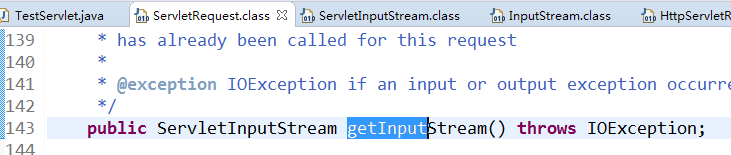
在流中有都有继承或重写父类InputStream的mark(int)方法，reset()方法，以及markSupported()方法，其中，markSupported()能够返回boolean值表示此流是否能够被标记和重置，即是否能调用mark(int)方法和reset()方法，mark(int)能够标识出一个位置，使得调用reset()方法时能够使指针回到标识位,标识位默认为0，即在流的开头。

我们在读取一个流的时候，标识位会随着我们的读取而增大直到流读取完毕，即指针会随着数据的读取完毕而指向流的末尾，对于markSupported()返回true的流，我们只需要调用reset()方法即可把指针重置到标识位，然后重复读取流中数据，但是对于markSupported()返回false的流，则无法重复读取，因为指针已经读取到流的末尾。InputStream基类中的markSupported()默认返回false,标识位默认为0,即不可重复读取。

从源码中我们可以看到，HttpServletRequest接口继承了servletRequest接口，而servletRequest接口的getInputStream()获取到的流是ServletInputStream;

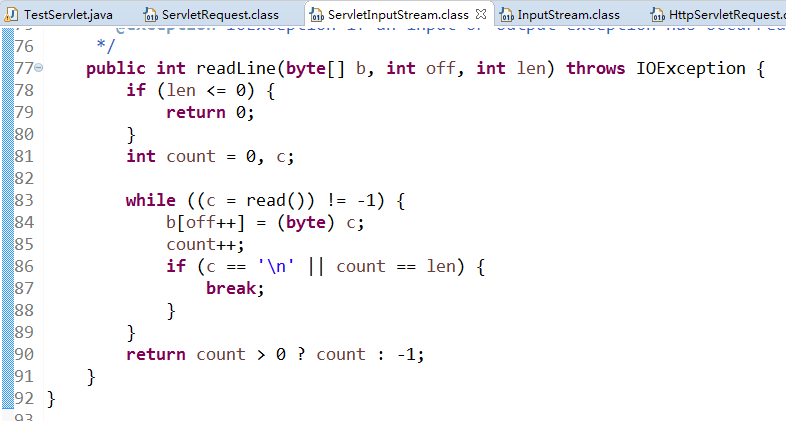




调用HttpServletRequest的getInputStream方法获取到的流就是调用的继承自父类的方法getInputStream()获取到的servletInputStream流。

然而ServletInputStream继承了抽象类InputStream流基类，并且只定义了readline方法，其他的方法和属性全部继承自父类InputStream，因此ServletInputStream调用markSupported()后返回false，即不支持标记与重置，所以request.getInputStream获取到的流不可重复读取：





那么我们怎么来解决这个问题呢？

方法一：

通过使用HttpServletRequestWrapper对HttpServletRequest进行包装:

1. 自定义类RequestMyWrapper继承HttpServletRequestWrapper,在自定类的构造器中调用父类的构造器，传入HttpServletRequest请求，在其父类HttpServletRequest的构造器请求中又调用了ServletRequest的构造器将请求传入，最终传递给一个ServletRequest，因此从这里看出自定义类RequestMyWrapper中的请求，除却重写父类的部分，其他的部分还是原请求。

|  |
| --- |
| package com.servlet;  import java.io.BufferedReader;  import java.io.ByteArrayInputStream;  import java.io.IOException;  import java.io.InputStream;  import java.io.InputStreamReader;  import javax.servlet.ServletInputStream;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletRequestWrapper;  public class RequestMyWrapper extends HttpServletRequestWrapper  {  CacheInputStream CacheInputStream;    private final byte[] body;    public RequestMyWrapper(HttpServletRequest request) throws IOException  {  super(request);  body = getBody(request).getBytes();  }    @Override  public ServletInputStream getInputStream() throws IOException  {  return new CacheInputStream(new ByteArrayInputStream(body));  }    private String getBody(HttpServletRequest request)  {  ServletInputStream inputStream = null;  BufferedReader bufferedReader = null;  StringBuffer data = new StringBuffer("");  try  {  inputStream = request.getInputStream();  bufferedReader = new BufferedReader(new InputStreamReader(  inputStream));  String str = "";  while ( (str = bufferedReader.readLine()) != null )  {  data.append(str);  }  }  catch (IOException e)  {  e.printStackTrace();  }  finally  {  try  {  bufferedReader.close();  }  catch (IOException e)  {  e.printStackTrace();  }  }  return data.toString();  }    @Override  public BufferedReader getReader() throws IOException  {  return new BufferedReader(new InputStreamReader(  new ByteArrayInputStream(body)));  }    private class CacheInputStream extends ServletInputStream  {  InputStream inputStream = null;    public CacheInputStream(ByteArrayInputStream inputStream)  {  this.inputStream = inputStream;  }    @Override  public boolean markSupported()  {  return true;  }    @Override  public int read() throws IOException  {  return inputStream.read();  }  }  } |

1. 在过滤器中重写doFilter方法，对请求进行判断，如果是http请求则new一个自定义的RequestMyWrapper类，并传递给doFilter方法（在这里要注意，如果是soap请求，也会被http的过滤器拦截，因为soap请求是http+xml形式的，最终还是http请求，这时候，自定义了RequestMyWrapper类后可能无法获取messageContext中的数据）

|  |
| --- |
| package com.servlet;  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import javax.servlet.Filter;  import javax.servlet.FilterChain;  import javax.servlet.FilterConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.http.HttpServletRequest;  public class ServletFilter implements Filter  {  public ServletFilter()  {  }    public void destroy()  {  }    public void doFilter(ServletRequest request, ServletResponse response,  FilterChain chain) throws IOException, ServletException  {  HttpServletRequest httpServletRequest = (HttpServletRequest) request;  String method = httpServletRequest.getMethod();  RequestMyWrapper requestMyWrapper = null;  if ("POST".equals(method))  {  requestMyWrapper = new RequestMyWrapper(httpServletRequest);  BufferedReader reader = new BufferedReader(new InputStreamReader(  requestMyWrapper.getInputStream()));;  StringBuffer sb = new StringBuffer("");  String line = "";  while ( (line = reader.readLine()) != null )  {  sb.append(line);  }  }  System.out.println("过滤器");  chain.doFilter(null == requestMyWrapper ? request : requestMyWrapper,  response);  }    public void init(FilterConfig filterConfig) throws ServletException  {  }  } |

1. 我们可以在以上过滤器中获取流中的数据，也可以在业务阶段获取流中数据，但是有一点要注意，在获取数据时应将getInputStream获取出来后再放到循环的条件中，不要将request.getInputStream直接放在循环条件中，这样每次都会获取到一个新的流对象，每次都只是读取了第一个字节。

1.配置过滤器时没有配置servletname属性，或者配置错误成了类名

现象：在启动web应用时显示cann't cast to ServletFilter

原因：在配置过滤器时，<filter-class>节点配置的过滤器继承的是ServletFilter,但是在转换的时候发现不能将过滤器转换成ServletFilter过滤器，可能是由于servlet-api.jar,jsp-api.jar使用maven打包，在打包的时候不需要这些jar包，由容器提供，应设置为provided的。

解决办法：设置servlet-api.jar,jsp-api.jar的scope属性为provided

2.对于一个servlet过滤器，如果过滤器的servletName指向的servletname不正确，则不能起到过滤作用

3.在编写java代码的过程中没有提示，可以来设置：window-->preference-->java-->Editor-->content assist-->Advanced-->java proposals