**Step 1: Create the Interceptor Class**

First, create a class that implements the javax.xml.ws.handler.soap.SOAPHandler interface. This class will intercept the SOAP messages and check the HTTP method.

**Java**

import javax.xml.ws.handler.soap.SOAPHandler;

import javax.xml.ws.handler.soap.SOAPMessageContext;

import javax.xml.ws.handler.MessageContext;

import javax.xml.namespace.QName;

import java.util.Set;

public class HTTPMethodInterceptor implements SOAPHandler<SOAPMessageContext> {

@Override

public boolean handleMessage(SOAPMessageContext context) {

Boolean outboundProperty = (Boolean) context.get(MessageContext.MESSAGE\_OUTBOUND\_PROPERTY);

if (!outboundProperty) {

String httpMethod = (String) context.get(MessageContext.HTTP\_REQUEST\_METHOD);

if ("OPTIONS".equals(httpMethod) || "HEAD".equals(httpMethod) ||

"CONNECT".equals(httpMethod) || "PATCH".equals(httpMethod)) {

throw new RuntimeException("HTTP method " + httpMethod + " is not allowed.");

}

}

return true;

}

@Override

public boolean handleFault(SOAPMessageContext context) {

return true;

}

@Override

public void close(MessageContext context) {}

@Override

public Set<QName> getHeaders() {

return null;

}

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**Step 2: Register the Interceptor**

Next, register the interceptor in your JAX-WS endpoint configuration. This can be done in the web.xml or directly in the service implementation class.

**Option 1: Register in**web.xml

**XML**

<web-app>

<servlet>

<servlet-name>MyService</servlet-name>

<servlet-class>com.example.MyService</servlet-class>

<init-param>

<param-name>jaxws.handler.chain</param-name>

<param-value>com.example.HTTPMethodInterceptor</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>MyService</servlet-name>

<url-pattern>/MyService</url-pattern>

</servlet-mapping>

</web-app>

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**Option 2: Register in Service Implementation**

**Java**

import javax.jws.WebService;

import javax.xml.ws.handler.Handler;

import javax.xml.ws.handler.HandlerResolver;

import javax.xml.ws.handler.PortInfo;

import javax.xml.ws.Endpoint;

import java.util.List;

import java.util.ArrayList;

@WebService

public class MyService {

public MyService() {

Endpoint endpoint = Endpoint.create(this);

endpoint.setHandlerResolver(new HandlerResolver() {

@Override

public List<Handler> getHandlerChain(PortInfo portInfo) {

List<Handler> handlerChain = new ArrayList<>();

handlerChain.add(new HTTPMethodInterceptor());

return handlerChain;

}

});

endpoint.publish("/MyService");

}

// Your service methods here

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**Step 3: Package and Deploy**

1. **Package the Application**: Ensure your interceptor class and service implementation are included in the EAR file.
2. **Deploy to WebSphere**: Deploy the EAR file to your WebSphere server using the administrative console or command-line tools.

**Step 4: Test the Interceptor**

Deploy the application and test it by sending requests with different HTTP methods. The interceptor should block the disallowed methods and allow others.

This setup ensures that your JAX-WS endpoints in the WebSphere server will reject requests with the specified HTTP methods. If you have any questions or need further assistance, feel free to ask!