一、三种注入方式

1、有参构造器（有循环依赖的问题）

2、Set

3、注解（先通过无参构造器创建实例，通过反射填充属性）

二、循环依赖

1、有参构造器注入的没法解决

2、<https://blog.csdn.net/chejinqiang/article/details/80003868>

<http://www.imooc.com/article/34150>



三、单例里获取多例

1、在bean A中引入ApplicationContext每次调用方法时用上下文的getBean(name,class)方法去重新获取bean B的实例。



2、使用@Lookup注解。

3、<https://somefuture.iteye.com/blog/2404846>

1. @Scope(value = ConfigurableBeanFactory.SCOPE\_PROTOTYPE, proxyMode = ScopedProxyMode.TARGET\_CLASS)

四、controller层的request使用@autowired没有线程安全问题

<https://blog.csdn.net/forlovedoit/article/details/53204667>

我们在Controller层中所注入的HttpServletRequest其实是一个JDK动态代理生成的对象

五、JDK动态代理和CGLIB字节码生成的区别

JDK动态代理只能对实现了接口的类生成代理，而不能针对类。

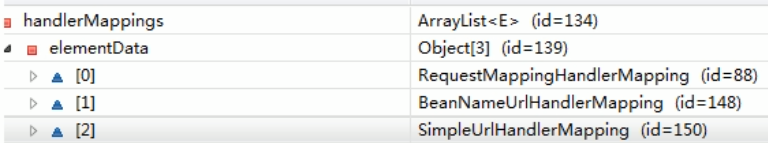
CGLIB是针对类实现代理，主要是对指定的类生成一个子类，覆盖其中的方法，并覆盖其中方法实现增强，但是因为采用的是继承，所以该类或方法最好不要声明成final，对于final类或方法，是无法继承的

AOP 全称是 Aspect Oriented Programming，即面向切面的编程，AOP 是一种开发理念。通过 AOP，我们可以把一些非业务逻辑的代码，比如安全检查，监控等代码从业务方法中抽取出来，以非侵入的方式与原方法进行协同。这样可以使原方法更专注于业务逻辑，代码结构会更加清晰，便于维护。

SpringMVC源码：

<https://blog.csdn.net/gududedabai/article/details/83352106>

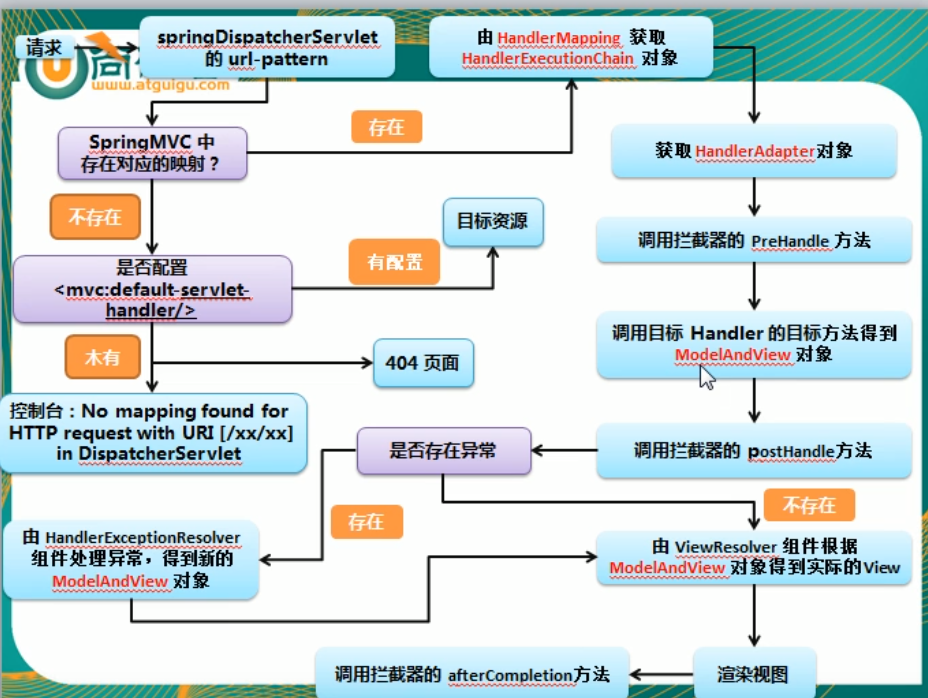
<https://blog.csdn.net/gududedabai/article/details/83375156>



handlerMappings（xml、注解、静态请求）

< url-pattern > / </ url-pattern > 不会匹配到\*.jsp，即：\*.jsp不会进入spring的 DispatcherServlet类 。

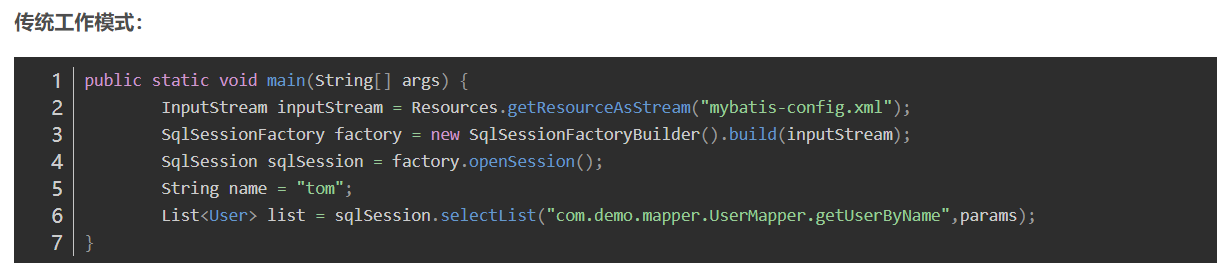
< url-pattern > /\* </ url-pattern > 会匹配\*.jsp，会出现返回jsp视图时再次进入spring的



InternalResourceViewResolver就是在返回结果前后加上/。。。。jsp，然后再forword转发

Mybatis流程：

<https://blog.csdn.net/weixin_43184769/article/details/91126687>



1. 创建SqlSessionFactoryBuilder对象，读入xml配置文件生成Configuration对象（将解析的xml数据封装到Configuration内部的属性中），然后创建SqlSessionFactory。
2. Configuration中包含属性mappedStatements，解析mapper配置文件时，会再将对应配置文件中的增删改查标签一 一封装成MappedStatement对象，存入mappedStatements中

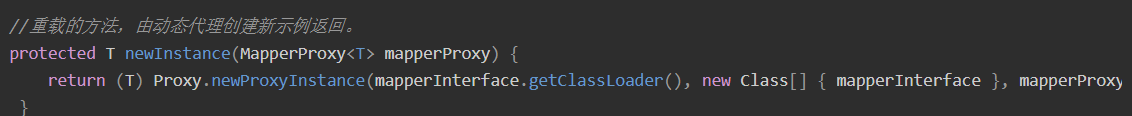


1. SqlSessionFactory创建SqlSession，SqlSession是MyBatis中用于和数据库交互的顶层类，通常将它与ThreadLocal绑定，一个会话使用一个SqlSession，并且在使用完毕后需要close。SqlSession中的两个最重要的参数，configuration与初始化时的相同，Executor为执行器。
2. SqlSession先获取MappedStatement，然后调用Executor中的方法处理sql，首先根据传入的参数动态获得SQL语句，最后返回用BoundSql对象表示，然后查看缓存中有没结果，如果没有就去数据库查询，查到后将结果放入缓存。查询的过程中会从连接池获取一个connection，然后创建jdbc中的prepareStatement对象，prepareStatement.execute()进行查询。

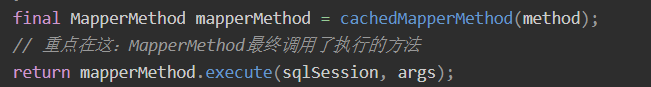
使用接口：获取的是mapper接口的动态代理，初始化mybatis时会为每个接口生成一个代理工厂，通过这些工厂产生接口代理对象。执行时，根据调用的方法method找到对应的mapper文件中的标签和详细信息，知道他是增删改查中的哪一个，然后执行SqlSession对应的方法（获取MappedStatement，参数合并成动态sql，连接池获取连接，statement.execute（））



1. MapperRegistry是Configuration中的一个属性，它内部维护一个HashMap用于存放mapper接口的工厂类，每个接口对应一个工厂类。MyBatis初始化解析到接口时，会创建此接口对应的MapperProxyFactory对象，存入HashMap中，key = 接口的字节码对象，value = 此接口对应的MapperProxyFactory对象。
2. sqlSession调用getMapper方法获取Mapper接口的代理对象。该方法最终调用的是MapperRegistry的getMapper方法。
3. MapperRegistry的getMapper方法先获取MapperRegistry里的一个代理工厂，然后调用MapperProxyFactory的newInstance(sqlSession)方法，该方法调用newInstance(mapperProxy)方法，其中mapperProxy是个InvocationHandler



1. invoke方法：根据接口需要执行的方法method生成mapperMethod



5、execute（）方法：根据增删改查调用SQLSession对应的方法。