Spring Boot中的SPI实现方式

Spring Boot中实现自动配置时，用到了SPI(Service Provider Interface)机制，其中核心是自动配置，依赖如下：

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-autoconfigure</artifactId>*

*<version>1.5.9.RELEASE</version>*

*</dependency>*

其中@EnableAutoConfiguration注解就是Spring Boot实现自动配置的关键。

# Java SPI

Java SPI(Service Provider Interface)是JDK内置的服务提供发现机制，其核心是动态代理机制，下面是SPI使用示例：

1. 定义接口及实现类

*public interface HelloInterface {*

*public void sayHello();*

*}*

*public class HelloMessage implements HelloInterface {*

*public void sayHello() {*

*System.out.println("Hello Message");*

*}*

*}*

*public class HelloService implements HelloInterface {*

*public void sayHello() {*

*System.out.println("Hello Service");*

*}*

*}*

1. 定义SPI配置文件，在目录META-INF/services中创建文件

*com.fys.springspi.HelloInterface， 内容如下：*

*com.fys.springspi.HelloMessage*

*com.fys.springspi.HelloService*

1. 测试程序

*public class SPIMain {*

*public static void main(String[] args) {*

*ServiceLoader<HelloInterface> loaders =*

*ServiceLoader.load(HelloInterface.class);*

*for(HelloInterface in : loaders) {*

*in.sayHello();*

*}*

*}*

*}*

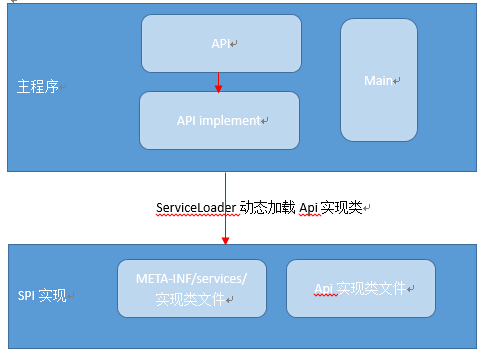
其输出如下：

*Hello Message*

*Hello Service*

1. 源码分析

JDK中查找服务的实现工具类是ServiceLoader，其动态加载META-INF中的实现类，其执行流程如下：



ServiceLoader加载源码如下：

*public void reload() {*

*providers.clear();*

*lookupIterator = new LazyIterator(service, loader);*

*}*

LazyIterator的初始化如下：

*private boolean hasNextService() {*

*....*

*if (configs == null) {*

*try {*

*String PREFIX = "META-INF/services/"*

*String fullName = PREFIX + service.getName();*

*if (loader == null)*

*configs = ClassLoader.getSystemResources(fullName);*

*else*

*configs = loader.getResources(fullName);*

*} ....*

*}*

在services目录中加载配置文件，然后获取接口对应类，核心方法为nextService

*private S nextService() {*

*String cn = nextName;*

*nextName = null;*

*Class<?> c = null;*

*try {*

*c = Class.forName(cn, false, loader);*

*S p = service.cast(c.newInstance()); //类构造器*

*providers.put(cn, p);*

*return p;*

*}*

*}*

在SPI中定义的接口通过Class.newInstance进行实例化，其有一定的使用局限，即生成对象只能调用无参的构造函数。

# Spring SPI

Spring Boot的SPI实质是通过EnableAutoConfigure自动从spring.factories中加载Bean到Spring 容器中，示例如下：

*org.springframework.boot.autoconfigure.EnableAutoConfiguration*

*=com.fys.springstarter.HelloAutoConfiguration*

EnableAutoConfiguration类的定义如下：

*@AutoConfigurationPackage*

*@Import(AutoConfigurationImportSelector.class)*

*public @interface EnableAutoConfiguration {*

*String ENABLED\_OVERRIDE\_PROPERTY = "spring.boot.enableautoconfiguration";*

*Class<?>[] exclude() default {};*

*String[] excludeName() default {};*

*}*

在@EnableAutoConfiguration上引入EnableAutoConfigurationImportSelector，其实现如下：

*public class AutoConfigurationImportSelector*

*implements DeferredImportSelector, BeanClassLoaderAware, ResourceLoaderAware,*

*BeanFactoryAware, EnvironmentAware, Ordered {*

*......*

*protected List<String> getCandidateConfigurations(AnnotationMetadata metadata,*

*AnnotationAttributes attributes) {*

*List<String> configurations = SpringFactoriesLoader.loadFactoryNames(*

*getSpringFactoriesLoaderFactoryClass(), getBeanClassLoader());*

*Assert.notEmpty(configurations,"No auto configuration classes found in*

*META-INF/spring.factories. "*

*return configurations;*

*}*

*......*

*}*

在Spring容器启动过程中，通过EnableAutoConfigurationImportSelector类导入下面的包：

*import org.springframework.core.io.support.SpringFactoriesLoader;*

其核心代码分析如下：

1. 默认加载路径，META-INF/spring.factories

*public static final String FACTORIES\_RESOURCE\_LOCATION =*

*"META-INF/spring.factories";*

*private static Map<String, List<String>> loadSpringFactories( ClassLoader classLoader) {*

*MultiValueMap<String, String> result = cache.get(classLoader);*

*try {*

*Enumeration<URL> urls = (classLoader != null ?*

*classLoader.getResources(FACTORIES\_RESOURCE\_LOCATION) :*

*ClassLoader.getSystemResources(FACTORIES\_RESOURCE\_LOCATION));*

*.....*

*}*

1. 从配置中，加载Bean类

*while (urls.hasMoreElements()) {*

*URL url = urls.nextElement();*

*UrlResource resource = new UrlResource(url);*

*Properties properties = PropertiesLoaderUtils.loadProperties(resource);*

*for (Map.Entry<?, ?> entry : properties.entrySet()) {*

*String factoryClassName = ((String) entry.getKey()).trim();*

*for (String factoryName : StringUtils.commaDelimitedListToStringArray((String)*

*entry.getValue())) {*

*result.add(factoryClassName, factoryName.trim());}}*

*}*

将工厂类与实现类的对应关系放到Map中。

1. 对工厂类进行实例化

*private static <T> T instantiateFactory(*

*String instanceClassName,*

*Class<T> factoryClass,*

*ClassLoader classLoader) {*

*try {*

*Class<?> instanceClass = ClassUtils.forName(instanceClassName, classLoader);*

*...*

*return (T) ReflectionUtils.accessibleConstructor(instanceClass).newInstance();*

*}*

*}*

使用反射技术实现实例化。

1. 将Bean对象放到BeanFactory中

*private ConfigurableListableBeanFactory beanFactory;*

上面的BeanFactory的实现类是DefaultListableBeanFactory，其将Bean对象存放到Map中，如下所示：

*private final Map<String, Object> singletonObjects =*

*new ConcurrentHashMap<String, Object>(256);*

SPI原理：

<https://www.cnblogs.com/java-zhao/p/7617143.html>

Spring SPI :

https://blog.csdn.net/qq\_28802119/article/details/83536305