链滴

Dubbo 源码分析 —【5】SPI 扩展机制 下

作者：zsr251

原文链接：https://ld246.com/article/1543740636017

来源网站：链滴

承上

上篇分析到了 injectExtension，其中 objectFactory 需要详细分析

Object object = objectFactory.getExtension(pt, property);

objectFactory 是在用私有构造方法初始化的 ExtensionFactory 的自适应实现类。

// 如果是扩展工厂类型 则返回 null
objectFactory = (type == ExtensionFactory.class ? null : ExtensionLoader.getExtensionLoader(
xtensionFactory.class).getAdaptiveExtension());

即最重要的是这一句：ExtensionLoader.getExtensionLoader(ExtensionFactory.class).getAdaptiv
Extension() 和简单使用方法对比一下，我们会发现它调用的是 getAdaptiveExtension 方法

第一层 getAdaptiveExtension 分析

getAdaptiveExtension 主要的作用是 获取

```
// 获得扩展实例
public T getAdaptiveExtension() {
    // 先从本地缓存中查找
    Object instance = cachedAdaptiveInstance.get();
    if (instance == null) {
        if (createAdaptiveInstanceError == null) {
            synchronized (cachedAdaptiveInstance) {
                // 解决多线程情况下 指令重排的问题 参数使用 volatile 修饰
                instance = cachedAdaptiveInstance.get();
                if (instance == null) {
                    try {
                        // 创建适应的实例
                        instance = createAdaptiveExtension();
                        cachedAdaptiveInstance.set(instance);
                    } catch (Throwable t) {
                        createAdaptiveInstanceError = t;
                        throw new IllegalStateException ...
                    }
                }
            }
        } else {
            throw new IllegalStateException ...
        }
    }

    return (T) instance;
}
```

第二层 createAdaptiveExtension 分析

createAdaptiveExtension 和上文分析的 createExtension 作用相似，但是实现有很大区别，这个方标示着 Dubbo 自己扩展的 SPI 机制增加的对扩展点自适应的支持。

```
private T createAdaptiveExtension() {
    try {
        // 上文已分析 `injectExtension` 是根据 setter 方法实现 IoC 的
        return injectExtension((T) getAdaptiveExtensionClass().newInstance());
    } catch (Exception e) {
        throw new IllegalStateException ...
    }
}
```

第三层 getAdaptiveExtensionClass 分析

getAdaptiveExtensionClass 的主要作用的是先从缓存中加载，如果从未加载过则再调用方法加载

```
private Class<?> getAdaptiveExtensionClass() {
    // 上文已分析 `getExtensionClasses` 的实现是为了加载指定路径的文件配置
    getExtensionClasses();
    // 先从本地缓存中加载
    if (cachedAdaptiveClass != null) {
        return cachedAdaptiveClass;
    }
    return cachedAdaptiveClass = createAdaptiveExtensionClass();
}
```

第四层 createAdaptiveExtensionClass 分析

创建一个扩展点的代理，将扩展的选择从Dubbo启动时，延迟到RPC调用时。非常有意思的是实现动编译的 Compiler 也使用的插件的方式。

```
private Class<?> createAdaptiveExtensionClass() {
    // 生成 SPI 的自适应实现类的 .java 代码
    // 先生成Java源代码，然后编译，加载到jvm中。通过这种方式，可以更好的控制生成的Java类。且这样也不用care各个字节码生成框架的api
    String code = createAdaptiveExtensionClassCode();
    ClassLoader classLoader = findClassLoader();
    // 选择一个编译框架
    org.apache.dubbo.common.compiler.Compiler compiler = ExtensionLoader.getExtensionLoader(org.apache.dubbo.common.compiler.Compiler.class).getAdaptiveExtension();
    // 编译 Java 源码
    return compiler.compile(code, classLoader);
}
```

第五层 createAdaptiveExtensionClassCode 分析

这个主要时生成 SPI 自适应实现类的 .java 代码，最重要也是相对比较复杂的，耐心看完这一点就离功不远了，刚把得！

```
// 创建扩展接口 实现类代码
private String createAdaptiveExtensionClassCode() {
    StringBuilder codeBuilder = new StringBuilder();
    // 获得扩展接口的所有方法
    Method[] methods = type.getMethods();
```

```
    boolean hasAdaptiveAnnotation = false;
    // 判断方法里是否有被 @Adaptive 注解的方法。如果没有则抛出异常 没有需要自动适配的方法
    for (Method m : methods) {
        if (m.isAnnotationPresent(Adaptive.class)) {
            hasAdaptiveAnnotation = true;
            break;
        }
    }
    // no need to generate adaptive class since there's no adaptive method found.
    if (!hasAdaptiveAnnotation) {
        throw new IllegalStateException("No adaptive method on extension " + type.getName()
 ", refuse to create the adaptive class!");
    }

    codeBuilder.append("package ").append(type.getPackage().getName()).append(";");
    codeBuilder.append("\nimport ").append(ExtensionLoader.class.getName()).append(";");
    // 生成的类名称是 SPI 接口名 + $Adaptive 例如：A 接口 生成的类名是 A$Adaptive
    codeBuilder.append("\npublic class ").append(type.getSimpleName()).append("$Adaptive")
append(" implements ").append(type.getCanonicalName()).append(" {");

    codeBuilder.append("\nprivate static final org.apache.dubbo.common.logger.Logger logger
= org.apache.dubbo.common.logger.LoggerFactory.getLogger(ExtensionLoader.class);");
    codeBuilder.append("\nprivate java.util.concurrent.atomic.AtomicInteger count = new java.
til.concurrent.atomic.AtomicInteger(0);\n");

    for (Method method : methods) {
        Class<?> rt = method.getReturnType();
        Class<?>[] pts = method.getParameterTypes();
        Class<?>[] ets = method.getExceptionTypes();

        Adaptive adaptiveAnnotation = method.getAnnotation(Adaptive.class);
        StringBuilder code = new StringBuilder(512);
        // 如果没有被 @Adaptive 注解的方法，调用的话会抛出异常
        if (adaptiveAnnotation == null) {
            code.append("throw new UnsupportedOperationException(\"method ")
                    .append(method.toString()).append(" of interface ")
                    .append(type.getName()).append(" is not adaptive method!\");");
        } else {
            // 这段代码是为了获取 URL 参数  -------start-------
            int urlTypeIndex = -1;
            // 判断接口参数中有没有 URL
            for (int i = 0; i < pts.length; ++i) {
                if (pts[i].equals(URL.class)) {
                    urlTypeIndex = i;
                    break;
                }
            }
            // found parameter in URL type
            if (urlTypeIndex != -1) {
                // Null Point check
                String s = String.format("\nif (arg%d == null) throw new IllegalArgumentException(
"url == null\");",
                        urlTypeIndex);
                code.append(s);
```

```
            s = String.format("\n%s url = arg%d;", URL.class.getName(), urlTypeIndex);
            code.append(s);
        }
        // did not find parameter in URL type
        else {
            String attribMethod = null;
            // 如果在参数中没有 URL，则从参数中查找是否有能够获得 URL 的方法
            // find URL getter method
            // 这个是一个 break 标示，跳出循环时，直接跳出外层循环
            LBL_PTS:
            for (int i = 0; i < pts.length; ++i) {
                Method[] ms = pts[i].getMethods();
                for (Method m : ms) {
                    String name = m.getName();
                    if ((name.startsWith("get") || name.length() > 3)
                            && Modifier.isPublic(m.getModifiers())
                            && !Modifier.isStatic(m.getModifiers())
                            && m.getParameterTypes().length == 0
                            && m.getReturnType() == URL.class) {
                        urlTypeIndex = i;
                        attribMethod = name;
                        // 直接跳出最外层循环 LBL_PTS 标示处
                        break LBL_PTS;
                    }
                }
            }
            // 如果参数的方法中也不能获得 URL 则抛出异常
            if (attribMethod == null) {
                throw new IllegalStateException("fail to create adaptive class for interface " + typ
.getName()
                        + ": not found url parameter or url attribute in parameters of method " + me
hod.getName());
            }

            // Null point check
            String s = String.format("\nif (arg%d == null) throw new IllegalArgumentException(
"%s argument == null\");",
                    urlTypeIndex, pts[urlTypeIndex].getName());
            code.append(s);
            s = String.format("\nif (arg%d.%s() == null) throw new IllegalArgumentException(\
%s argument %s() == null\");",
                    urlTypeIndex, attribMethod, pts[urlTypeIndex].getName(), attribMethod);
            code.append(s);

            s = String.format("%s url = arg%d.%s();", URL.class.getName(), urlTypeIndex, attrib
ethod);
            code.append(s);
        }
        // 这段代码是为了获取 URL 参数  -------end-------

        String[] value = adaptiveAnnotation.value();
        // value is not set, use the value generated from class name as the key
        if (value.length == 0) {
```

```
            char[] charArray = type.getSimpleName().toCharArray();
            StringBuilder sb = new StringBuilder(128);
            for (int i = 0; i < charArray.length; i++) {
                if (Character.isUpperCase(charArray[i])) {
                    if (i != 0) {
                        sb.append(".");
                    }
                    sb.append(Character.toLowerCase(charArray[i]));
                } else {
                    sb.append(charArray[i]);
                }
            }
            value = new String[]{sb.toString()};
        }

        boolean hasInvocation = false;
        for (int i = 0; i < pts.length; ++i) {
            if (("org.apache.dubbo.rpc.Invocation").equals(pts[i].getName())) {
                // Null Point check
                String s = String.format("\nif (arg%d == null) throw new IllegalArgumentExcepti
n(\"invocation == null\");", i);
                code.append(s);
                s = String.format("\nString methodName = arg%d.getMethodName();", i);
                code.append(s);
                hasInvocation = true;
                break;
            }
        }

        String defaultExtName = cachedDefaultName;
        String getNameCode = null;
        for (int i = value.length - 1; i >= 0; --i) {
            if (i == value.length - 1) {
                if (null != defaultExtName) {
                    if (!"protocol".equals(value[i])) {
                        if (hasInvocation) {
                            getNameCode = String.format("url.getMethodParameter(methodName, \
%s\", \"%s\")", value[i], defaultExtName);
                        } else {
                            getNameCode = String.format("url.getParameter(\"%s\", \"%s\")", value[i],
defaultExtName);
                        }
                    } else {
                        getNameCode = String.format("( url.getProtocol() == null ? \"%s\" : url.getP
otocol() )", defaultExtName);
                    }
                } else {
                    if (!"protocol".equals(value[i])) {
                        if (hasInvocation) {
                            getNameCode = String.format("url.getMethodParameter(methodName, \
%s\", \"%s\")", value[i], defaultExtName);
                        } else {
                            getNameCode = String.format("url.getParameter(\"%s\")", value[i]);
                        }
```

```
                    } else {
                        getNameCode = "url.getProtocol()";
                    }
                }
            } else {
                if (!"protocol".equals(value[i])) {
                    if (hasInvocation) {
                        getNameCode = String.format("url.getMethodParameter(methodName, \"%
\", \"%s\")", value[i], defaultExtName);
                    } else {
                        getNameCode = String.format("url.getParameter(\"%s\", %s)", value[i], getN
meCode);
                    }
                } else {
                    getNameCode = String.format("url.getProtocol() == null ? (%s) : url.getProtoco
()", getNameCode);
                }
            }
        }
        code.append("\nString extName = ").append(getNameCode).append(";");
        // check extName == null?
        String s = String.format("\nif(extName == null) " +
                        "throw new IllegalStateException(\"Fail to get extension(%s) name from url(\
 + url.toString() + \") use keys(%s)\");",
                type.getName(), Arrays.toString(value));
        code.append(s);
        // 真正执行的方法出现了！ getExtensionLoader().getExtension() 就是上一篇分析的简单
用
        code.append(String.format("\n%s extension = null;\n try {\nextension = (%<s)%s.getE
tensionLoader(%s.class).getExtension(extName);\n}catch(Exception e){\n",
                type.getName(), ExtensionLoader.class.getSimpleName(), type.getName()));
        code.append(String.format("if (count.incrementAndGet() == 1) {\nlogger.warn(\"Failed
to find extension named \" + extName + \" for type %s, will use default extension %s instead.
", e);\n}\n",
                type.getName(), defaultExtName));
        code.append(String.format("extension = (%s)%s.getExtensionLoader(%s.class).getExte
sion(\"%s\");\n}",
                type.getName(), ExtensionLoader.class.getSimpleName(), type.getName(), default
xtName));

        // return statement
        if (!rt.equals(void.class)) {
            code.append("\nreturn ");
        }

        s = String.format("extension.%s(", method.getName());
        code.append(s);
        for (int i = 0; i < pts.length; i++) {
            if (i != 0) {
                code.append(", ");
            }
            code.append("arg").append(i);
        }
        code.append(");");
```

```
        }

        codeBuilder.append("\npublic ").append(rt.getCanonicalName()).append(" ").append(me
thod.getName()).append("(");
        for (int i = 0; i < pts.length; i++) {
            if (i > 0) {
                codeBuilder.append(", ");
            }
            codeBuilder.append(pts[i].getCanonicalName());
            codeBuilder.append(" ");
            codeBuilder.append("arg").append(i);
        }
        codeBuilder.append(")");
        if (ets.length > 0) {
            codeBuilder.append(" throws ");
            for (int i = 0; i < ets.length; i++) {
                if (i > 0) {
                    codeBuilder.append(", ");
                }
                codeBuilder.append(ets[i].getCanonicalName());
            }
        }
        codeBuilder.append(" {");
        codeBuilder.append(code.toString());
        codeBuilder.append("\n}");
    }
    codeBuilder.append("\n}");
    if (logger.isDebugEnabled()) {
        logger.debug(codeBuilder.toString());
    }
    return codeBuilder.toString();
}
```

Compiler 解析

Dubbo会自动为我们创建一个，默认使用Javaassist

```
@SPI("javassist")
public interface Compiler {
    // 你会发现 这个方法上没有 @Adaptive 注解，根据上文分析应该会报错吧?
    // 不用担心不会，它的dubbo源码中唯二 实现类上有 @Adaptive 的，走不到那段生成代码的逻辑
    // 另一个是 ExtensionFactory
    Class<?> compile(String code, ClassLoader classLoader);
}
```

自适应实现

```
// 在实现类上直接有 @Adaptive 注解，在调用 `getAdaptiveExtension` 方法时不会再生成自适应
现类代码
@Adaptive
public class AdaptiveCompiler implements Compiler {
    ...
    @Override
    public Class<?> compile(String code, ClassLoader classLoader) {
```

```
        Compiler compiler;
        // 真正调用的时候 再选择实际调用的实现类
        ExtensionLoader<Compiler> loader = ExtensionLoader.getExtensionLoader(Compiler.cla
s);
        String name = DEFAULT_COMPILER; // copy reference
        if (name != null && name.length() > 0) {
            compiler = loader.getExtension(name);
        } else {
            // 如果没有指定别名 则加载默认的 即 javassist
            compiler = loader.getDefaultExtension();
        }
        return compiler.compile(code, classLoader);
    }

}
```

总结

getActivateExtension 就不再分析了。另外实现 AOP 功能的Wrapper类也不再详细分析了。

理解了 SPI 的实现能够帮助我们分析 Dubbo 源码的实现。不就是各个插件组装嘛，对吧：）

- 扩展点自动包装
  - 由 wrapperClass.getConstructor(type).newInstance(instance) 实现
  - 自动包装扩展点的 Wrapper 类
  - 根据代码分析 只要扩展点有拷贝构造函数，则判定为扩展点 Wrapper 类。
  - 新加的 Wrapper 在所有的扩展点上添加了逻辑，有些类似 AOP，即 Wrapper 代理了扩展点。
- 扩展点自动装配
  - 由 injectExtension 实现
  - ExtensionLoader 通过扫描扩展点实现类的所有 setter 方法来判定其成员。
  - 自动注入依赖的扩展点，实现 IoC
- 扩展点自适应
  - 由 getAdaptiveExtension 实现
  - ExtensionLoader 注入的依赖扩展点是一个 Adaptive 实例，直到扩展点方法执行时才决定调是一个扩展点实现。
  - Dubbo 使用 URL 对象（包含了Key-Value）传递配置信息。
- 扩展点自动激活
  - 由 getActivateExtension 实现
  - 同时加载多个实现，可以用自动激活来简化配置