有道云链接：http://note.youdao.com/noteshare?id=d945e8d5b97ef56a5a94d7c824c0e4bd&sub=B6F11EBE3D05411B83E2F0E9973727E6

整合核心思路

由很多框架都需要和Spring进行整合，而整合的核心思想就是把其他框架所产生的对象放到Spring容器中，让其成为Bean。

比如Mybatis，Mybatis框架可以单独使用，而单独使用Mybatis框架就需要用到Mybatis所提供的一些类构造出对应的对象，然后使用该对象，就能使用到Mybatis框架给我们提供的功能，和Mybatis整合Spring就是为了将这些对象放入Spring容器中成为Bean，只要成为了Bean，在我们的Spring项目中就能很方便的使用这些对象了，也就能很方便的使用Mybatis框架所提供的功能了。

Mybatis-Spring 1.3.2版本底层源码执行流程

1. 通过@MapperScan导入了MapperScannerRegistrar类
2. MapperScannerRegistrar类实现了ImportBeanDefinitionRegistrar接口，所以Spring在启动时会调用MapperScannerRegistrar类中的registerBeanDefinitions方法
3. 在registerBeanDefinitions方法中定义了一个ClassPathMapperScanner对象，用来扫描mapper
4. 设置ClassPathMapperScanner对象可以扫描到接口，因为在Spring中是不会扫描接口的
5. 同时因为ClassPathMapperScanner中重写了isCandidateComponent方法，导致isCandidateComponent只会认为接口是备选者Component
6. 通过利用Spring的扫描后，会把接口扫描出来并且得到对应的BeanDefinition
7. 接下来把扫描得到的BeanDefinition进行修改，把BeanClass修改为MapperFactoryBean，把AutowireMode修改为byType
8. 扫描完成后，Spring就会基于BeanDefinition去创建Bean了，相当于每个Mapper对应一个FactoryBean
9. 在MapperFactoryBean中的getObject方法中，调用了getSqlSession()去得到一个sqlSession对象，然后根据对应的Mapper接口生成一个Mapper接口代理对象，这个代理对象就成为Spring容器中的Bean
10. sqlSession对象是Mybatis中的，一个sqlSession对象需要SqlSessionFactory来产生
11. MapperFactoryBean的AutowireMode为byType，所以Spring会自动调用set方法，有两个set方法，一个setSqlSessionFactory，一个setSqlSessionTemplate，而这两个方法执行的前提是根据方法参数类型能找到对应的bean，所以Spring容器中要存在SqlSessionFactory类型的bean或者SqlSessionTemplate类型的bean。
12. 如果你定义的是一个SqlSessionFactory类型的bean，那么最终也会被包装为一个SqlSessionTemplate对象，并且赋值给sqlSession属性
13. 而在SqlSessionTemplate类中就存在一个getMapper方法，这个方法中就产生一个Mapper接口代理对象
14. 到时候，当执行该代理对象的某个方法时，就会进入到Mybatis框架的底层执行流程，详细的请看下图

Spring整合Mybatis之后SQL执行流程：
https://www.processon.com/view/link/6152cc385653bb6791db436c

Mybatis-Spring 2.0.6版本(最新版)底层源码执行流程

1. 通过@MapperScan导入了MapperScannerRegistrar类
2. MapperScannerRegistrar类实现了ImportBeanDefinitionRegistrar接口，所以Spring在启动时会调用MapperScannerRegistrar类中的registerBeanDefinitions方法
3. **在registerBeanDefinitions方法中注册一个MapperScannerConfigurer类型的BeanDefinition**
4. 而MapperScannerConfigurer实现了BeanDefinitionRegistryPostProcessor接口，所以Spring在启动过程中时会调用它的postProcessBeanDefinitionRegistry()方法
5. 在postProcessBeanDefinitionRegistry方法中会生成一个ClassPathMapperScanner对象，然后进行扫描
6. 后续的逻辑和1.3.2版本一样。

带来的好处是，可以不使用@MapperScan注解，而可以直接定义一个Bean，比如：

```
@Bean
public MapperScannerConfigurer mapperScannerConfigurer() {
    MapperScannerConfigurer mapperScannerConfigurer = new MapperScannerConfigurer();
    mapperScannerConfigurer.setBasePackage("com.luban");
    return mapperScannerConfigurer;
}
```

Spring整合Mybatis后一级缓存失效问题

先看下图： Spring整合Mybatis之后SQL执行流程：
https://www.processon.com/view/link/6152cc385653bb6791db436c

Mybatis中的一级缓存是基于SqlSession来实现的，所以在执行同一个sql时，如果使用的是同一个SqlSession对象，那么就能利用到一级缓存，提高sql的执行效率。

但是在Spring整合Mybatis后，如果没有执行某个方法时，该方法上没有加@Transactional注解，也就是没有开启Spring事务，那么后面在执行具体sql时，没执行一个sql时都会新生成一个SqlSession对象来执行该sql，这就是我们说的一级缓存失效（也就是没有使用同一个SqlSession对象），而如果开启了Spring事务，那么该Spring事务中的多个sql，在执行时会使用同一个SqlSession对象，从而一级缓存生效，具体的底层执行流程在上图。

个人理解：实际上Spring整合Mybatis后一级缓存失效并**不是问题**，是正常的实现，因为，一个方法如果没有开启Spring事务，那么在执行sql时候，那就是每个sql单独一个事务来执行，也就是单独一个SqlSession对象来执行该sql，如果开启了Spring事务，那就是多个sql属于同一个事务，那自然就应该用一个SqlSession来执行这多个sql。所以，在没有开启Spring事务的时候，SqlSession的一级缓存并不是**失效**了，而是存在的生命周期太短了（执行完一个sql后就被销毁了，下一个sql执行时又是一个新的SqlSession了）。