搜云库

SpringBoot中获取请求的Json格式并解决request的请求流只能读取一次的问题

2018/12/21 request.getParameterMap

公司有个小需求，需要从Spring拦截器中获取请求参数，用于记录用户的访问统计，把数据发到Kafka，例如：浏览器名称，浏览器版本，操作系统名称，操作系统版本，请求参数， 请求来源地址，等等，做的过程中发现一个问题就是GET 请求用 `request.getParameterMap()` 获取请求参数是可以的,但是PSOT 获取请求参数就是获取不到。

代码如下:

就是获取的请求参数json对象。

```
Map<String, String[]> paramMap = request.getParameterMap();
```

问题原因

在网上查找资料后发现，基本就是这样的原因

如果是POST请求 `contentType` 设置为 `application/x-www-form-urlencoded` 是可以通过request.getInputStream()来读取请求参数

如果上述条件没有满足，则相关的表单数据不会被设置进request的parameter集合中，相关的数据可以通过request.getInputStream()来访问获取。

但是如果在拦截器中，通过request.getInputStream()读取流之后Controller 中就获取不到了。在网上查找资料后发现，因为request的输入流只能读取一次，那么这是为什么呢?

下面是答案:

那是因为流对应的是数据，数据放在内存中，有的是部分放在内存中。read 一次标记一次当前位置（mark position），第二次read就从标记位置继续读（从内存中copy）数据。 所以这就是为什么读了一次第二次是空了。 怎么让它不为空呢？只要inputstream 中的pos 变成0就可以重写读取当前内存中的数据。javaAPI中有一个方法public void reset() 这个方法就是可以重置pos为起始位置，但是不是所有的IO读取流都可以调用该方法！ServletInputStream是不能调用reset方法，这就导致了只能调用一次getInputStream()。

摘自：https://blog.csdn.net/sdut406/article/details/81369983

解决办法

这种方法就是通过重写HttpServletRequestWrapper把request的保存下来，然后通过过滤器保存下来的request在填充进去，这样就可以多次读取request了

步骤一

** 继承HttpServletRequestWrapper类，重写该类方法**

```
/**
 * Request请求参数获取处理类
 */
public class BodyReaderHttpServletRequestWrapper extends HttpServletRequestWrapper {
    private final byte[] body;

    public BodyReaderHttpServletRequestWrapper(HttpServletRequest request) throws IO
        super(request);
        String sessionStream = getBodyString(request);
        body = sessionStream.getBytes(Charset.forName("UTF-8"));
    }

    /**
     * 获取请求Body
     *
     * @param request
     * @return
     */
    public String getBodyString(final ServletRequest request) {
        StringBuilder sb = new StringBuilder();
        InputStream inputStream = null;
        BufferedReader reader = null;
        try {
            inputStream = cloneInputStream(request.getInputStream());
            reader = new BufferedReader(new InputStreamReader(inputStream, Charset.f
            String line = "";
            while ((line = reader.readLine()) != null) {
                sb.append(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (inputStream != null) {
                try {
                    inputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (reader != null) {
```

```
                try {
                    reader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return sb.toString();
    }

    /**
     * Description: 复制输入流</br>
     *
     * @param inputStream
     * @return</br>
     */
    public InputStream cloneInputStream(ServletInputStream inputStream) {
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        int len;
        try {
            while ((len = inputStream.read(buffer)) > -1) {
                byteArrayOutputStream.write(buffer, 0, len);
            }
            byteArrayOutputStream.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
        InputStream byteArrayInputStream = new ByteArrayInputStream(byteArrayOutputS
        return byteArrayInputStream;
    }

    @Override
    public BufferedReader getReader() throws IOException {
        return new BufferedReader(new InputStreamReader(getInputStream()));
    }

    @Override
    public ServletInputStream getInputStream() throws IOException {

        final ByteArrayInputStream bais = new ByteArrayInputStream(body);

        return new ServletInputStream() {

            @Override
            public int read() throws IOException {
                return bais.read();
            }

            @Override
            public boolean isFinished() {
```

```
                return false;
            }

            @Override
            public boolean isReady() {
                return false;
            }

            @Override
            public void setReadListener(ReadListener readListener) {
            }
        };
    }
}
```

步骤二

使用过滤器Filter，用来把request传递下去

```
import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;

/**
 * 过滤器Filter，用来把request传递下去
 */
@WebFilter(urlPatterns = "/*",filterName = "channelFilter")
public class ChannelFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("--------------过滤器初始化------------");
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletRespo
        System.out.println("--------------执行过滤操作------------");

        // 防止流读取一次后就没有了，所以需要将流继续写出去
        HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;
        ServletRequest requestWrapper = new   BodyReaderHttpServletRequestWrapper(ht

        filterChain.doFilter(requestWrapper, servletResponse);
    }

    @Override
    public void destroy() {
        System.out.println("--------------过滤器销毁------------");
```

```
    }

}
```

步骤三

启动类加上注册过滤器注解

```
@SpringBootApplication
@ServletComponentScan(value = {"com.xxx.xxx"}) //注册过滤器注解
@ComponentScan(value = {"com.xxx.xxx"})
public class Startup {

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

步骤四

**拦截器中使用getBodyString() 获取请求参数 **

```
@Configuration
@Component(value = "requestHandlerInterceptor")
public class RequestHandlerInterceptor extends HandlerInterceptorAdapter implements

        // 省略 preHandle, postHandle, afterPropertiesSet 方法

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse resp

        JSONObject parameterMap = JSON.parseObject(new BodyReaderHttpServletRequestW
    }

}
```

最后完美解决

参考：https://blog.csdn.net/qq_33206732/article/details/78421793

往期精彩

- **100篇：搜云库技术团队，整理了一年的技术干货**
- 百度、腾讯、阿里、谷歌 面试题视频详解合集
- 大型分布式系统中的缓存架构

- 美团面试经历，贡献出来一起学习
- 干货：MySQL索引与优化实践
- 微服务架构：搭建网站扫码登录的功能设计
- 技术变化那么快，学 Docker 看这篇就够了
- 一文看懂 MySQL 高性能优化技巧实践
- 分布式事务不理解？一次给你讲清楚
- 动画+原理+代码+优化，解读十大经典排序算法

Show Disqus Comments

Like

Issue Page

Error: Comments Not Initialized

Write Preview Login with GitHub

Leave a comment

Styling with Markdown is supported

Comment

Powered by Gitment

Search

Search

关注 - 微信公众号

TOP