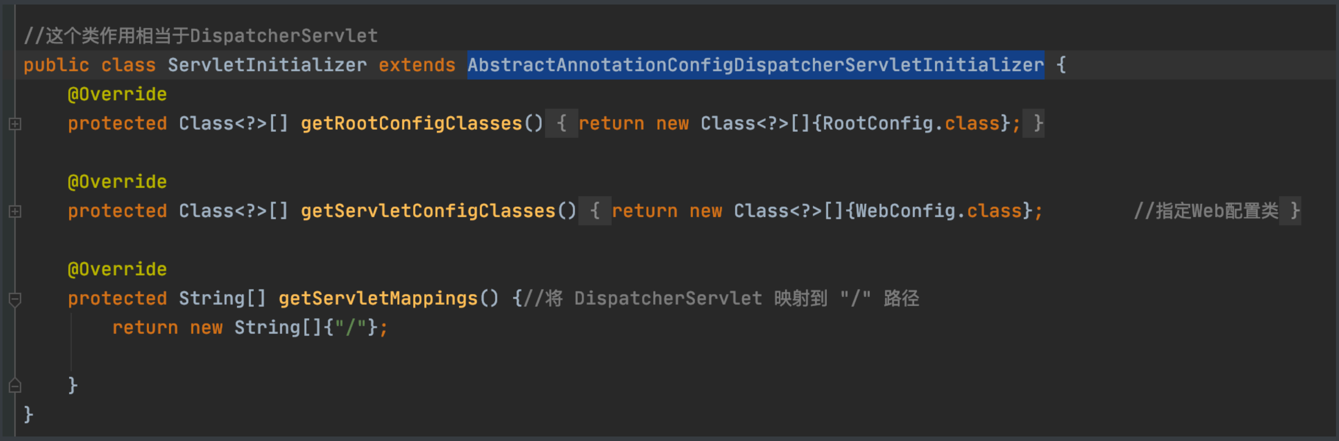
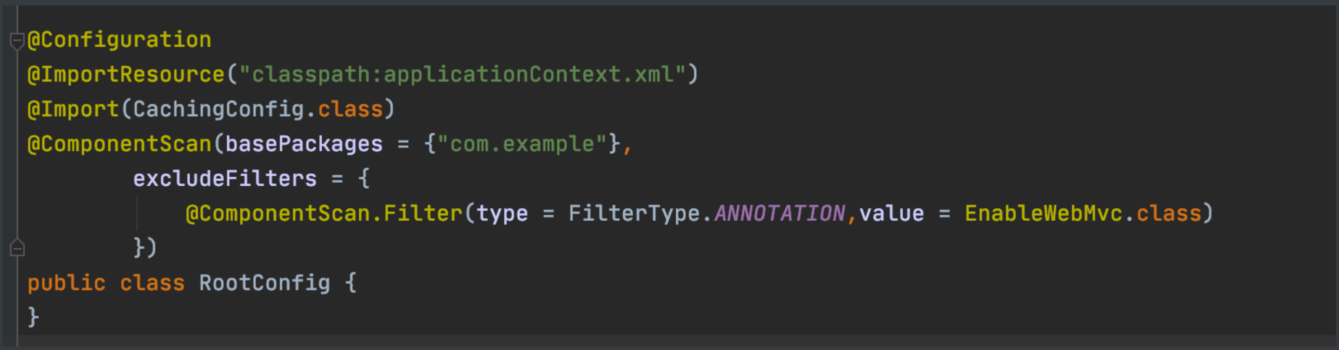
阶段测试

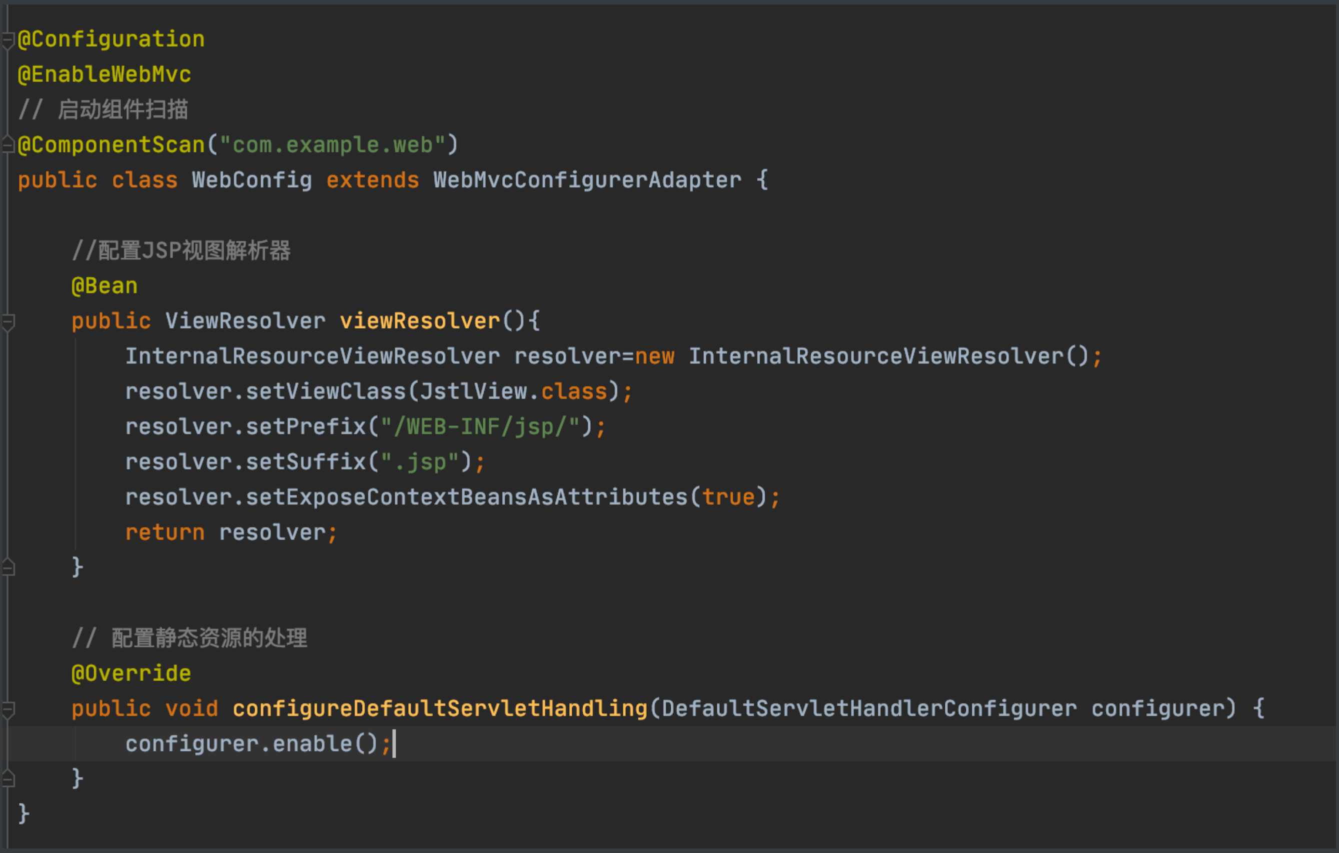
1. 主要是增加了三个类，其中ServletInitializer继承自AbstractAnnotationConfigDispatcherServletInitializer，并且引用RootConfig、WebConfig来进行配置



```java
//这个类作用相当于DispatcherServlet
public class ServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() { return new Class<?>[]{RootConfig.class}; }

    @Override
    protected Class<?>[] getServletConfigClasses() { return new Class<?>[]{WebConfig.class};        //指定Web配置类 }

    @Override
    protected String[] getServletMappings() {//将 DispatcherServlet 映射到 "/" 路径
        return new String[]{"/"};

    }
}
```

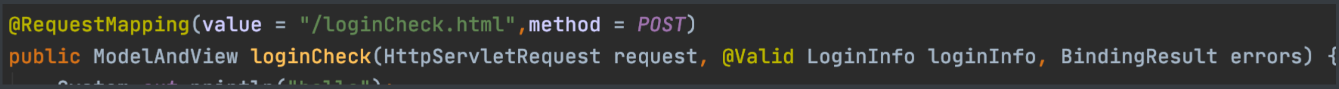


```java
@Configuration
@ImportResource("classpath:applicationContext.xml")
@Import(CachingConfig.class)
@ComponentScan(basePackages = {"com.example"},
        excludeFilters = {
            @ComponentScan.Filter(type = FilterType.ANNOTATION,value = EnableWebMvc.class)
        })
public class RootConfig {
}
```

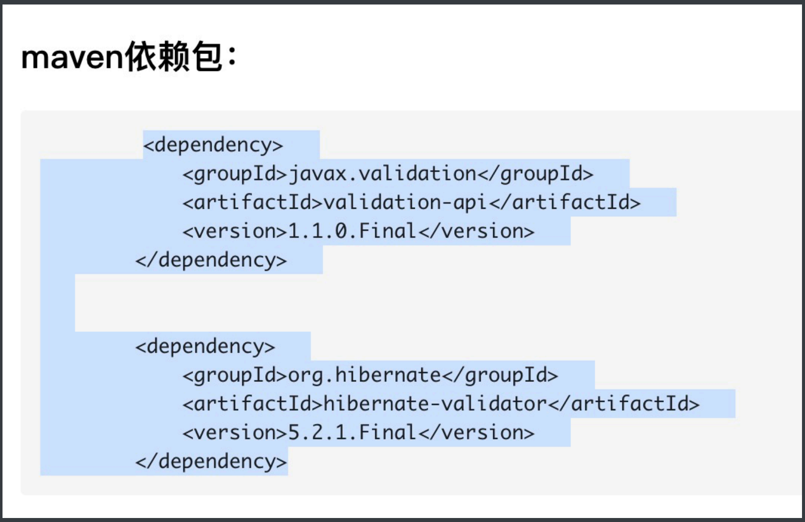


```java
@Configuration
@EnableWebMvc
// 启动组件扫描
@ComponentScan("com.example.web")
public class WebConfig extends WebMvcConfigurerAdapter {

    //配置JSP视图解析器
    @Bean
    public ViewResolver viewResolver(){
        InternalResourceViewResolver resolver=new InternalResourceViewResolver();
        resolver.setViewClass(JstlView.class);
        resolver.setPrefix("/WEB-INF/jsp/");
        resolver.setSuffix(".jsp");
        resolver.setExposeContextBeansAsAttributes(true);
        return resolver;
    }

    // 配置静态资源的处理
    @Override
    public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {
        configurer.enable();
    }
}
```

2. 在LoginInfo里面添加valid所需注解（需要添加依赖），在LoginController控制层中优化loginCheck方法，并添加@Valid（Validated好像也行）

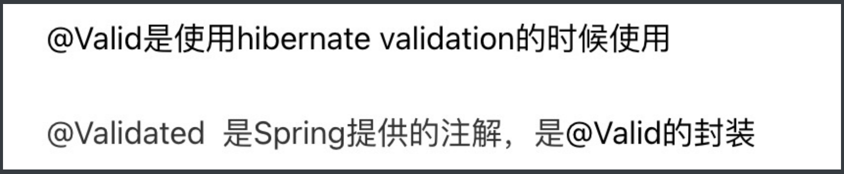


```java
@RequestMapping(value = "/loginCheck.html",method = POST)
public ModelAndView loginCheck(HttpServletRequest request, @Valid LoginInfo loginInfo, BindingResult errors) {
```



maven依赖包：

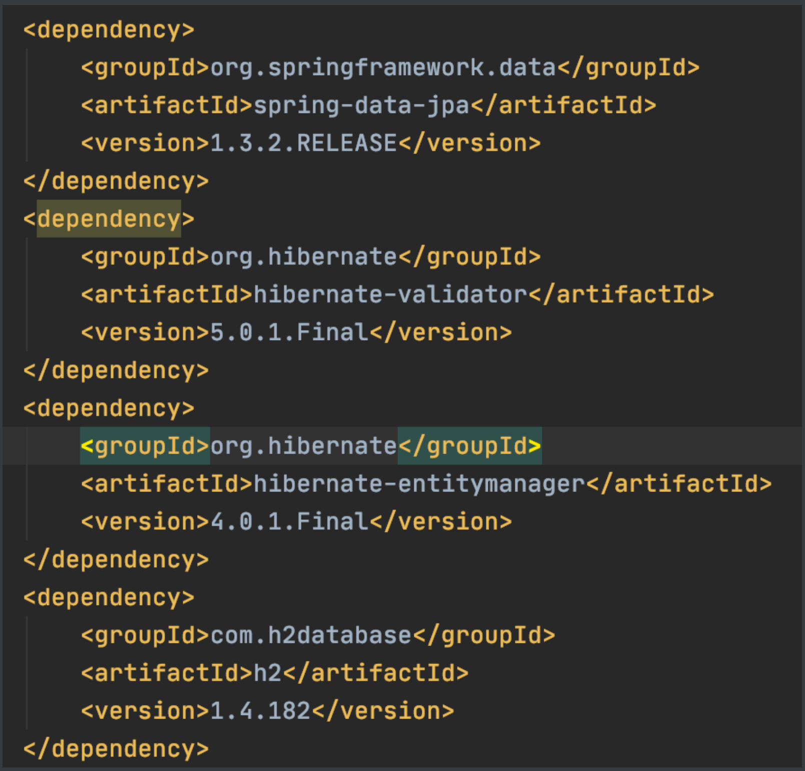
```xml
        <dependency>
            <groupId>javax.validation</groupId>
            <artifactId>validation-api</artifactId>
            <version>1.1.0.Final</version>
        </dependency>


        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-validator</artifactId>
            <version>5.2.1.Final</version>
        </dependency>
```



@Valid是使用hibernate validation的时候使用

@Validated 是Spring提供的注解，是@Valid的封装

3. （第三问、第四问耦合比较大，一起做）导入依赖

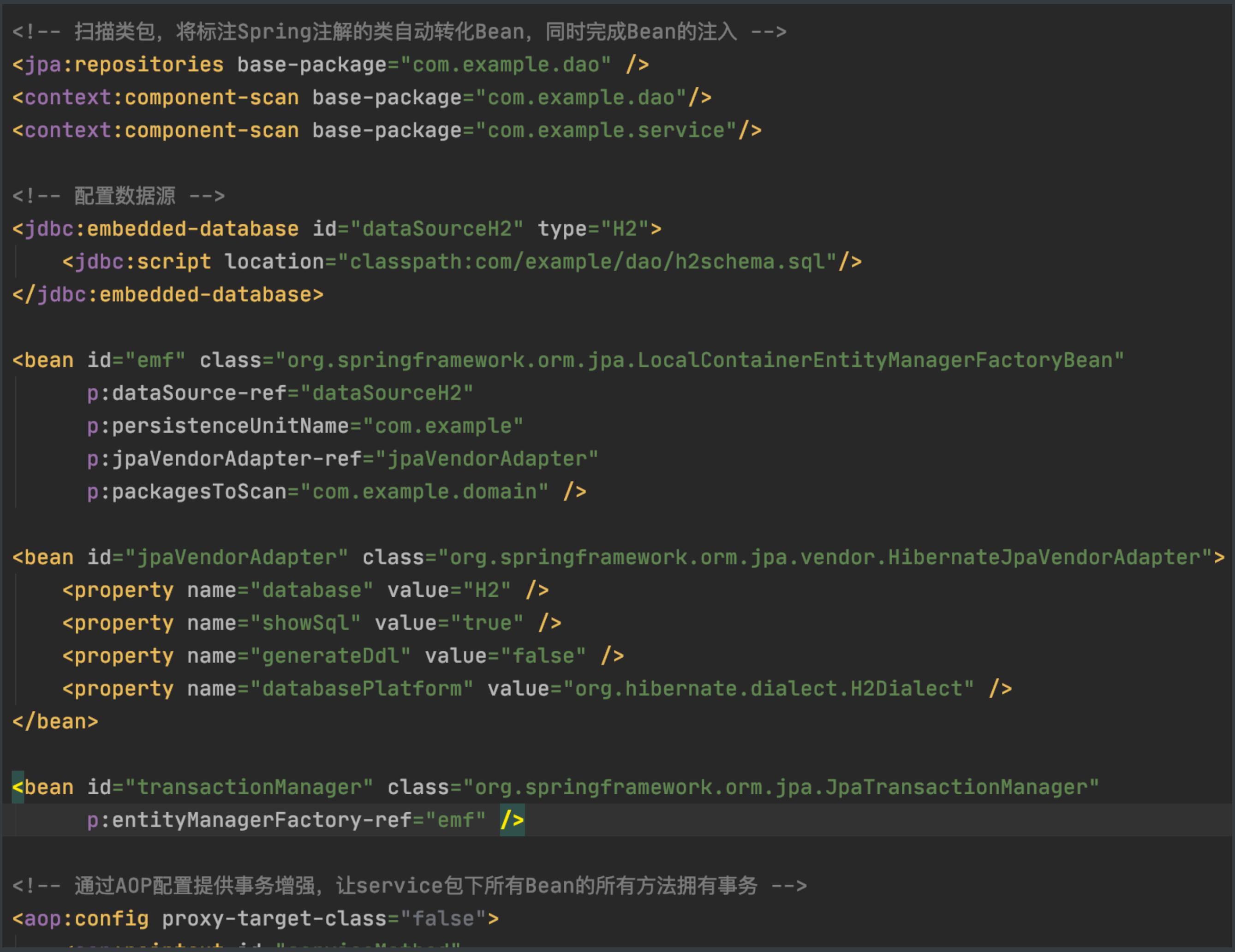


```xml
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>1.3.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.0.1.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.0.1.Final</version>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.182</version>
</dependency>
```

修改Dao层，增加Custom和Impl

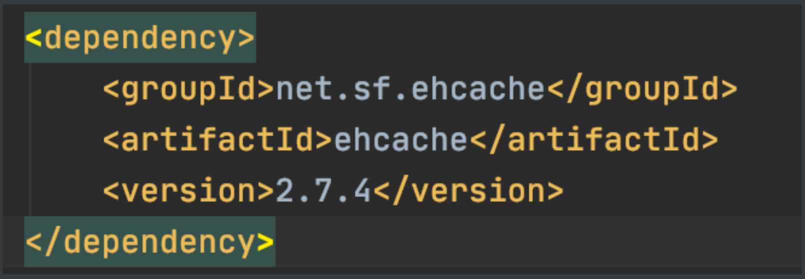
修改domain中的Bean对象，添加注释

修改applicationContext.xml（注意细节）



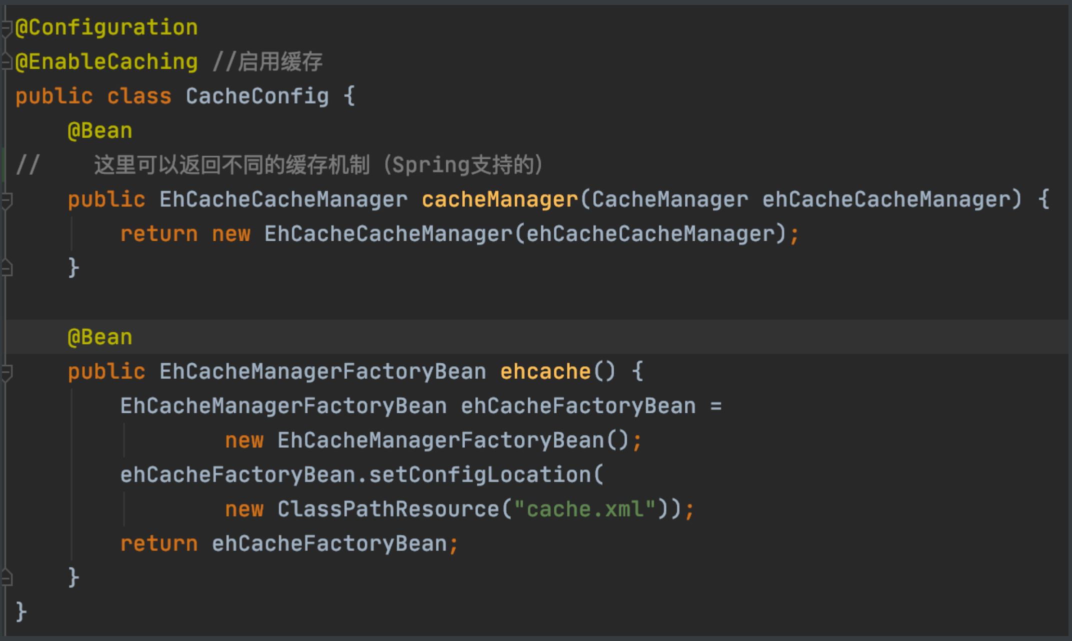
```xml
<!-- 扫描类包，将标注Spring注解的类自动转化Bean，同时完成Bean的注入 -->
<jpa:repositories base-package="com.example.dao" />
<context:component-scan base-package="com.example.dao"/>
<context:component-scan base-package="com.example.service"/>

<!-- 配置数据源 -->
<jdbc:embedded-database id="dataSourceH2" type="H2">
    <jdbc:script location="classpath:com/example/dao/h2schema.sql"/>
</jdbc:embedded-database>

<bean id="emf" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
      p:dataSource-ref="dataSourceH2"
      p:persistenceUnitName="com.example"
      p:jpaVendorAdapter-ref="jpaVendorAdapter"
      p:packagesToScan="com.example.domain" />

<bean id="jpaVendorAdapter" class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <property name="database" value="H2" />
    <property name="showSql" value="true" />
    <property name="generateDdl" value="false" />
    <property name="databasePlatform" value="org.hibernate.dialect.H2Dialect" />
</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager"
      p:entityManagerFactory-ref="emf" />

<!-- 通过AOP配置提供事务增强，让service包下所有Bean的所有方法拥有事务 -->
<aop:config proxy-target-class="false">
```

4. 添加依赖



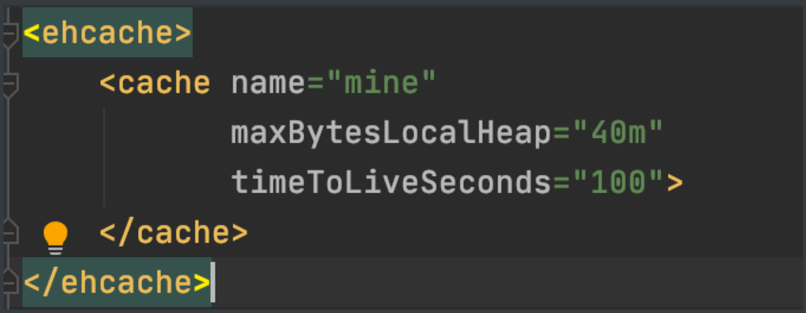
```xml
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
    <version>2.7.4</version>
</dependency>
```

增加CacheConfig配置文件



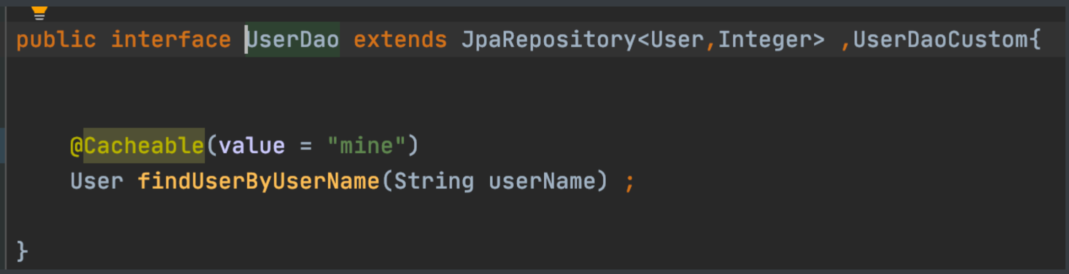
```java
@Configuration
@EnableCaching //启用缓存
public class CacheConfig {
    @Bean
//    这里可以返回不同的缓存机制（Spring支持的）
    public EhCacheCacheManager cacheManager(CacheManager ehCacheCacheManager) {
        return new EhCacheCacheManager(ehCacheCacheManager);
    }

    @Bean
    public EhCacheManagerFactoryBean ehcache() {
        EhCacheManagerFactoryBean ehCacheFactoryBean =
                new EhCacheManagerFactoryBean();
        ehCacheFactoryBean.setConfigLocation(
                new ClassPathResource("cache.xml"));
        return ehCacheFactoryBean;
    }
}
```

添加ehcache.xml



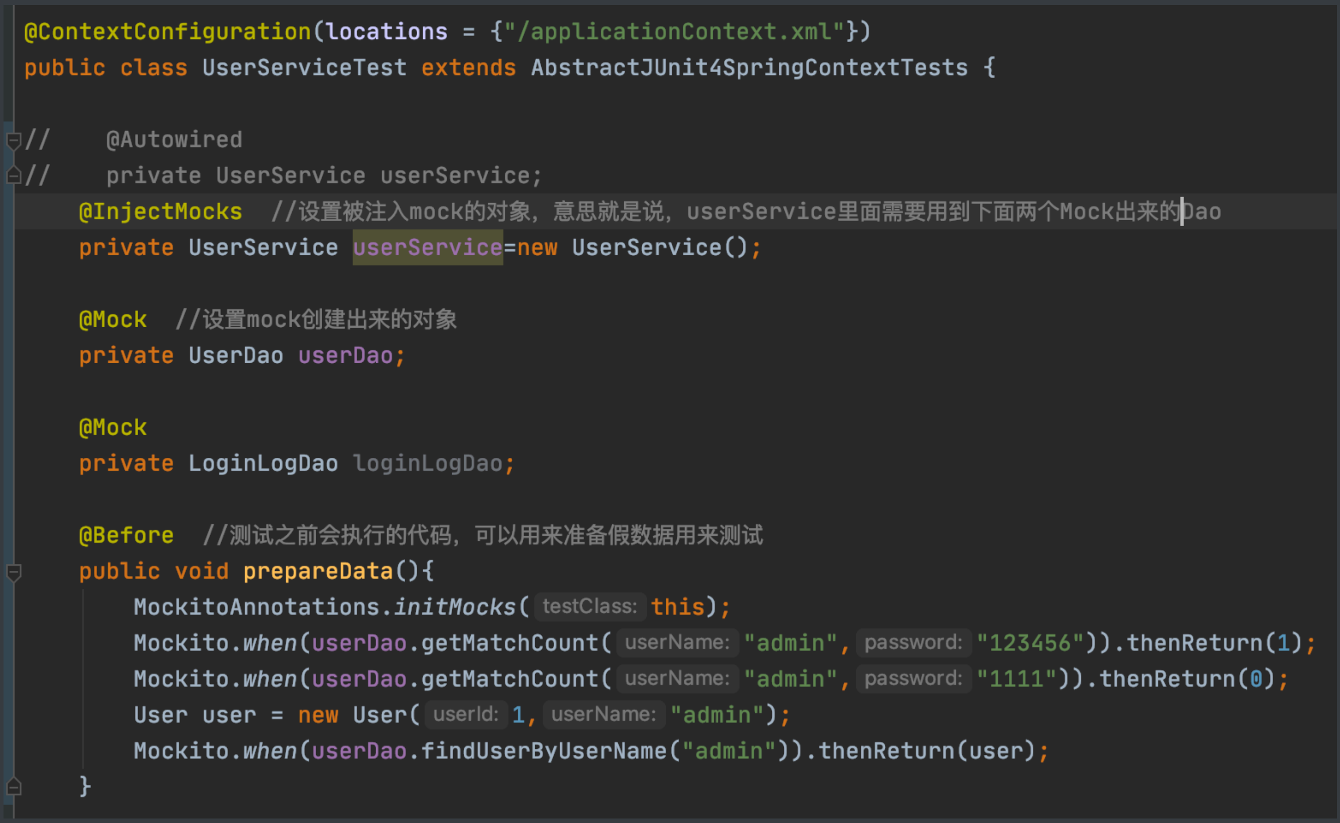
```xml
<ehcache>
    <cache name="mine"
           maxBytesLocalHeap="40m"
           timeToLiveSeconds="100">
    </cache>
</ehcache>
```

给对应的方法添加注释



```java
public interface UserDao extends JpaRepository<User,Integer> ,UserDaoCustom{


    @Cacheable(value = "mine")
    User findUserByUserName(String userName) ;

}
```

5. 修改UserServiceTest



```
@ContextConfiguration(locations = {"/applicationContext.xml"})
public class UserServiceTest extends AbstractJUnit4SpringContextTests {

//    @Autowired
//    private UserService userService;
    @InjectMocks  //设置被注入mock的对象，意思就是说，userService里面需要用到下面两个Mock出来的Dao
    private UserService userService=new UserService();

    @Mock  //设置mock创建出来的对象
    private UserDao userDao;

    @Mock
    private LoginLogDao loginLogDao;

    @Before  //测试之前会执行的代码，可以用来准备假数据用来测试
    public void prepareData(){
        MockitoAnnotations.initMocks( testClass: this);
        Mockito.when(userDao.getMatchCount( userName: "admin", password: "123456")).thenReturn(1);
        Mockito.when(userDao.getMatchCount( userName: "admin", password: "1111")).thenReturn(0);
        User user = new User( userId: 1, userName: "admin");
        Mockito.when(userDao.findUserByUserName("admin")).thenReturn(user);
    }
```