1、@EnableCaching

@Import 的原理参见《Spring 注解》第 8 节

驱动注解@EnableXXX，在这个注解中@Import 具体实现，具体实现有以下三种：

- 基于 Configuration Class
- 基于 ImportSelector 接口实现
- 基于 ImportBeanDefinitionRegistrar 接口实现

@EnableCaching 的定义如下

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Import(CachingConfigurationSelector.class)
public @interface EnableCaching {

	boolean proxyTargetClass() default false;

	AdviceMode mode() default AdviceMode.PROXY;

	int order() default Ordered.LOWEST_PRECEDENCE;

}
```

可见是按照第二种方式使用@Import 的。**那么 CachingConfigurationSelector#selectImports 方法是何时调用的呢**？是在 SpringBoot 启动后刷新容器上下文的 refresh 方法发起的，通过 AbstractApplicationContext#invokeBeanFactoryPostProcessors 定制 BeanFactory 完成的，这个方法最终会调用处理

ConfigurationClassPostProcessor#processConfigBeanDefinitions 方法处理上下文中所有的 ConfigurationClass(Represents a user-defined @Configuration class.)。

大体上调用链如下：

invokeBeanFactoryPostProcessors -->

invokeBeanDefinitionRegistryPostProcessors -->

ConfigurationClassPostProcessor#processConfigBeanDefinitions -->

processConfigurationClass -->

processImports --> selectImports

处理过程如下，会使用 ConfigurationClassParser#getImports 方法得到配置类（研究的例子是 SpringBoot 的启动类）上所有@EnableXXX 注解上@Import 注解引入的类，对于缓存来说，处理 **CachingConfigurationSelector** 类，然后调用其 selectImports 方法，返回了两个用于缓存配置的类：AutoProxyRegistrar 和 ProxyCachingConfiguration。

CachingConfigurationSelector#processImports 方法

这个方法会处理一开始说的实现@EnableXXX 的三种方法中涉及到的类，即 ImportSelector、ImportBeanDefinitionRegistrar 和普通的 Configuration Class。其中处理 ImportSelector 类时是递归的，即 ImportSelector 可以注解 ImportSelector。最终递归到 Configuration Class 为止，再调用 ConfigurationClassParser#processConfigurationClass-->doProcessConfigurationClass 方法处理配置类。

在 doProcessConfigurationClass 中还会处理@PropertySource、@ComponentScan 和配置类上的@Import 引入的类；此外还有@ImportResource，然后处

理配置类中的@Bean 注解，把所有@Bean 注解的方法放入到 ConfigurationClass 的一个 Set 成员变量中。

最后会在 ConfigurationClassParser#processConfigurationClass 中把配置类都添加到 ConfigurationClassParser#configurationClasses 中（一个 Map）。

1.1、AutoProxyRegistrar#registerBeanDefinitions

这里给出启动类的代码

```
@SpringBootApplication(scanBasePackages = "io.kimmking.cache")
@MapperScan("io.kimmking.cache.mapper")
@EnableCaching
public class CacheApplication {

    public static void main(String[] args) {
        SpringApplication.run(CacheApplication.class, args);
    }

}
```

registerBeanDefinitions 执行的时候，会依次处理启动类上的三个注解类。因为 AutoProxyRegistrar 是用来处理注解@EnableCaching 的，所以会寻找相关的 mode 和 proxyTargetClass 属性，自然前 2 个注解会被跳过去。因为 AdviceMode.PROXY 是默认的选项，所以会调用 AopConfigUtils#registerAutoProxyCreatorIfNecessary 方法，在上下文中注册一个 internalAutoProxyCreator 的 BeanDefinition。

1.2、ProxyCachingConfiguration

通过层层代理，会调用这个类的方法，分别创建了 BeanFactoryCacheOperationSourceAdvisor、CacheOperationSource、CacheInterceptor 三个 Bean。这三个 BEAN 都是为 Spring 缓存抽象服务的。CacheOperationSource 的方法最有意思，一个实现 AnnotationCacheOperationSource 的注释如下：

Implementation of the CacheOperationSource interface for working with caching metadata in annotation format.

This class reads Spring's Cacheable, CachePut and CacheEvict annotations and exposes corresponding caching operation definition to Spring's cache infrastructure. This class may also serve as base class for a custom CacheOperationSource.

2、CacheAutoConfiguration 自动配置类

在 org\springframework\boot\spring-boot-autoconfigure\2.0.9.RELEASE\spring-boot-autoconfigure-2.0.9.RELEASE.jar!\META-INF\spring.factories 中有如下的和缓存自动配置相关的类

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,\
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration,\
```

CacheAutoConfiguration 上有若干相关注解，

```
@Configuration
@ConditionalOnClass(CacheManager.class)
@ConditionalOnBean(CacheAspectSupport.class)
```

```
@ConditionalOnMissingBean(value = CacheManager.class, name = "cacheResolver")
@EnableConfigurationProperties(CacheProperties.class)
@AutoConfigureAfter({ CouchbaseAutoConfiguration.class, HazelcastAutoConfiguration.class,
        HibernateJpaAutoConfiguration.class, RedisAutoConfiguration.class })
@Import(CacheConfigurationImportSelector.class)
public class CacheAutoConfiguration {
    ......
}
```

程序启动时，会执行 CacheConfigurationImportSelector#selectImports 方法，调起的过程和第一节 @EnableCaching 注解相关的 CachingConfigurationSelector#selectImports 调起的过程类似。

```
static class CacheConfigurationImportSelector implements ImportSelector {

    @Override
    public String[] selectImports(AnnotationMetadata importingClassMetadata) {
        CacheType[] types = CacheType.values();
        String[] imports = new String[types.length];
        for (int i = 0; i < types.length; i++) {
            imports[i] = CacheConfigurations.getConfigurationClass(types[i]);
        }
        return imports;
    }

}
```

其中 CacheType.values()中定义了支持的所有缓存类型

```
public enum CacheType {
    GENERIC,
    JCACHE,
    EHCACHE,
    HAZELCAST,
    INFINISPAN,
```

```
    COUCHBASE,
    REDIS,
    CAFFEINE,
    SIMPLE,
    NONE;

    private CacheType() {
    }
}
```

根据上面的得到所有的配置类，REDIS 的配置类为 org.springframework.boot.autoconfigure.cache.RedisCacheConfiguration，这个配置类在 CachingConfigurationSelector#processImports 方法中被处理（参见第 1 节的说明）。配置类中会定义一个 RedisCacheManager 类型的 bean。