

2_全局配置加载

仅供 编程导航 <<https://www.code-nav.cn/post/1816420035119853569>> 内部成员观看，请勿对外分享！

从本节教程开始，我们将对 RPC 框架进行一系列的扩展。

一、需求分析

在 RPC 框架运行的过程中，会涉及到很多的配置信息，比如注册中心的地址、序列化方式、网络服务器端口号等等。

之前的简易版 RPC 项目中，我们是在程序里硬编码了这些配置，不利于维护。

而且 RPC 框架是需要被其他项目作为服务提供者或者服务消费者引入的，我们应当允许引入框架的项目通过编写配置文件来 **自定义配置**。并且一般情况下，服务提供者和服务消费者需要编写相同的 RPC 配置。

因此，我们需要一套全局配置加载功能。能够让 RPC 框架轻松地从配置文件中读取配置，并且维护一个全局配置对象，便于框架快速获取到一致的配置。

二、设计方案

配置项

首先我们梳理需要的配置项，刚开始就一切从简，只提供以下几个配置项即可：

- name 名称
- version 版本号
- serverHost 服务器主机名
- serverPort 服务器端口号

后续随着框架功能的扩展，我们会不断地新增配置项，还可以适当地对配置项进行分组。

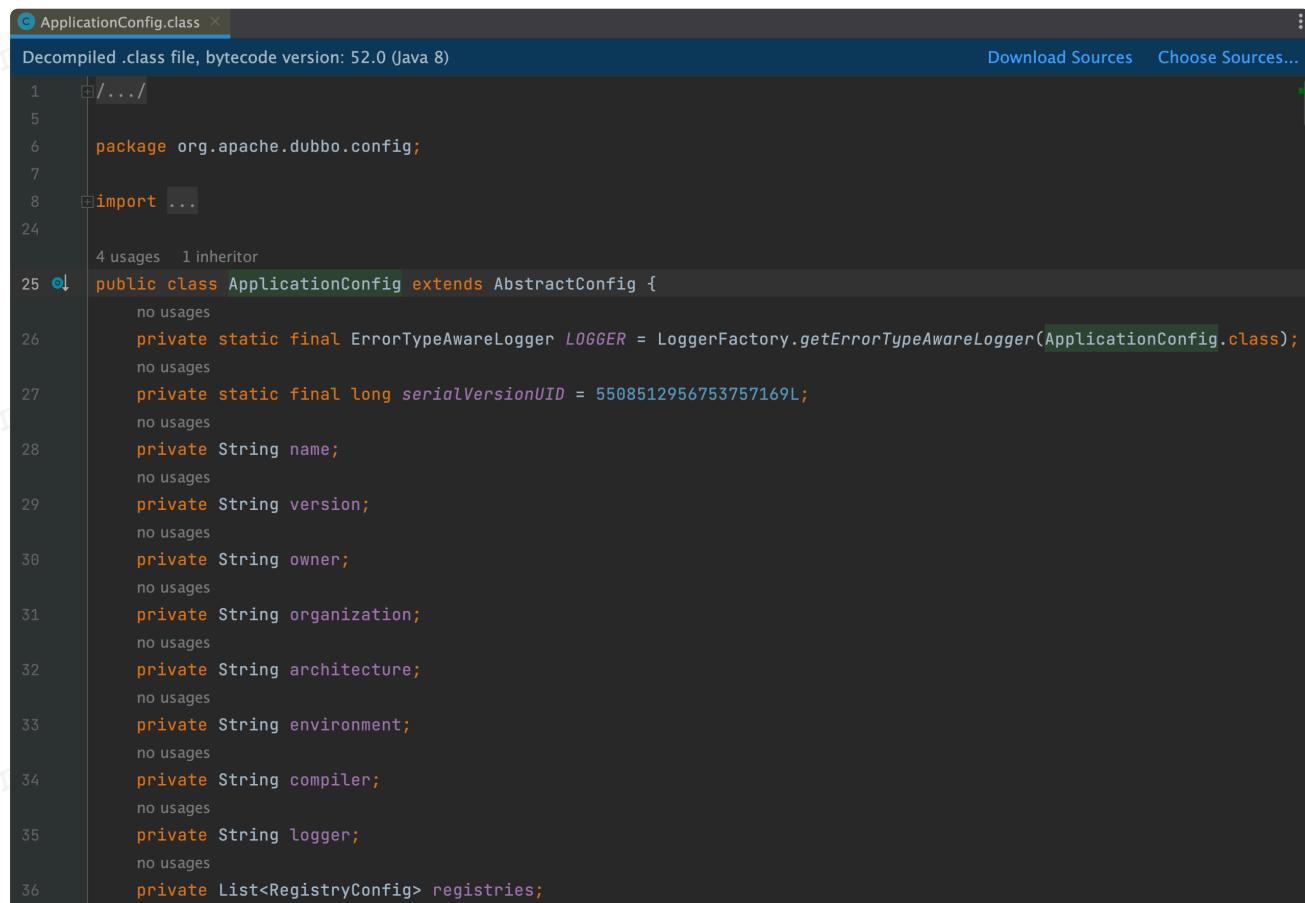
比如以下是一些常见的 RPC 框架配置项，仅做了解即可：

1. 注册中心地址：服务提供者和服务消费者都需要指定注册中心的地址，以便进行服务的注册和发现。
2. 服务接口：服务提供者需要指定提供的服务接口，而服务消费者需要指定要调用的服务接口。
3. 序列化方式：服务提供者和服务消费者都需要指定序列化方式，以便在网络中传输数据时进行序列化和反序列化。
4. 网络通信协议：服务提供者和服务消费者都需要选择合适的网络通信协议，比如 TCP、HTTP 等。
5. 超时设置：服务提供者和服务消费者都需要设置超时时间，以便在调用服务时进行超时处理。
6. 负载均衡策略：服务消费者需要指定负载均衡策略，以决定调用哪个服务提供者实例。
7. 服务端线程模型：服务提供者需要指定服务端线程模型，以决定如何处理客户端请求。

感兴趣的同学可以了解下 Dubbo RPC 框架的配置项，包括应用配置、注册中心配置、服务配置等。

参考 Dubbo: <https://cn.dubbo.apache.org/zh-cn/overview/manual/java-sdk/reference-manual/config/api/> <<https://cn.dubbo.apache.org/zh-cn/overview/manual/java-sdk/reference-manual/config/api/>>

任意项目引入 Dubbo 依赖后，就可以查看到 `ApplicationConfig` 配置类，如图：



The screenshot shows a Java code editor with the file `ApplicationConfig.class` open. The code is displayed in a dark-themed interface. The class definition is as follows:

```
1  /...
5
6  package org.apache.dubbo.config;
7
8  import ...
24
25  public class ApplicationConfig extends AbstractConfig {
26      no usages
27      private static final ErrorTypeAwareLogger LOGGER = LoggerFactory.getErrorTypeAwareLogger(ApplicationConfig.class);
28      no usages
29      private static final long serialVersionUID = 5508512956753757169L;
30      no usages
31      private String name;
32      no usages
33      private String version;
34      no usages
35      private String owner;
36      no usages
37      private String organization;
38      no usages
39      private String architecture;
40      no usages
41      private String environment;
42      no usages
43      private String compiler;
44      no usages
45      private String logger;
46      no usages
47      private List<RegistryConfig> registries;
```

读取配置文件

如何读取配置文件呢？这里可以使用 Java 的 Properties 类自行编写，但是更推荐使用一些第三方工具库，比如 Hutool 的 Setting 模块，可以直接读取指定名称的配置文件中的部分配置信息，并且转换成 Java 对象，非常方便。

参考官方文档：<https://doc.hutool.cn/pages/Props/>

<<https://doc.hutool.cn/pages/Props/>. >

一般情况下，我们读取的配置文件名称为 `application.properties`，还可以通过指定文件名称后缀的方式来区分多环境，比如 `application-prod.properties` 表示生产环境、`application-test.properties` 表示测试环境。

三、开发实现

1、项目初始化

1) 先新建 `yu-rpc-core` 模块，后面的 RPC 项目开发及扩展均在该项目进行。

可以直接复制 `yu-rpc-easy` 的代码并改名，就得到了这个项目。

2) 然后给项目引入日志库和单元测试依赖，便于后续开发：

```
1  <!-- https://mvnrepository.com/artifact/ch.qos.logback/logback-classic -->
2  <dependency>
3      <groupId>ch.qos.logback</groupId>
4
5      <artifactId>logback-classic</artifactId>
6
7      <version>1.3.12</version>
8
9  </dependency>
10
11 <dependency>
12     <groupId>junit</groupId>
13
14     <artifactId>junit</artifactId>
15
16     <version>RELEASE</version>
17
18     <scope>test</scope>
19
20 </dependency>
21
```

3) 将 `example-consumer` 和 `example-provider` 项目引入的 RPC 依赖都替换成 `yu-rpc-core`，代码如下：

```
1  <dependency>
2      <groupId>com.yupi</groupId>
3
4      <artifactId>yu-rpc-core</artifactId>
5
6      <version>1.0-SNAPSHOT</version>
7
8  </dependency>
9
```

2、配置加载

1) 在 config 包下新建配置类 `RpcConfig`，用于保存配置信息。

可以给属性指定一些默认值，完整代码如下：

```
1 package com.yupi.yurpc.config;
2
3 import lombok.Data;
4
5 /**
6  * RPC 框架配置
7  */
8 @Data
9 public class RpcConfig {
10
11 /**
12  * 名称
13 */
14 private String name = "yu-rpc";
15
16 /**
17  * 版本号
18 */
19 private String version = "1.0";
20
21 /**
22  * 服务器主机名
23 */
24 private String serverHost = "localhost";
25
26 /**
27  * 服务器端口号
28 */
29 private Integer serverPort = 8080;
30
31 }
```

2) 在 utils 包下新建工具类 `ConfigUtils`，作用是读取配置文件并返回配置对象，可以简化调用。

工具类应当尽量通用，和业务不强绑定，提高使用的灵活性。比如支持外层传入要读取的配置内容前缀、支持传入环境等。

完整代码如下：

```
1 package com.yupi.yurpc.utils;
2
3 import cn.hutool.core.util.StrUtil;
4 import cn.hutool.setting.dialect.Props;
5
6 /**
7 * 配置工具类
8 */
9 public class ConfigUtils {
10
11     /**
12      * 加载配置对象
13      *
14      * @param tClass
15      * @param prefix
16      * @param <T>
17      * @return
18      */
19     public static <T> T loadConfig(Class<T> tClass, String prefix) {
20         return loadConfig(tClass, prefix, "");
21     }
22
23     /**
24      * 加载配置对象，支持区分环境
25      *
26      * @param tClass
27      * @param prefix
28      * @param environment
29      * @param <T>
30      * @return
31      */
32     public static <T> T loadConfig(Class<T> tClass, String prefix, String env
33         StringBuilder configfileBuilder = new StringBuilder("application");
34     if (StrUtil.isNotBlank(environment)) {
35         configfileBuilder.append("-").append(environment);
36     }
37     configfileBuilder.append(".properties");
38     Props props = new Props(configfileBuilder.toString());
39     return props.toBean(tClass, prefix);
40 }
41 }
```

之后，调用 `ConfigUtils` 的静态方法就能读取配置了。

3) 在 `constant` 包中新建 `RpcConstant` 接口，用于存储 RPC 框架相关的常量。

比如默认配置文件的加载前缀为 `rpc`：

```
1 package com.yupi.yurpc.constant;
2
3 /**
4  * RPC 相关常量
5  *
6  * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
7  *
8  * @learn <a href="https://codefather.cn">鱼皮的编程宝典</a>
9  *
10 * @from <a href="https://yupi.icu">编程导航学习圈</a>
11 */
12
13 public interface RpcConstant {
14
15 /**
16  * 默认配置文件加载前缀
17  */
18 String DEFAULT_CONFIG_PREFIX = "rpc";
19 }
```

可以读取到类似下面的配置：

```
1 rpc.name=yurpc
2 rpc.version=2.0
3 rpc.serverPort=8081
```

3、维护全局配置对象

RPC 框架中需要维护一个全局的配置对象。在引入 RPC 框架的项目启动时，从配置文件中读取配置并创建对象实例，之后就可以集中地从这个对象中获取配置信息，而不用每次加载配置时再重新读取配置、并创建新的对象，减少了性能开销。

使用设计模式中的 **单例模式**，就能够很轻松地实现这个需求了。

一般情况下，我们会使用 `holder` 来维护全局配置对象实例。在我们的项目中，可以换一个更优雅的命名，使用 `RpcApplication` 类作为 RPC 项目的启动入口、并且维护项目全局用到的变量。

完整代码如下：

```
1 package com.yupi.yurpc;
2
3 import com.yupi.yurpc.config.RpcConfig;
4 import com.yupi.yurpc.constant.RpcConstant;
5 import com.yupi.yurpc.utils.ConfigUtils;
6 import lombok.extern.slf4j.Slf4j;
7
8 /**
9  * RPC 框架应用
10 * 相当于 holder，存放了项目全局用到的变量。双检锁单例模式实现
11 */
12 @Slf4j
13 public class RpcApplication {
14
15     private static volatile RpcConfig rpcConfig;
16
17     /**
18      * 框架初始化，支持传入自定义配置
19      *
20      * @param newRpcConfig
21      */
22     public static void init(RpcConfig newRpcConfig) {
23         rpcConfig = newRpcConfig;
24         log.info("rpc init, config = {}", newRpcConfig.toString());
25     }
26
27     /**
28      * 初始化
29      */
30     public static void init() {
31         RpcConfig newRpcConfig;
32         try {
33             newRpcConfig = ConfigUtils.loadConfig(RpcConfig.class, RpcConstan
34         } catch (Exception e) {
35             // 配置加载失败，使用默认值
36             newRpcConfig = new RpcConfig();
37         }
38         init(newRpcConfig);
39     }
40
41     /**
42      * 获取配置
43      *
44      * @return
45      */
46     public static RpcConfig getRpcConfig() {
47         if (rpcConfig == null) {
48             synchronized (RpcApplication.class) {
49                 if (rpcConfig == null) {
50                     init();
```

```
51             }
52         }
53     }
54     return rpcConfig;
55 }
56 }
```

上述代码其实就是 双检锁单例模式 的经典实现，支持在获取配置时才调用 init 方法实现懒加载。

为了便于扩展，还支持自己传入配置对象；如果不传入，则默认调用前面写好的 ConfigUtils 来加载配置。

以后 RPC 框架内只需要写一行代码，就能正确加载到配置：

```
1 RpcConfig rpc = RpcApplication.getRpcConfig();
```

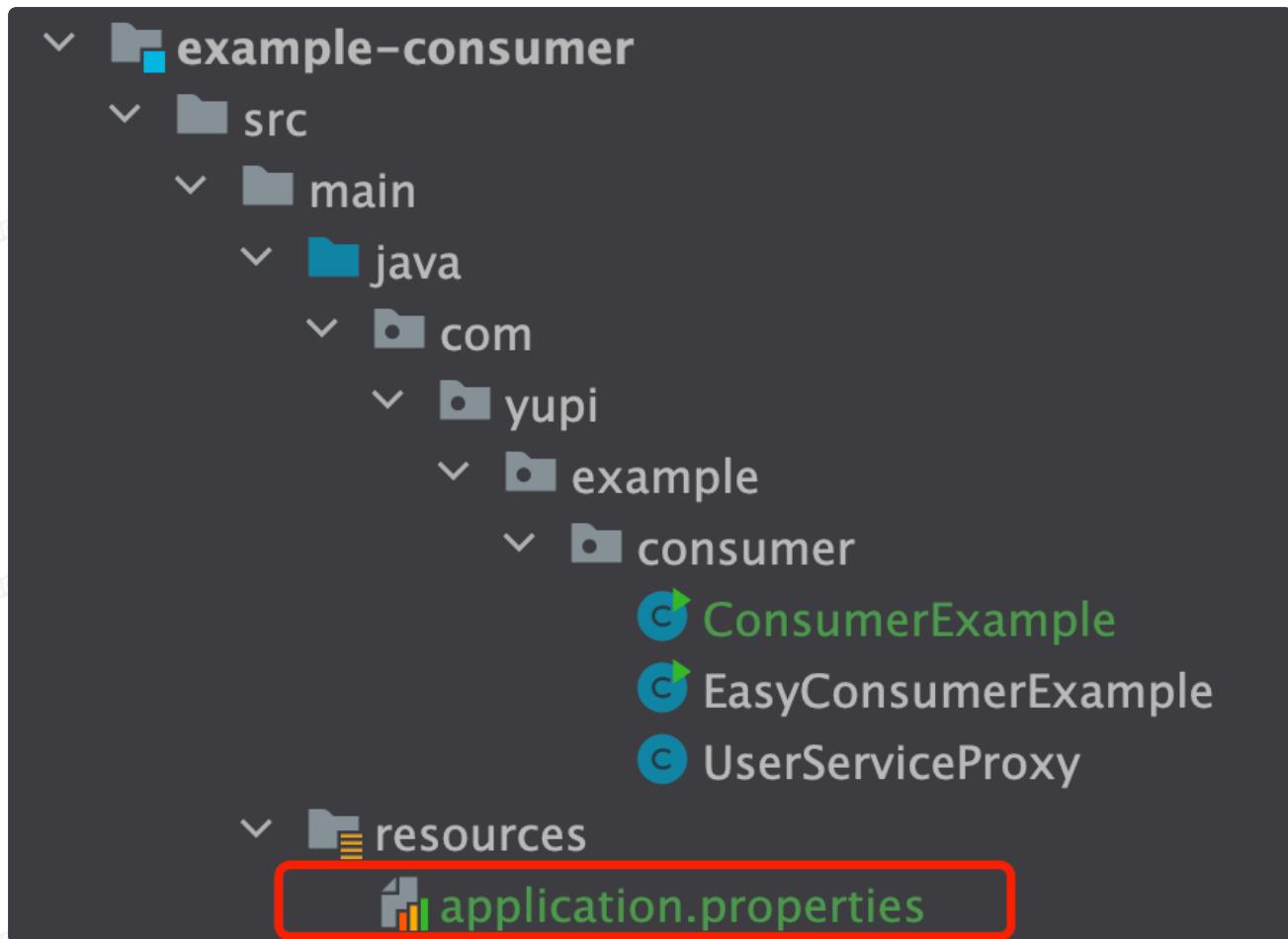
四、测试

1、测试配置文件读取

在 example-consumer 项目的 resources 目录下编写配置文件 application.properties，代码如下：

```
1 rpc.name=yurpc
2 rpc.version=2.0
3 rpc.serverPort=8081
```

如图：



创建 `ConsumerExample` 作为扩展后 RPC 项目的示例消费者类，测试配置文件读取。

代码如下：

```
1  /**  
2   * 简易服务消费者示例  
3   *  
4   * @author <a href="https://github.com/liyupi">程序员鱼皮</a>  
5   * @learn <a href="https://codefather.cn">编程宝典</a>  
6   * @from <a href="https://yupi.icu">编程导航知识星球</a>  
7   *  
8   */  
9  
10  public class ConsumerExample {  
11  
12  
13  public static void main(String[] args) {  
14      RpcConfig rpc = ConfigUtils.loadConfig(RpcConfig.class, "rpc");  
15      System.out.println(rpc);  
16      ...  
17  }  
18 }
```

能够正确输出配置。

2、测试全局配置对象加载

在 `example-provider` 项目中创建 `ProviderExample` 服务提供者示例类，能够根据配置动态地在不同端口启动 web 服务。

代码如下：

```
1 package com.yupi.example.provider;
2
3 import com.yupi.example.common.service.UserService;
4 import com.yupi.yurpc.RpcApplication;
5 import com.yupi.yurpc.registry.LocalRegistry;
6 import com.yupi.yurpc.server.HttpServer;
7 import com.yupi.yurpc.server.VertxHttpServer;
8
9 /**
10  * 简易服务提供者示例
11 *
12  * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
13  * @learn <a href="https://codefather.cn">编程宝典</a>
14  * @from <a href="https://yupi.icu">编程导航知识星球</a>
15 *
16 */
17
18 public class EasyProviderExample {
19
20
21     public static void main(String[] args) {
22         // RPC 框架初始化
23         RpcApplication.init();
24
25         // 注册服务
26         LocalRegistry.register(UserService.class.getName(), UserServiceImpl.c
27
28         // 启动 web 服务
29         HttpServer httpServer = new VertxHttpServer();
30         httpServer.doStart(RpcApplication.getRpcConfig().getServerPort());
31     }
32 }
```

五、扩展

提供以下扩展思路，可自行实现：

- 1) 支持读取 `application.yml`、`application.yaml` 等不同格式的配置文件。

2) 支持监听配置文件的变更，并自动更新配置对象。

参考思路：使用 Hutool 工具类的 `props.autoLoad()` 可以实现配置文件变更的监听和自动加载。

3) 配置文件支持中文。

参考思路：需要注意编码问题

4) 配置分组。后续随着配置项的增多，可以考虑对配置项进行分组。

参考思路：可以通过嵌套配置类实现。