，默认就是方法的参数；

\* key是按照某种策略生成的；默认是使用keyGenerator生成的，默认使用SimpleKeyGenerator生成key；

\* SimpleKeyGenerator生成key的默认策略；

\* 如果没有参数；key=new SimpleKey()；

\* 如果有一个参数：key=参数的值

\* 如果有多个参数：key=new SimpleKey(params)；

\* 3、没有查到缓存就调用目标方法；

\* 4、将目标方法返回的结果，放进缓存中

\*

\* @Cacheable标注的方法执行之前先来检查缓存中有没有这个数据，默认按照参数的值作为key去查询缓存，

\* 如果没有就运行方法并将结果放入缓存；以后再来调用就可以直接使用缓存中的数据；

\*

\* 核心：

\* 1）、使用CacheManager【ConcurrentMapCacheManager】按照名字得到Cache【ConcurrentMapCache】组件

\* 2）、key使用keyGenerator生成的，默认是SimpleKeyGenerator

\*

\*

\* 几个属性：

\* cacheNames/value：指定缓存组件的名字;将方法的返回结果放在哪个缓存中，是数组的方式，可以指定多个缓存；

\*

\* key：缓存数据使用的key；可以用它来指定。默认是使用方法参数的值 1-方法的返回值

\* 编写SpEL； #i d;参数id的值 #a0 #p0 #root.args[0]

\* getEmp[2]

\*

\* keyGenerator：key的生成器；可以自己指定key的生成器的组件id

\* key/keyGenerator：二选一使用;

如何自定义生成keyGenerator：configuration配置类中配置生成一个keyGenerator类，在缓存注解参数中指定它的benId

\*

\*

\* cacheManager：指定缓存管理器；或者cacheResolver指定获取解析器

\*

\* condition：指定符合条件的情况下才缓存；

\* ,condition = "#id>0"

\* condition = "#a0>1"：第一个参数的值》1的时候才进行缓存

\*

\* unless:否定缓存；当unless指定的条件为true，方法的返回值就不会被缓存；可以获取到结果进行判断

\* unless = "#result == null"

\* unless = "#a0==2":如果第一个参数的值是2，结果不缓存；

\* sync：是否使用异步模式

/\*\*

\* @CachePut：既调用方法，又更新缓存数据；同步更新缓存

\* 修改了数据库的某个数据，同时更新缓存；

\* 运行时机：

\* 1、先调用目标方法

\* 2、将目标方法的结果缓存起来

\* key = "#result.id"：使用返回后的id

\* @Cacheable的key是不能用#result，因为@Cacheable是运行方法前,不知道result的结果

\*

\*/

/\*\*

\* @CacheEvict：缓存清除

\* key：指定要清除的数据

\* allEntries = true：指定清除这个缓存中所有的数据

\* beforeInvocation = false：缓存的清除是否在方法之前执行

\* 默认代表缓存清除操作是在方法执行之后执行;如果出现异常缓存就不会清除

\*

\* beforeInvocation = true：

\* 代表清除缓存操作是在方法运行之前执行，无论方法是否出现异常，缓存都清除

\*

\*

\*/

// @Caching 组合注解 定义复杂的缓存规则

@Caching(

cacheable = {

@Cacheable(value="emp",key = "#lastName")

},

put = {

@CachePut(value="emp",key = "#result.id"),

@CachePut(value="emp",key = "#result.email")

}

)

@CacheConfig注解在类上，配置方法公共的配置，比如CacheNames,KeyGeneratory等等

**Rabbitmq**

1.配置参数

**spring**:  
 **rabbitmq**:  
 **port**: 5672  
 **username**: guest  
 **password**: guest  
 **virtual-host**: /

2.注入RabbitTemplete，使用其发送消息

[3.使用@RabbitListener](mailto:3.使用@RabbitListener)注解在方法上，参数写消息实体或者Message表示队列的数据

[4.使用了@RabbitListener](mailto:4.使用了@RabbitListener)注解，就在住程序类上添加@EnableRabbit开启注解配置

1. 默认使用jdk序列化，要想使用Json序列化，配置MessageConverter类，配置json就行
2. 使用adminRabbit类，管理交换机和队列

**Springboot-elasticSearch**

/\*\*

\* SpringBoot默认支持两种技术来和ES交互；

\* 1、Jest（默认不生效）更多操作：https://github.com/searchbox-io/Jest/tree/master/jest

\* 需要导入jest的工具包（io.searchbox.client.JestClient）

\* 2、SpringData ElasticSearch【ES版本有可能不合适】

\* 版本适配说明：https://github.com/spring-projects/spring-data-elasticsearch

\* 如果版本不适配：2.4.6

\* 1）、升级SpringBoot版本

\* 2）、安装对应版本的ES

\*

\* 1）、Client 节点信息clusterNodes；clusterName

\* 2）、ElasticsearchTemplate 操作es

\* 3）、编写一个 ElasticsearchRepository 的子接口来操作ES；

\* 两种用法：https://github.com/spring-projects/spring-data-elasticsearch

\* 1）、编写一个 ElasticsearchRepository

\*/

**Springboot任务**

异步任务

方法是标注：@Async

启动类开启注解：@EnableAsync

定时任务：

@EnableScheduling、@Scheduled



**Springboot邮箱**

**Springboot-security**

/\*\*

\* 1、引入SpringSecurity；

\* 2、编写SpringSecurity的配置类；

\* @EnableWebSecurity extends WebSecurityConfigurerAdapter

\* 3、控制请求的访问权限：

\* configure(HttpSecurity http) {

\* http.authorizeRequests().antMatchers("/").permitAll()

\* .antMatchers("/level1/\*\*").hasRole("VIP1")

\* }

\* 4、定义认证规则：

\* configure(AuthenticationManagerBuilder auth){

\* auth.inMemoryAuthentication()

\* .withUser("zhangsan").password("123456").roles("VIP1","VIP2")

\* }

\* 5、开启自动配置的登陆功能：

\* configure(HttpSecurity http){

不要默认的登陆页面，自定义的登陆页面

\* http.formLogin().loginPage(“/userlogin”);

\* }

\* 6、注销：http.logout();

\* 7、记住我：Remeberme()；

\*/

@EnableWebSecurity

public class MySecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

//super.configure(http);

//定制请求的授权规则

http.authorizeRequests().antMatchers("/").permitAll()

.antMatchers("/level1/\*\*").hasRole("VIP1")

.antMatchers("/level2/\*\*").hasRole("VIP2")

.antMatchers("/level3/\*\*").hasRole("VIP3");

//开启自动配置的登陆功能，效果，如果没有登陆，没有权限就会来到登陆页面

http.formLogin().usernameParameter("user").passwordParameter("pwd")

.loginPage("/userlogin");

//1、/login来到登陆页

//2、重定向到/login?error表示登陆失败

//3、更多详细规定

//4、默认post形式的 /login代表处理登陆

//5、一但定制loginPage；那么 loginPage的post请求就是登陆

//开启自动配置的注销功能。

http.logout().logoutSuccessUrl("/");//注销成功以后来到首页

//1、访问 /logout 表示用户注销，清空session

//2、注销成功会返回 /login?logout 页面；

//开启记住我功能

http.rememberMe().rememberMeParameter("remeber");

//登陆成功以后，将cookie发给浏览器保存，以后访问页面带上这个cookie，只要通过检查就可以免登录

//点击注销会删除cookie

}

//定义认证规则

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

//super.configure(auth);

auth.inMemoryAuthentication()

.withUser("zhangsan").password("123456").roles("VIP1","VIP2")

.and()

.withUser("lisi").password("123456").roles("VIP2","VIP3")

.and()

.withUser("wangwu").password("123456").roles("VIP1","VIP3");

}

}