链滴

SpringBoot 仿微服务 CoreArchetype 与 UtilArchetype 的搭配使用

作者：liumapp

原文链接：https://ld246.com/article/1513844127335

来源网站：链滴

CoreArchetype作为微服务独立部署运行，占据独有的端口，UtilArchetype做为一个工具库，以依的形式为CoreArchetype提供服务。

前言

先上示范案例的源代码：

spring-boot-core-util-demo

安装使用

一个典型的Maven项目，引入IDEA之后，直接运行core-module下的启动类即可。（您可能需要检一下配置文件，检查相关端口是否被占用）

或者

- mvn install
- cd core-module/target/
- java -jar ./core-module-v1.0.0.jar

源代码中的util-module是通过UtilArchetype自动生成的。

core-module是通过CoreArchetype自动生成的。

关于这两个Archetype的安装使用，在上两篇博文中已经提及过了。

分别是：

SpringBoot工具库UtilArchetype

SpringBoot仿微服务CoreArchetype

同时在下文中，util-module的内容我会简称为util，core-module的内容将被成为core。

配置

具体的配置文件为core下的application.yml文件，也就是说，我们只在能够运行的core模块下去配参数。

对于util而言，我们将它看作一个工具库，所以它只需要提供外界需要配置的参数，而至于参数的内是什么，util本身不需要关心。

所以在util下，跟配置相关的就三个核心的类注意：

- Main

没错，即便是工具库，本身也具备了一个Main类，其代码为：

```
@Configuration
@Import({UtilConfig.class})
public  class  Main {
```

```
}
```

- UtilConfig

初始化Bean对象的配置类

```
@Configuration
public class UtilConfig {

 @Bean
 @ConfigurationProperties(prefix = "liumapp.test.module.util")
   public UtilParams utilParams(){
      UtilParams utilParams = new UtilParams();
      return utilParams;
   }

  @Bean
  public Guest guest(UtilParams utilParams) {
      Guest guest = new Guest();
      guest.setAppKey(utilParams.getAppKey());
      return guest;
  }

}
```

- UtilParams

定义了该工具库需要的参数

```
@Component
public class UtilParams {

    private String appKey;

    public String getAppKey() {
        return appKey;
    }

    public void setAppKey(String appKey) {
        this.appKey = appKey;
    }
}
```

其中的UtilConfig将会去接收在core下的application.yml配置文件中的

```
liumapp:
  test:
    module:
      util:
        *****
```

下面的参数，并自动写入UtilParams中，当然了，前提是配置的参数必须是UtilParams的实际属性，且要具有写方法的属性。

pom

core下需要引入util，除了配置，还需要在pom的配置文件中去加载util：

```
<dependency>
  <groupId>com.liumapp.test.module.util</groupId>
  <artifactId>util-module</artifactId>
  <version>v1.0.0</version>
</dependency>
```

加载之后，在core的Main类中，使用Import注解，导入util的Main类即可剩余的工作。

```
@Configuration
@Import({UtilConfig.class})
public  class  Main  {

}
```

效果

效果很简单，运行core的启动类，打开浏览器访问之后，便能够看到core利用Spring的依赖注入，使util下的Guest类，来输出打印配置文件的AppKey的内容。

当然，这并没有什么实际的意义，更多的作用是作为一个Demo来演示两个模块之间的对应关系。