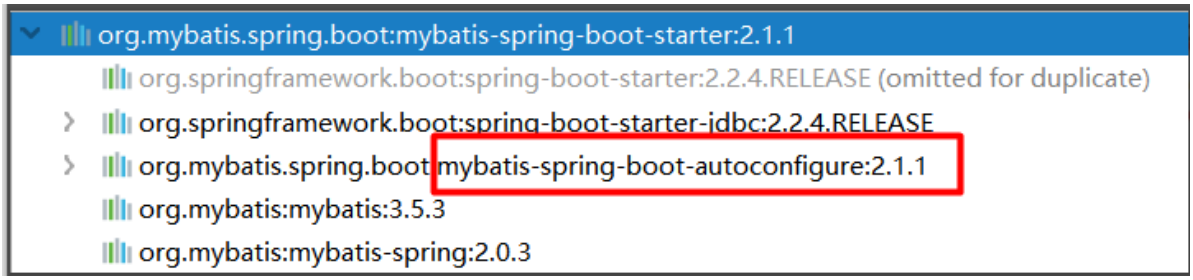
【第34话：这个流程有点长Spring Boot自定义启动器流程】

Hello 小伙伴们，这节课给大家讲解一下“Spring Boot自定义启动器流程”。

既然问题是问我们如何自定义一个启动器，最好的办法就是先看看别人如何编写启动器。

以mybatis-spring-boot-starter举例，让小伙伴们看看Apache是如何自定义启动器的。

首先，看一下启动的依赖。每个启动器都会有个自动化配置的依赖，这个依赖都是把启动器名称中starter换成autoconfigure。mybatis-spring-boot-starter的依赖就是mybatis-spring-boot-autoconfigure



接下来在mybatis-spring-boot-autoconfigure中/META-INF/spring.factories文件中看一下，发现MyBatisAutoConfiguration类时自动化配置类。

```
# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.mybatis.spring.boot.autoconfigure.MybatisLanguageDriverAutoConfiguration,\
org.mybatis.spring.boot.autoconfigure.MybatisAutoConfiguration
```

进入到MyBatisAutoConfiguration，类上面配置了自动化配置条件的注解。比较多，我们一个一个解释。

```
@Configuration
@ConditionalOnClass({SqlSessionFactory.class, SqlSessionFactoryBean.class})
@ConditionalOnSingleCandidate(DataSource.class)
@EnableConfigurationProperties({MybatisProperties.class})
@AutoConfigureAfter({DataSourceAutoConfiguration.class, MybatisLanguageDriverAutoConfiguration.class})
public class MybatisAutoConfiguration implements InitializingBean {
    private static final Logger logger = LoggerFactory.getLogger(MybatisAutoConfiguration.class);
    private final MybatisProperties properties;
    private final Interceptor[] interceptors;
    private final TypeHandler[] typeHandlers;
    private final LanguageDriver[] languageDrivers;
    private final ResourceLoader resourceLoader;
    private final DatabaseIdProvider databaseIdProvider;
    private final List<ConfigurationCustomizer> configurationCustomizers;
```

@ConditionalOnBean// 当给定的在bean存在时,则实例化当前Bean

@ConditionalOnMissingBean// 当给定的在bean不存在时,则实例化当前Bean

@ConditionalOnClass// 当给定的类名在类路径上存在，则实例化当前Bean

@ConditionalOnMissingClass// 当给定的类名在类路径上不存在，则实例化当前Bean

@ConditionalOnSingleCandidate表示当指定Bean在容器中只有一个，或者虽然有多个但是指定首选Bean

@ConfigurationProperties注解的beans将自动被Environment属性配置

@AutoConfigureAfter 在加载配置的类之后再加载当前类

在查看MybatisProperties类，这个类中包含了启动器导入后所支持的所有配置。

@ConfigurationProperties中prefix表示所有的配置都是以mybatis开头。

```
@ConfigurationProperties(
    prefix = "mybatis"
)
public class MybatisProperties {
    public static final String MYBATIS_PREFIX = "mybatis";
    private static final ResourcePatternResolver resourceResolver = new PathMatchingResourcePatternResolver();
    private String configLocation;
    private String[] mapperLocations;
    private String typeAliasesPackage;
    private Class<?> typeAliasesSuperType;
    private String typeHandlersPackage;
    private boolean checkConfigLocation = false;
    private ExecutorType executorType;
    private Class<? extends LanguageDriver> defaultScriptingLanguageDriver;
    private Properties configurationProperties;
    @NestedConfigurationProperty
    private Configuration configuration;
```

继续看MyBatisAutoConfiguration中提供了两个Bean

SqlSessionFactory的Bean

```
@Bean
@ConditionalOnMissingBean
public SqlSessionFactory sqlSessionFactory(DataSource dataSource) throws Exception {
  SqlSessionFactoryBean factory = new SqlSessionFactoryBean();
  factory.setDataSource(dataSource);
```

SqlSessionTemplate的Bean

```
@Bean
@ConditionalOnMissingBean
public SqlSessionTemplate sqlSessionTemplate(SqlSessionFactory sqlSessionFactory) {
  ExecutorType executorType = this.properties.getExecutorType();
  if (executorType != null) {
    return new SqlSessionTemplate(sqlSessionFactory, executorType);
  } else {
    return new SqlSessionTemplate(sqlSessionFactory);
  }
}
```

提供了两个static的内部类AutoConfiguredMapperScannerRegistrar和MapperScannerRegistrarNotFoundConfiguration

提供了MapperScannerConfigurer扫描注解

```
if (propertyNames.contains("lazyInitialization")) {
  // Need to mybatis-spring 2.0.2+
  builder.addPropertyValue( name: "lazyInitialization",  value: "${mybatis.lazy-initialization:false}");
}
if (propertyNames.contains("defaultScope")) {
  // Need to mybatis-spring 2.0.6+
  builder.addPropertyValue( name: "defaultScope",  value: "${mybatis.mapper-default-scope:}");
}
registry.registerBeanDefinition(MapperScannerConfigurer.class.getName(), builder.getBeanDefinition());
```

所以总结起来自动装配内部流程：

1. 启动类中@SpringBootApplication中包含@EnableAutoConfiguration
2. 加载项目依赖的META-INF/spring.factories中EnableAutoConfiguration对应类，进行实例化。
3. 实例化自动化配置类时会同时加载属性类，从配置文件中读取。
4. 生效自动化配置，把对应的Bean放入到Spring容器中。