整合Mybatis的核心思路

由于很多框架都需要和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的容器中,让它们成为Bean.
比如在Mybatis中,Mybatis框架可以单独使用,而单独使用Mybatis框架就需要用到Mybatis所提供一些类构造出对应的对象,然后使用对象,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的容器中成为Bean,只要成为了Bean,在我们的Spring项目中就能很方便的使用到这些对象了,也就能很方便的使用Mybatis框架所提供的功能了.

需要进行的步骤

- 我们需要指定路径下扫描我们需要的Mapper接口,所以我们可以想到重写ImportBeanDefinitionRegistrar,通过重写它的方法,获取到我们自定义@MapperScan的属性值,去拿到需要扫描的路径,然后将我们扫描到的类都添加到includeFilter中去,让它们都成为Bean对象,谈到这里就是我们包扫描的逻辑了,只有在includeFilters里的才会是Bean对象
- 我们需要将指定路径下的Mapper接口扫描到并且放到BeanDefinitionMap里面去,到这里可以想到我们可以使用ClassPathBeanDefinitionScanner,我们可以选择重写它的部分逻辑,首先是我们需要的是Mapper接口成为Bean,那么我们需要在重写isCandidateComponent,这是阻碍Mapper接口成为Bean的条件所以需要重写,其次我们需要重写scan方法,其实际是重写doScan方法,这里要考虑到我们先扫描完得到所有的BeanDefinitionHolder之后了,我们需要指定我们需要转换的对象类型,以及指定beanDefinition的类型,这里需要到下一步就明白了
- 我们自定义一个FactoryBean,指定属性mapperInterface用于传入指定接口,后续对应这个接口,使用jdk动态代理,为在service层的Mapper生成一个代理对象放到Spring容器中,还有就是我们需要执行对应的方法,这里可以考虑直接使用SqlSession去完成,而SqlSession又不是Spring的Bean对象,所以我们得提前把它注册成为一个Bean对象.对于这里我们可以考虑使用两种方式去处理,可以通过推断构造方式根据类型去注入AUTOWIRE_BY_TYPE,或者直接通过@Autowired先byType再byName

手写模拟Spring整合Mybatis

```
public interface UserMapper {
     @Select("select 'user'")
     String selectById();
  }
```

```
public interface OrderMapper {
     @Select("select 'user'")
     String selectById();
  }
```

```
  @Component
  public class OrderService {
     /**

     * Mybatis UserMapper代理对象-->Bean
         */
       @Autowired
       private UserMapper userMapper;

    public void test() {
       System.out.println(userMapper.selectById());
    }
}
```

```
@Component
public class UserService {
    /**

    * Mybatis UserMapper代理对象-->Bean
      */
         @Autowired
         private UserMapper userMapper;
         @Autowired
         private OrderMapper orderMapper;

    public void test() {
       System.out.println(userMapper.selectById());
       System.out.println(orderMapper.selectById());
    }
}
```

```
public class MybatisFactoryBean implements FactoryBean {
    private Class<?> mapperInterface;
    private SqlSession sqlSession;

    public MybatisFactoryBean(Class<?> mapperInterface) {
       this.mapperInterface = mapperInterface;
    }

    @Autowired
    public void setSqlSession(SqlSessionFactory sqlSessionFactory) {
       sqlSessionFactory.getConfiguration().addMapper(mapperInterface);
       this.sqlSession = sqlSessionFactory.openSession();
    }


    @Override
    public Object getObject() throws Exception {
       return sqlSession.getMapper(mapperInterface);
```

```
      //    return Proxy.newProxyInstance(this.getClass().getClassLoader(), new Class[]
{mapperInterface}, (proxy, method, args) -> {
      //       System.out.println(method.getName() + "");
      //       return null;
      //    });

   }

   @Override
   public Class<?> getObjectType() {
      return mapperInterface;
   }
}
```

```
public class MyBeanDefinitionScanner extends ClassPathBeanDefinitionScanner {
   public MyBeanDefinitionScanner(BeanDefinitionRegistry registry) {
      super(registry);
   }

   @Override
   protected boolean isCandidateComponent(AnnotatedBeanDefinition beanDefinition) {
      return beanDefinition.getMetadata().isInterface();
   }

   @Override
   protected Set<BeanDefinitionHolder> doScan(String... basePackages) {
      Set<BeanDefinitionHolder> beanDefinitionHolders = super.doScan(basePackages);
      for (BeanDefinitionHolder beanDefinitionHolder : beanDefinitionHolders) {
         BeanDefinition beanDefinition = beanDefinitionHolder.getBeanDefinition();

beanDefinition.getConstructorArgumentValues().addGenericArgumentValue(beanDefinition.getB
eanClassName());
         beanDefinition.setBeanClassName(MybatisFactoryBean.class.getName());
      }
      return beanDefinitionHolders;
   }
}
```

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Import({MyMybatisImportBeanDefinitionRegistry.class})
public @interface MyMapperScan {
   String value() default "com.qf.mapper";
}
```

```
@Component
public class MyMybatisImportBeanDefinitionRegistry implements
ImportBeanDefinitionRegistrar {
```

```
    @Override
    public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata,
BeanDefinitionRegistry registry, BeanNameGenerator importBeanNameGenerator) {
        // 扫描路径
        Map<String, Object> annotationAttributes =
importingClassMetadata.getAnnotationAttributes(MyMapperScan.class.getName());
        String path = (String) annotationAttributes.get("value");
        MyBeanDefinitionScanner scanner = new MyBeanDefinitionScanner(registry);
        // 重写spring的扫描逻辑
        scanner.addIncludeFilter((metadataReader, metadataReaderFactory) -> true);
        scanner.scan(path);
    }
}
```

```
@Configuration
@ComponentScan("com.qf")
@MyMapperScan
public class AppConfig {
    @Bean
    public SqlSessionFactory sqlSessionFactory() throws IOException {
        InputStream inputStream = Resources.getResourceAsStream("mybatis.xml");
        return new SqlSessionFactoryBuilder().build(inputStream);
    }
}
```

```
public class Main {

    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext();
        context.register(AppConfig.class);
        context.refresh();
        UserService userService = (UserService) context.getBean("userService");
        userService.test();
    }
}
```

Mybatis底层源码执行流程

- 通过@MapperScan导入了MapperScannerRegistrar类
- 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
- 在registerBeanDefinitions方法中定义了一个ClassPathMapperScanner对象,用来扫描Mapper接口
- 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
- 同时因为ClassPathMapperScanner中重写了isCandidateComponent方法
- 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
- 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

- 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
- 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了Spring容器中的Bean
- sqlSession对象是Mybatis中的，一个sqlSession对象需要SqlSessionFactory来产生
- 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。
- 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
- 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
- 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