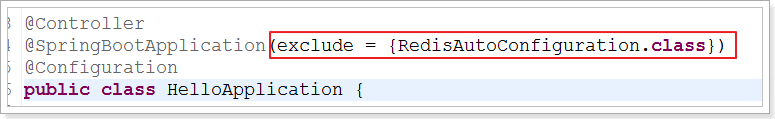
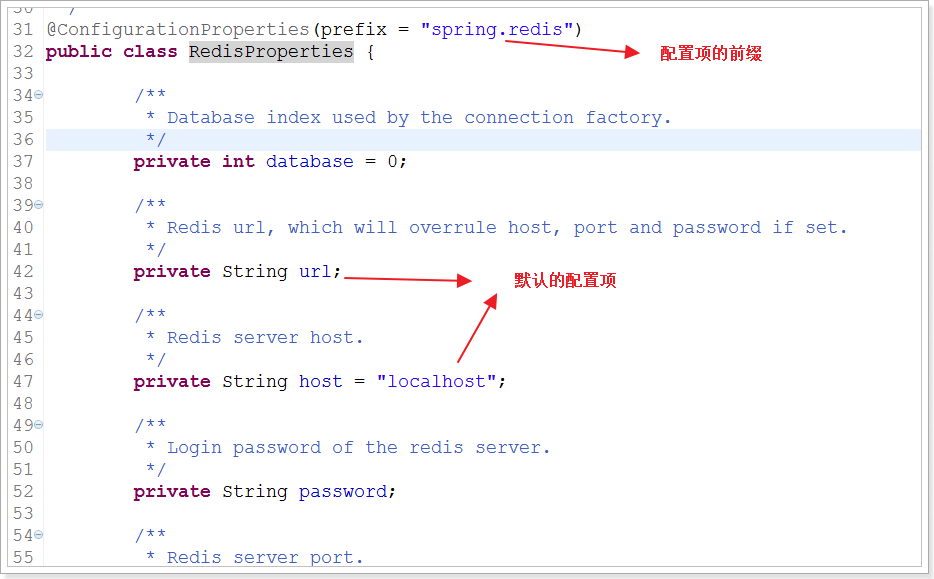
不想自动配置Redis，手动配置

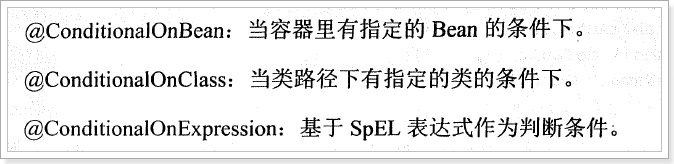


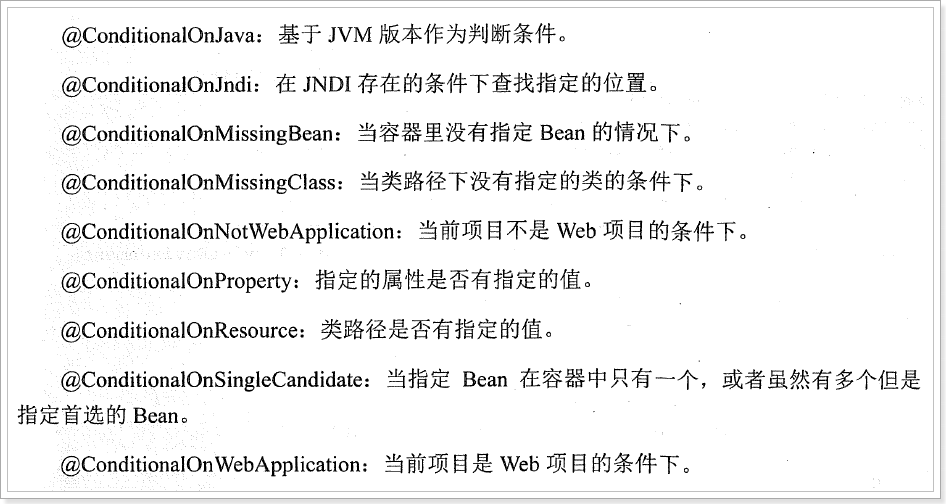
### Redis的自动配置



】

### 条件注解





@Configuration//申明这是一个配置

**publicclass** MySrpingMVCConfig **extends** WebMvcConfigurerAdapter{

// 自定义拦截器

@Override

**publicvoid** addInterceptors(InterceptorRegistry registry) {

HandlerInterceptor handlerInterceptor = **new** HandlerInterceptor() {

@Override

**publicboolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)

**throws** Exception {

System.***out***.println("自定义拦截器............");

**returntrue**;

}

### 设置Mybatis和Spring Boot整合

@Configuration

**publicclass** MyBatisConfig {

@Bean

@ConditionalOnMissingBean//当容器里没有指定的Bean的情况下创建该对象

**public** SqlSessionFactoryBean sqlSessionFactory(DataSource dataSource) {

SqlSessionFactoryBean sqlSessionFactoryBean = **new** SqlSessionFactoryBean();

// 设置数据源

sqlSessionFactoryBean.setDataSource(dataSource);

// 设置mybatis的主配置文件

ResourcePatternResolver resolver = **new**PathMatchingResourcePatternResolver();

Resource mybatisConfigXml = resolver.getResource("classpath:mybatis/mybatis-config.xml");

sqlSessionFactoryBean.setConfigLocation(mybatisConfigXml);

// 设置别名包

sqlSessionFactoryBean.setTypeAliasesPackage("com.taotao.cart.pojo");

**return**sqlSessionFactoryBean;

}

}

**import** org.mybatis.spring.mapper.MapperScannerConfigurer;

**import** org.springframework.boot.autoconfigure.AutoConfigureAfter;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

@Configuration

@AutoConfigureAfter(MyBatisConfig.**class**) //保证在MyBatisConfig实例化之后再实例化该类

**publicclass** MapperScannerConfig {

// mapper接口的扫描器

@Bean

**public** MapperScannerConfigurer mapperScannerConfigurer() {

MapperScannerConfigurer mapperScannerConfigurer = **new** MapperScannerConfigurer();

mapperScannerConfigurer.setBasePackage("com.taotao.cart.mapper");

**return**mapperScannerConfigurer;

}

}

### 设置事务管理

在Spring Boot中推荐使用@Transactional注解来申明事务。

首先需要导入依赖：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

当引入jdbc依赖之后，Spring Boot会自动默认分别注入DataSourceTransactionManager或JpaTransactionManager

在Service中添加@Transactional注解

@Configuration

@PropertySource(value = "classpath:redis.properties")

**publicclass** RedisSpringConfig {

@Value("${redis.maxTotal}")

**private** Integer redisMaxTotal;

@Value("${redis.node1.host}")

**private** String redisNode1Host;

@Value("${redis.node1.port}")

**private** Integer redisNode1Port;

**private** JedisPoolConfig jedisPoolConfig() {

JedisPoolConfig jedisPoolConfig = **new**JedisPoolConfig();

jedisPoolConfig.setMaxTotal(redisMaxTotal);

**return**jedisPoolConfig;

}

@Bean

**public**ShardedJedisPool shardedJedisPool() {

List<JedisShardInfo>jedisShardInfos = **new** ArrayList<JedisShardInfo>();

jedisShardInfos.add(**new** JedisShardInfo(redisNode1Host, redisNode1Port));

**returnnew** ShardedJedisPool(jedisPoolConfig(), jedisShardInfos);

}

}

### 设置Httpclient和Spring的整合

**import** org.apache.http.client.config.RequestConfig;

**import** org.apache.http.impl.client.CloseableHttpClient;

**import** org.apache.http.impl.client.HttpClients;

**import** org.apache.http.impl.conn.PoolingHttpClientConnectionManager;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.context.annotation.PropertySource;

**import** org.springframework.context.annotation.Scope;

**import** com.taotao.common.httpclient.IdleConnectionEvictor;

@Configuration

@PropertySource(value = "classpath:httpclient.properties")

**publicclass** HttpclientSpringConfig {

@Value("${http.maxTotal}")

**private** Integer httpMaxTotal;

@Value("${http.defaultMaxPerRoute}")

**private** Integer httpDefaultMaxPerRoute;

@Value("${http.connectTimeout}")

**private** Integer httpConnectTimeout;

@Value("${http.connectionRequestTimeout}")

**private** Integer httpConnectionRequestTimeout;

@Value("${http.socketTimeout}")

**private** Integer httpSocketTimeout;

@Value("${http.staleConnectionCheckEnabled}")

**private** Boolean httpStaleConnectionCheckEnabled;

@Autowired

**private** PoolingHttpClientConnectionManager manager;

@Bean

**public** PoolingHttpClientConnectionManager poolingHttpClientConnectionManager() {

PoolingHttpClientConnectionManager poolingHttpClientConnectionManager = **new** PoolingHttpClientConnectionManager();

// 最大连接数

poolingHttpClientConnectionManager.setMaxTotal(httpMaxTotal);

// 每个主机的最大并发数

poolingHttpClientConnectionManager.setDefaultMaxPerRoute(httpDefaultMaxPerRoute);

**return**poolingHttpClientConnectionManager;

}

// 定期关闭无效连接

@Bean

**public** IdleConnectionEvictor idleConnectionEvictor() {

**returnnew** IdleConnectionEvictor(manager);

}

// 定义Httpclient对

@Bean

@Scope("prototype")

**public** CloseableHttpClient closeableHttpClient() {

**return** HttpClients.*custom*().setConnectionManager(**this**.manager).build();

}

// 请求配置

@Bean

**public** RequestConfig requestConfig() {

**return** RequestConfig.*custom*().setConnectTimeout(httpConnectTimeout) // 创建连接的最长时间

.setConnectionRequestTimeout(httpConnectionRequestTimeout) // 从连接池中获取到连接的最长时间

.setSocketTimeout(httpSocketTimeout) // 数据传输的最长时间

.~~setStaleConnectionCheckEnabled~~(httpStaleConnectionCheckEnabled)// 提交请求前测试连接是否可用

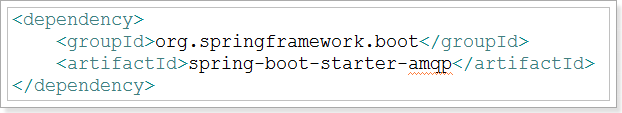
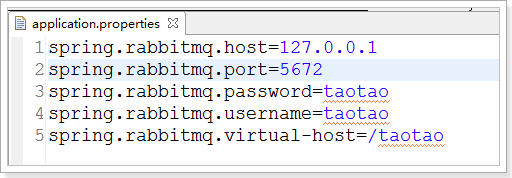
.build();

}

}

### 设置RabbitMQ和Spring的整合

我们之前使用的Spring-Rabbit的xml方式，现在我们要改造成java方式，并且Spring Boot对RabbitMQ的使用做了自动配置，更加的简化了我们的使用。

1. 在导入spring-boot-starter-amqp的依赖；  
   
2. 在application.properties文件中配置RabbitMQ的连接信息  
   
3. 编写Rabbit的Spring配置类  
   **import** org.springframework.amqp.core.Queue;

**import** org.springframework.amqp.rabbit.connection.ConnectionFactory;

**import** org.springframework.amqp.rabbit.core.RabbitAdmin;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

@Configuration

**publicclass** RabbitMQSpringConfig {

@Autowired

**private** ConnectionFactory connectionFactory;

// 管理

@Bean

**public** RabbitAdmin rabbitAdmin() {

**returnnew** RabbitAdmin(connectionFactory);

}

// 声明队列

@Bean

**public** Queue taotaoCartLoginQueue() {

// 默认就是自动声明的

**returnnew** Queue("TAOTAO-CART-LOGIN-QUEUE", **true**);

}

// 声明队列

@Bean

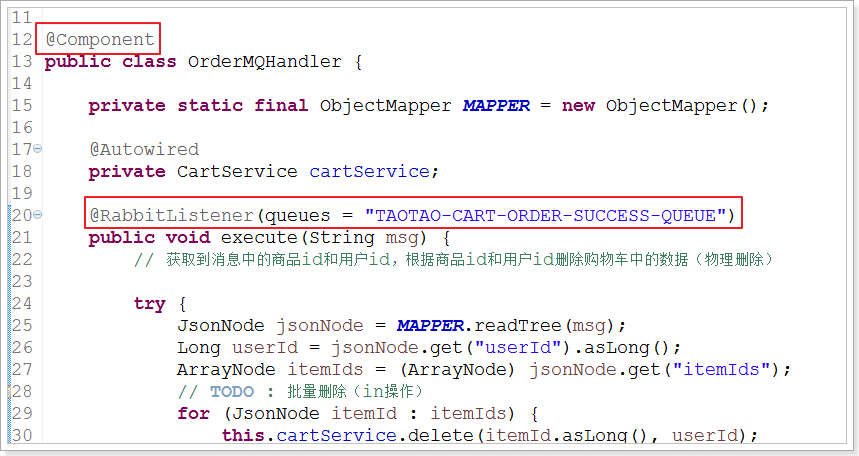
**public** Queue taotaoCartOrderSuccessQueue() {

// 默认就是自动声明的

**returnnew** Queue("TAOTAO-CART-ORDER-SUCCESS-QUEUE", **true**);

}

}

1. 设置监听  
     
   

### 设置SpringMVC的配置

原有配置：

