Spring Boot集成ELK

参考链接：

https://github.com/qos-ch/logback/tree/v_1.2.3/logback-examples

https://github.com/logfellow/logstash-logback-encoder/tree/logstash-logback-encoder-6.6

1 分场景配置

为了方便我们查看日志，我们把日志分为以下四种：

- 调试日志：最全日志，包含了应用中所有`DEBUG`级别以上的日志，仅在开发、测试环境中开启收集；
- 错误日志：只包含应用中所有`ERROR`级别的日志，所有环境只都开启收集；
- 业务日志：在我们应用`对应包下`打印的日志，可用于查看我们自己在应用中打印的业务日志；
- 记录日志：每个接口的`访问记录`，可以用来查看接口执行效率，获取接口访问参数。

下面来看一个完整的配置示例，配置文件命名为`logback-spring.xml`，放在`classpath`下（resources根目录）。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration>
<configuration>
    <!-- 引用默认日志配置 -->
    <include resource="org/springframework/boot/logging/logback/defaults.xml"/>
    <!-- 使用默认的控制台日志输出实现 -->
    <include resource="org/springframework/boot/logging/logback/console-appender.xml"/>
    <!-- 日志文件保存路径 -->
    <property name="LOG_FILE_PATH" value="${LOG_FILE:-${LOG_PATH:-${LOG_TEMP:-${java.io.tmpdir:-/tmp}}}/logs}"/>
    <!-- 项目名称 -->
    <springProperty name="PROJ_NAME" scope="context" source="application.title" defaultValue="01-star"/>
    <!-- 应用名称 -->
    <springProperty name="APP_NAME" scope="context" source="spring.application.name" defaultValue="spring-boot"/>
    <!-- LogStash访问host -->
    <springProperty name="LOG_STASH_HOST" scope="context" source="logstash.host" defaultValue="localhost"/>

    <!-- DEBUG日志输出到文件 -->
    <appender name="FILE_DEBUG" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- 输出DEBUG以上级别日志 -->
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>DEBUG</level>
        </filter>
```

```
        <encoder>
            <!-- 设置为默认的文件日志格式 -->
            <pattern>${FILE_LOG_PATTERN}</pattern>
            <charset>UTF-8</charset>
        </encoder>
        <rollingPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
            <!-- 设置文件命名格式 -->
            <fileNamePattern>${LOG_FILE_PATH}/debug/${APP_NAME}-%d{yyyy-MM-
dd}-%i.log</fileNamePattern>
            <!-- 设置日志文件大小，超过就重新生成文件，默认10M -->
            <maxFileSize>${LOG_FILE_MAX_SIZE:-10MB}</maxFileSize>
            <!-- 日志文件保留天数，默认30天 -->
            <maxHistory>${LOG_FILE_MAX_HISTORY:-30}</maxHistory>
        </rollingPolicy>
    </appender>

    <!-- ERROR日志输出到文件 -->
    <appender name="FILE_ERROR"
              class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- 只输出ERROR级别的日志 -->
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>ERROR</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
        <encoder>
            <!-- 设置为默认的文件日志格式 -->
            <pattern>${FILE_LOG_PATTERN}</pattern>
            <charset>UTF-8</charset>
        </encoder>
        <rollingPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
            <!-- 设置文件命名格式 -->
            <fileNamePattern>${LOG_FILE_PATH}/error/${APP_NAME}-%d{yyyy-MM-
dd}-%i.log</fileNamePattern>
            <!-- 设置日志文件大小，超过就重新生成文件，默认10M -->
            <maxFileSize>${LOG_FILE_MAX_SIZE:-10MB}</maxFileSize>
            <!-- 日志文件保留天数，默认30天 -->
            <maxHistory>${LOG_FILE_MAX_HISTORY:-30}</maxHistory>
        </rollingPolicy>
    </appender>

    <!-- DEBUG日志输出到LogStash -->
    <appender name="LOG_STASH_DEBUG"
class="net.logstash.logback.appender.LogstashTcpSocketAppender">
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>DEBUG</level>
        </filter>
        <destination>${LOG_STASH_HOST}:4560</destination>
        <encoder charset="UTF-8"
class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
            <providers>
                <timestamp>
                    <timeZone>Asia/Shanghai</timeZone>
```

```
                </timestamp>
                <!-- 自定义日志输出格式 -->
                <pattern>
                    <pattern>
                        {
                        "project": "${PROJ_NAME:-}",
                        "level": "%level",
                        "service": "${APP_NAME:-}",
                        "pid": "${PID:-}",
                        "thread": "%thread",
                        "class": "%logger",
                        "message": "%message",
                        "stack_trace": "%exception{20}"
                        }
                    </pattern>
                </pattern>
            </providers>
        </encoder>
        <!-- 当有多个LogStash服务时，设置访问策略为轮询 -->
        <connectionStrategy>
            <roundRobin>
                <connectionTTL>5 minutes</connectionTTL>
            </roundRobin>
        </connectionStrategy>
    </appender>

    <!-- ERROR日志输出到LogStash -->
    <appender name="LOG_STASH_ERROR"
class="net.logstash.logback.appender.LogstashTcpSocketAppender">
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>ERROR</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
        <destination>${LOG_STASH_HOST}:4561</destination>
        <encoder charset="UTF-8"
class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
            <providers>
                <timestamp>
                    <timeZone>Asia/Shanghai</timeZone>
                </timestamp>
                <!-- 自定义日志输出格式 -->
                <pattern>
                    <pattern>
                        {
                        "project": "${PROJ_NAME:-}",
                        "level": "%level",
                        "service": "${APP_NAME:-}",
                        "pid": "${PID:-}",
                        "thread": "%thread",
                        "class": "%logger",
                        "message": "%message",
                        "stack_trace": "%exception{20}"
                        }
                    </pattern>
```

```
                </pattern>
            </providers>
        </encoder>
        <!-- 当有多个LogStash服务时，设置访问策略为轮询 -->
        <connectionStrategy>
            <roundRobin>
                <connectionTTL>5 minutes</connectionTTL>
            </roundRobin>
        </connectionStrategy>
    </appender>

    <!-- 业务日志输出到LogStash -->
    <appender name="LOG_STASH_BUSINESS"
class="net.logstash.logback.appender.LogstashTcpSocketAppender">
        <destination>${LOG_STASH_HOST}:4562</destination>
        <encoder charset="UTF-8"
class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
            <providers>
                <timestamp>
                    <timeZone>Asia/Shanghai</timeZone>
                </timestamp>
                <!-- 自定义日志输出格式 -->
                <pattern>
                    <pattern>
                        {
                        "project": "${PROJ_NAME:-}",
                        "level": "%level",
                        "service": "${APP_NAME:-}",
                        "pid": "${PID:-}",
                        "thread": "%thread",
                        "class": "%logger",
                        "message": "%message",
                        "stack_trace": "%exception{20}"
                        }
                    </pattern>
                </pattern>
            </providers>
        </encoder>
        <!-- 当有多个LogStash服务时，设置访问策略为轮询 -->
        <connectionStrategy>
            <roundRobin>
                <connectionTTL>5 minutes</connectionTTL>
            </roundRobin>
        </connectionStrategy>
    </appender>

    <!-- 接口访问记录日志输出到LogStash -->
    <appender name="LOG_STASH_RECORD"
class="net.logstash.logback.appender.LogstashTcpSocketAppender">
        <destination>${LOG_STASH_HOST}:4563</destination>
        <encoder charset="UTF-8"
class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
            <providers>
                <timestamp>
                    <timeZone>Asia/Shanghai</timeZone>
```

```
                </timestamp>
                <!-- 自定义日志输出格式 -->
                <pattern>
                    <pattern>
                        {
                        "project": "${PROJ_NAME:-}",
                        "level": "%level",
                        "service": "${APP_NAME:-}",
                        "class": "%logger",
                        "message": "%message"
                        }
                    </pattern>
                </pattern>
            </providers>
        </encoder>
        <!-- 当有多个LogStash服务时，设置访问策略为轮询 -->
        <connectionStrategy>
            <roundRobin>
                <connectionTTL>5 minutes</connectionTTL>
            </roundRobin>
        </connectionStrategy>
    </appender>

    <!-- root appender DEBUG级别以上日志 -->
    <root level="DEBUG">
        <!-- 控制台日志 -->
        <appender-ref ref="CONSOLE"/>
        <!-- 调试日志 -->
        <appender-ref ref="LOG_STASH_DEBUG"/>
        <!-- 错误日志 -->
        <appender-ref ref="LOG_STASH_ERROR"/>
    </root>
    <!-- 业务日志 -->
    <logger name="com.zeroone.star.projectlog" level="DEBUG">
        <appender-ref ref="LOG_STASH_BUSINESS"/>
    </logger>
    <!-- 记录日志 -->
    <logger name="com.zeroone.star.projectlog.component.WebLogAspect"
level="DEBUG">
        <appender-ref ref="LOG_STASH_RECORD"/>
    </logger>

    <!--控制框架输出日志-->
    <logger name="org.slf4j" level="INFO"/>
    <logger name="springfox" level="INFO"/>
    <logger name="io.swagger" level="INFO"/>
    <logger name="org.springframework" level="INFO"/>
    <logger name="org.hibernate.validator" level="INFO"/>
</configuration>
```

后面我们分开来对每个配置要点进行解释。

2 Logback配置

2.1 控制台日志配置

一般我们不需要自定义控制台输出，可以采用默认配置。

具体配置参考`console-appender.xml`，该文件在`spring-boot-${version}.jar`下面。

```
<!-- 引用默认日志配置 -->
<include resource="org/springframework/boot/logging/logback/defaults.xml"/>
<!-- 使用默认的控制台日志输出实现 -->
<include resource="org/springframework/boot/logging/logback/console-appender.xml"/>
```

2.2 springProperty

该标签可以从Spring Boot的配置文件中获取配置属性，比如说在不同环境下我们的`Logstash`服务地址是不一样的，我们就可以把该地址定义在`application.yml`来使用。

例如在`application-dev.yml`中定义了这些属性：

```
logstash:
  host: localhost
```

在`logback-spring.xml`中就可以直接这样使用：

```
<!-- 应用名称 -->
<springProperty name="APP_NAME" scope="context"
                source="spring.application.name" defaultValue="spring-boot"/>
<!-- LogStash访问host -->
<springProperty name="LOG_STASH_HOST" scope="context"
                source="logstash.host" defaultValue="localhost"/>
```

2.3 filter

在`Logback`中有两种不同的过滤器，用来过滤日志输出。

`ThresholdFilter`：临界值过滤器，过滤掉低于指定临界值的日志，比如下面的配置将过滤掉所有低于INFO级别的日志。

```
<filter class="ch.qos.logback.classic.filter.ThresholdFilter">
    <level>INFO</level>
</filter>
```

`LevelFilter`：级别过滤器，根据日志级别进行过滤，比如下面的配置将过滤掉所有非ERROR级别的日志。

```
<filter class="ch.qos.logback.classic.filter.LevelFilter">
    <level>ERROR</level>
    <onMatch>ACCEPT</onMatch>
    <onMismatch>DENY</onMismatch>
</filter>
```

2.4 appender

Appender可以用来控制日志的输出形式，主要有下面三种。

`ConsoleAppender`：控制日志输出到控制台的形式，比如在`console-appender.xml`中定义的默认控制台输出。

```
<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
        <pattern>${CONSOLE_LOG_PATTERN}</pattern>
    </encoder>
</appender>
```

`RollingFileAppender`：控制日志输出到文件的形式，可以控制日志文件生成策略，比如文件名称格式、超过多大重新生成文件以及删除超过多少天的文件。

```
<!-- ERROR日志输出到文件 -->
<appender name="FILE_ERROR"
          class="ch.qos.logback.core.rolling.RollingFileAppender">
    <rollingPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
        <!-- 设置文件命名格式 -->
        <fileNamePattern>${LOG_FILE_PATH}/error/${APP_NAME}-%d{yyyy-MM-dd}-
%i.log</fileNamePattern>
        <!-- 设置日志文件大小，超过就重新生成文件，默认10M -->
        <maxFileSize>${LOG_FILE_MAX_SIZE:-10MB}</maxFileSize>
        <!-- 日志文件保留天数，默认30天 -->
        <maxHistory>${LOG_FILE_MAX_HISTORY:-30}</maxHistory>
    </rollingPolicy>
</appender>
```

`LogstashTcpSocketAppender`：控制日志输出到`Logstash`的形式，可以用来配置`Logstash`的地址、访问策略以及日志的格式。

```
<!-- DEBUG日志输出到LogStash -->
<appender name="LOG_STASH_DEBUG"
class="net.logstash.logback.appender.LogstashTcpSocketAppender">
    <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
        <level>DEBUG</level>
    </filter>
    <destination>${LOG_STASH_HOST}:4560</destination>
    <encoder charset="UTF-8"
class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
        <providers>
            <timestamp>
                <timeZone>Asia/Shanghai</timeZone>
            </timestamp>
            <!-- 自定义日志输出格式 -->
            <pattern>
                <pattern>
                    {
                    "project": "${PROJ_NAME:-}",
                    "level": "%level",
                    "service": "${APP_NAME:-}",
```

```
                    "pid": "${PID:-}",
                    "thread": "%thread",
                    "class": "%logger",
                    "message": "%message",
                    "stack_trace": "%exception{20}"
                    }
                </pattern>
            </pattern>
        </providers>
    </encoder>
    <!-- 当有多个LogStash服务时，设置访问策略为轮询 -->
    <connectionStrategy>
        <roundRobin>
            <connectionTTL>5 minutes</connectionTTL>
        </roundRobin>
    </connectionStrategy>
</appender>
```

2.5 logger

只有配置到logger节点上的appender才会被使用，logger用于配置哪种条件下的日志被打印，root是一种特殊的appender，下面介绍下日志划分的条件。

- 调试日志：所有的DEBUG级别以上日志；
- 错误日志：所有的ERROR级别日志；
- 业务日志：`com.zeroone.star.projectlog`包下的所有DEBUG级别以上日志；
- 记录日志：`com.zeroone.star.projectlog.component.WebLogAspect`类下所有DEBUG级别以上日志，该类是统计接口访问信息的`AOP`切面类。

还有一些使用框架内部的日志，DEBUG级别的日志对我们并没有啥用处，都可以设置为了INFO以上级别。

```
<!-- 控制框架输出日志 -->
<logger name="org.slf4j" level="INFO"/>
<logger name="springfox" level="INFO"/>
<logger name="io.swagger" level="INFO"/>
<logger name="org.springframework" level="INFO"/>
<logger name="org.hibernate.validator" level="INFO"/>
```

3 `Logstash`配置

接下来我们需要配置下`Logstash`，让它可以分场景收集不同的日志，下面详细介绍下使用到的配置。

```
input {
  tcp {
    mode => "server"
    host => "0.0.0.0"
    port => 4560
    codec => json_lines
    type => "debug"
  }
  tcp {
    mode => "server"
    host => "0.0.0.0"
```

```
    port => 4561
    codec => json_lines
    type => "error"
  }
  tcp {
    mode => "server"
    host => "0.0.0.0"
    port => 4562
    codec => json_lines
    type => "business"
  }
  tcp {
    mode => "server"
    host => "0.0.0.0"
    port => 4563
    codec => json_lines
    type => "record"
  }
}
filter{
  if [type] == "record" {
    mutate {
      remove_field => "port"
      remove_field => "host"
      remove_field => "@version"
    }
    json {
      source => "message"
      remove_field => ["message"]
    }
  }
}
output {
  elasticsearch {
    hosts => "es:9200"
    action => "index"
    codec => json
    index => "project-%{type}-%{+YYYY.MM.dd}"
    template_name => "project"
  }
}
```

3.1 配置要点

- input：使用不同端口收集不同类型的日志，从4560~4563开启四个端口；
- filter：对于记录类型的日志，直接将`JSON`格式的message转化到source中去，便于搜索查看；
- output：按类型、时间自定义索引格式。

4 Spring Boot配置

在Spring Boot中的配置可以直接用来覆盖`Logback`中的配置，比如`logging.level.root`就可以覆盖`<root>`节点中的`level`配置。

开发环境配置：`application-dev.yml`

```yaml
logstash:
  host: 192.168.220.127
logging:
  level:
    root: debug
```

测试环境配置：`application-test.yml`

```yaml
logstash:
  host: 192.168.220.128
logging:
  level:
    root: debug
```

生产环境配置：`application-prod.yml`

```yaml
logstash:
  host: 192.168.220.129
logging:
  level:
    root: info
```

5 Spring Boot集成案例

5.1 添加依赖

```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>

    <groupId>com.zeroone.star</groupId>
    <artifactId>project-log</artifactId>
    <version>1.0.0-SNAPSHOT</version>

    <properties>
        <java.version>1.8</java.version>
        <spring-boot.version>2.3.12.RELEASE</spring-boot.version>
        <logstash-logback-encoder.version>6.6</logstash-logback-
encoder.version>
        <knife4j.version>2.0.8</knife4j.version>
        <hutool.version>5.8.3</hutool.version>
        <mybaits.plus.version>3.4.3.4</mybaits.plus.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-aop</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <!-- logstash logback encoder -->
        <dependency>
            <groupId>net.logstash.logback</groupId>
            <artifactId>logstash-logback-encoder</artifactId>
        </dependency>
        <!-- knife4j -->
        <dependency>
            <groupId>com.github.xiaoymin</groupId>
            <artifactId>knife4j-spring-boot-starter</artifactId>
        </dependency>
        <!-- hutool -->
        <dependency>
            <groupId>cn.hutool</groupId>
            <artifactId>hutool-all</artifactId>
        </dependency>
        <!-- mybatis-plus-extension -->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-extension</artifactId>
        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <!-- spring boot -->
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>${spring-boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <!-- logstash logback encoder -->
            <dependency>
                <groupId>net.logstash.logback</groupId>
                <artifactId>logstash-logback-encoder</artifactId>
                <version>${logstash-logback-encoder.version}</version>
            </dependency>
            <!-- knife4j -->
            <dependency>
                <groupId>com.github.xiaoymin</groupId>
                <artifactId>knife4j-spring-boot-starter</artifactId>
                <version>${knife4j.version}</version>
            </dependency>
            <!-- hutool -->
            <dependency>
                <groupId>cn.hutool</groupId>
```

```
                <artifactId>hutool-all</artifactId>
                <version>${hutool.version}</version>
                <optional>true</optional>
            </dependency>
            <!-- mybatis-plus-extension -->
            <dependency>
                <groupId>com.baomidou</groupId>
                <artifactId>mybatis-plus-extension</artifactId>
                <version>${mybaits.plus.version}</version>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>${spring-boot.version}</version>
                <executions>
                    <execution>
                        <id>repackage</id>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

5.2 添加`logback`配置

参考：1 分场景配置

5.3 修改项目配置

下面是`application.yml`配置

```yaml
application:
  title: 测试项目
spring:
  application:
    name: PROJECT-LOG
  profiles:
    active: dev
server:
  port: 8080
```

多环境配置，参考4 Spring Boot配置

5.4 关键代码

5.4.1 访问记录对象

创建一个访问记录类，用于记录访问。

```java
@Data
public class WebLog {
    /**
     * 操作描述
     */
    private String description;
    /**
     * 操作用户
     */
    private String username;
    /**
     * 操作时间
     */
    private Long startTime;
    /**
     * 消耗时间
     */
    private Integer spendTime;
    /**
     * 根路径
     */
    private String basePath;
    /**
     * URI
     */
    private String uri;
    /**
     * URL
     */
    private String url;
    /**
     * 请求类型
     */
    private String method;
    /**
     * IP地址
     */
    private String ip;
    /**
     * 请求参数
     */
    private Object parameter;
    /**
     * 请求返回的结果
     */
    private Object result;
}
```

5.4.2 AOP切面

书写一个AOP切面拦截请求，生成请求记录日志

```
@Aspect
@Component
@Order(1)
public class WebLogAspect {
    private static final Logger LOGGER = LoggerFactory.getLogger(WebLogAspect.class);
    @Pointcut("execution(public * com.zeroone.star.projectlog.controller.*.*(..))")
    public void webLog() {
    }
    @Around("webLog()")
    public Object doAround(ProceedingJoinPoint joinPoint) throws Throwable {
        long startTime = System.currentTimeMillis();
        //获取当前请求对象
        ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();
        HttpServletRequest request = Objects.requireNonNull(attributes).getRequest();
        //记录请求信息
        WebLog webLog = new WebLog();
        Object result = joinPoint.proceed();
        Signature signature = joinPoint.getSignature();
        MethodSignature methodSignature = (MethodSignature) signature;
        Method method = methodSignature.getMethod();
        if (method.isAnnotationPresent(ApiOperation.class)) {
            ApiOperation apiOperation = method.getAnnotation(ApiOperation.class);
            webLog.setDescription(apiOperation.value());
        }
        long endTime = System.currentTimeMillis();
        String urlStr = request.getRequestURL().toString();
        webLog.setBasePath(StrUtil.removeSuffix(urlStr, URLUtil.url(urlStr).getPath()));
        webLog.setIp(request.getRemoteUser());
        webLog.setMethod(request.getMethod());
        webLog.setParameter(getParameter(method, joinPoint.getArgs()));
        webLog.setResult(result);
        webLog.setSpendTime((int) (endTime - startTime));
        webLog.setStartTime(startTime);
        webLog.setUri(request.getRequestURI());
        webLog.setUrl(request.getRequestURL().toString());
        Map<String, Object> logMap = new HashMap<>(5);
        logMap.put("url", webLog.getUrl());
        logMap.put("method", webLog.getMethod());
        logMap.put("parameter", webLog.getParameter());
        logMap.put("spendTime", webLog.getSpendTime());
        logMap.put("description", webLog.getDescription());
        LOGGER.info(Markers.appendEntries(logMap), JSONUtil.parse(webLog).toString());
        return result;
    }
```

```
    /**
     * 根据方法和传入的参数获取请求参数
     */
    private Object getParameter(Method method, Object[] args) {
        List<Object> argList = new ArrayList<>();
        Parameter[] parameters = method.getParameters();
        for (int i = 0; i < parameters.length; i++) {
            //将RequestBody注解修饰的参数作为请求参数
            RequestBody requestBody =
parameters[i].getAnnotation(RequestBody.class);
            if (requestBody != null) {
                argList.add(args[i]);
            }
            //将RequestParam注解修饰的参数作为请求参数
            RequestParam requestParam =
parameters[i].getAnnotation(RequestParam.class);
            String clsName = args[i].getClass().getName();
            if (requestParam != null) {
                Map<String, Object> map = new HashMap<>(1);
                String key = parameters[i].getName();
                if (!StringUtils.isEmpty(requestParam.value())) {
                    key = requestParam.value();
                }
                map.put(key, args[i]);
                argList.add(map);
            }
            //处理对象类型的RequestParam参数
            else if (clsName.contains(".dto.") ||
clsName.contains(".query.")) {
                argList.add(args[i]);
            }
        }
        if (argList.size() == 0) {
            return null;
        } else if (argList.size() == 1) {
            return argList.get(0);
        } else {
            return argList;
        }
    }
}
```

5.4.3 Swagger配置

参考项目演示示例代码中，我们集成的是knife4j

5.4.4 其他领域模型

其他领域模型类参考示例源码

5.4.5 测试控制器

书写一个测试控制器

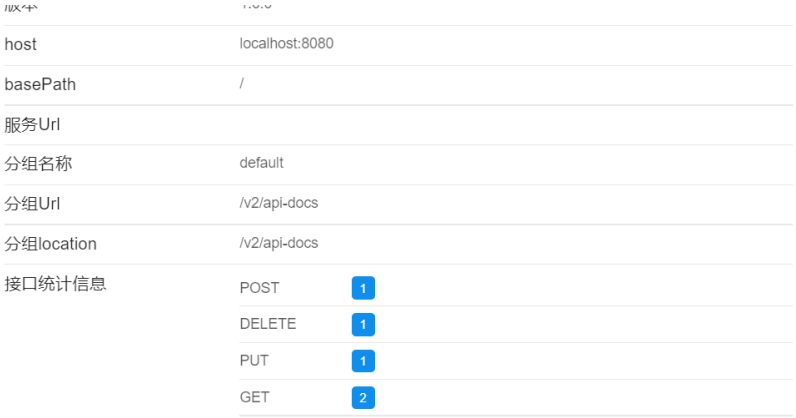
```
@Api(tags = "测试API")
@RestController
@RequestMapping("/test")
public class TestController {

    @ApiOperation("获取所有")
    @GetMapping("query-all")
    public JsonVO<PageVO<SampleVO>> queryAll() {
        List<SampleVO> voList = new ArrayList<>(10);
        for (int i = 0; i < 10; i++) {
            SampleVO sampleVO = new SampleVO();
            sampleVO.setId(i + 1);
            sampleVO.setAge(11);
            sampleVO.setName("李四-" + (i + 1));
            sampleVO.setSex("男");
            voList.add(sampleVO);
        }
        PageVO<SampleVO> pv = new PageVO<>(1L, 12L, 10L, 1L, voList);
        return JsonVO.success(pv);
    }

    @ApiOperation("添加数据")
    @PostMapping("add")
    public JsonVO<Long> add(SampleDTO data) {
        return JsonVO.success(0L);
    }

    @ApiOperation("修改数据")
    @PutMapping("modify")
    public JsonVO<Long> modify(SampleDTO data) {
        return JsonVO.success(1L);
    }

    @ApiOperation("删除数据")
    @DeleteMapping("delete/{id}")
    public JsonVO<Long> delete(@PathVariable("id") Long id) {
        return JsonVO.success(id);
    }

    @ApiOperation("条件查询")
    @GetMapping("query")
    public JsonVO<PageVO<SampleVO>> query(SampleQuery query) {
        SampleVO sampleVO = new SampleVO();
        sampleVO.setId(1);
        sampleVO.setAge(11);
        sampleVO.setName(query.getName());
        if ("张三".equals(query.getName())) {
            sampleVO.setSex("男");
        } else {
            sampleVO.setSex("女");
        }
```

```
52          List<SampleVO> voList = new ArrayList<>(1);
53          voList.add(sampleVO);
54          PageVO<SampleVO> pv = new PageVO<>(1L, 12L, 1L, 1L, voList);
55          return JsonVO.success(pv);
56      }
57  }
```

5.5 启动项目

启动项目，观察控制台是否有报错，如果没有报错，访问API文档页面，http://localhost:8080/doc.html

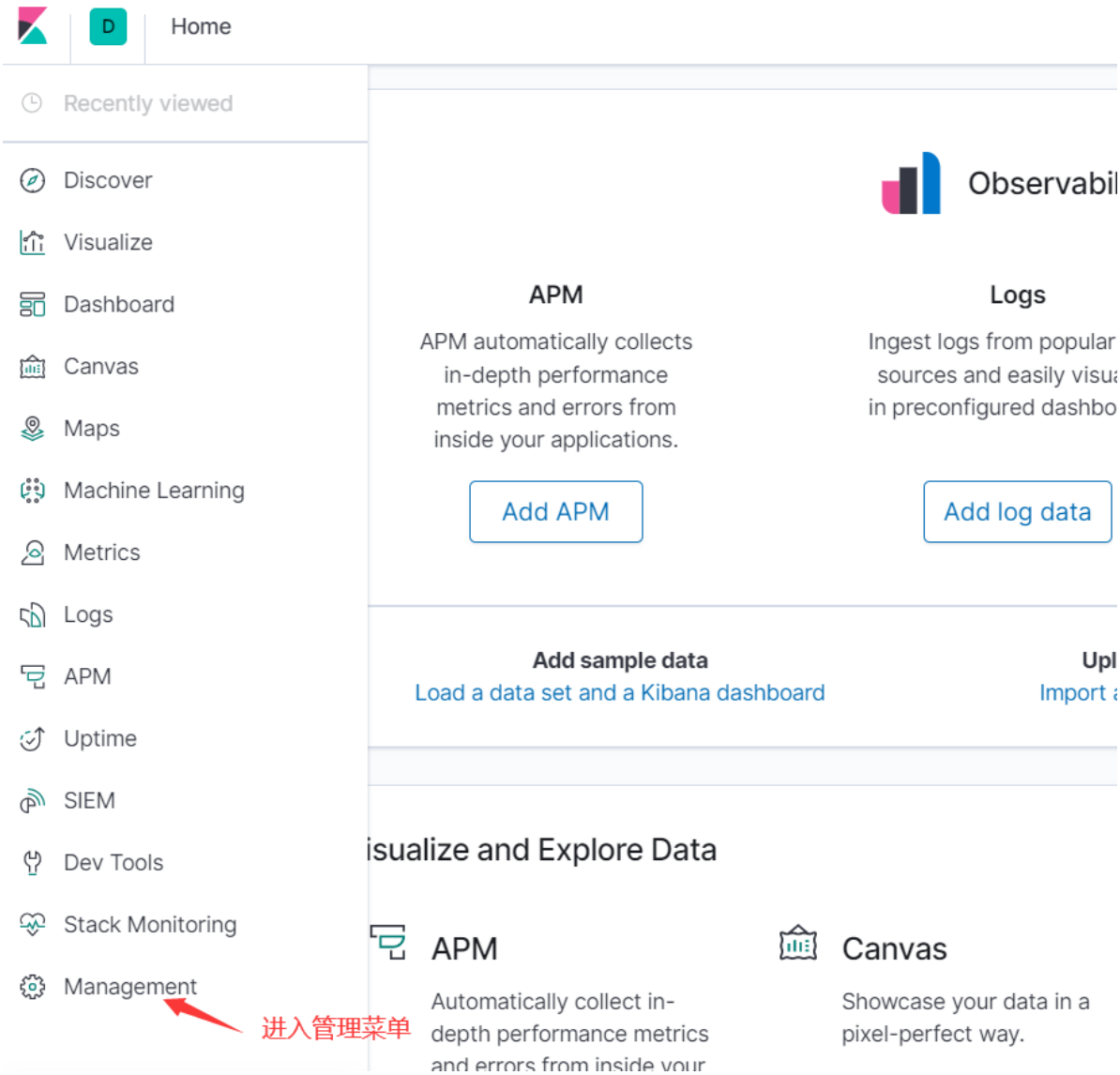
看到如下图所示的页面



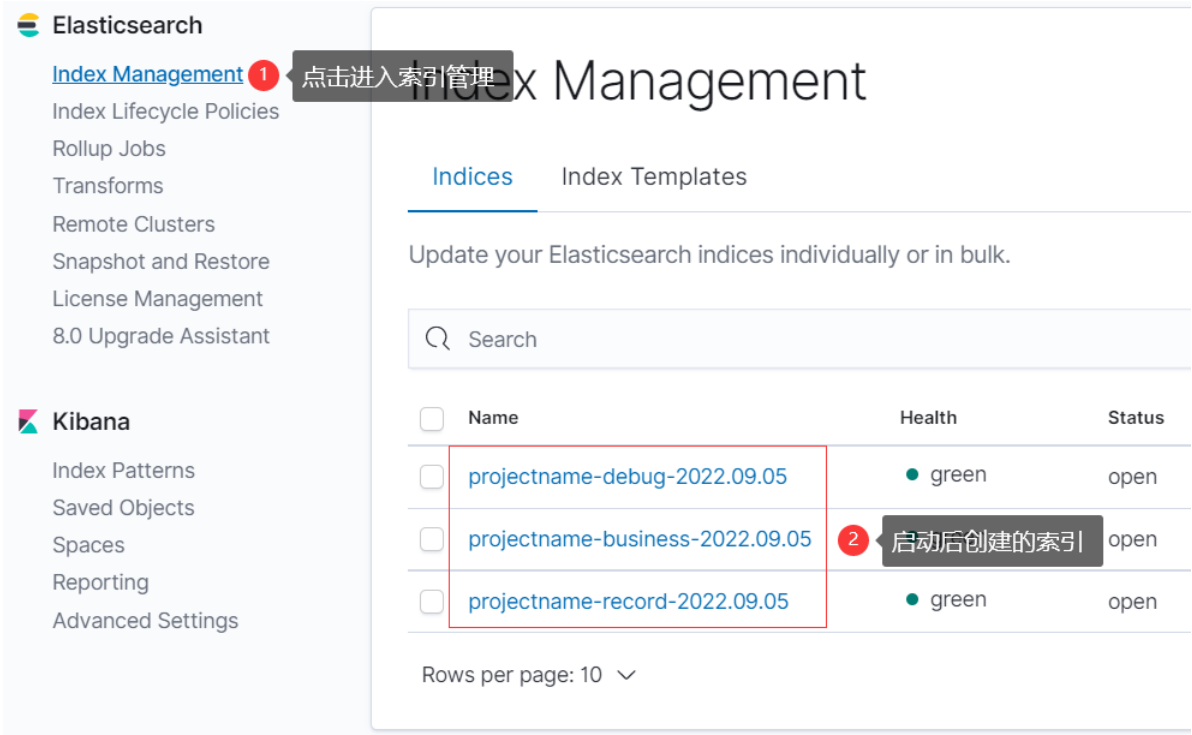
5.6 在`Kibana`中查看

访问你的`Kibana`，没有修改端口的访问地址为，http://ip:5601

5.6.1 查看索引

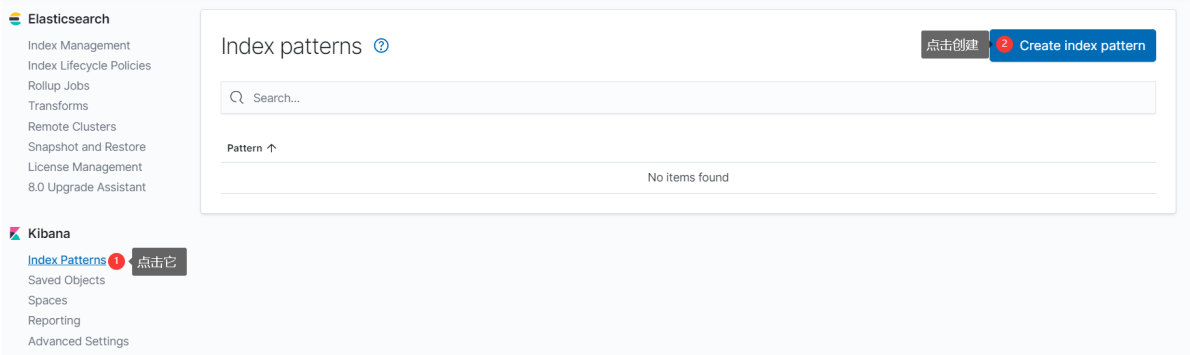


点击，可以查看已经产生的索引

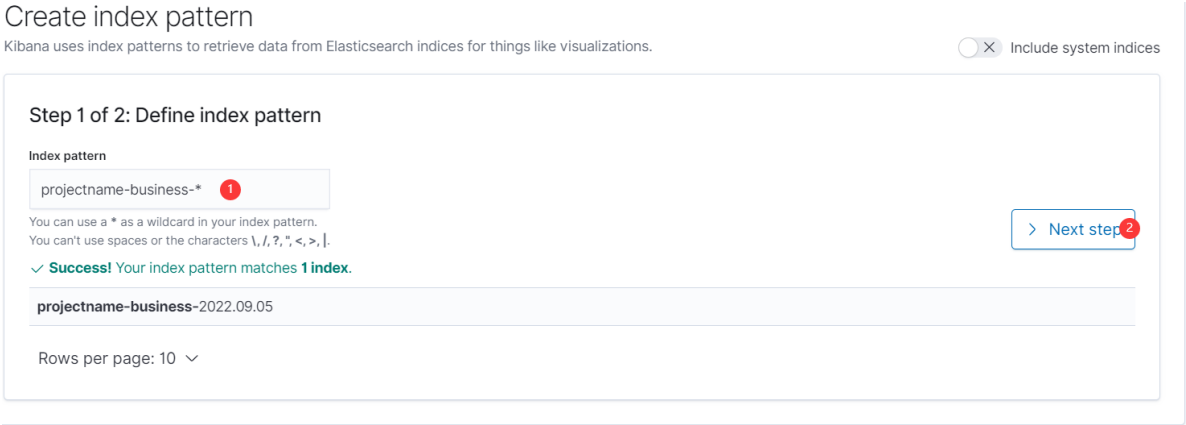


5.6.2 索引模式

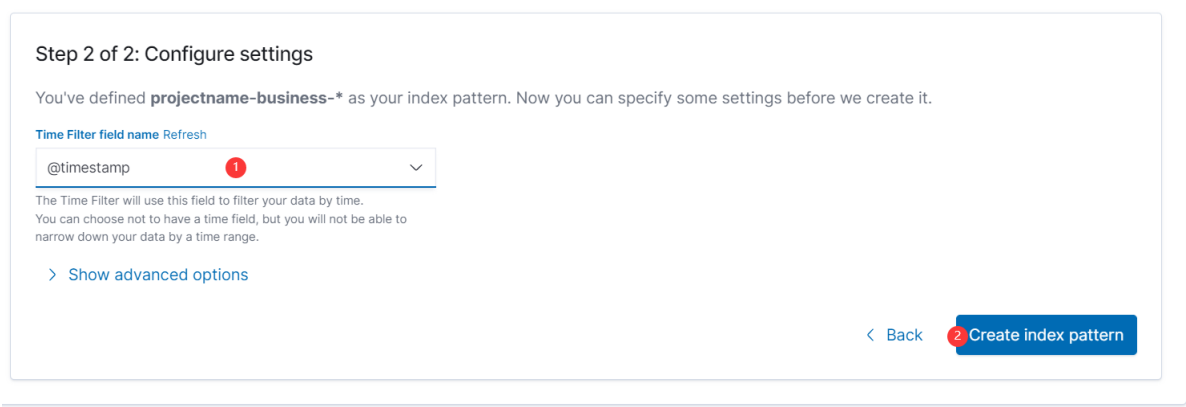
要使用`Kibana`查看索引详细信息，首先创建索引模式。



步骤1

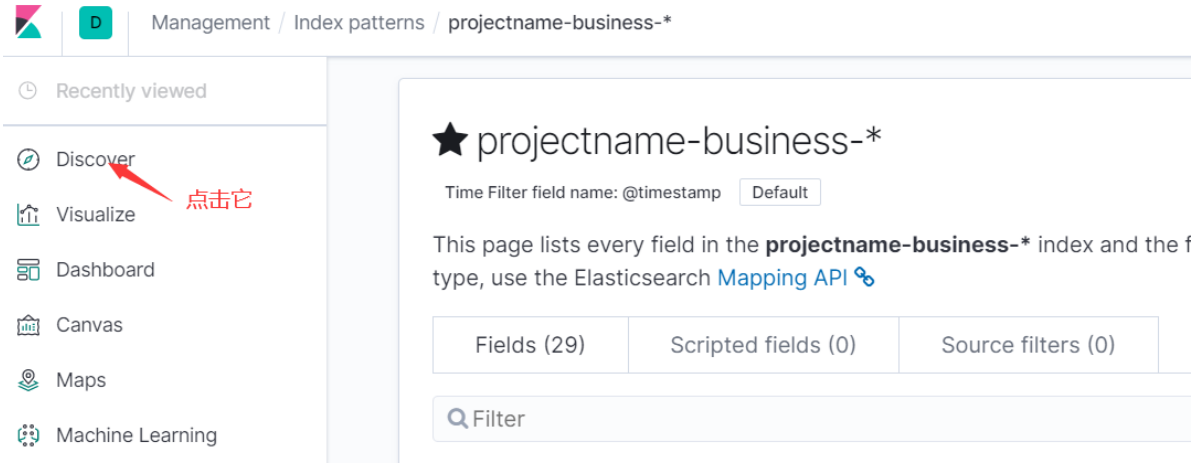


步骤2

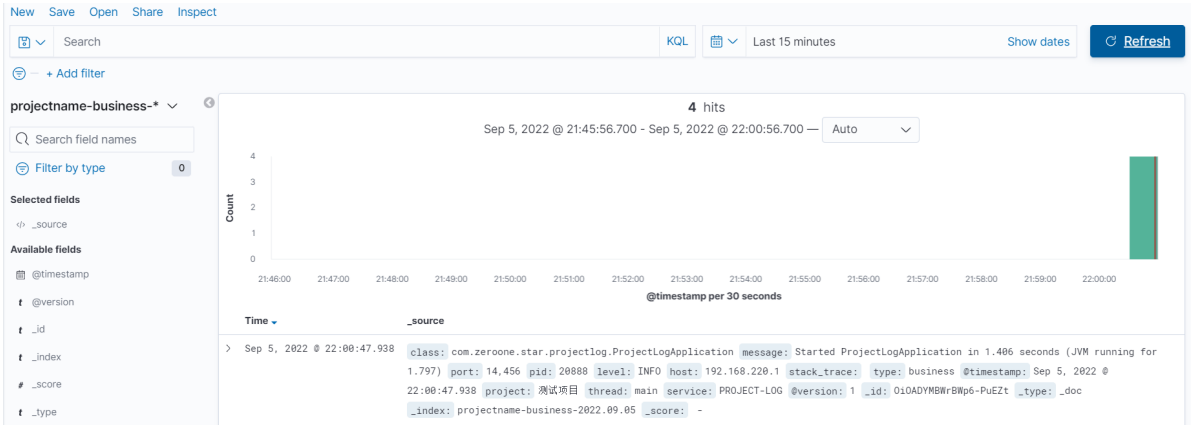


5.6.3 查看日志

进入发现面板

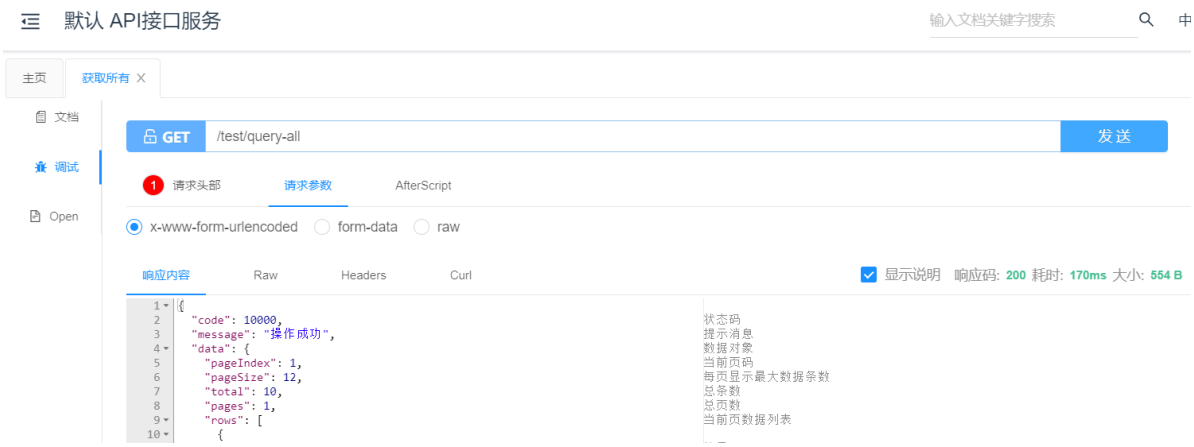


进入面板后可以看到已经有一些日志了，如下图所示



然后再API页面访问接口，查看日志生成情况

测试访问

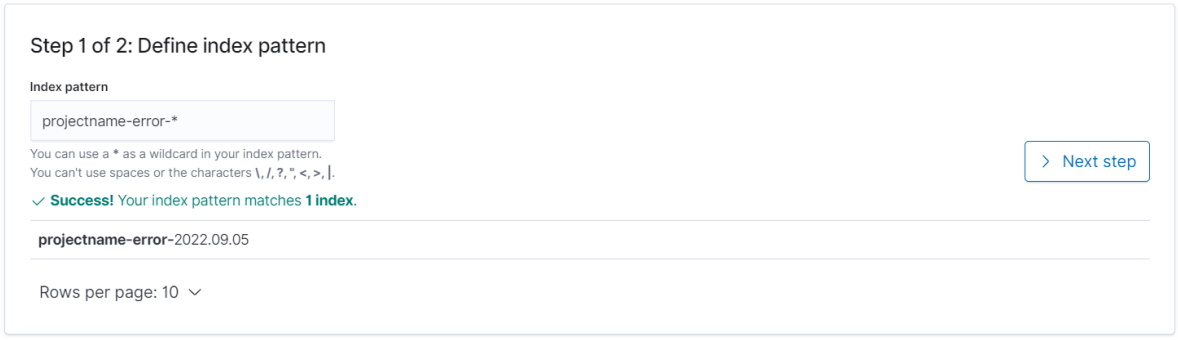


查看测试访问记录日志，示例如下图所示

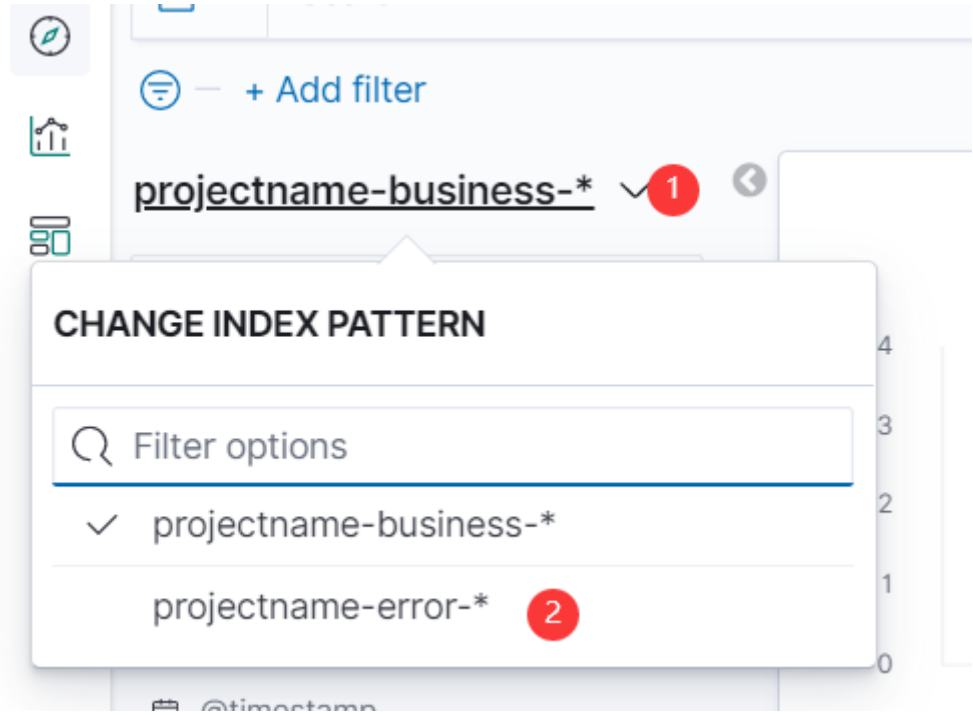
测试异常

```java
@ApiOperation("获取所有")
@GetMapping("query-all")
public JsonVO<PageVO<SampleVO>> queryAll() {
    //测试异常
    int b = 1 / 0;
    List<SampleVO> voList = new ArrayList<>( initialCapacity: 10);
    for (int i = 0; i < 10; i++) {
        SampleVO sampleVO = new SampleVO();
        sampleVO.setId(i + 1);
        sampleVO.setAge(11);
        sampleVO.setName("李四-" + (i + 1));
        sampleVO.setSex("男");
        voList.add(sampleVO);
    }
```

重启Spring Boot服务，然后访问一下query-all接口，然后在`Kibana`中创建索引模式。



在发现面板中查看错误日志



异常详情

Sep 5, 2022 @ 22:05:2: class: org.apache.catalina.core.ContainerBase.[Tomcat].[localhost].[/].[dispatcherServlet] message: Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request processing failed; nested exception is java.lang.ArithmeticException: / by zero] with root cause port: 1,225 pid: 17840 level: ERROR host: 192.168.220.1 stack_trace: java.lang.ArithmeticException: / by zero at com.zeroone.star.projectlog.controller.TestController.queryAll(TestController.java:30) at

Expanded document View surrounding documents View single document

Table JSON


```
{
  "_index": "projectname-error-2022.09.05",
  "_type": "_doc",
  "_id": "WiOFDYMBWrBWp6-PRkaC",
  "_version": 1,
  "_score": null,
  "_source": {
    "class": "org.apache.catalina.core.ContainerBase.[Tomcat].[localhost].[/].[dispatcherServlet]",
    "message": "Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request processing failed; nested exception is java.lang.ArithmeticException: / by zero] with root cause",
    "port": 1225,
    "pid": "17840",
    "level": "ERROR",
    "host": "192.168.220.1",
```


除0异常

t stack_trace

```
java.lang.ArithmeticException: / by zero
        at com.zeroone.star.projectlog.controller.TestController.queryAll(TestController.java:30)
        at com.zeroone.star.projectlog.controller.TestController$$FastClassBySpringCGLIB$$def39c.invoke(<generated>)
        at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218)
        at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:779)
        at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
        at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:750)
        at org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint.proceed(MethodInvocationProceedingJoinPoin
t.java:88)
        at com.zeroone.star.projectlog.component.WebLogAspect.doAround(WebLogAspect.java:54)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethodWithGivenArgs(AbstractAspectJAdvice.jav
a:644)
        at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethod(AbstractAspectJAdvice.java:633)
        at org.springframework.aop.aspectj.AspectJAroundAdvice.invoke(AspectJAroundAdvice.java:70)
        at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:186)
        at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:750)
        at org.springframework.aop.interceptor.ExposeInvocationInterceptor.invoke(ExposeInvocationInterceptor.java:95)
        at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:186)
        at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:750)
```