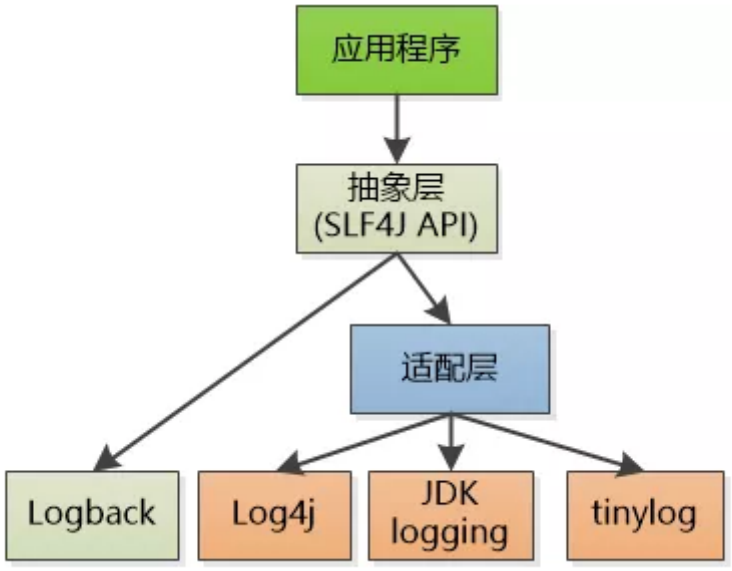
第七步：启动程序
第八步：访问
第九步：更改UserController
第十步：重启服务并访问

--

logback

介绍

Logback继承自log4j。Logback的架构非常的通用，适用于不同的使用场景。



logback和Log4j都是slf4j规范的具体实现，我们在程序中直接调用的API其实都是slf4j的api，底层则是真正的日志实现组件---logback或者log4j。

Logback 构建在三个主要的类上：Logger，Appender 和 Layout。这三个不同类型的组件一起作用能够让开发者根据消息的类型以及日志的级别来打印日志。

Logger作为日志的记录器，把它关联到应用的对应的context后，主要用于存放日志对象，也可以定义日志类型、级别。各个logger 都被关联到一个 LoggerContext，LoggerContext负责制造logger，也负责以树结构排列各 logger。

Appender主要用于指定日志输出的目的地，目的地可以是控制台、文件、 数据库等。

Layout 负责把事件转换成字符串，输出格式化的日志信息。

logback的maven坐标：

```
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-core</artifactId>
    <version>1.2.3</version>
</dependency>
```

logback层级

在 logback中每一个 logger 都依附在 LoggerContext 上，它负责产生 logger，并且通过一个**树状**的层级结构来进行管理。

一个 Logger 被当作为一个实体，它们的命名是大小写敏感的，并且遵循以下规则：

- 如果一个logger的名字加上一个.作为另一个logger名字的前缀，那么该logger就是另一个logger的祖先。如果一个logger与另一个logger之间没有其它的logger，则该logger就是另一个logger的父级。

在logback中有一个root logger，它是logger层次结构的最高层，它是一个特殊的logger，因为它是每一个层次结构的一部分

logback日志输出等级

logback的日志输出等级分为：TRACE, DEBUG, INFO, WARN, ERROR。

如果一个给定的logger没有指定一个日志输出等级，那么它就会继承离它最近的一个祖先的层级。

为了确保所有的logger都有一个日志输出等级，root logger会有一个默认输出等级 --- DEBUG。

logback初始化步骤

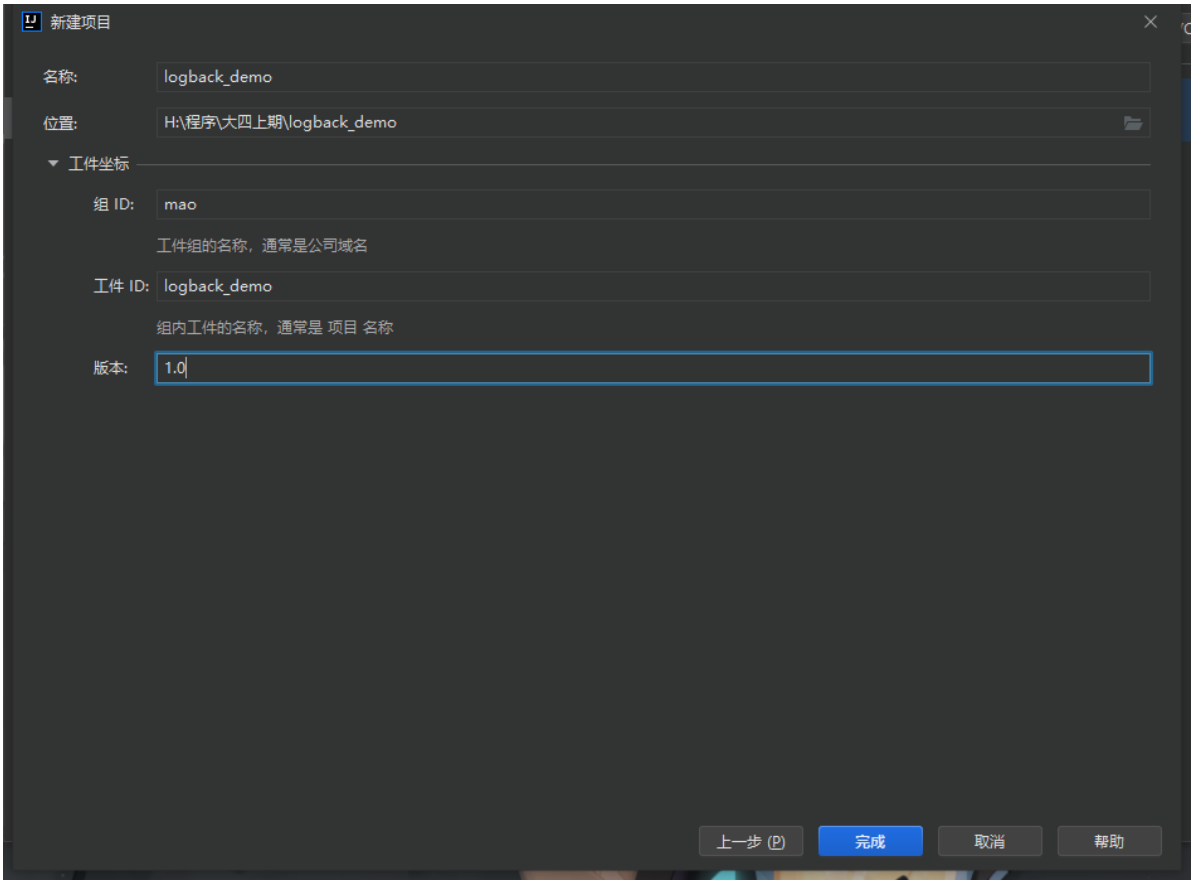
1. logback会在类路径下寻找名为logback-test.xml的文件
2. 如果没有找到，logback会继续寻找名为logback.groovy的文件
3. 如果没有找到，logback会继续寻找名为logback.xml的文件
4. 如果没有找到，将会在类路径下寻找文件META-INFO/services/ch.qos.logback.classic.spi.Configurator，该文件的内容为实现了Configurator接口的实现类的全限定类名
5. 如果以上都没有成功，logback会通过BasicConfigurator为自己进行配置，并且日志将会全部在控制台打印出来

最后一步的目的是为了保证在所有的配置文件都没有被找到的情况下，提供一个默认的配置。

logback入门案例

案例一

第一步：创建maven工程logback_demo



第二步：修改pom文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <!--
      -maven项目核心配置文件-
    Project name(项目名称)：logback_demo
    Author(作者）: mao
    Author QQ：1296193245
    GitHub：https://github.com/maomao124/
    Date(创建日期)： 2022/10/31
    Time(创建时间)： 18:52
    -->
    <groupId>mao</groupId>
    <artifactId>logback_demo</artifactId>
    <version>1.0</version>

    <properties>
        <maven.compiler.source>16</maven.compiler.source>
        <maven.compiler.target>16</maven.compiler.target>
    </properties>

    <dependencies>
```

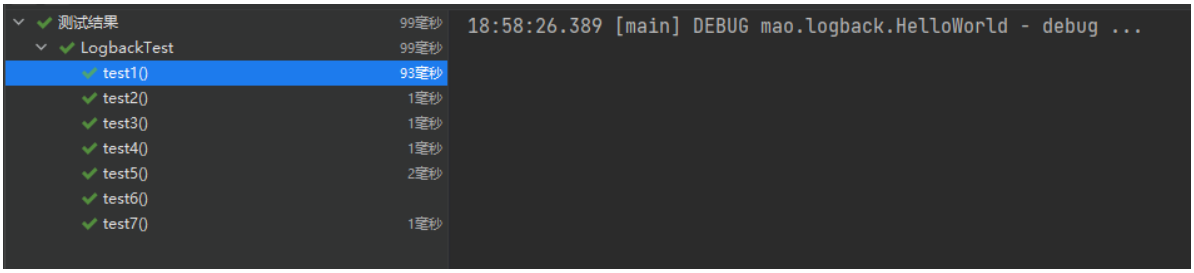
```
25
26          <dependency>
27              <groupId>ch.qos.logback</groupId>
28              <artifactId>logback-classic</artifactId>
29              <version>1.3.4</version>
30          </dependency>
31          <dependency>
32              <groupId>ch.qos.logback</groupId>
33              <artifactId>logback-core</artifactId>
34              <version>1.3.4</version>
35          </dependency>
36
37          <!-- 测试框架 -->
38          <dependency>
39              <groupId>org.junit.jupiter</groupId>
40              <artifactId>junit-jupiter</artifactId>
41              <version>RELEASE</version>
42              <scope>test</scope>
43          </dependency>
44
45      </dependencies>
46
47      <build>
48          <finalName>logback_demo</finalName>
49          <plugins>
50              <plugin>
51                  <groupId>org.apache.maven.plugins</groupId>
52                  <artifactId>maven-jar-plugin</artifactId>
53                  <configuration>
54                      <archive>
55                          <manifest>
56                              <mainClass>Test</mainClass>
57                              <!--更改项，主类名-->
58                          </manifest>
59                      </archive>
60                  </configuration>
61              </plugin>
62          </plugins>
63      </build>
64
65  </project>
```

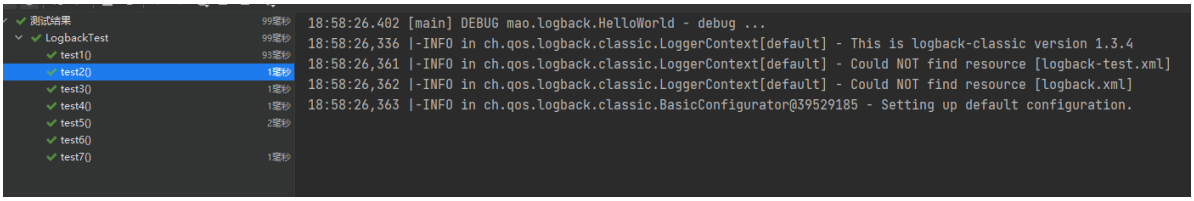
第三步：编写单元测试

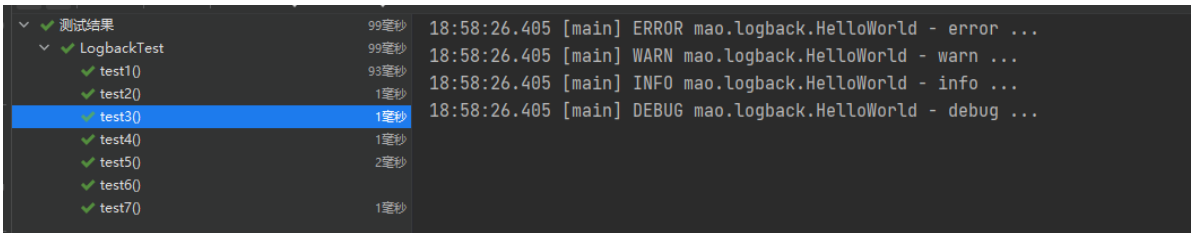
```
1  package mao;
2
3  import ch.qos.logback.classic.Level;
4  import ch.qos.logback.classic.LoggerContext;
5  import ch.qos.logback.core.util.StatusPrinter;
6  import org.junit.jupiter.api.Test;
```

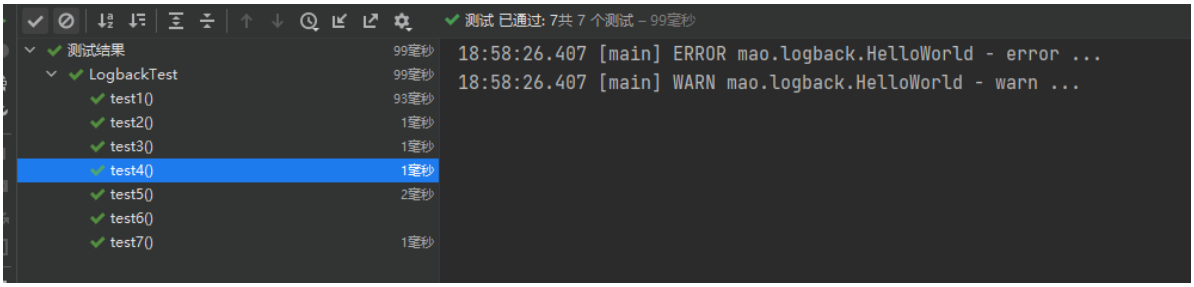
```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Project name(项目名称)：logback_demo
 * Package(包名): mao
 * Class(类名): LogbackTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 18:56
 * Version(版本): 1.0
 * Description(描述)： 无
 */


public class LogbackTest
{
    //简单使用
    @Test
    public void test1()
    {
        Logger logger = LoggerFactory.getLogger("mao.logback.HelloWorld");
        logger.debug("debug ...");
    }

    //打印日志内部状态
    @Test
    public void test2()
    {
        Logger logger = LoggerFactory.getLogger("mao.logback.HelloWorld");
        logger.debug("debug ...");
        // 打印内部的状态
        LoggerContext lc = (LoggerContext)
LoggerFactory.getILoggerFactory();
        StatusPrinter.print(lc);
    }

    /*
     * 日志输出级别：ERROR > WARN > INFO > DEBUG > TRACE
     * */

    //测试默认的日志输出级别
    @Test
    public void test3()
    {
        Logger logger = LoggerFactory.getLogger("mao.logback.HelloWorld");
        logger.error("error ...");
        logger.warn("warn ...");
        logger.info("info ...");
        logger.debug("debug ...");
        //因为默认的输出级别为debug，所以这一条日志不会输出
        logger.trace("trace ...");
    }

    //设置日志输出级别
    @Test
```

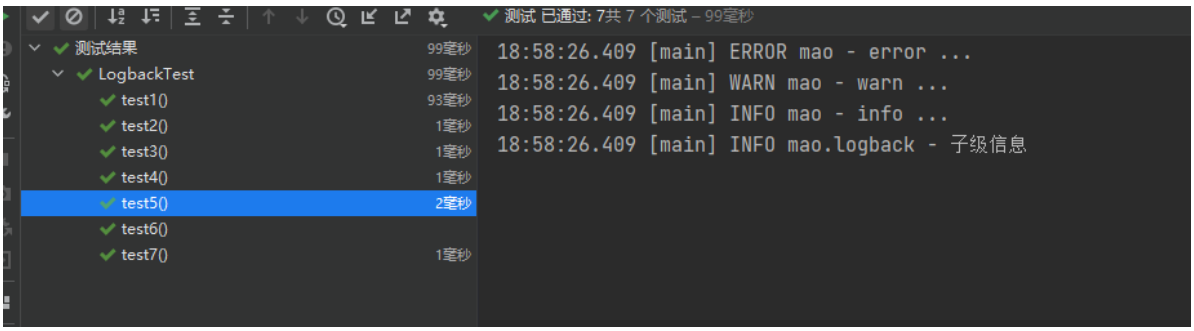
```
64      public void test4()
65      {
66          ch.qos.logback.classic.Logger logger =
(ch.qos.logback.classic.Logger)
LoggerFactory.getLogger("mao.logback.HelloWorld");
67          logger.setLevel(Level.WARN);
68          logger.error("error ...");
69          logger.warn("warn ...");
70          logger.info("info ...");
71          logger.debug("debug ...");
72          logger.trace("trace ...");
73      }
74
75      //测试Logger的继承
76      @Test
77      public void test5()
78      {
79          ch.qos.logback.classic.Logger logger =
80                  (ch.qos.logback.classic.Logger)
LoggerFactory.getLogger("mao");
81          logger.setLevel(Level.INFO);
82          logger.error("error ...");
83          logger.warn("warn ...");
84          logger.info("info ...");
85          logger.debug("debug ...");
86          logger.trace("trace ...");
87
88          // "mao.logback" 会继承 "mao" 的有效级别
89          Logger barLogger = LoggerFactory.getLogger("mao.logback");
90          // 这条日志会打印，因为 INFO >= INFO
91          barLogger.info("子级信息");
92          // 这条日志不会打印，因为 DEBUG < INFO
93          barLogger.debug("子级调试信息");
94      }
95
96      //Logger获取，根据同一个名称获得的logger都是同一个实例
97      @Test
98      public void test6()
99      {
100         Logger logger1 = LoggerFactory.getLogger("mao");
101         Logger logger2 = LoggerFactory.getLogger("mao");
102         System.out.println(logger1 == logger2);
103     }
104
105     //参数化日志
106     @Test
107     public void test7()
108     {
109         Logger logger = LoggerFactory.getLogger("mao");
110         logger.debug("hello {}", "world");
111     }
112 }
```

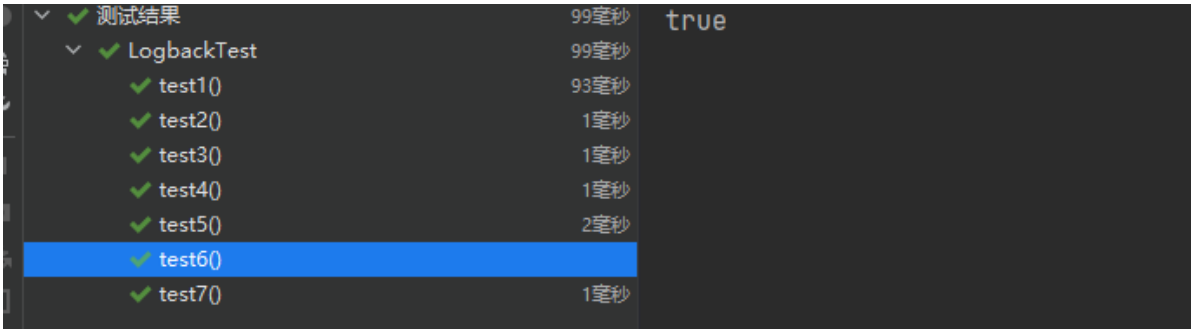

测试结果 99毫秒
LogbackTest 99毫秒
test1() 93毫秒
test2() 1毫秒
test3() 1毫秒
test4() 1毫秒
test5() 2毫秒
test6()
test7() 1毫秒
18:58:26.389 [main] DEBUG mao.logback.HelloWorld - debug ...

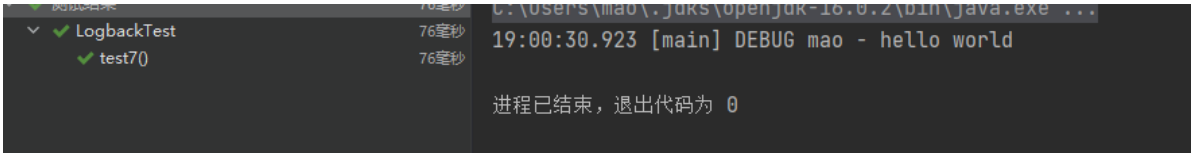

18:58:26.402 [main] DEBUG mao.logback.HelloWorld - debug ...
18:58:26,336 |-INFO in ch.qos.logback.classic.LoggerContext[default] - This is logback-classic version 1.3.4
18:58:26,361 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback-test.xml]
18:58:26,362 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback.xml]
18:58:26,363 |-INFO in ch.qos.logback.classic.BasicConfigurator@39529185 - Setting up default configuration.


测试结果 99毫秒
LogbackTest 99毫秒
test1() 93毫秒
test2() 1毫秒
test3() 1毫秒
test4() 1毫秒
test5() 2毫秒
test6()
test7() 1毫秒
18:58:26.405 [main] ERROR mao.logback.HelloWorld - error ...
18:58:26.405 [main] WARN mao.logback.HelloWorld - warn ...
18:58:26.405 [main] INFO mao.logback.HelloWorld - info ...
18:58:26.405 [main] DEBUG mao.logback.HelloWorld - debug ...


测试 已通过: 7共 7 个测试 – 99毫秒
测试结果 99毫秒
LogbackTest 99毫秒
test1() 93毫秒
test2() 1毫秒
test3() 1毫秒
test4() 1毫秒
test5() 2毫秒
test6()
test7() 1毫秒
18:58:26.407 [main] ERROR mao.logback.HelloWorld - error ...
18:58:26.407 [main] WARN mao.logback.HelloWorld - warn ...

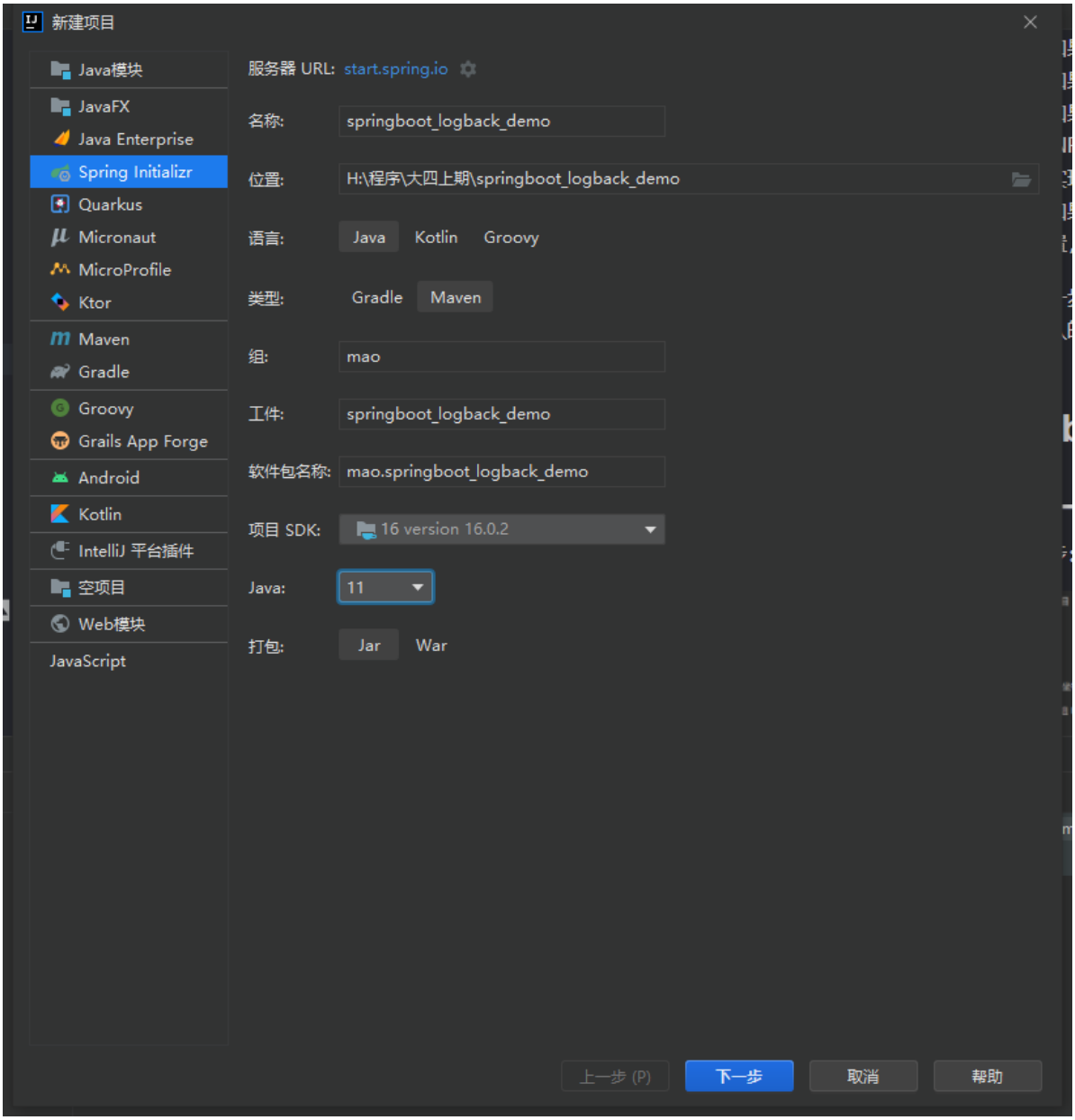

测试 已通过: 7共 7 个测试 – 99毫秒
测试结果 99毫秒
LogbackTest 99毫秒
test1() 93毫秒
test2() 1毫秒
test3() 1毫秒
test4() 1毫秒
test5() 2毫秒
test6()
test7() 1毫秒
18:58:26.409 [main] ERROR mao - error ...
18:58:26.409 [main] WARN mao - warn ...
18:58:26.409 [main] INFO mao - info ...
18:58:26.409 [main] INFO mao.logback - 子级信息





案例二

第一步：创建springboot工程springboot_logback_demo



第二步：修改pom文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>mao</groupId>
```

```
    <artifactId>springboot_logback_demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>springboot_logback_demo</name>
    <description>springboot_logback_demo</description>
    <properties>
        <java.version>11</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <!--logback-->
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.2.3</version>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-core</artifactId>
            <version>1.2.3</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>

```

第三步：在resources下编写logback配置文件logback-base.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<included>
    <contextName>logback</contextName>
```

```
    <!--
      name的值是变量的名称，value的值时变量定义的值
      定义变量后，可以使“${}”来使用变量
   -->
    <property name="log.path" value="./logs"/>

    <!-- 彩色日志 -->
    <!-- 彩色日志依赖的渲染类 -->
    <conversionRule
            conversionWord="clr"

 converterClass="org.springframework.boot.logging.logback.ColorConverter"/>
    <conversionRule
            conversionWord="wex"

 converterClass="org.springframework.boot.logging.logback.WhitespaceThrowablProxyConverter"/>
    <conversionRule conversionWord="wEx"

 converterClass="org.springframework.boot.logging.logback.ExtendedWhitespaceThrowableProxyConverter"/>
    <!-- 彩色日志格式 -->
    <property name="CONSOLE_LOG_PATTERN"
              value="${CONSOLE_LOG_PATTERN:-%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint} %clr(${LOG_LEVEL_PATTERN:-%5p}) %clr(${PID:- }){magenta} %clr(---){faint} %clr([%15.15t]){faint} %clr(%-40.40logger{39}){cyan} %clr(:){faint} %m%n${LOG_EXCEPTION_CONVERSION_WORD:-%wEx}}"/>

    <!--输出到控制台-->
    <appender name="LOG_CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <Pattern>${CONSOLE_LOG_PATTERN}</Pattern>
            <!-- 设置字符集 -->
            <charset>UTF-8</charset>
        </encoder>
    </appender>

    <!--输出到文件-->
    <appender name="LOG_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- 正在记录的日志文件的路径及文件名 -->
        <file>${log.path}/logback.log</file>
        <!--日志文件输出格式-->
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>
            <charset>UTF-8</charset>
        </encoder>
        <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- 每天日志归档路径以及格式 -->
            <fileNamePattern>${log.path}/info/log-info-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>10MB</maxFileSize>
```

```
            </timeBasedFileNamingAndTriggeringPolicy>
            <!--日志文件保留天数-->
            <maxHistory>365</maxHistory>
        </rollingPolicy>
    </appender>
</included>
```

第四步：在resources下编写logback配置文件logback-spring.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <!--引入其他配置文件-->
    <include resource="logback-base.xml"/>
    <!--
    <logger>用来设置某一个包或者具体的某一个类的日志打印级别、
    以及指定<appender>。<logger>仅有一个name属性，
    一个可选的level和一个可选的addtivity属性。
    name:用来指定受此logger约束的某一个包或者具体的某一个类。
    level:用来设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL 和 OFF，
          如果未设置此属性，那么当前logger将会继承上级的级别。
    addtivity:是否向上级logger传递打印信息。默认是true。
     -->

    <!--开发环境-->
<!--    <springProfile name="dev">-->
<!--        <logger name="包名" additivity="false" level="debug">-->
<!--            <appender-ref ref="LOG_CONSOLE"/>-->
<!--        </logger>-->
<!--    </springProfile>-->
<!--    &lt;!&ndash;生产环境&ndash;&gt;-->
<!--    <springProfile name="pro">-->
<!--        <logger name="包名" additivity="false" level="info">-->
<!--            <appender-ref ref="LOG_FILE"/>-->
<!--        </logger>-->
<!--    </springProfile>-->

    <!--
    root节点是必选节点，用来指定最基础的日志输出级别，只有一个level属性
    level:设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL 和 OFF
    默认是DEBUG
    可以包含零个或多个元素，标识这个appender将会添加到这个logger。
    -->
    <root level="info">
        <appender-ref ref="LOG_CONSOLE"/>
        <appender-ref ref="LOG_FILE"/>
    </root>
</configuration>
```

第五步：编写application.yml文件

```yaml
logging:
  config: classpath:logback-spring.xml

spring:
  profiles:
    active: dev
```

第六步：创建并编写UserController

```java
package mao.springboot_logback_demo.controller;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * Project name(项目名称)：springboot_logback_demo
 * Package(包名): mao.springboot_logback_demo.controller
 * Class(类名): UserController
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 19:24
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@RestController
@RequestMapping("/user")
public class UserController
{
    private static final Logger log =
LoggerFactory.getLogger(UserController.class);

    @GetMapping("/get")
    public String get()
    {
        log.trace("trace...");
        log.debug("debug...");
        log.info("info...");
        log.warn("warn...");
```

```
35          log.error("error...");
36          return "OK";
37      }
38  }
```

第七步：启动并访问

http://localhost:8080/user/get

```
2022-10-31 19:35:08.917  INFO 15552 --- [           main] m.s.SpringbootLogbackDemoApplication     : Started SpringbootLogbackDemoApplicatio
2022-10-31 19:36:14.275  INFO 15552 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet '
2022-10-31 19:36:14.276  INFO 15552 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2022-10-31 19:36:14.276  INFO 15552 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 0 ms
2022-10-31 19:36:14.290  INFO 15552 --- [nio-8080-exec-1] m.s.controller.UserController            : info...
2022-10-31 19:36:14.290  WARN 15552 --- [nio-8080-exec-1] m.s.controller.UserController            : warn...
2022-10-31 19:36:14.290 ERROR 15552 --- [nio-8080-exec-1] m.s.controller.UserController            : error...
```

可以看到控制台已经开始输出日志信息。

修改application.yml文件中的开发模式为pro，重启项目这日志输出到了文件中。

Spring Event

介绍

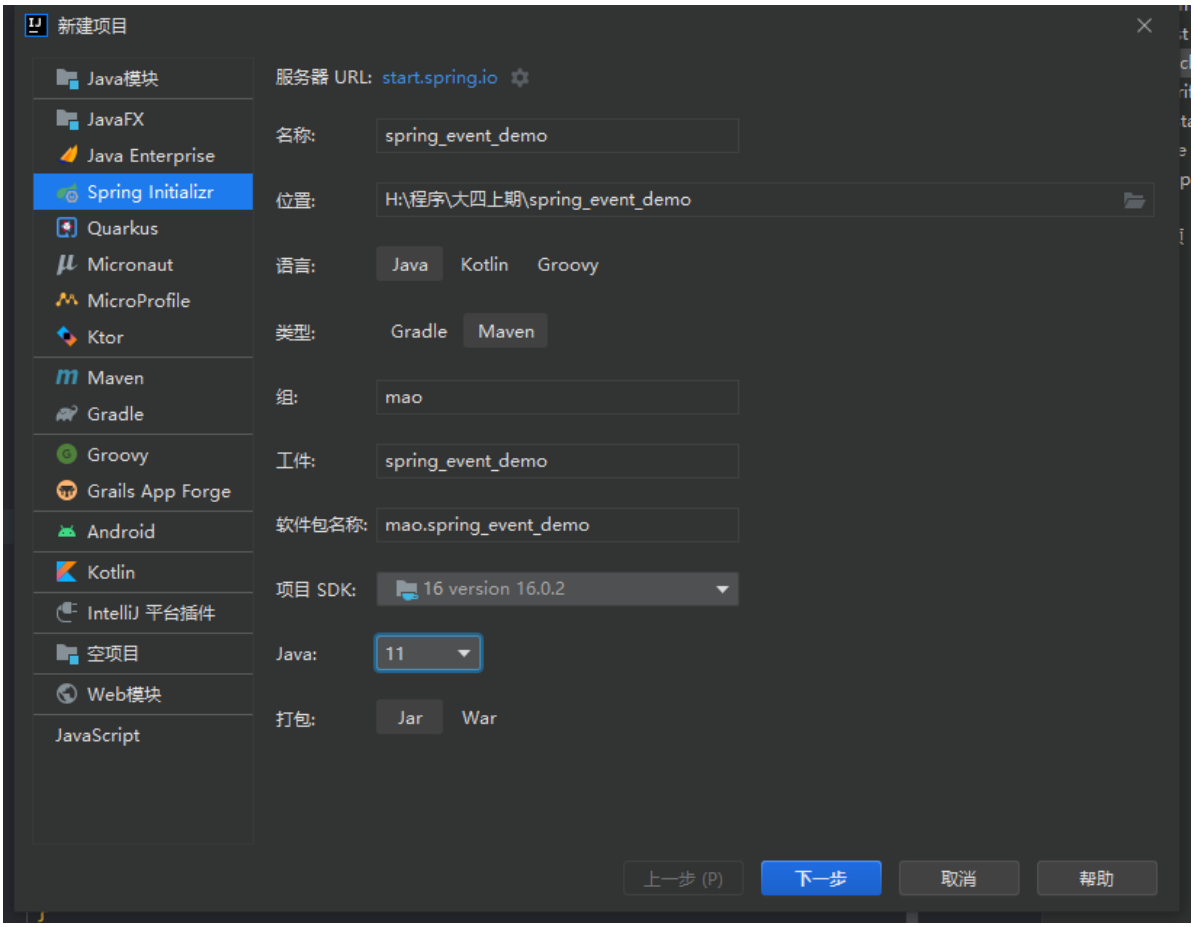
Spring Event是Spring的事件通知机制，可以将相互耦合的代码解耦，从而方便功能的修改与添加。Spring Event是监听者模式的一个具体实现。

监听者模式包含了监听者Listener、事件Event、事件发布者EventPublish，过程就是EventPublish发布一个事件，被监听者捕获到，然后执行事件相应的方法。

Spring Event的相关API在spring-context包中。

Spring Event入门案例

第一步：创建工程spring_event_demo



pom文件：

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>mao</groupId>
    <artifactId>spring_event_demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>spring_event_demo</name>
    <description>spring_event_demo</description>
    <properties>
```

```
        <java.version>11</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

第二步：创建OptLogDTO类，用于封装操作日志信息

```
package mao.spring_event_demo.entity;

/**
 * Project name(项目名称)：spring_event_demo
 * Package(包名): mao.spring_event_demo.entity
 * Class(类名): OptLogDTO
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 20:01
 * Version(版本): 1.0
 * Description(描述)： 无
 */


public class OptLogDTO
{
    /**
     * 操作IP
     */
    private String requestIp;
```

```
23
24      /**
25       * 日志类型 LogType{OPT:操作类型;EX:异常类型}
26       */
27      private String type;
28
29      /**
30       * 操作人
31       */
32      private String userName;
33
34      /**
35       * 操作描述
36       */
37      private String description;
38
39      /**
40       * Instantiates a new Opt log dto.
41       */
42      public OptLogDTO()
43      {
44
45      }
46
47      /**
48       * Instantiates a new Opt log dto.
49       *
50       * @param requestIp   the request ip
51       * @param type        the type
52       * @param userName    the user name
53       * @param description the description
54       */
55      public OptLogDTO(String requestIp, String type, String userName, String
    description)
56      {
57          this.requestIp = requestIp;
58          this.type = type;
59          this.userName = userName;
60          this.description = description;
61      }
62
63      /**
64       * Gets request ip.
65       *
66       * @return the request ip
67       */
68      public String getRequestIp()
69      {
70          return requestIp;
71      }
72
73      /**
74       * Sets request ip.
75       *
76       * @param requestIp the request ip
77       */
78      public void setRequestIp(String requestIp)
79      {
```

```
80          this.requestIp = requestIp;
81      }
82
83      /**
84       * Gets type.
85       *
86       * @return the type
87       */
88      public String getType()
89      {
90          return type;
91      }
92
93      /**
94       * Sets type.
95       *
96       * @param type the type
97       */
98      public void setType(String type)
99      {
100         this.type = type;
101     }
102
103     /**
104      * Gets user name.
105      *
106      * @return the user name
107      */
108     public String getUserName()
109     {
110         return userName;
111     }
112
113     /**
114      * Sets user name.
115      *
116      * @param userName the user name
117      */
118     public void setUserName(String userName)
119     {
120         this.userName = userName;
121     }
122
123     /**
124      * Gets description.
125      *
126      * @return the description
127      */
128     public String getDescription()
129     {
130         return description;
131     }
132
133     /**
134      * Sets description.
135      *
136      * @param description the description
137      */
```

```
    public void setDescription(String description)
    {
        this.description = description;
    }

    @Override
    public boolean equals(Object o)
    {
        if (this == o)
        {
            return true;
        }
        if (o == null || getClass() != o.getClass())
        {
            return false;
        }

        OptLogDTO optLogDTO = (OptLogDTO) o;

        if (getRequestIp() != null ? !getRequestIp().equals(optLogDTO.getRequestIp()) : optLogDTO.getRequestIp() != null)
        {
            return false;
        }
        if (getType() != null ? !getType().equals(optLogDTO.getType()) : optLogDTO.getType() != null)
        {
            return false;
        }
        if (getUserName() != null ? !getUserName().equals(optLogDTO.getUserName()) : optLogDTO.getUserName() != null)
        {
            return false;
        }
        return getDescription() != null ? getDescription().equals(optLogDTO.getDescription()) : optLogDTO.getDescription() == null;
    }

    @Override
    public int hashCode()
    {
        int result = getRequestIp() != null ? getRequestIp().hashCode() : 0;
        result = 31 * result + (getType() != null ? getType().hashCode() : 0);
        result = 31 * result + (getUserName() != null ? getUserName().hashCode() : 0);
        result = 31 * result + (getDescription() != null ? getDescription().hashCode() : 0);
        return result;
    }

    @Override
    public String toString()
    {
```

```
        final StringBuffer stringBuffer = new StringBuffer("OptLogDTO{");
        stringBuffer.append("requestIp='").append(requestIp).append('\'');
        stringBuffer.append(", type='").append(type).append('\'');
        stringBuffer.append(", userName='").append(userName).append('\'');
        stringBuffer.append(",
 description='").append(description).append('\'');
        stringBuffer.append('}');
        return stringBuffer.toString();
    }
}

```

第三步：创建事件类SysLogEvent

```
package mao.spring_event_demo.event;

import mao.spring_event_demo.entity.OptLogDTO;
import org.springframework.context.ApplicationEvent;

/**
 * Project name(项目名称)：spring_event_demo
 * Package(包名): mao.spring_event_demo.event
 * Class(类名): SysLogEvent
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 20:03
 * Version(版本): 1.0
 * Description(描述)： 无
 */

public class SysLogEvent extends ApplicationEvent
{
    public SysLogEvent(OptLogDTO optLogDTO)
    {
        super(optLogDTO);
    }
}
```

第四步：创建监听器类SysLogListener

```
package mao.spring_event_demo.listener;

```

```java
import mao.spring_event_demo.entity.OptLogDTO;
import mao.spring_event_demo.event.SysLogEvent;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.event.EventListener;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;

/**
 * Project name(项目名称)：spring_event_demo
 * Package(包名): mao.spring_event_demo.listener
 * Class(类名): SysLogListener
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 20:05
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@Component
public class SysLogListener
{
    private static final Logger log =
LoggerFactory.getLogger(SysLogListener.class);

    @Async
    @EventListener(SysLogEvent.class)
    public void saveSysLog(SysLogEvent event)
    {
        OptLogDTO sysLog = (OptLogDTO) event.getSource();
        long id = Thread.currentThread().getId();
        log.info("监听到日志操作事件：" + sysLog + " 线程id：" + id);
        //将日志信息保存到数据库或者其它地方

    }

    @PostConstruct
    public void init()
    {
        log.info("初始化SysLogListener");
    }
}

```

第五步：创建Controller，用于发布事件

```java
package mao.spring_event_demo.controller;

import mao.spring_event_demo.entity.OptLogDTO;
import mao.spring_event_demo.event.SysLogEvent;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationEvent;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * Project name(项目名称)：spring_event_demo
 * Package(包名): mao.spring_event_demo.controller
 * Class(类名): UserController
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 20:09
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@RestController
@RequestMapping("/user")
public class UserController
{
    @Autowired
    private ApplicationContext applicationContext;

    private static final Logger log =
LoggerFactory.getLogger(UserController.class);

    @GetMapping("/getUser")
    public String getUser()
    {
        //构造操作日志信息
        OptLogDTO logInfo = new OptLogDTO();
        logInfo.setRequestIp("127.0.0.1");
        logInfo.setUserName("admin");
        logInfo.setType("OPT");
        logInfo.setDescription("查询用户信息");

        //构造事件对象
        ApplicationEvent event = new SysLogEvent(logInfo);

        //发布事件
        applicationContext.publishEvent(event);

        long id = Thread.currentThread().getId();
        log.info("发布事件,线程id：" + id);
        return "OK";
    }
}

```

第六步：在启动类上添加EnableAsync注解

```java
package mao.spring_event_demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableAsync;

@SpringBootApplication
@EnableAsync
public class SpringEventDemoApplication
{

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

}
```

第七步：启动程序

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.1)

2022-10-31 20:28:16.647  INFO 11836 --- [           main] m.s.SpringEventDemoApplication           : Starting SpringEventDemoApplication using Java 16.0.2 on mao with PID 11836 (H:\程序\大四上期\spring_event_demo\target\classes started by mao in H:\程序\大四上期\spring_event_demo)
2022-10-31 20:28:16.649  INFO 11836 --- [           main] m.s.SpringEventDemoApplication           : No active profile set, falling back to 1 default profile: "default"
2022-10-31 20:28:17.281  INFO 11836 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2022-10-31 20:28:17.287  INFO 11836 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
```

```
2022-10-31 20:28:17.287  INFO 11836 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.64]
2022-10-31 20:28:17.357  INFO 11836 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-10-31 20:28:17.358  INFO 11836 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 672 ms
2022-10-31 20:28:17.393  INFO 11836 --- [           main] m.s.listener.SysLogListener              : 初始化SysLogListener
2022-10-31 20:28:17.632  INFO 11836 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-10-31 20:28:17.641  INFO 11836 --- [           main] m.s.SpringEventDemoApplication           : Started SpringEventDemoApplication in 1.259 seconds (JVM running for 1.697)
```

第八步：访问

http://localhost:8080/user/getUser

```
2022-10-31 20:28:17.393  INFO 11836 --- [           main] m.s.listener.SysLogListener              : 初始化SysLogListener
2022-10-31 20:28:17.632  INFO 11836 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-10-31 20:28:17.641  INFO 11836 --- [           main] m.s.SpringEventDemoApplication           : Started SpringEventDemoApplication in 1.259 seconds (JVM running for 1.697)
2022-10-31 20:28:30.387  INFO 11836 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-10-31 20:28:30.387  INFO 11836 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2022-10-31 20:28:30.388  INFO 11836 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
2022-10-31 20:28:30.404  INFO 11836 --- [nio-8080-exec-1] m.s.controller.UserController            : 发布事件,线程id：35
2022-10-31 20:28:30.412  INFO 11836 --- [         task-1] m.s.listener.SysLogListener              : 监听到日志操作事件：OptLogDTO{requestIp='127.0.0.1', type='OPT', userName='admin', description='查询用户信息'} 线程id：53
2022-10-31 20:28:30.572  INFO 11836 --- [nio-8080-exec-2] m.s.controller.UserController            : 发布事件,线程id：36
```

```
2022-10-31 20:28:30.572  INFO 11836 --- [         task-2] m.s.listener.SysLogListener              : 监听到日志操作事件：OptLogDTO{requestIp='127.0.0.1', type='OPT', userName='admin', description='查询用户信息'} 线程id：54
2022-10-31 20:28:30.789  INFO 11836 --- [nio-8080-exec-3] m.s.controller.UserController            : 发布事件,线程id：37
2022-10-31 20:28:30.789  INFO 11836 --- [         task-3] m.s.listener.SysLogListener              : 监听到日志操作事件：OptLogDTO{requestIp='127.0.0.1', type='OPT', userName='admin', description='查询用户信息'} 线程id：55
2022-10-31 20:28:31.936  INFO 11836 --- [nio-8080-exec-4] m.s.controller.UserController            : 发布事件,线程id：38
2022-10-31 20:28:31.936  INFO 11836 --- [         task-4] m.s.listener.SysLogListener              : 监听到日志操作事件：OptLogDTO{requestIp='127.0.0.1', type='OPT', userName='admin', description='查询用户信息'} 线程id：56
2022-10-31 20:28:32.369  INFO 11836 --- [nio-8080-exec-5] m.s.controller.UserController            : 发布事件,线程id：39
2022-10-31 20:28:32.370  INFO 11836 --- [         task-5] m.s.listener.SysLogListener              : 监听到日志操作事件：OptLogDTO{requestIp='127.0.0.1', type='OPT', userName='admin', description='查询用户信息'} 线程id：57
```

自定义spring boot starter

tools-log的开发步骤为：

1、定义日志操作事件类SysLogEvent

2、定义@SysLog注解，用于在Controller的方法上标注当前方法需要进行操作日志的保存处理

3、定义切面类SysLogAspect

4、在切面类SysLogAspect中定义切点，拦截Controller中添加@SysLog注解的方法

5、在切面类SysLogAspect中定义前置通知，在前置通知方法recordLog中收集操作日志相关信息封装为OptLogDTO对象并保存到ThreadLocal中

6、在切面类SysLogAspect中定义后置通知，在后置通知方法doAfterReturning中通过ThreadLocal 获取OptLogDTO并继续设置其他的操作信息到OptLogDTO

7、在切面类SysLogAspect的后置通知方法doAfterReturning中发布事件SysLogEvent

8、定义监听器SysLogListener，监听日志发布事件SysLogEvent

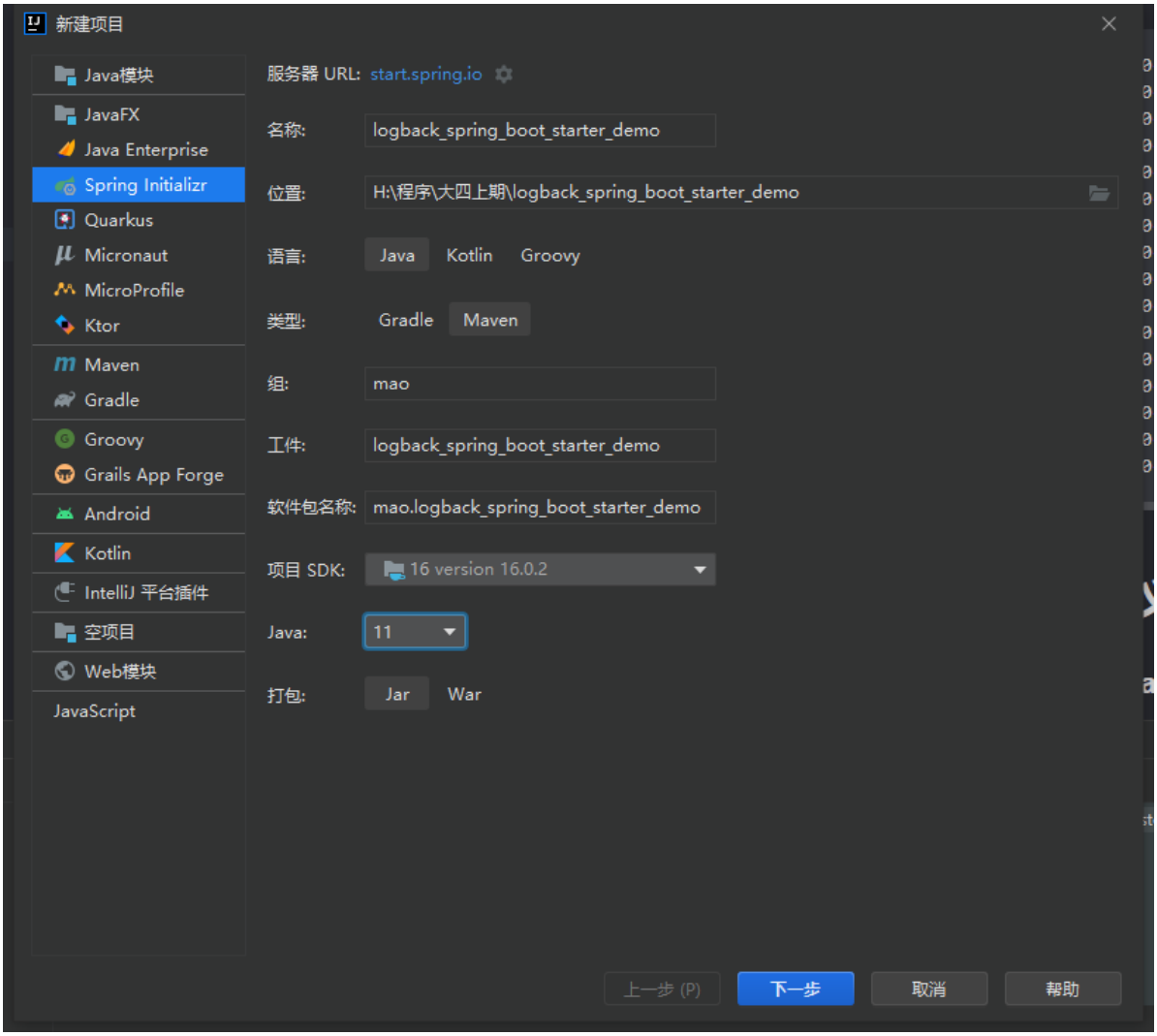
9、定义配置类LogAutoConfiguration，用于自动配置切面SysLogAspect对象

10、定义starter所需的META-INF/spring.factories文件，并配置自动配置类LogAutoConfiguration

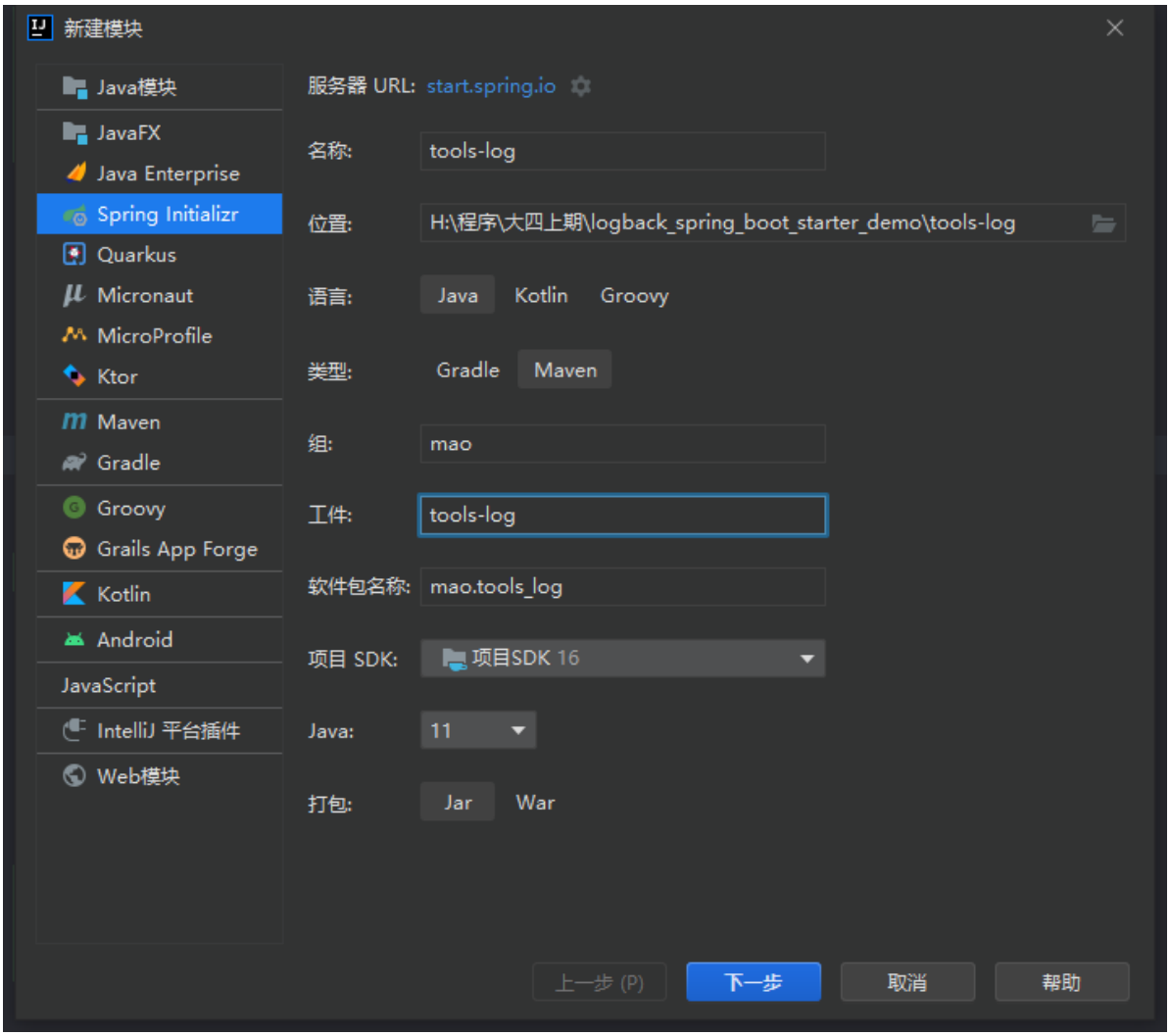
开发starter

第一步：初始化项目

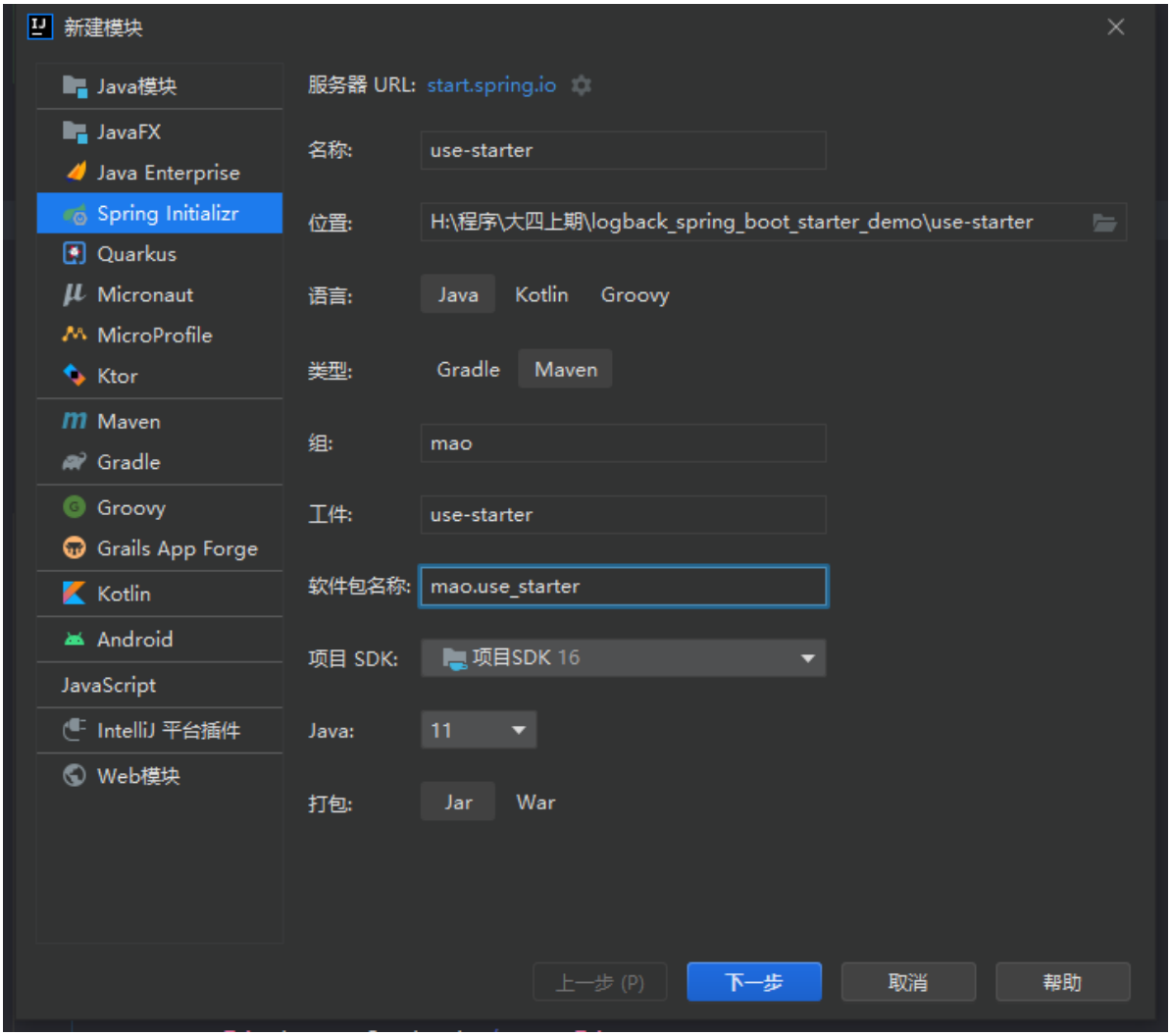
创建父工程logback_spring_boot_starter_demo



创建子工程tools-log



创建子工程use-starter



第二步：修改pom文件

父工程logback_spring_boot_starter_demo的pom文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <groupId>mao</groupId>
    <artifactId>logback_spring_boot_starter_demo</artifactId>
```

```
    <version>0.0.1-SNAPSHOT</version>
    <name>logback_spring_boot_starter_demo</name>
    <description>logback_spring_boot_starter_demo</description>
    <packaging>pom</packaging>

    <properties>
        <java.version>11</java.version>
    </properties>

    <dependencies>

    </dependencies>

    <modules>
        <module>tools-log</module>
        <module>use-starter</module>
    </modules>

    <dependencyManagement>
        <dependencies>

        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>

```

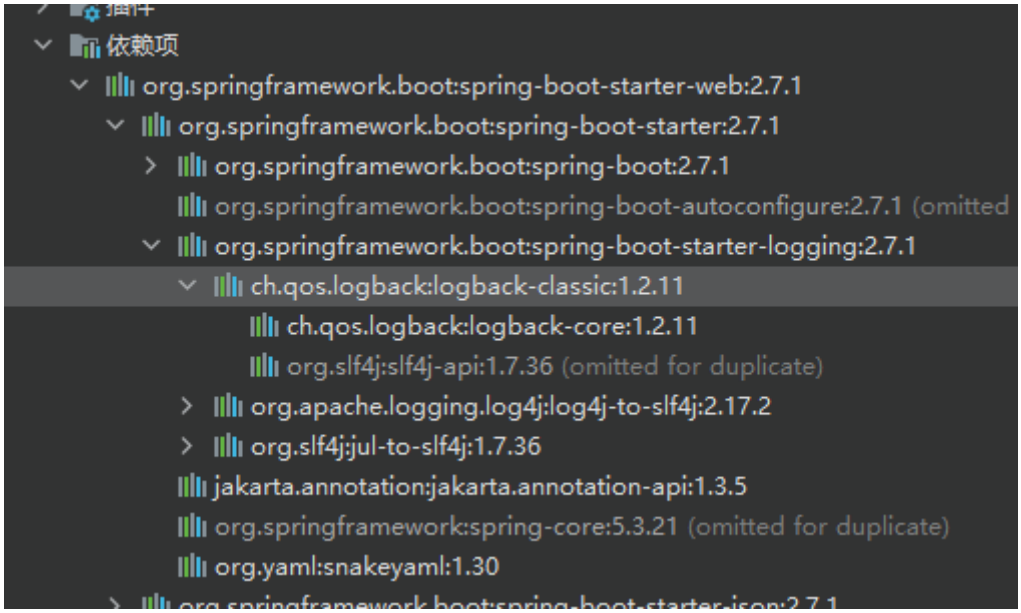
子工程tools-log的pom文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <artifactId>logback_spring_boot_starter_demo</artifactId>
        <groupId>mao</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>

    <artifactId>tools-log</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>tools-log</name>
    <description>tools-log</description>
```

```
    <properties>

    </properties>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!--logback-->
<!--        <dependency>-->
<!--            <groupId>ch.qos.logback</groupId>-->
<!--            <artifactId>logback-classic</artifactId>-->
<!--            <version>1.2.3</version>-->
<!--        </dependency>-->
<!--        <dependency>-->
<!--            <groupId>ch.qos.logback</groupId>-->
<!--            <artifactId>logback-core</artifactId>-->
<!--            <version>1.2.3</version>-->
<!--        </dependency>-->


        <dependency>
            <groupId>org.lionsoul</groupId>
            <artifactId>ip2region</artifactId>
            <version>1.7.2</version>
        </dependency>
        <dependency>
            <groupId>eu.bitwalker</groupId>
            <artifactId>UserAgentUtils</artifactId>
            <version>1.21</version>
        </dependency>
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.11.0</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>cn.hutool</groupId>
            <artifactId>hutool-all</artifactId>
            <version>5.1.0</version>
        </dependency>

        <dependency>
            <groupId>com.github.xiaoymin</groupId>
            <artifactId>knife4j-spring-boot-starter</artifactId>
            <version>2.0.1</version>
```

```
        </dependency>

        <!--阿里巴巴的FastJson json解析-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.79</version>
        </dependency>


        <!--spring boot starter开发依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-autoconfigure</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-configuration-processor</artifactId>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <skip>true</skip>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

不需要在导入logback了，因为spring-boot-starter-web已经包含了logback



工程use-starter的pom文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <artifactId>logback_spring_boot_starter_demo</artifactId>
        <groupId>mao</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>

    <artifactId>use-starter</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>use-starter</name>
    <description>use-starter</description>

    <properties>

    </properties>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
```

```
            <scope>test</scope>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

第三步：编写工具类AddressUtil

```
package mao.tools_log.utils;

import org.lionsoul.ip2region.DataBlock;
import org.lionsoul.ip2region.DbConfig;
import org.lionsoul.ip2region.DbSearcher;
import org.lionsoul.ip2region.Util;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import cn.hutool.core.io.resource.ResourceUtil;
import cn.hutool.core.util.StrUtil;
import org.apache.commons.io.FileUtils;

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.lang.reflect.Method;


/**
 * Project name(项目名称)：logback_spring_boot_starter_demo
 * Package(包名): mao.tools_log.utils
 * Class(类名): AddressUtil
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 22:17
 * Version(版本): 1.0
 * Description(描述)： 解析ip地址的工具类
 */

```

```
public class AddressUtil
{
    private static final String JAVA_TEMP_DIR = "java.io.tmpdir";

    private static final Logger log =
LoggerFactory.getLogger(AddressUtil.class);

    static DbConfig config = null;
    static DbSearcher searcher = null;


    /**
     * 根据ip查询地址
     *
     * @param ip ip地址
     * @return {@link String}
     */
    /*public static String getCityInfo(String ip)
    {
        DbSearcher searcher = null;
        try
        {
            String dbPath =
AddressUtil.class.getResource("/ip2region/ip2region.db").getPath();
            File file = new File(dbPath);
            if (!file.exists())
            {
                String tmpDir =
System.getProperties().getProperty(JAVA_TEMP_DIR);
                dbPath = tmpDir + "ip2region.db";
                file = new File(dbPath);
                String classPath = "classpath:ip2region/ip2region.db";
                InputStream resourceAsStream =
ResourceUtil.getStreamSafe(classPath);
                if (resourceAsStream != null)
                {
                    FileUtils.copyInputStreamToFile(resourceAsStream,
file);
                }
            }
            DbConfig config = new DbConfig();
            searcher = new DbSearcher(config, file.getPath());
            Method method = searcher.getClass().getMethod("btreeSearch",
String.class);
            if (!Util.isIpAddress(ip))
            {
                log.error("Error: Invalid ip address");
            }
            DataBlock dataBlock = (DataBlock) method.invoke(searcher, ip);
            return dataBlock.getRegion();
        }
        catch (Exception e)
        {
            log.error("获取地址信息异常，", e);
            return StrUtil.EMPTY;
        }
        finally
        {
```

```
            if (searcher != null)
            {
                try
                {
                    searcher.close();
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
            }
        }
    }*/

    /*
     * 初始化IP库
     */
    static
    {
        try
        {
            // 因为jar无法读取文件,复制创建临时文件
//            String tmpDir = System.getProperty("user.dir") +
File.separator + "temp";
//            String dbPath = tmpDir + File.separator + "ip2region.db";
//            log.info("init ip region db path [{}]", dbPath);
//            File file = new File(dbPath);
//
FileUtils.copyInputStreamToFile(AddressUtil.class.getClassLoader().getResou
rceAsStream("ip2region/ip2region.db"), file);
            String dbPath =
AddressUtil.class.getResource("/ip2region/ip2region.db").getPath();
            File file = new File(dbPath);
            if (!file.exists())
            {
                String tmpDir =
System.getProperties().getProperty(JAVA_TEMP_DIR);
                dbPath = tmpDir + "ip2region.db";
                file = new File(dbPath);
                String classPath = "classpath:ip2region/ip2region.db";
                InputStream resourceAsStream =
ResourceUtil.getStreamSafe(classPath);
                if (resourceAsStream != null)
                {
                    FileUtils.copyInputStreamToFile(resourceAsStream,
file);
                }
            }
            config = new DbConfig();
            searcher = new DbSearcher(config, dbPath);
            log.info("bean [{}]", config);
            log.info("bean [{}]", searcher);
        }
        catch (Exception e)
        {
            log.error("init ip region error:", e);
        }
    }
```

```
136
137
138     /**
139      * 解析ip
140      *
141      * @param ip ip地址
142      * @return {@link String}
143      */
144     public static String getRegion(String ip)
145     {
146         try
147         {
148             //db
149             if (searcher == null || StrUtil.isEmpty(ip))
150             {
151                 log.error("DbSearcher is null");
152                 return StrUtil.EMPTY;
153             }
154             long startTime = System.currentTimeMillis();
155             //查询算法
156             int algorithm = DbSearcher.MEMORY_ALGORITYM;
157             Method method = null;
158             switch (algorithm)
159             {
160                 case DbSearcher.BTREE_ALGORITHM:
161                     method = searcher.getClass().getMethod("btreeSearch",
String.class);
162                     break;
163                 case DbSearcher.BINARY_ALGORITHM:
164                     method = searcher.getClass().getMethod("binarySearch",
String.class);
165                     break;
166                 case DbSearcher.MEMORY_ALGORITYM:
167                     method = searcher.getClass().getMethod("memorySearch",
String.class);
168                     break;
169             }
170
171             DataBlock dataBlock = null;
172             if (!Util.isIpAddress(ip))
173             {
174                 log.warn("warning: Invalid ip address");
175             }
176             dataBlock = (DataBlock) method.invoke(searcher, ip);
177             String result = dataBlock.getRegion();
178             long endTime = System.currentTimeMillis();
179             log.debug("region use time[{}] result[{}]", endTime -
startTime, result);
180             return result;
181
182         }
183         catch (Exception e)
184         {
185             log.error("error:", e);
186         }
187         return StrUtil.EMPTY;
188     }
189
```

```
    public static void main(String[] args)
    {
        System.out.println(AddressUtil.getRegion("113.222.142.84"));
        System.out.println(AddressUtil.getRegion("113.221.141.84"));
        System.out.println(AddressUtil.getRegion("113.192.142.84"));
        System.out.println(AddressUtil.getRegion("113.224.142.84"));
        System.out.println(AddressUtil.getRegion("114.222.142.84"));
        System.out.println(AddressUtil.getRegion("115.222.142.84"));
        System.out.println(AddressUtil.getRegion("117.222.142.84"));
        System.out.println(AddressUtil.getRegion("119.222.142.84"));
        System.out.println(AddressUtil.getRegion("13.222.142.84"));
        System.out.println(AddressUtil.getRegion("14.222.142.84"));
        System.out.println(AddressUtil.getRegion("15.222.142.84"));
        System.out.println(AddressUtil.getRegion("16.222.142.84"));
    }
}
```

第四步：编写工具类LogUtil

```
package mao.tools_log.utils;

import mao.tools_log.annotation.SysLog;
import org.aspectj.lang.JoinPoint;

import java.io.PrintWriter;
import java.io.StringWriter;
import java.lang.reflect.Method;

/**
 * Project name(项目名称)：logback_spring_boot_starter_demo
 * Package(包名): mao.tools_log.utils
 * Class(类名): LogUtil
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 22:16
 * Version(版本): 1.0
 * Description(描述)： 无
 */

public class LogUtil
{
    /***
     * 获取操作信息
     * @param point JoinPoint对象
     * @return String
     */
    public static String getControllerMethodDescription(JoinPoint point)
    {
```

```
        try
        {
            // 获取连接点目标类名
            String targetName = point.getTarget().getClass().getName();
            // 获取连接点签名的方法名
            String methodName = point.getSignature().getName();
            //获取连接点参数
            Object[] args = point.getArgs();
            //根据连接点类的名字获取指定类
            Class targetClass = Class.forName(targetName);
            //获取类里面的方法
            Method[] methods = targetClass.getMethods();
            String description = "";
            for (Method method : methods)
            {
                if (method.getName().equals(methodName))
                {
                    Class[] clazzs = method.getParameterTypes();
                    if (clazzs.length == args.length)
                    {
                        description =
method.getAnnotation(SysLog.class).value();
                        break;
                    }
                }
            }
            return description;
        }
        catch (Exception e)
        {
            return "";
        }
    }


    /**
     * 获取堆栈信息
     *
     * @param throwable throwable
     * @return {@link String}
     */
    public static String getStackTrace(Throwable throwable)
    {
        StringWriter sw = new StringWriter();
        try (PrintWriter pw = new PrintWriter(sw))
        {
            throwable.printStackTrace(pw);
            return sw.toString();
        }
    }
}
```

第五步：编写工具类NumberHelper

```java
package mao.tools_log.utils;

import java.util.function.Function;


/**
 * 数字类型 帮助类
 */
public class NumberHelper
{

    private static <T, R> R valueOfDef(T t, Function<T, R> function, R def)
    {
        try
        {
            return function.apply(t);
        }
        catch (Exception e)
        {
            return def;
        }
    }

    public static Long longValueOfNil(String value)
    {
        return valueOfDef(value, Long::valueOf, null);
    }

    public static Long longValueOf0(String value)
    {
        return valueOfDef(value, Long::valueOf, 0L);
    }

    public static Long longValueOfNil(Object value)
    {
        return valueOfDef(value, (val) -> Long.valueOf(val.toString()),
null);
    }

    public static Long longValueOf0(Object value)
    {
        return valueOfDef(value, (val) -> Long.valueOf(val.toString()), 0L);
    }

    public static Boolean boolValueOf0(Object value)
    {
        return valueOfDef(value, (val) -> Boolean.valueOf(val.toString()),
false);
    }

    public static Integer intValueOfNil(String value)
    {
        return valueOfDef(value, Integer::valueOf, null);
```

```
    }

    public static Integer intValueOf0(String value)
    {
        return intValueOf(value, 0);
    }

    public static Integer intValueOf(String value, Integer def)
    {
        return valueOfDef(value, Integer::valueOf, def);
    }

    public static Integer intValueOfNil(Object value)
    {
        return valueOfDef(value, (val) -> Integer.valueOf(val.toString()), null);
    }

    public static Integer intValueOf0(Object value)
    {
        return valueOfDef(value, (val) -> Integer.valueOf(val.toString()), 0);
    }

    public static Integer getOrDef(Integer val, Integer def)
    {
        return val == null ? def : val;
    }

    public static Long getOrDef(Long val, Long def)
    {
        return val == null ? def : val;
    }

    public static Boolean getOrDef(Boolean val, Boolean def)
    {
        return val == null ? def : val;
    }

}
```

第六步：编写工具类StrHelper

```
package mao.tools_log.utils;

import java.net.URLDecoder;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;

import cn.hutool.core.util.StrUtil;
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;

/**
 * 字符串帮助类
 */

public class StrHelper
{

    private static final Logger log =
LoggerFactory.getLogger(StrHelper.class);

    public static String getObjectValue(Object obj)
    {
        return obj == null ? "" : obj.toString();
    }

    public static String encode(String value)
    {
        try
        {
            return URLEncoder.encode(value, StandardCharsets.UTF_8);
        }
        catch (Exception e)
        {
            return "";
        }
    }

    public static String decode(String value)
    {
        try
        {
            return URLDecoder.decode(value, StandardCharsets.UTF_8);
        }
        catch (Exception e)
        {
            return "";
        }
    }

    public static String getOrDef(String val, String def)
    {
        return StrUtil.isEmpty(val) ? def : val;
    }
}
```

第七步：编写类ApplicationLoggerInitializer

```
package mao.tools_log.init;

```

```
import org.springframework.context.ApplicationContextInitializer;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.core.env.ConfigurableEnvironment;

/**
 * Project name(项目名称)：logback_spring_boot_starter_demo
 * Package(包名): mao.tools_log.init
 * Class(类名): ApplicationLoggerInitializer
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 21:43
 * Version(版本): 1.0
 * Description(描述)：
 * <p>
 * 通过环境变量的形式注入 logging.file
 * 自动维护 Spring Boot Admin Logger Viewer
 */

public class ApplicationLoggerInitializer implements
ApplicationContextInitializer<ConfigurableApplicationContext>
{
    @Override
    public void initialize(ConfigurableApplicationContext
applicationContext)
    {
        ConfigurableEnvironment environment =
applicationContext.getEnvironment();
        String logBase = environment.getProperty("logging.path",
"/data/projects/logs");
        String appName = environment.getProperty("spring.application.name");
        // spring boot admin 直接加载日志
        System.setProperty("logging.file", String.format("%s/%s/root.log",
logBase, appName));

        // nacos的日志文件路径
        System.setProperty("nacos.logging.path", String.format("%s/%s",
logBase, appName));
        //这里设置了无效，跟启动时，传递 -
Dcom.alibaba.nacos.naming.log.level=warn 一样，可能是nacos的bug
//        System.setProperty("com.alibaba.nacos.naming.log.level", "warn");
//        System.setProperty("com.alibaba.nacos.config.log.level", "info");
    }
}
```

第八步：编写接口BaseExceptionCode

```
package mao.tools_log.exception.code;

/**
```

```
 *
 */
public interface BaseExceptionCode
{
    /**
     * 异常编码
     *
     * @return int
     */
    int getCode();

    /**
     * 异常消息
     *
     * @return String
     */
    String getMsg();
}
```

第九步：编写类ExceptionCode

```
package mao.tools_log.exception.code;


/**
 * 全局错误码 10000-15000
 * <p>
 * 预警异常编码     范围：  30000~34999
 * 标准服务异常编码 范围：35000~39999
 * 邮件服务异常编码 范围：40000~44999
 * 短信服务异常编码 范围：45000~49999
 * 权限服务异常编码 范围：50000-59999
 * 文件服务异常编码 范围：60000~64999
 * 日志服务异常编码 范围：65000~69999
 * 消息服务异常编码 范围：70000~74999
 * 开发者平台异常编码 范围：75000~79999
 * 搜索服务异常编码 范围：80000-84999
 * 共享交换异常编码 范围：85000-89999
 * 移动终端平台 异常码 范围：90000-94999
 * <p>
 * 安全保障平台     范围：          95000-99999
 * 软硬件平台 异常编码 范围：     100000-104999
 * 运维服务平台 异常编码 范围：   105000-109999
 * 统一监管平台异常 编码 范围：   110000-114999
 * 认证方面的异常编码   范围：115000-115999
 */
public enum ExceptionCode implements BaseExceptionCode
{

    //系统相关 start
    SUCCESS(0, "成功"),
```

```
    SYSTEM_BUSY(-1, "系统繁忙~请稍后再试~"),
    SYSTEM_TIMEOUT(-2, "系统维护中~请稍后再试~"),
    PARAM_EX(-3, "参数类型解析异常"),
    SQL_EX(-4, "运行SQL出现异常"),
    NULL_POINT_EX(-5, "空指针异常"),
    ILLEGALA_ARGUMENT_EX(-6, "无效参数异常"),
    MEDIA_TYPE_EX(-7, "请求类型异常"),
    LOAD_RESOURCES_ERROR(-8, "加载资源出错"),
    BASE_VALID_PARAM(-9, "统一验证参数异常"),
    OPERATION_EX(-10, "操作异常"),


    OK(200, "OK"),
    BAD_REQUEST(400, "错误的请求"),
    /**
     * {@code 401 Unauthorized}.
     *
     * @see <a href="http://tools.ietf.org/html/rfc7235#section-
3.1">HTTP/1.1: Authentication, section 3.1</a>
     */
    UNAUTHORIZED(401, "未经授权"),
    /**
     * {@code 404 Not Found}.
     *
     * @see <a href="http://tools.ietf.org/html/rfc7231#section-
6.5.4">HTTP/1.1: Semantics and Content, section 6.5.4</a>
     */
    NOT_FOUND(404, "没有找到资源"),
    METHOD_NOT_ALLOWED(405, "不支持当前请求类型"),

    TOO_MANY_REQUESTS(429, "请求超过次数限制"),
    INTERNAL_SERVER_ERROR(500, "内部服务错误"),
    BAD_GATEWAY(502, "网关错误"),
    GATEWAY_TIMEOUT(504, "网关超时"),
    //系统相关 end

    REQUIRED_FILE_PARAM_EX(1001, "请求中必须至少包含一个有效文件"),
    //jwt token 相关 start

    JWT_TOKEN_EXPIRED(40001, "会话超时，请重新登录"),
    JWT_SIGNATURE(40002, "不合法的token，请认真比对 token 的签名"),
    JWT_ILLEGAL_ARGUMENT(40003, "缺少token参数"),
    JWT_GEN_TOKEN_FAIL(40004, "生成token失败"),
    JWT_PARSER_TOKEN_FAIL(40005, "解析token失败"),
    JWT_USER_INVALID(40006, "用户名或密码错误"),
    JWT_USER_ENABLED(40007, "用户已经被禁用！"),
    //jwt token 相关 end

    ;

    private int code;
    private String msg;

    ExceptionCode(int code, String msg)
    {
        this.code = code;
        this.msg = msg;
    }
```

```
87
88      @Override
89      public int getCode()
90      {
91          return code;
92      }
93
94      @Override
95      public String getMsg()
96      {
97          return msg;
98      }
99
100
101     public ExceptionCode build(String msg, Object... param)
102     {
103         this.msg = String.format(msg, param);
104         return this;
105     }
106
107     public ExceptionCode param(Object... param)
108     {
109         msg = String.format(msg, param);
110         return this;
111     }
112 }
```

第十步：编写接口BaseException

```
1  package mao.tools_log.exception;
2
3  /**
4   * 异常接口类
5   */
6  public interface BaseException
7  {
8
9      /**
10      * 统一参数验证异常码
11      */
12     int BASE_VALID_PARAM = -9;
13
14     /**
15      * 返回异常信息
16      *
17      * @return String
18      */
19     String getMessage();
20
21     /**
22      * 返回异常编码
```

```
     *
     * @return int
     */
    int getCode();

}
```

第十一步：编写类BaseUncheckedException

```
package mao.tools_log.exception;

/**
 * 非运行期异常基类，所有自定义非运行时异常继承该类
 */
public class BaseUncheckedException extends RuntimeException implements
BaseException
{

    private static final long serialVersionUID = -778887391066124051L;

    /**
     * 异常信息
     */
    protected String message;

    /**
     * 具体异常码
     */
    protected int code;

    public BaseUncheckedException(int code, String message)
    {
        super(message);
        this.code = code;
        this.message = message;
    }

    public BaseUncheckedException(int code, String format, Object... args)
    {
        super(String.format(format, args));
        this.code = code;
        this.message = String.format(format, args);
    }


    @Override
    public String getMessage()
    {
        return message;
    }

```

```
    @Override
    public int getCode()
    {
        return code;
    }
}
```

第十二步：编写类BizException

```
package mao.tools_log.exception;


import mao.tools_log.exception.code.BaseExceptionCode;

/**
 * 业务异常
 * 用于在处理业务逻辑时，进行抛出的异常。
 */
public class BizException extends BaseUncheckedException
{

    private static final long serialVersionUID = -3843907364558373817L;

    public BizException(String message)
    {
        super(-1, message);
    }

    public BizException(int code, String message)
    {
        super(code, message);
    }

    public BizException(int code, String message, Object... args)
    {
        super(code, message, args);
    }

    /**
     * 实例化异常
     *
     * @param code    自定义异常编码
     * @param message 自定义异常消息
     * @param args    已定义异常参数
     * @return BizException
     */
    public static BizException wrap(int code, String message, Object...
args)
    {
        return new BizException(code, message, args);
    }
```

```
42
43      public static BizException wrap(String message, Object... args)
44      {
45          return new BizException(-1, message, args);
46      }
47
48      public static BizException validFail(String message, Object... args)
49      {
50          return new BizException(-9, message, args);
51      }
52
53      public static BizException wrap(BaseExceptionCode ex)
54      {
55          return new BizException(ex.getCode(), ex.getMsg());
56      }
57
58      @Override
59      public String toString()
60      {
61          return "BizException [message=" + message + ", code=" + code + "]";
62      }
63
64  }
```

第十三步：编写实体类OptLogDTO

```
1   package mao.tools_log.entity;
2
3   import java.time.LocalDateTime;
4
5
6   /**
7    * Project name(项目名称)：logback_spring_boot_starter_demo
8    * Package(包名): mao.tools_log.entity
9    * Class(类名): OptLogDTO
10   * Author(作者）: mao
11   * Author QQ：1296193245
12   * GitHub：https://github.com/maomao124/
13   * Date(创建日期)： 2022/10/31
14   * Time(创建时间)： 21:47
15   * Version(版本): 1.0
16   * Description(描述)： 无
17   */
18  public class OptLogDTO
19  {
20
21      private static final long serialVersionUID = 1L;
22
23      /**
24       * 操作IP
```

```
25          */
26         private String requestIp;
27
28         /**
29          * 日志类型
30          * #LogType{OPT:操作类型;EX:异常类型}
31          */
32         private String type;
33
34         /**
35          * 操作人
36          */
37         private String userName;
38
39         /**
40          * 操作描述
41          */
42         private String description;
43
44         /**
45          * 类路径
46          */
47         private String classPath;
48
49         /**
50          * 请求类型
51          */
52         private String actionMethod;
53
54         /**
55          * 请求地址
56          */
57         private String requestUri;
58
59         /**
60          * 请求类型
61          * #HttpMethod{GET:GET请求;POST:POST请求;PUT:PUT请求;DELETE:DELETE请
    求;PATCH:PATCH请求;
62          * TRACE:TRACE请求;HEAD:HEAD请求;OPTIONS:OPTIONS请求;}
63          */
64         private String httpMethod;
65
66         /**
67          * 请求参数
68          */
69         private String params;
70
71         /**
72          * 返回值
73          */
74         private String result;
75
76         /**
77          * 异常详情信息
78          */
79         private String exDesc;
80
81         /**
```

```
82          * 异常描述
83          */
84         private String exDetail;
85
86         /**
87          * 开始时间
88          */
89         private LocalDateTime startTime;
90
91         /**
92          * 完成时间
93          */
94         private LocalDateTime finishTime;
95
96         /**
97          * 消耗时间
98          */
99         private Long consumingTime;
100
101        /**
102         * 浏览器
103         */
104        private String ua;
105
106        /**
107         * 创建用户
108         */
109        private Long createUser;
110
111
112        /**
113         * Instantiates a new Opt log dto.
114         */
115        public OptLogDTO()
116        {
117
118        }
119
120        /**
121         * Instantiates a new Opt log dto.
122         *
123         * @param requestIp     the request ip
124         * @param type          the type
125         * @param userName      the user name
126         * @param description   the description
127         * @param classPath     the class path
128         * @param actionMethod  the action method
129         * @param requestUri    the request uri
130         * @param httpMethod    the http method
131         * @param params        the params
132         * @param result        the result
133         * @param exDesc        the ex desc
134         * @param exDetail      the ex detail
135         * @param startTime     the start time
136         * @param finishTime    the finish time
137         * @param consumingTime the consuming time
138         * @param ua            the ua
139         * @param createUser    the create user
```

```
     */
    public OptLogDTO(String requestIp, String type, String userName, String description,
                     String classPath, String actionMethod, String requestUri,
                     String httpMethod, String params, String result, String exDesc,
                     String exDetail, LocalDateTime startTime, LocalDateTime finishTime,
                     Long consumingTime, String ua, Long createUser)
    {
        this.requestIp = requestIp;
        this.type = type;
        this.userName = userName;
        this.description = description;
        this.classPath = classPath;
        this.actionMethod = actionMethod;
        this.requestUri = requestUri;
        this.httpMethod = httpMethod;
        this.params = params;
        this.result = result;
        this.exDesc = exDesc;
        this.exDetail = exDetail;
        this.startTime = startTime;
        this.finishTime = finishTime;
        this.consumingTime = consumingTime;
        this.ua = ua;
        this.createUser = createUser;
    }

    /**
     * Gets request ip.
     *
     * @return the request ip
     */
    public String getRequestIp()
    {
        return requestIp;
    }

    /**
     * Sets request ip.
     *
     * @param requestIp the request ip
     */
    public void setRequestIp(String requestIp)
    {
        this.requestIp = requestIp;
    }

    /**
     * Gets type.
     *
     * @return the type
     */
    public String getType()
    {
        return type;
```

```
194     }
195
196     /**
197      * Sets type.
198      *
199      * @param type the type
200      */
201     public void setType(String type)
202     {
203         this.type = type;
204     }
205
206     /**
207      * Gets user name.
208      *
209      * @return the user name
210      */
211     public String getUserName()
212     {
213         return userName;
214     }
215
216     /**
217      * Sets user name.
218      *
219      * @param userName the user name
220      */
221     public void setUserName(String userName)
222     {
223         this.userName = userName;
224     }
225
226     /**
227      * Gets description.
228      *
229      * @return the description
230      */
231     public String getDescription()
232     {
233         return description;
234     }
235
236     /**
237      * Sets description.
238      *
239      * @param description the description
240      */
241     public void setDescription(String description)
242     {
243         this.description = description;
244     }
245
246     /**
247      * Gets class path.
248      *
249      * @return the class path
250      */
251     public String getClassPath()
```

```
252     {
253         return classPath;
254     }
255 
256     /**
257      * Sets class path.
258      *
259      * @param classPath the class path
260      */
261     public void setClassPath(String classPath)
262     {
263         this.classPath = classPath;
264     }
265 
266     /**
267      * Gets action method.
268      *
269      * @return the action method
270      */
271     public String getActionMethod()
272     {
273         return actionMethod;
274     }
275 
276     /**
277      * Sets action method.
278      *
279      * @param actionMethod the action method
280      */
281     public void setActionMethod(String actionMethod)
282     {
283         this.actionMethod = actionMethod;
284     }
285 
286     /**
287      * Gets request uri.
288      *
289      * @return the request uri
290      */
291     public String getRequestUri()
292     {
293         return requestUri;
294     }
295 
296     /**
297      * Sets request uri.
298      *
299      * @param requestUri the request uri
300      */
301     public void setRequestUri(String requestUri)
302     {
303         this.requestUri = requestUri;
304     }
305 
306     /**
307      * Gets http method.
308      *
309      * @return the http method
```

```
310         */
311        public String getHttpMethod()
312        {
313            return httpMethod;
314        }
315
316        /**
317         * Sets http method.
318         *
319         * @param httpMethod the http method
320         */
321        public void setHttpMethod(String httpMethod)
322        {
323            this.httpMethod = httpMethod;
324        }
325
326        /**
327         * Gets params.
328         *
329         * @return the params
330         */
331        public String getParams()
332        {
333            return params;
334        }
335
336        /**
337         * Sets params.
338         *
339         * @param params the params
340         */
341        public void setParams(String params)
342        {
343            this.params = params;
344        }
345
346        /**
347         * Gets result.
348         *
349         * @return the result
350         */
351        public String getResult()
352        {
353            return result;
354        }
355
356        /**
357         * Sets result.
358         *
359         * @param result the result
360         */
361        public void setResult(String result)
362        {
363            this.result = result;
364        }
365
366        /**
367         * Gets ex desc.
```

```
368         *
369         * @return the ex desc
370         */
371        public String getExDesc()
372        {
373            return exDesc;
374        }
375
376        /**
377         * Sets ex desc.
378         *
379         * @param exDesc the ex desc
380         */
381        public void setExDesc(String exDesc)
382        {
383            this.exDesc = exDesc;
384        }
385
386        /**
387         * Gets ex detail.
388         *
389         * @return the ex detail
390         */
391        public String getExDetail()
392        {
393            return exDetail;
394        }
395
396        /**
397         * Sets ex detail.
398         *
399         * @param exDetail the ex detail
400         */
401        public void setExDetail(String exDetail)
402        {
403            this.exDetail = exDetail;
404        }
405
406        /**
407         * Gets start time.
408         *
409         * @return the start time
410         */
411        public LocalDateTime getStartTime()
412        {
413            return startTime;
414        }
415
416        /**
417         * Sets start time.
418         *
419         * @param startTime the start time
420         */
421        public void setStartTime(LocalDateTime startTime)
422        {
423            this.startTime = startTime;
424        }
425
```

```
426     /**
427      * Gets finish time.
428      *
429      * @return the finish time
430      */
431     public LocalDateTime getFinishTime()
432     {
433         return finishTime;
434     }
435 
436     /**
437      * Sets finish time.
438      *
439      * @param finishTime the finish time
440      */
441     public void setFinishTime(LocalDateTime finishTime)
442     {
443         this.finishTime = finishTime;
444     }
445 
446     /**
447      * Gets consuming time.
448      *
449      * @return the consuming time
450      */
451     public Long getConsumingTime()
452     {
453         return consumingTime;
454     }
455 
456     /**
457      * Sets consuming time.
458      *
459      * @param consumingTime the consuming time
460      */
461     public void setConsumingTime(Long consumingTime)
462     {
463         this.consumingTime = consumingTime;
464     }
465 
466     /**
467      * Gets ua.
468      *
469      * @return the ua
470      */
471     public String getUa()
472     {
473         return ua;
474     }
475 
476     /**
477      * Sets ua.
478      *
479      * @param ua the ua
480      */
481     public void setUa(String ua)
482     {
483         this.ua = ua;
```

```
484     }
485
486     /**
487      * Gets create user.
488      *
489      * @return the create user
490      */
491     public Long getCreateUser()
492     {
493         return createUser;
494     }
495
496     /**
497      * Sets create user.
498      *
499      * @param createUser the create user
500      */
501     public void setCreateUser(Long createUser)
502     {
503         this.createUser = createUser;
504     }
505
506     @Override
507     public boolean equals(Object o)
508     {
509         if (this == o)
510         {
511             return true;
512         }
513         if (o == null || getClass() != o.getClass())
514         {
515             return false;
516         }
517
518         OptLogDTO optLogDTO = (OptLogDTO) o;
519
520         if (getRequestIp() != null ?
!getRequestIp().equals(optLogDTO.getRequestIp()) : optLogDTO.getRequestIp()
!= null)
521         {
522             return false;
523         }
524         if (getType() != null ? !getType().equals(optLogDTO.getType()) :
optLogDTO.getType() != null)
525         {
526             return false;
527         }
528         if (getUserName() != null ?
!getUserName().equals(optLogDTO.getUserName()) : optLogDTO.getUserName() !=
null)
529         {
530             return false;
531         }
532         if (getDescription() != null ?
!getDescription().equals(optLogDTO.getDescription()) :
optLogDTO.getDescription() != null)
533         {
534             return false;
```

```
535         }
536         if (getClassPath() != null ?
    !getClassPath().equals(optLogDTO.getClassPath()) : optLogDTO.getClassPath()
    != null)
537         {
538             return false;
539         }
540         if (getActionMethod() != null ?
    !getActionMethod().equals(optLogDTO.getActionMethod()) :
    optLogDTO.getActionMethod() != null)
541         {
542             return false;
543         }
544         if (getRequestUri() != null ?
    !getRequestUri().equals(optLogDTO.getRequestUri()) :
    optLogDTO.getRequestUri() != null)
545         {
546             return false;
547         }
548         if (getHttpMethod() != null ?
    !getHttpMethod().equals(optLogDTO.getHttpMethod()) :
    optLogDTO.getHttpMethod() != null)
549         {
550             return false;
551         }
552         if (getParams() != null ?
    !getParams().equals(optLogDTO.getParams()) : optLogDTO.getParams() != null)
553         {
554             return false;
555         }
556         if (getResult() != null ?
    !getResult().equals(optLogDTO.getResult()) : optLogDTO.getResult() != null)
557         {
558             return false;
559         }
560         if (getExDesc() != null ?
    !getExDesc().equals(optLogDTO.getExDesc()) : optLogDTO.getExDesc() != null)
561         {
562             return false;
563         }
564         if (getExDetail() != null ?
    !getExDetail().equals(optLogDTO.getExDetail()) : optLogDTO.getExDetail() !=
    null)
565         {
566             return false;
567         }
568         if (getStartTime() != null ?
    !getStartTime().equals(optLogDTO.getStartTime()) : optLogDTO.getStartTime()
    != null)
569         {
570             return false;
571         }
572         if (getFinishTime() != null ?
    !getFinishTime().equals(optLogDTO.getFinishTime()) :
    optLogDTO.getFinishTime() != null)
573         {
574             return false;
575         }
```

```
        if (getConsumingTime() != null ? !getConsumingTime().equals(optLogDTO.getConsumingTime()) : optLogDTO.getConsumingTime() != null)
        {
            return false;
        }
        if (getUa() != null ? !getUa().equals(optLogDTO.getUa()) : optLogDTO.getUa() != null)
        {
            return false;
        }
        return getCreateUser() != null ? getCreateUser().equals(optLogDTO.getCreateUser()) : optLogDTO.getCreateUser() == null;
    }

    @Override
    public int hashCode()
    {
        int result1 = getRequestIp() != null ? getRequestIp().hashCode() : 0;
        result1 = 31 * result1 + (getType() != null ? getType().hashCode() : 0);
        result1 = 31 * result1 + (getUserName() != null ? getUserName().hashCode() : 0);
        result1 = 31 * result1 + (getDescription() != null ? getDescription().hashCode() : 0);
        result1 = 31 * result1 + (getClassPath() != null ? getClassPath().hashCode() : 0);
        result1 = 31 * result1 + (getActionMethod() != null ? getActionMethod().hashCode() : 0);
        result1 = 31 * result1 + (getRequestUri() != null ? getRequestUri().hashCode() : 0);
        result1 = 31 * result1 + (getHttpMethod() != null ? getHttpMethod().hashCode() : 0);
        result1 = 31 * result1 + (getParams() != null ? getParams().hashCode() : 0);
        result1 = 31 * result1 + (getResult() != null ? getResult().hashCode() : 0);
        result1 = 31 * result1 + (getExDesc() != null ? getExDesc().hashCode() : 0);
        result1 = 31 * result1 + (getExDetail() != null ? getExDetail().hashCode() : 0);
        result1 = 31 * result1 + (getStartTime() != null ? getStartTime().hashCode() : 0);
        result1 = 31 * result1 + (getFinishTime() != null ? getFinishTime().hashCode() : 0);
        result1 = 31 * result1 + (getConsumingTime() != null ? getConsumingTime().hashCode() : 0);
        result1 = 31 * result1 + (getUa() != null ? getUa().hashCode() : 0);
        result1 = 31 * result1 + (getCreateUser() != null ? getCreateUser().hashCode() : 0);
        return result1;
    }

    @Override
    public String toString()
```

```
    {
        final StringBuilder sb = new StringBuilder("OptLogDTO{");
        sb.append("requestIp='").append(requestIp).append('\'');
        sb.append(", type='").append(type).append('\'');
        sb.append(", userName='").append(userName).append('\'');
        sb.append(", description='").append(description).append('\'');
        sb.append(", classPath='").append(classPath).append('\'');
        sb.append(", actionMethod='").append(actionMethod).append('\'');
        sb.append(", requestUri='").append(requestUri).append('\'');
        sb.append(", httpMethod='").append(httpMethod).append('\'');
        sb.append(", params='").append(params).append('\'');
        sb.append(", result='").append(result).append('\'');
        sb.append(", exDesc='").append(exDesc).append('\'');
        sb.append(", exDetail='").append(exDetail).append('\'');
        sb.append(", startTime=").append(startTime);
        sb.append(", finishTime=").append(finishTime);
        sb.append(", consumingTime=").append(consumingTime);
        sb.append(", ua='").append(ua).append('\'');
        sb.append(", createUser=").append(createUser);
        sb.append('}');
        return sb.toString();
    }
}
```

第十四步：编写实体类R<T>

```
package mao.tools_log.entity;

import java.util.Map;

import com.alibaba.fastjson.JSONObject;
import com.google.common.collect.Maps;


import io.swagger.annotations.ApiModelProperty;
import mao.tools_log.exception.BizException;
import mao.tools_log.exception.code.BaseExceptionCode;


@SuppressWarnings({"AlibabaClassNamingShouldBeCamel"})
public class R<T>
{
    public static final String DEF_ERROR_MESSAGE = "系统繁忙，请稍候再试";
    public static final String HYSTRIX_ERROR_MESSAGE = "请求超时，请稍候再试";
    public static final int SUCCESS_CODE = 0;
    public static final int FAIL_CODE = -1;
    public static final int TIMEOUT_CODE = -2;
    /**
     * 统一参数验证异常
     */
```

```
    public static final int VALID_EX_CODE = -9;
    public static final int OPERATION_EX_CODE = -10;
    /**
     * 调用是否成功标识，0：成功，-1:系统繁忙，此时请开发者稍候再试 详情见[ExceptionCode]
     */
    @ApiModelProperty(value = "响应编码:0/200-请求处理成功")
    private int code;

    /**
     * 调用结果
     */
    @ApiModelProperty(value = "响应数据")
    private T data;

    /**
     * 结果消息，如果调用成功，消息通常为空T
     */
    @ApiModelProperty(value = "提示消息")
    private String msg = "ok";

    @ApiModelProperty(value = "请求路径")
    private String path;
    /**
     * 附加数据
     */
    @ApiModelProperty(value = "附加数据")
    private Map<String, Object> extra;

    /**
     * 响应时间
     */
    @ApiModelProperty(value = "响应时间戳")
    private long timestamp = System.currentTimeMillis();

    private R()
    {
        super();
    }

    public R(int code, T data, String msg)
    {
        this.code = code;
        this.data = data;
        this.msg = msg;
    }

    public static <E> R<E> result(int code, E data, String msg)
    {
        return new R<>(code, data, msg);
    }

    /**
     * 请求成功消息
     *
     * @param data 结果
     * @return RPC调用结果
     */
```

```
    public static <E> R<E> success(E data)
    {
        return new R<>(SUCCESS_CODE, data, "ok");
    }

    public static R<Boolean> success()
    {
        return new R<>(SUCCESS_CODE, true, "ok");
    }

    /**
     * 请求成功方法 ，data返回值，msg提示信息
     *
     * @param data 结果
     * @param msg  消息
     * @return RPC调用结果
     */
    public static <E> R<E> success(E data, String msg)
    {
        return new R<>(SUCCESS_CODE, data, msg);
    }

    /**
     * 请求失败消息
     *
     * @param msg 消息
     * @return RPC调用结果
     */
    public static <E> R<E> fail(int code, String msg)
    {
        return new R<>(code, null, (msg == null || msg.isEmpty()) ? DEF_ERROR_MESSAGE : msg);
    }

    public static <E> R<E> fail(String msg)
    {
        return fail(OPERATION_EX_CODE, msg);
    }

    public static <E> R<E> fail(String msg, Object... args)
    {
        String message = (msg == null || msg.isEmpty()) ? DEF_ERROR_MESSAGE : msg;
        return new R<>(OPERATION_EX_CODE, null, String.format(message, args));
    }

    public static <E> R<E> fail(BaseExceptionCode exceptionCode)
    {
        return validFail(exceptionCode);
    }

    public static <E> R<E> fail(BizException exception)
    {
        if (exception == null)
        {
            return fail(DEF_ERROR_MESSAGE);
        }
```

```
        return new R<>(exception.getCode(), null, exception.getMessage());
    }

    /**
     * 请求失败消息，根据异常类型，获取不同的提供消息
     *
     * @param throwable 异常
     * @return RPC调用结果
     */
    public static <E> R<E> fail(Throwable throwable)
    {
        return fail(FAIL_CODE, throwable != null ? throwable.getMessage() :
DEF_ERROR_MESSAGE);
    }

    public static <E> R<E> validFail(String msg)
    {
        return new R<>(VALID_EX_CODE, null, (msg == null || msg.isEmpty())
? DEF_ERROR_MESSAGE : msg);
    }

    public static <E> R<E> validFail(String msg, Object... args)
    {
        String message = (msg == null || msg.isEmpty()) ? DEF_ERROR_MESSAGE
: msg;
        return new R<>(VALID_EX_CODE, null, String.format(message, args));
    }

    public static <E> R<E> validFail(BaseExceptionCode exceptionCode)
    {
        return new R<>(exceptionCode.getCode(), null,
                (exceptionCode.getMsg() == null ||
exceptionCode.getMsg().isEmpty()) ? DEF_ERROR_MESSAGE :
exceptionCode.getMsg());
    }

    public static <E> R<E> timeout()
    {
        return fail(TIMEOUT_CODE, HYSTRIX_ERROR_MESSAGE);
    }


    public R<T> put(String key, Object value)
    {
        if (this.extra == null)
        {
            this.extra = Maps.newHashMap();
        }
        this.extra.put(key, value);
        return this;
    }

    /**
     * 逻辑处理是否成功
     *
     * @return 是否成功
     */
    public Boolean getIsSuccess()
```

```
    {
        return this.code == SUCCESS_CODE || this.code == 200;
    }

    /**
     * 逻辑处理是否失败
     *
     * @return 是否失败
     */
    public Boolean getIsError()
    {
        return !getIsSuccess();
    }

    @Override
    public String toString()
    {
        return JSONObject.toJSONString(this);
    }

    //------------------------------------------------------

    public int getCode()
    {
        return code;
    }

    public void setCode(int code)
    {
        this.code = code;
    }

    public T getData()
    {
        return data;
    }

    public void setData(T data)
    {
        this.data = data;
    }

    public String getMsg()
    {
        return msg;
    }

    public void setMsg(String msg)
    {
        this.msg = msg;
    }

    public String getPath()
    {
        return path;
    }

    public void setPath(String path)
```

```
    {
        this.path = path;
    }

    public Map<String, Object> getExtra()
    {
        return extra;
    }

    public void setExtra(Map<String, Object> extra)
    {
        this.extra = extra;
    }

    public long getTimestamp()
    {
        return timestamp;
    }

    public void setTimestamp(long timestamp)
    {
        this.timestamp = timestamp;
    }
}
```

第十五步：编写类SysLogEvent

```
package mao.tools_log.event;

import mao.tools_log.entity.OptLogDTO;
import org.springframework.context.ApplicationEvent;

/**
 * Project name(项目名称)：logback_spring_boot_starter_demo
 * Package(包名): mao.tools_log.event
 * Class(类名): SysLogEvent
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 21:44
 * Version(版本): 1.0
 * Description(描述)： 系统日志事件
 */

public class SysLogEvent extends ApplicationEvent
{
    public SysLogEvent(OptLogDTO source)
    {
        super(source);
    }
```

```
}
```

第十六步：编写类SysLogListener

```
package mao.tools_log.event;

import mao.tools_log.entity.OptLogDTO;
import org.springframework.context.event.EventListener;
import org.springframework.core.annotation.Order;
import org.springframework.scheduling.annotation.Async;

import java.util.function.Consumer;

/**
 * Project name(项目名称)：logback_spring_boot_starter_demo
 * Package(包名): mao.tools_log.event
 * Class(类名): SysLogListener
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 21:45
 * Version(版本): 1.0
 * Description(描述)： 无
 */

public class SysLogListener
{
    private final Consumer<OptLogDTO> consumer;

    public SysLogListener(Consumer<OptLogDTO> consumer)
    {
        this.consumer = consumer;
    }

    @Async
    @Order
    @EventListener(SysLogEvent.class)
    public void saveSysLog(SysLogEvent event)
    {
        OptLogDTO optLog = (OptLogDTO) event.getSource();
        //BaseContextHandler.setDatabase(database);
        consumer.accept(optLog);
    }
}
```

第十七步：编写常量工具类BaseContextConstants

```java
package mao.tools_log.context;

/**
 * 常量工具类
 */
public class BaseContextConstants
{
    /**
     *
     */
    public static final String TOKEN_NAME = "token";
    /**
     *
     */
    public static final String JWT_KEY_USER_ID = "userid";
    /**
     *
     */
    public static final String JWT_KEY_NAME = "name";
    /**
     *
     */
    public static final String JWT_KEY_ACCOUNT = "account";

    /**
     * 组织id
     */
    public static final String JWT_KEY_ORG_ID = "orgid";
    /**
     * 岗位id
     */
    public static final String JWT_KEY_STATION_ID = "stationid";

    /**
     * 动态数据库名前缀。   每个项目配置死的
     */
    public static final String DATABASE_NAME = "database_name";
}
```

第十八步：编写类BaseContextHandler

```java
package mao.tools_log.context;


import mao.tools_log.utils.NumberHelper;
import mao.tools_log.utils.StrHelper;

```

```java
import java.util.HashMap;
import java.util.Map;

/**
 * 获取当前域中的 用户id appid 用户昵称
 * 注意： appid 通过token解析， 用户id 和 用户昵称必须在前端 通过请求头的方法传入。否则这里无法获取
 */
public class BaseContextHandler
{
    private static final ThreadLocal<Map<String, String>> THREAD_LOCAL = new ThreadLocal<>();

    public static void set(String key, Long value)
    {
        Map<String, String> map = getLocalMap();
        map.put(key, value == null ? "0" : String.valueOf(value));
    }

    public static void set(String key, String value)
    {
        Map<String, String> map = getLocalMap();
        map.put(key, value == null ? "" : value);
    }

    public static void set(String key, Boolean value)
    {
        Map<String, String> map = getLocalMap();
        map.put(key, value == null ? "false" : value.toString());
    }


    public static Map<String, String> getLocalMap()
    {
        Map<String, String> map = THREAD_LOCAL.get();
        if (map == null)
        {
            map = new HashMap<>(10);
            THREAD_LOCAL.set(map);
        }
        return map;
    }

    public static void setLocalMap(Map<String, String> threadLocalMap)
    {
        THREAD_LOCAL.set(threadLocalMap);
    }


    public static String get(String key)
    {
        Map<String, String> map = getLocalMap();
        return map.getOrDefault(key, "");
    }

    /**
     * 账号id
     *
```

```
     * @return
     */
    public static Long getUserId()
    {
        Object value = get(BaseContextConstants.JWT_KEY_USER_ID);
        return NumberHelper.longValueOf0(value);
    }

    /**
     * 账号id
     *
     * @param userId
     */
    public static void setUserId(Long userId)
    {
        set(BaseContextConstants.JWT_KEY_USER_ID, userId);
    }

    public static void setUserId(String userId)
    {
        setUserId(NumberHelper.longValueOf0(userId));
    }

    /**
     * 账号表中的name
     *
     * @return
     */
    public static String getAccount()
    {
        Object value = get(BaseContextConstants.JWT_KEY_ACCOUNT);
        return returnObjectValue(value);
    }

    /**
     * 账号表中的name
     *
     * @param name
     */
    public static void setAccount(String name)
    {
        set(BaseContextConstants.JWT_KEY_ACCOUNT, name);
    }


    /**
     * 登录的账号
     *
     * @return
     */
    public static String getName()
    {
        Object value = get(BaseContextConstants.JWT_KEY_NAME);
        return returnObjectValue(value);
    }

    /**
     * 登录的账号
```

```
     *
     * @param account
     */
    public static void setName(String account)
    {
        set(BaseContextConstants.JWT_KEY_NAME, account);
    }

    /**
     * 获取用户token
     *
     * @return
     */
    public static String getToken()
    {
        Object value = get(BaseContextConstants.TOKEN_NAME);
        return StrHelper.getObjectValue(value);
    }

    public static void setToken(String token)
    {
        set(BaseContextConstants.TOKEN_NAME, token);
    }

    public static Long getOrgId()
    {
        Object value = get(BaseContextConstants.JWT_KEY_ORG_ID);
        return NumberHelper.longValueOf0(value);
    }

    public static void setOrgId(String val)
    {
        set(BaseContextConstants.JWT_KEY_ORG_ID, val);
    }


    public static Long getStationId()
    {
        Object value = get(BaseContextConstants.JWT_KEY_STATION_ID);
        return NumberHelper.longValueOf0(value);
    }

    public static void setStationId(String val)
    {
        set(BaseContextConstants.JWT_KEY_STATION_ID, val);
    }

    public static String getDatabase()
    {
        Object value = get(BaseContextConstants.DATABASE_NAME);
        return StrHelper.getObjectValue(value);
    }

    public static void setDatabase(String val)
    {
        set(BaseContextConstants.DATABASE_NAME, val);
    }

```

```
179
180     private static String returnObjectValue(Object value)
181     {
182         return value == null ? "" : value.toString();
183     }
184
185     public static void remove()
186     {
187         if (THREAD_LOCAL != null)
188         {
189             THREAD_LOCAL.remove();
190         }
191     }
192
193 }
```

第十九步：编写注解SysLog

```
1  package mao.tools_log.annotation;
2
3  import java.lang.annotation.*;
4
5  @Target(ElementType.METHOD)
6  @Retention(RetentionPolicy.RUNTIME)
7  @Documented
8  public @interface SysLog
9  {
10     /**
11      * 描述
12      *
13      * @return {String}
14      */
15     String value();
16
17     /**
18      * 记录执行参数
19      *
20      * @return boolean
21      */
22     boolean recordRequestParam() default true;
23
24     /**
25      * 记录返回参数
26      *
27      * @return boolean
28      */
29     boolean recordResponseParam() default true;
30 }
```

第二十步：编写类SysLogAspect

```
package mao.tools_log.aspect;

import cn.hutool.core.convert.Convert;
import cn.hutool.core.util.StrUtil;
import cn.hutool.core.util.URLUtil;
import cn.hutool.extra.servlet.ServletUtil;
import com.alibaba.fastjson.JSONObject;
import io.swagger.annotations.Api;
import mao.tools_log.context.BaseContextHandler;
import mao.tools_log.entity.OptLogDTO;
import mao.tools_log.entity.R;
import mao.tools_log.event.SysLogEvent;

import mao.tools_log.utils.LogUtil;
import org.aspectj.lang.annotation.Aspect;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.*;

import org.springframework.context.ApplicationContext;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;

import javax.servlet.http.HttpServletRequest;
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;
import java.util.Arrays;
import java.util.Objects;
import java.util.function.Consumer;

/**
 * Project name(项目名称)：logback_spring_boot_starter_demo
 * Package(包名): mao.tools_log.aspect
 * Class(类名): SysLogAspect
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 21:52
 * Version(版本): 1.0
 * Description(描述)： 操作日志使用spring event异步入库
 */

@Aspect
public class SysLogAspect
{
    /**
```

```
     * 事件发布是由ApplicationContext对象管控的，我们发布事件前需要注入ApplicationContext对象调用publishEvent方法完成事件发布
     **/
    @Autowired
    private ApplicationContext applicationContext;

    private static final ThreadLocal<OptLogDTO> THREAD_LOCAL = new ThreadLocal<>();

    private static final Logger log = LoggerFactory.getLogger(SysLogAspect.class);

    /***
     * 定义controller切入点拦截规则，拦截SysLog注解的方法
     */
    @Pointcut("@annotation(mao.tools_log.annotation.SysLog)")
    public void sysLogAspect()
    {

    }

    private OptLogDTO get()
    {
        OptLogDTO sysLog = THREAD_LOCAL.get();
        if (sysLog == null)
        {
            return new OptLogDTO();
        }
        return sysLog;
    }

    @Before(value = "sysLogAspect()")
    public void recordLog(JoinPoint joinPoint) throws Throwable
    {
        tryCatch((val) ->
        {
            // 开始时间
            OptLogDTO sysLog = get();
            sysLog.setCreateUser(BaseContextHandler.getUserId());
            sysLog.setUserName(BaseContextHandler.getName());
            String controllerDescription = "";
            Api api = joinPoint.getTarget().getClass().getAnnotation(Api.class);
            if (api != null)
            {
                String[] tags = api.tags();
                if (tags != null && tags.length > 0)
                {
                    controllerDescription = tags[0];
                }
            }

            String controllerMethodDescription = LogUtil.getControllerMethodDescription(joinPoint);
            if (StrUtil.isEmpty(controllerDescription))
            {
                sysLog.setDescription(controllerMethodDescription);
            }
```

```
            else
            {
                sysLog.setDescription(controllerDescription + "-" +
controllerMethodDescription);
            }

            // 类名

 sysLog.setClassPath(joinPoint.getTarget().getClass().getName());
            //获取执行的方法名
            sysLog.setActionMethod(joinPoint.getSignature().getName());


            // 参数
            Object[] args = joinPoint.getArgs();

            String strArgs = "";
            HttpServletRequest request = ((ServletRequestAttributes)
Objects.requireNonNull(RequestContextHolder.getRequestAttributes())).getReq
uest();
            try
            {
                if (!request.getContentType().contains("multipart/form-
data"))
                {
                    strArgs = JSONObject.toJSONString(args);
                }
            }
            catch (Exception e)
            {
                try
                {
                    strArgs = Arrays.toString(args);
                }
                catch (Exception ex)
                {
                    log.warn("解析参数异常", ex);
                }
            }
            sysLog.setParams(getText(strArgs));

            if (request != null)
            {
                sysLog.setRequestIp(ServletUtil.getClientIP(request));

 sysLog.setRequestUri(URLUtil.getPath(request.getRequestURI()));
                sysLog.setHttpMethod(request.getMethod());
                sysLog.setUa(StrUtil.sub(request.getHeader("user-agent"),
0, 500));
            }
            sysLog.setStartTime(LocalDateTime.now());

            THREAD_LOCAL.set(sysLog);
        });
    }


    private void tryCatch(Consumer<String> consumer)
```

```
    {
        try
        {
            consumer.accept("");
        }
        catch (Exception e)
        {
            log.warn("记录操作日志异常", e);
            THREAD_LOCAL.remove();
        }
    }

    /**
     * 返回通知
     *
     * @param ret 对象R
     * @throws Throwable
     */
    @AfterReturning(returning = "ret", pointcut = "sysLogAspect()")
    public void doAfterReturning(Object ret)
    {
        tryCatch((aaa) ->
        {
            R r = Convert.convert(R.class, ret);
            OptLogDTO sysLog = get();
            if (r == null)
            {
                sysLog.setType("OPT");
            }
            else
            {
                if (r.getIsSuccess())
                {
                    sysLog.setType("OPT");
                }
                else
                {
                    sysLog.setType("EX");
                    sysLog.setExDetail(r.getMsg());
                }
                sysLog.setResult(getText(r.toString()));
            }

            publishEvent(sysLog);
        });

    }

    private void publishEvent(OptLogDTO sysLog)
    {
        sysLog.setFinishTime(LocalDateTime.now());

 sysLog.setConsumingTime(sysLog.getStartTime().until(sysLog.getFinishTime()
, ChronoUnit.MILLIS));
        applicationContext.publishEvent(new SysLogEvent(sysLog));
        THREAD_LOCAL.remove();
    }

```

```
211     /**
212      * 异常通知
213      *
214      * @param e Throwable
215      */
216     @AfterThrowing(pointcut = "sysLogAspect()", throwing = "e")
217     public void doAfterThrowable(Throwable e)
218     {
219         tryCatch((aaa) ->
220         {
221             OptLogDTO sysLog = get();
222             sysLog.setType("EX");
223 
224             // 异常对象
225             sysLog.setExDetail(LogUtil.getStackTrace(e));
226             // 异常信息
227             sysLog.setExDesc(e.getMessage());
228 
229             publishEvent(sysLog);
230         });
231     }
232 
233 
234     /**
235      * 截取指定长度的字符串
236      *
237      * @param val String字符串
238      * @return {@link String}
239      */
240     private String getText(String val)
241     {
242         return StrUtil.sub(val, 0, 65535);
243     }
244 
245 //    @Around("@annotation(sLog)")
246 //    @SneakyThrows
247 //    public Object around(ProceedingJoinPoint point, SysLog sLog) {
248 //        log.info("当前线程id={}", Thread.currentThread().getId());
249 //
250 //        String strClassName = point.getTarget().getClass().getName();
251 //        String strMethodName = point.getSignature().getName();
252 //
253 //        log.info("[类名]:{},[方法]:{}", strClassName, strMethodName);
254 //        Log sysLog = Log.builder().build();
255 //
256 //        // 开始时间
257 //        Long startTime = Instant.now().toEpochMilli();
258 //        HttpServletRequest request = ((ServletRequestAttributes)
    Objects.requireNonNull(RequestContextHolder.getRequestAttributes())).getReq
    uest();
259 //        BaseContextHandler.getAccount();
260 //        sysLog.setCreateUser(BaseContextHandler.getUserId());
261 //        sysLog.setRequestIp(ServletUtil.getClientIP(request));
262 //        sysLog.setUserName(BaseContextHandler.getNickName());
263 //
    sysLog.setDescription(LogUtil.getControllerMethodDescription(point));
264 //
265 //        // 类名
```

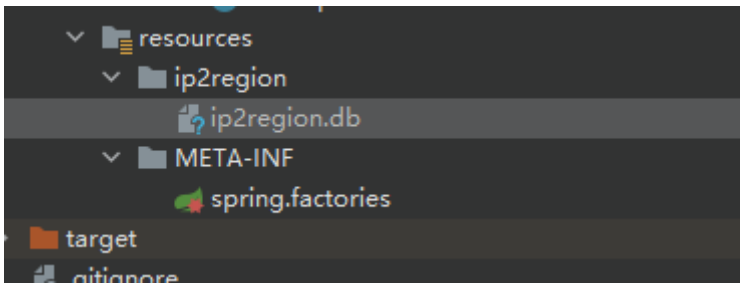
```
//        sysLog.setClassPath(point.getTarget().getClass().getName());
//        //获取执行的方法名
//        sysLog.setActionMethod(point.getSignature().getName());
//        sysLog.setRequestUri(URLUtil.getPath(request.getRequestURI()));
//        sysLog.setHttpMethod(HttpMethod.get(request.getMethod()));
//        // 参数
//        Object[] args = point.getArgs();
//        sysLog.setParams(getText(JSONObject.toJSONString(args)));
//
//        sysLog.setStartTime(LocalDateTime.now());
//        sysLog.setUa(request.getHeader("user-agent"));
//
//        // 发送异步日志事件
//        Object obj = point.proceed();
//
//        R r = Convert.convert(R.class, obj);
//        if (r.getIsSuccess()) {
//            sysLog.setType(LogType.OPT);
//        } else {
//            sysLog.setType(LogType.EX);
//            sysLog.setExDetail(r.getMsg());
//        }
//        if (r != null) {
//            sysLog.setResult(getText(r.toString()));
//        }
//
//        sysLog.setFinishTime(LocalDateTime.now());
//        long endTime = Instant.now().toEpochMilli();
//        sysLog.setConsumingTime(endTime - startTime);
//
//        applicationContext.publishEvent(new SysLogEvent(sysLog));
//        return obj;
//    }

}
```

第二十一步：编写配置类LogAutoConfiguration

```
package mao.tools_log.config;

import mao.tools_log.aspect.SysLogAspect;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnWebApplication
;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

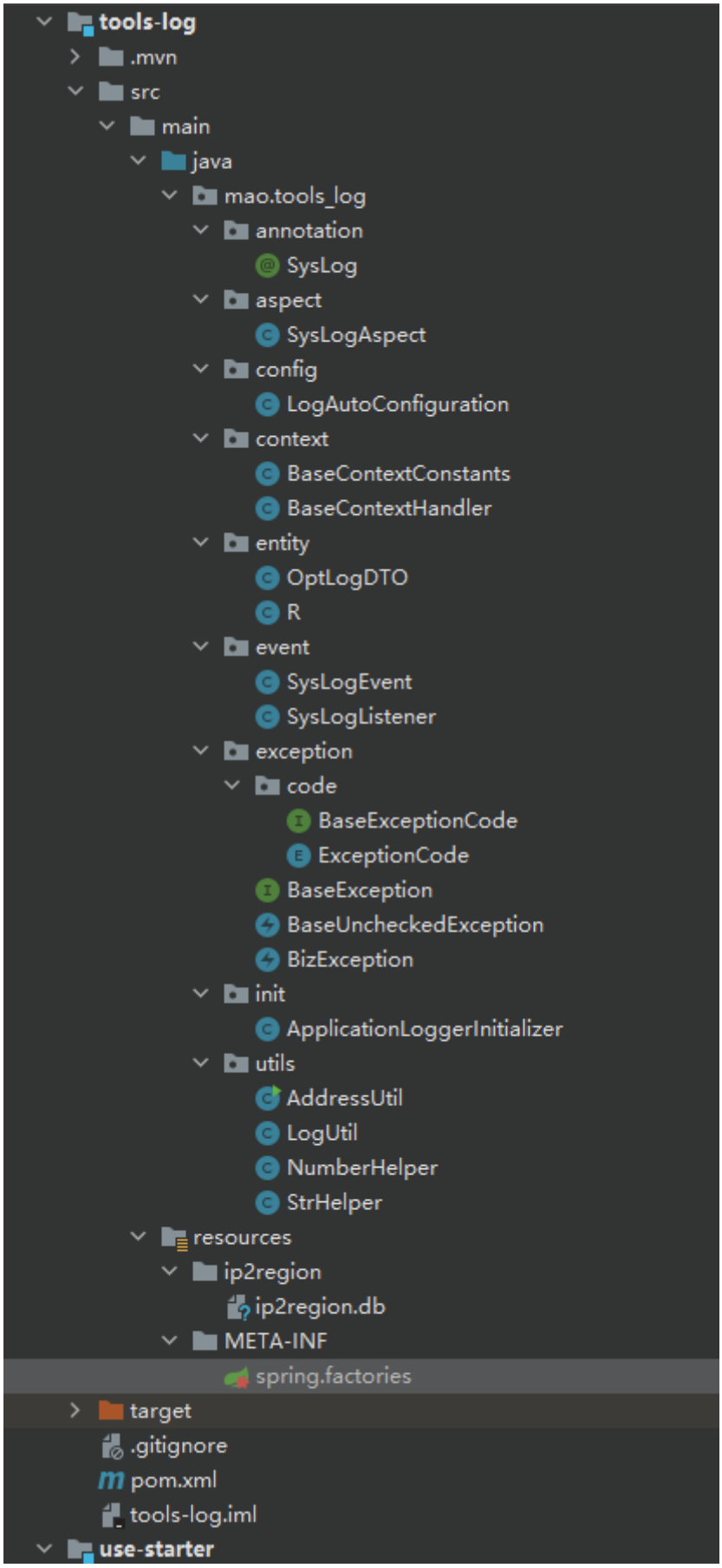
```java
import org.springframework.scheduling.annotation.EnableAsync;

/**
 * Project name(项目名称)：logback_spring_boot_starter_demo
 * Package(包名): mao.tools_log.config
 * Class(类名): LogAutoConfiguration
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 22:29
 * Version(版本): 1.0
 * Description(描述)：
 * <p>
 * 启动条件：
 * 1，存在web环境
 * 2，配置文件中log.enabled=true
 * 3，配置文件中不存在：log.enabled 值
 */

@EnableAsync
@Configuration
@ConditionalOnWebApplication
@ConditionalOnProperty(name = "log.enabled", havingValue = "true",
matchIfMissing = true)
public class LogAutoConfiguration
{

    @Bean
    @ConditionalOnMissingBean
    public SysLogAspect sysLogAspect()
    {
        return new SysLogAspect();
    }


}
```

第二十二步：拷贝下载的ip2region.db文件到资源目录下



第二十三步：编写spring.factories文件

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
    mao.tools_log.config.LogAutoConfiguration
org.springframework.context.ApplicationContextInitializer=\
    mao.tools_log.init.ApplicationLoggerInitializer
```


tools-log
.mvn
src
main
java
mao.tools_log
annotation
SysLog
aspect
SysLogAspect
config
LogAutoConfiguration
context
BaseContextConstants
BaseContextHandler
entity
OptLogDTO
R
event
SysLogEvent
SysLogListener
exception
code
BaseExceptionCode
ExceptionCode
BaseException
BaseUncheckedException
BizException
init
ApplicationLoggerInitializer
utils
AddressUtil
LogUtil
NumberHelper
StrHelper
resources
ip2region
ip2region.db
META-INF
spring.factories
target
.gitignore
pom.xml
tools-log.iml
use-starter

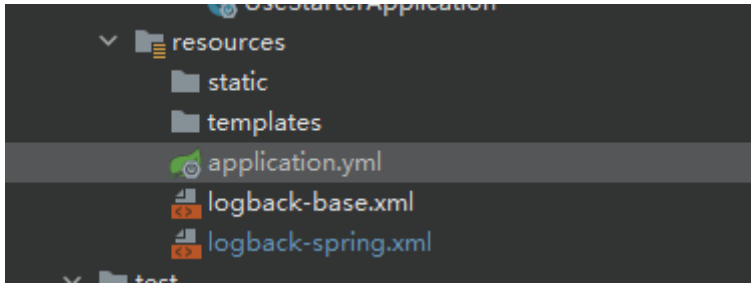
使用starter

第一步：导入tools-log的依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <artifactId>logback_spring_boot_starter_demo</artifactId>
        <groupId>mao</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>

    <artifactId>use-starter</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>use-starter</name>
    <description>use-starter</description>

    <properties>

    </properties>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>mao</groupId>
```

```
            <artifactId>tools-log</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

第二步：拷贝logback的配置文件到此项目的资源目录中



第三步：编写LogService

```java
package mao.use_starter.service;

import mao.tools_log.entity.OptLogDTO;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;

/**
 * Project name(项目名称)：logback_spring_boot_starter_demo
 * Package(包名): mao.use_starter.service
 * Class(类名): LogService
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
```

```
 * Time(创建时间)：  22:40
 * Version(版本): 1.0
 * Description(描述)：  无
 */

@Service
public class LogService
{

    private static final Logger log =
LoggerFactory.getLogger(LogService.class);


    /**
     * 将日志信息保存
     *
     * @param optLogDTO OptLogDTO对象
     */
    public void saveLog(OptLogDTO optLogDTO)
    {
        //此处只是将日志信息进行输出，实际项目中可以将日志信息保存到数据库
        log.info("保存日志信息：" + optLogDTO);
    }
}
```

第四步：修改application.yml文件

```
log:
  enabled: true

logging:
  #在Spring Boot项目中默认加载类路径下的logback-spring.xml文件
  config: classpath:logback-spring.xml

spring:
  profiles:
    active: dev
```

第五步：编写UserController

```
package mao.use_starter.controller;

import mao.tools_log.annotation.SysLog;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
```

```java
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * Project name(项目名称)：logback_spring_boot_starter_demo
 * Package(包名): mao.use_starter.controller
 * Class(类名): UserController
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 22:41
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@RestController
@RequestMapping("/user")
public class UserController
{
    @SysLog("分页查询用户")//记录操作日志
    @GetMapping(value = "page/{pageNum}/{pageSize}")
    public String findByPage(@PathVariable Integer pageNum, @PathVariable Integer pageSize)
    {
        try
        {
            Thread.sleep(300);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        return "OK";
    }
}
```

第六步：编写配置类LogAutoConfig

```java
package mao.use_starter.config;

import mao.tools_log.entity.OptLogDTO;
import mao.tools_log.event.SysLogListener;
import mao.use_starter.service.LogService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

```

```java
/**
 * Project name(项目名称)：logback_spring_boot_starter_demo
 * Package(包名): mao.use_starter.config
 * Class(类名): LogAutoConfig
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 22:38
 * Version(版本): 1.0
 * Description(描述)： 日志配置类
 */


@Configuration
public class LogAutoConfig
{

    private static final Logger log =
LoggerFactory.getLogger(LogAutoConfig.class);

    //自动配置日志监听器组件
    @Bean
    @ConditionalOnMissingBean
    public SysLogListener sysLogListener(LogService logService)
    {
        return new SysLogListener(logService::saveLog);
    }
}
```

第七步：启动程序

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.1)

2022-11-01 14:06:39.953  INFO 14052 --- [           main] mao.use_starter.UseStarterApplication    : Starting UseStarterApplication using Java 16.0.2 on mao with PID 14052 (H:\程序\大四上期\logback_spring_boot_starter_demo\use-starter\target\classes started by mao in H:\程序\大四上期\logback_spring_boot_starter_demo)
2022-11-01 14:06:39.956 DEBUG 14052 --- [           main] mao.use_starter.UseStarterApplication    : Running with Spring Boot v2.7.1, Spring v5.3.21
```

```
2022-11-01 14:06:39.956  INFO 14052 --- [           main] mao.use_starter.UseStarterApplication    : The following 1 profile is active: "dev"
2022-11-01 14:06:40.720  INFO 14052 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2022-11-01 14:06:40.727  INFO 14052 --- [           main] o.a.coyote.http11.Http11NioProtocol      : Initializing ProtocolHandler ["http-nio-8080"]
2022-11-01 14:06:40.727  INFO 14052 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-11-01 14:06:40.727  INFO 14052 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.64]
2022-11-01 14:06:40.822  INFO 14052 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-11-01 14:06:40.823  INFO 14052 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 835 ms
2022-11-01 14:06:41.169  INFO 14052 --- [           main] o.a.coyote.http11.Http11NioProtocol      : Starting ProtocolHandler ["http-nio-8080"]
2022-11-01 14:06:41.184  INFO 14052 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-11-01 14:06:41.192  INFO 14052 --- [           main] mao.use_starter.UseStarterApplication    : Started UseStarterApplication in 1.599 seconds (JVM running for 2.088)
```

第八步：访问

http://localhost:8080/user/page/1/24

http://localhost:8080/user/page/3/14

http://localhost:8080/user/page/2/4

```
2022-11-01 14:07:58.821  INFO 14052 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-11-01 14:07:58.821  INFO 14052 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2022-11-01 14:07:58.822  INFO 14052 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 0 ms
2022-11-01 14:07:59.186  INFO 14052 --- [         task-1] mao.use_starter.service.LogService       : 保存日志信息:
OptLogDTO{requestIp='0:0:0:0:0:0:0:1', type='OPT', userName='', description='分页查询用户', classPath='mao.use_starter.controller.UserController', actionMethod='findByPage', requestUri='/user/page/1/24', httpMethod='GET', params='[1, 24]', result='null', exDesc='null', exDetail='null', startTime=2022-11-01T14:07:58.854014400, finishTime=2022-11-01T14:07:59.181036200, consumingTime=327, ua='Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36 Edg/107.0.1418.26', createUser=0}
2022-11-01 14:08:08.033  INFO 14052 --- [         task-2] mao.use_starter.service.LogService       : 保存日志信息:
OptLogDTO{requestIp='0:0:0:0:0:0:0:1', type='OPT', userName='', description='分页查询用户', classPath='mao.use_starter.controller.UserController', actionMethod='findByPage', requestUri='/user/page/3/14', httpMethod='GET', params='[3, 14]', result='null', exDesc='null', exDetail='null', startTime=2022-11-01T14:08:07.719294200, finishTime=2022-11-01T14:08:08.033604, consumingTime=314, ua='Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36 Edg/107.0.1418.26', createUser=0}
2022-11-01 14:08:16.123  INFO 14052 --- [         task-3] mao.use_starter.service.LogService       : 保存日志信息:
OptLogDTO{requestIp='0:0:0:0:0:0:0:1', type='OPT', userName='', description='分页查询用户', classPath='mao.use_starter.controller.UserController', actionMethod='findByPage', requestUri='/user/page/2/4', httpMethod='GET', params='[2, 4]', result='null', exDesc='null', exDetail='null', startTime=2022-11-01T14:08:15.816589600, finishTime=2022-11-01T14:08:16.123406500, consumingTime=306, ua='Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36 Edg/107.0.1418.26', createUser=0}
```

第九步：更改UserController

```java
package mao.use_starter.controller;

import mao.tools_log.annotation.SysLog;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
```

```java
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * Project name(项目名称)：logback_spring_boot_starter_demo
 * Package(包名): mao.use_starter.controller
 * Class(类名): UserController
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/10/31
 * Time(创建时间)： 22:41
 * Version(版本): 1.0
 * Description(描述)： 无
 */

@RestController
@RequestMapping("/user")
public class UserController
{
    @SysLog("分页查询用户")//记录操作日志
    @GetMapping(value = "page/{pageNum}/{pageSize}")
    public String findByPage(@PathVariable Integer pageNum, @PathVariable
Integer pageSize)
    {
        try
        {
            Thread.sleep(300);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        return "OK";
    }

    @SysLog("获取使用用户的信息")
    @GetMapping()
    public String getAll()
    {
        throw new RuntimeException("不允许使用");
    }
}
```

第十步：重启服务并访问

http://localhost:8080/user

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Nov 01 14:14:51 CST 2022
There was an unexpected error (type=Internal Server Error, status=500).

```
2022-11-01 14:14:51.523  INFO 11404 --- [         task-2] mao.use_starter.service.LogService       : 保存日志信息:OptLogDTO{requestIp='0:0:0:0:0:0:0:1', type='EX', userName='', description='获取使用用户的信息', classPath='mao.use_starter.controller.UserController', actionMethod='getAll', requestUri='/user', httpMethod='GET', params='[]', result='null', exDesc='不允许使用', exDetail='java.lang.RuntimeException: 不允许使用
    at mao.use_starter.controller.UserController.getAll(UserController.java:45)
    at mao.use_starter.controller.UserController$$FastClassBySpringCGLIB$$828362cf.invoke(<generated>)
    at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:793)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:763)
    at org.springframework.aop.aspectj.AspectJAfterThrowingAdvice.invoke(AspectJAfterThrowingAdvice.java:64)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:175)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:763)
    at org.springframework.aop.framework.adapter.AfterReturningAdviceInterceptor.invoke(AfterReturningAdviceInterceptor.java:57)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:175)
```

```
    at
org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:763)
    at
org.springframework.aop.framework.adapter.MethodBeforeAdviceInterceptor.invoke(MethodBeforeAdviceInterceptor.java:58)
    at
org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:175)
    at
org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:763)
    at
org.springframework.aop.interceptor.ExposeInvocationInterceptor.invoke(ExposeInvocationInterceptor.java:97)
    at
org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:186)
    at
org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:763)
    at
org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:708)
    at
mao.use_starter.controller.UserController$$EnhancerBySpringCGLIB$$c6751aa7.getAll(<generated>)
    at
java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)
    at
java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:78)
    at
java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:567)
    at
org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:205)
    at
org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:150)
    at
org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:117)
    at
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:895)
    at
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:808)
    at
org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:87)
    at
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:1067)
```

```
33      at
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServle
t.java:963)
34      at
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkSer
vlet.java:1006)
35      at
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java
:898)
36      at javax.servlet.http.HttpServlet.service(HttpServlet.java:655)
37      at
org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.ja
va:883)
38      at javax.servlet.http.HttpServlet.service(HttpServlet.java:764)
39      at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(Application
FilterChain.java:227)
40      at
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterCh
ain.java:162)
41      at
org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)
42      at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(Application
FilterChain.java:189)
43      at
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterCh
ain.java:162)
44      at
org.springframework.web.filter.RequestContextFilter.doFilterInternal(Request
ContextFilter.java:100)
45      at
org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestF
ilter.java:117)
46      at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(Application
FilterChain.java:189)
47      at
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterCh
ain.java:162)
48      at
org.springframework.web.filter.FormContentFilter.doFilterInternal(FormConten
tFilter.java:93)
49      at
org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestF
ilter.java:117)
50      at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(Application
FilterChain.java:189)
51      at
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterCh
ain.java:162)
52      at
org.springframework.web.filter.CharacterEncodingFilter.doFilterInternal(Char
acterEncodingFilter.java:201)
53      at
org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestF
ilter.java:117)
```

```
54      at
    org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(Application
    FilterChain.java:189)
55      at
    org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterCh
    ain.java:162)
56      at
    org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.ja
    va:197)
57      at
    org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.ja
    va:97)
58      at
    org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase
    .java:541)
59      at
    org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:135
    )
60      at
    org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:92)
61      at
    org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java
    :78)
62      at
    org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:360)
63      at
    org.apache.coyote.http11.Http11Processor.service(Http11Processor.java:399)
64      at
    org.apache.coyote.AbstractProcessorLight.process(AbstractProcessorLight.java
    :65)
65      at
    org.apache.coyote.AbstractProtocol$ConnectionHandler.process(AbstractProtoco
    l.java:890)
66      at
    org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.jav
    a:1787)
67      at
    org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBase.java:
    49)
68      at
    org.apache.tomcat.util.threads.ThreadPoolExecutor.runWorker(ThreadPoolExecut
    or.java:1191)
69      at
    org.apache.tomcat.util.threads.ThreadPoolExecutor$Worker.run(ThreadPoolExecu
    tor.java:659)
70      at
    org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.ja
    va:61)
71      at java.base/java.lang.Thread.run(Thread.java:831)
72  ', startTime=2022-11-01T14:14:51.522486900, finishTime=2022-11-
    01T14:14:51.523499, consumingTime=1, ua='Mozilla/5.0 (Windows NT 10.0;
    Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0
    Safari/537.36 Edg/107.0.1418.26', createUser=0}
73  2022-11-01 14:14:51.523 ERROR 11404 --- [nio-8080-exec-5] o.a.c.c.C.[.[.[/].
    [dispatcherServlet]    : Servlet.service() for servlet [dispatcherServlet]
    in context with path [] threw exception [Request processing failed; nested
    exception is java.lang.RuntimeException: 不允许使用] with root cause
74
```

```
java.lang.RuntimeException: 不允许使用
    at mao.use_starter.controller.UserController.getAll(UserController.java:45)
    at mao.use_starter.controller.UserController$$FastClassBySpringCGLIB$$828362cf.invoke(<generated>)
    at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:793)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:763)
...
...
...
```

---

end

---

by mao

2022 11 01
