# Introduction

By the end of this section, we will be able to manipulate the **JAX-WS** handlers (what, why, when). As we create service oriented applications and develop several web services endpoints and web services clients.

We need to address some crosscutting concerns or non-functional requirements, which have to be applied across both web services clients and web services endpoints. This requirement might not have anything to do with business logic, but they need us to manipulate the **SOAP** message (**SOAP** header, body) or a different requirement, which our application needs. That is where the handlers available in the **JAX-WS** standard come into picture.

These handlers are bunch of classes that we develop by implementing certain interfaces in the **JAX**-**WS** **API.**

Web services engines like Apache CXF will call into these handlers when

* The client request is sent.
* The request comes into the endpoint.
* The response goes back from the endpoint.
* The response comes back on to the client.

These handlers are very similar to **servlet filters**, if you are aware of servlet filters; except for these handlers can be applied on both the client side as well as the server side. Whatever logic, we write in the methods in the handlers will be called by web services stacks like **CXF.**

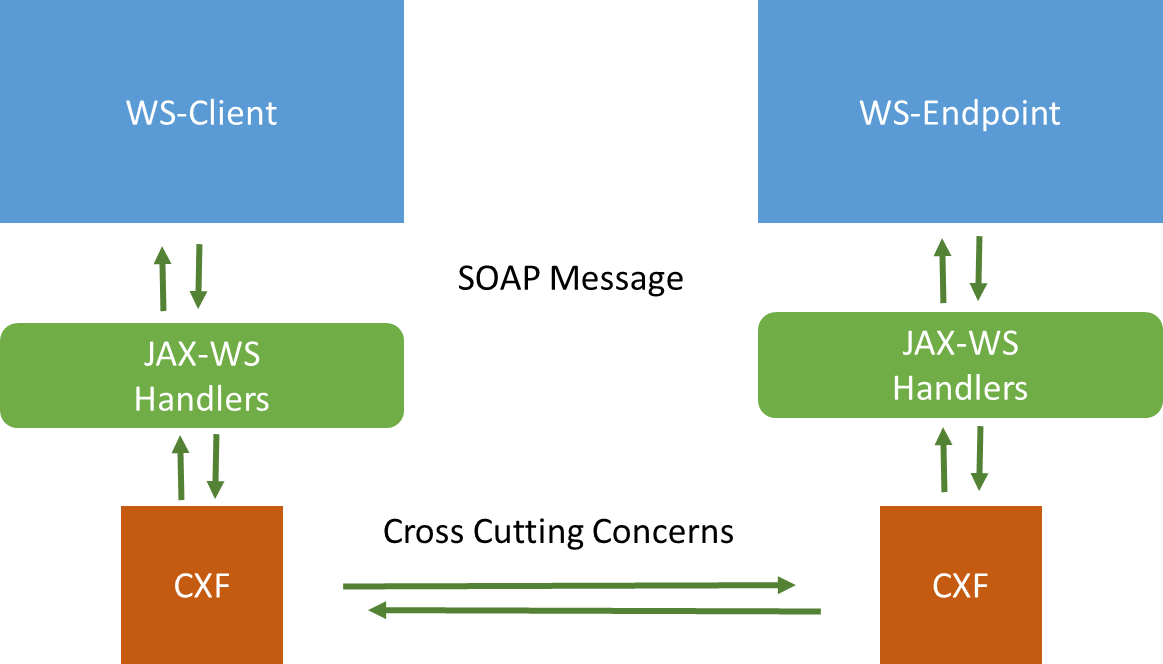
**Scenario:** handlers can be used to implement custom authentication mechanism

* We do not want to use username or password that included in **WS Standards** and including our SOAP headers).
* We can create handlers on both the provider and client side of our application.
* We can manipulate the **SOAP** headers inside the handlers.
* All the **SOAP** message information will be available to us when we create a handler.

**Caching:** we can create a handler that caches the responses and can check the request to see if the same request, which came earlier, and then it can be send back the response from the cache instead of calling the web services endpoint and executing the entire business logic and database operations again.

**Versioning:** to maintain the legacy application which had different versions of the same exact web service in the same exact application. When the client send a request, we have to decide which particular service will take care of the particular request based on the version that comes in the **SOAP** header. Therefore, our handler is going to dispatch to the exact endpoint.

Sometimes we do the client side and sometimes we them on both sides. So depending on our application requirements, handlers can be written and configured for either the providers or the clients.

****

# Types of JAX-WS handlers

There is two types of handlers:

1. **SOAP handlers**: have access to the entire **SOAP** message. They have access to the protocol information like HTTP headers, SOAP headers and then the entire **SOAP** body. We implement a **SOAP** **Handler** by implementing the **SOAPHandler** interface. Usually, we use the **SOAPMessageContext**, which wraps the entire **SOAP** information. It’s life cycles are:

* **handleMessage (called on both sides the client and provider ).**
* **handleFault (called when there is a soapFault).**
* **getHeaders (called on both sides the client and provider ).**
* **close (called at then of the entire flow).**

1. **Logical Handlers**: we implement them to access just the payload information. That is whatever goes in the **SOAP** body. We implement **Logical** **Handler** by implementing the logical interface. It’s life cycles are:

* **handleMessage**
* **handleFault**
* **close**

# Use case

# Steps

# Create the Handler Class

# Implement the handleMessage method

# Extract the Header

# Configure the handler

# SoapUI Test

# The Handler Flow

# getHeaders Explained