spriongboot 核心流程

Spring @Configure注解 的加载处理

```
入口 :
    1. org.springframework.boot.autoconfigure.SpringBootApplication
    2. org.springframework.boot.autoconfigure.EnableAutoConfiguration
    3. org.springframework.boot.autoconfigure.AutoConfigurationImportSelector
        负责通过 spring.factories 的配置加载 Configure
    4. org.springframework.boot.autoconfigure.AutoConfigurationImportFilter
        Configure 的过滤接口
```

```java
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
        @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
        @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {

    /**
     * Exclude specific auto-configuration classes such that they will never be applied.
     * @return the classes to exclude
     */
    @AliasFor(annotation = EnableAutoConfiguration.class)
    Class<?>[] exclude() default {};

    /**
     * Exclude specific auto-configuration class names such that they will never be
     * applied.
     * @return the class names to exclude
     * @since 1.3.0
     */
    @AliasFor(annotation = EnableAutoConfiguration.class)
    String[] excludeName() default {};
```

```
 * Auto-configuration classes are regular Spring {@link Configuration} beans. They are
 * located using the {@link SpringFactoriesLoader} mechanism (keyed against this class).
 * Generally auto-configuration beans are {@link Conditional @Conditional} beans (most
 * often using {@link ConditionalOnClass @ConditionalOnClass} and
 * {@link ConditionalOnMissingBean @ConditionalOnMissingBean} annotations).
 *
 * @author Phillip Webb
 * @author Stephane Nicoll
 * @see ConditionalOnBean
 * @see ConditionalOnMissingBean
 * @see ConditionalOnClass
 * @see AutoConfigureAfter
 * @see SpringBootApplication
 */
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import(AutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {

    String ENABLED_OVERRIDE_PROPERTY = "spring.boot.enableautoconfiguration";
```

AutoConfigurationImportSelector

```
@Override
public String[] selectImports(AnnotationMetadata annotationMetadata) {
    if (!isEnabled(annotationMetadata)) {
        return NO_IMPORTS;
    }
    AutoConfigurationMetadata autoConfigurationMetadata = AutoConfigurationMetadataLoader
            .loadMetadata(this.beanClassLoader);
    AnnotationAttributes attributes = getAttributes(annotationMetadata);
    List<String> configurations = getCandidateConfigurations(annotationMetadata,
            attributes);
    configurations = removeDuplicates(configurations);
    Set<String> exclusions = getExclusions(annotationMetadata, attributes);
    checkExcludedClasses(configurations, exclusions);
    configurations.removeAll(exclusions);
    configurations = filter(configurations, autoConfigurationMetadata);
    fireAutoConfigurationImportEvents(configurations, exclusions);
    return StringUtils.toStringArray(configurations);
}
```

1. getCandidateCongiguretion() 使用 springFactoriesLoader 加载所有需要初始化的bean
2. filter() 加载 AutoConfigurationImportFilter (也是通过 springFactorieLoader 加载的),
用filter.match 对 所有类进行判断， 这里主要是用于 conditonOn*** 注释

spring factories loader

1. 类加载器，
2. classpath
3. FACTORIES_RESOURCE_LOCATION:META-INF/spring.factories
4. 截图中包含了核心函数

```
private static Map<String, List<String>> loadSpringFactories(@Nullable ClassLoader classLoader) {
    MultiValueMap<String, String> result = cache.get(classLoader);
    if (result != null) {
        return result;
    }

    try {
        Enumeration<URL> urls = (classLoader != null ?
                classLoader.getResources(FACTORIES_RESOURCE_LOCATION) :
                ClassLoader.getSystemResources(FACTORIES_RESOURCE_LOCATION));
        result = new LinkedMultiValueMap<>();
        while (urls.hasMoreElements()) {
            URL url = urls.nextElement();
            UrlResource resource = new UrlResource(url);
            Properties properties = PropertiesLoaderUtils.loadProperties(resource);
            for (Map.Entry<?, ?> entry : properties.entrySet()) {
                List<String> factoryClassNames = Arrays.asList(
                        StringUtils.commaDelimitedListToStringArray((String) entry.getValue()));
                result.addAll((String) entry.getKey(), factoryClassNames);
            }
        }
        cache.put(classLoader, result);
        return result;
    }
    catch (IOException ex) {
        throw new IllegalArgumentException("Unable to load factories from location [" +
```

Conditional

@Conditional(OnClassCondition.class) OnClassCondition extend org.springframework.boot.autoconfigure.condition.SpringBootCondition 提供判断接口 mathchs, getMatchOutcome 等等函数，进行判断是否匹配

应用启动

```java
@MapperScan(basePackages = "com.netease.nieweb.cupid.mapper")
@SpringBootApplication
//@SpringBootApplication(scanBasePackages = { "com.netease.nieweb.cupid","com.netease.
@EnableScheduling
@EnableAsync
@EnableAspectJAutoProxy
@EnableDiscoveryClient
@EnableFeignClients
public class Main {

    /**
     * The entry of the application
     *
     * @param args
     * @throws Exception
     */
    public static void main(String[] args) throws Exception {
        SpringApplication.run(Main.class, args);
    }

}
```

org.springframework.boot.SpringApplication#run(java.lang.String...)
然后就是进入普通的spring生命周期

spring boot 与 spring

org.springframework.context.annotation.ImportSelector
springboot 注解SpringBootApplication ， 引入了 AutoConfigurationImportSelector<继承于ImportSelector>

关于import 与 importSelector 的作用详见 spring.md

spring 容器的初始化会 加载所有实现了 ImportSelector 接口的类，对接口返回的List<String>

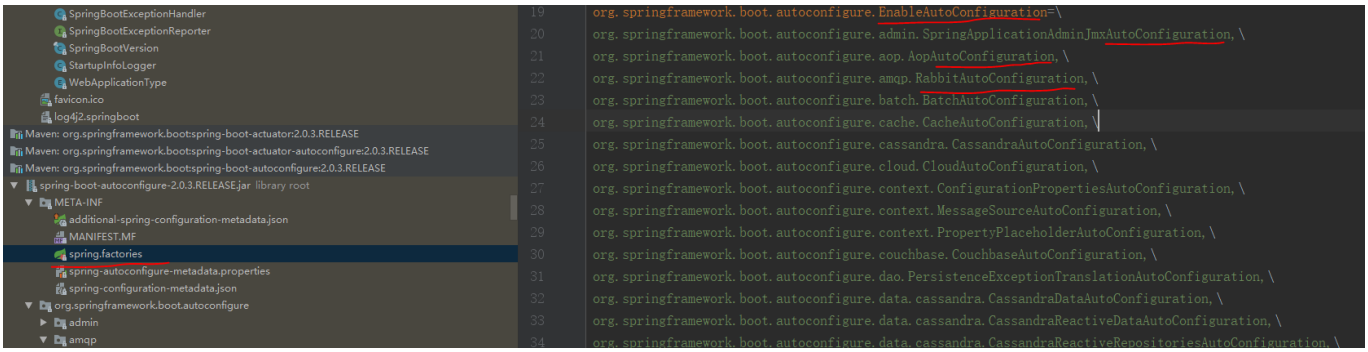
```java
public void refresh() throws BeansException, IllegalStateException {
    synchronized (this.startupShutdownMonitor) {
        // Prepare this context for refreshing.
        prepareRefresh();

        // Tell the subclass to refresh the internal bean factory.
        ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();

        // Prepare the bean factory for use in this context.
        prepareBeanFactory(beanFactory);

        try {
            // Allows post-processing of the bean factory in context subclasses.
            postProcessBeanFactory(beanFactory);

            // Invoke factory processors registered as beans in the context.
            invokeBeanFactoryPostProcessors(beanFactory);

            // Register bean processors that intercept bean creation.
            registerBeanPostProcessors(beanFactory);

            // Initialize message source for this context.
            initMessageSource();

            // Initialize event multicaster for this context.
            initApplicationEventMulticaster();

            // Initialize other special beans in specific context subclasses.
            onRefresh();
```

```
0.
org.springframework.context.support.AbstractApplicationContext#invokeBeanFactoryPo
stProcessors
0.
org.springframework.context.support.PostProcessorRegistrationDelegate#invokeBeanFa
ctoryPostProcessors(org.springframework.beans.factory.config.ConfigurableListableB
eanFactory,
java.util.List<org.springframework.beans.factory.config.BeanFactoryPostProcessor>)
0.
org.springframework.context.annotation.ConfigurationClassPostProcessor#processConf
igBeanDefinitions
0.
org.springframework.context.annotation.ConfigurationClassParser#processConfigurati
onClass
1.
org.springframework.context.annotation.ConfigurationClassParser#doProcessConfigura
tionClass
```

2. org.springframework.context.annotation.ConfigurationClassParser#processImports

ConfigurationClassParser#processImports 说明对于 ImportSelector 是以配置<@Configuration>的形式进行加载的， 因此 spring boot 的核心是 @Configuration 注解

下图中 ***AutoConfiguration 对应的类，都是 @Configuration 注解的类




```
 * @see EnableAspectJAutoProxy
 */
@Configuration
@ConditionalOnClass({ EnableAspectJAutoProxy.class, Aspect.class, Advice.class,
        AnnotatedElement.class })
@ConditionalOnProperty(prefix = "spring.aop", name = "auto", havingValue = "true", matchIfMissing = true)
public class AopAutoConfiguration {

    @Configuration
    @EnableAspectJAutoProxy(proxyTargetClass = false)
    @ConditionalOnProperty(prefix = "spring.aop", name = "proxy-target-class", havingValue = "false", matchIfMissing = fa
    public static class JdkDynamicAutoProxyConfiguration {

    }

    @Configuration
    @EnableAspectJAutoProxy(proxyTargetClass = true)
    @ConditionalOnProperty(prefix = "spring.aop", name = "proxy-target-class", havingValue = "true", matchIfMissing = tru
    public static class CglibAutoProxyConfiguration {

    }
```