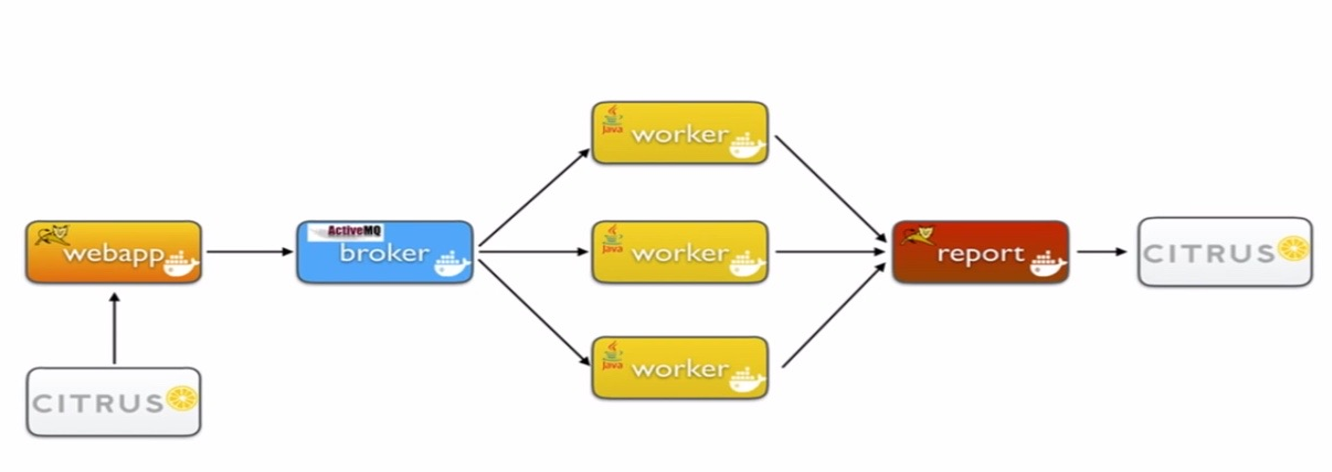
**Citrus Framework**

Допустим взаимодействую два сервиса : сервис А и сервис В. Сервис В может быть не доступен по разным причинам, поэтому возникает необходимость создания эмулятора (Mock, заглушка).

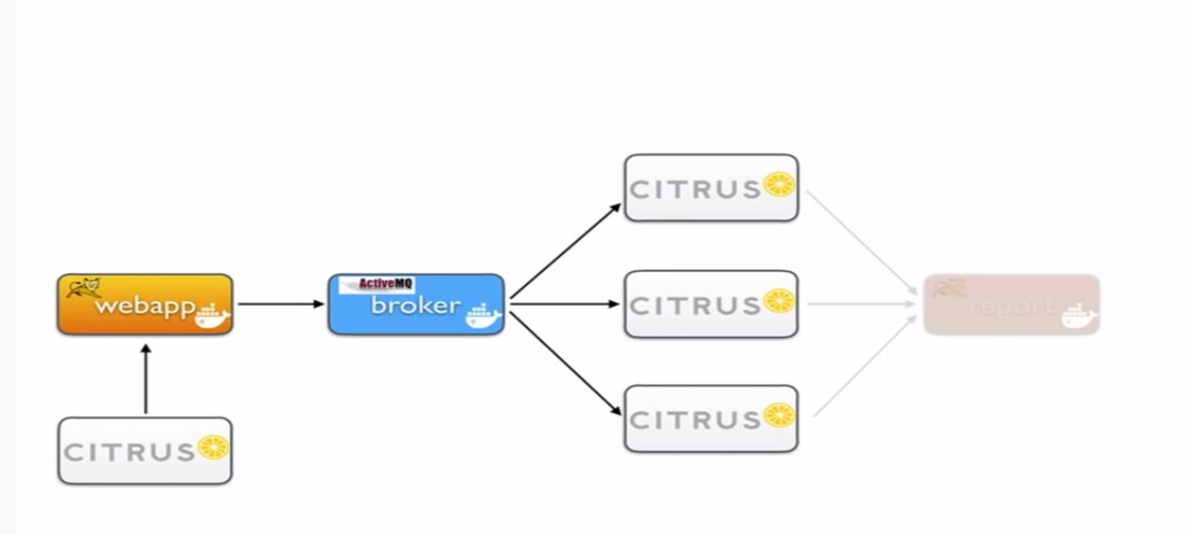


Пример использования заглушек для цепочки сервисов:

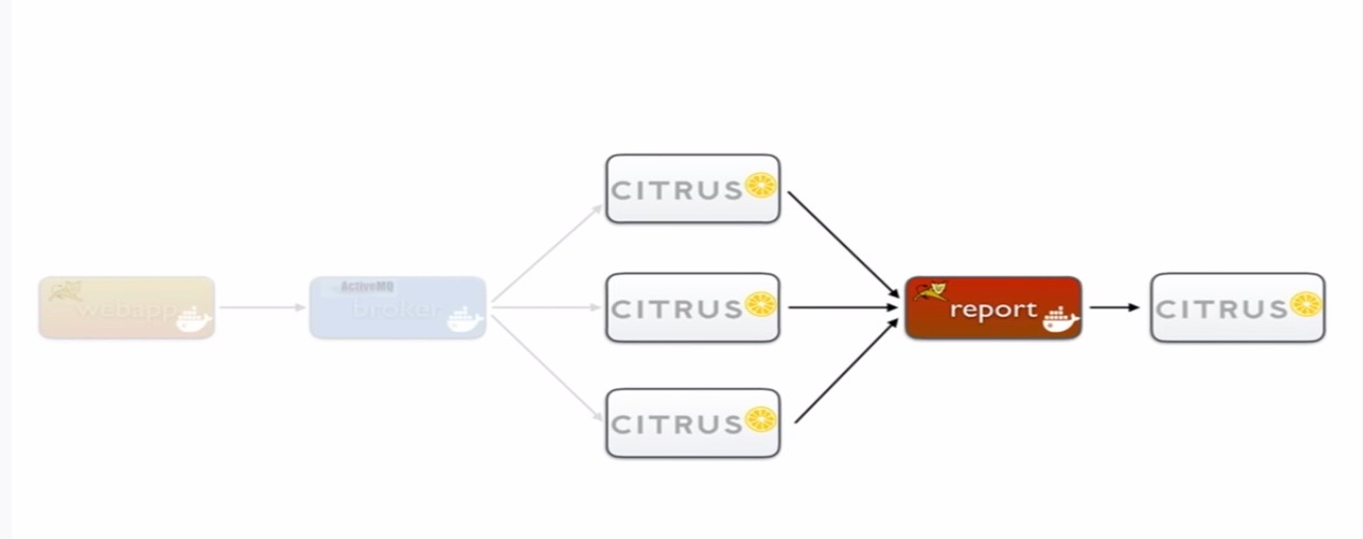


Flow end-to-end закрывается заглушками citrus.

Другой вариант заглушки: проверяем как брокер сообщений распределяет заявки по нашим worker.

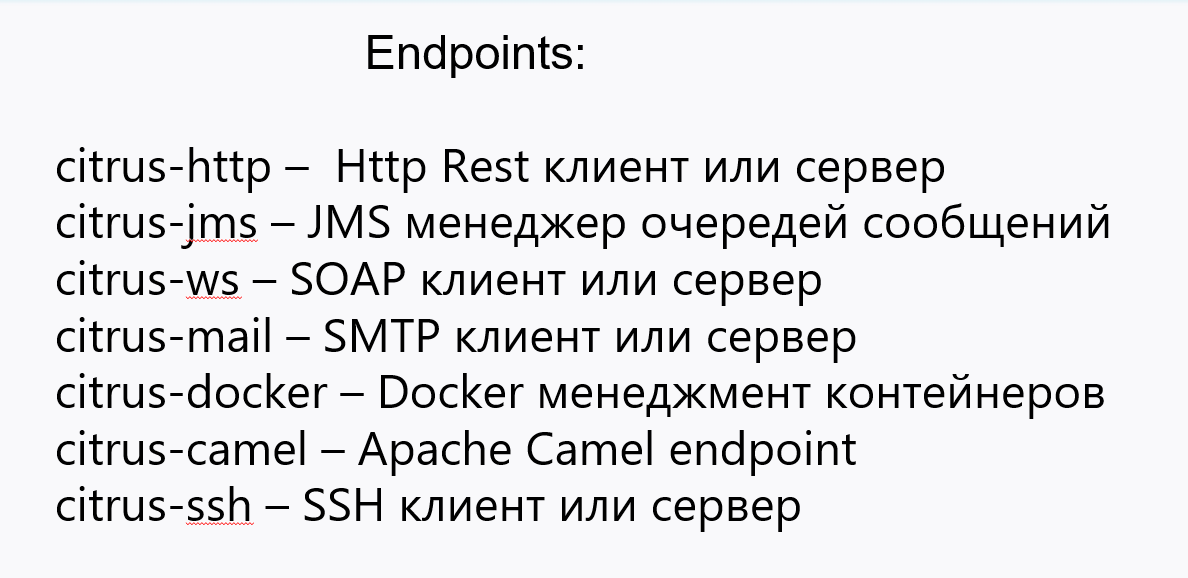


Еще вариант: атомарное тестирование сервиса отчета

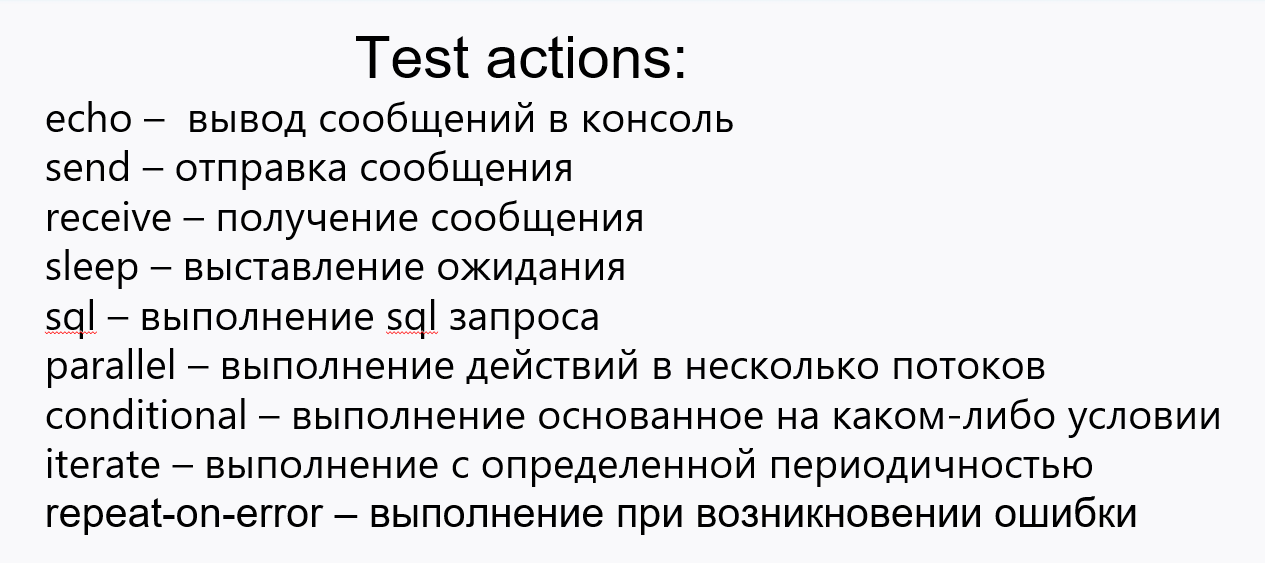


Фраемворк Citrus позиционирует себя для интеграционного тестирования, т.е. связь между микросервисами.

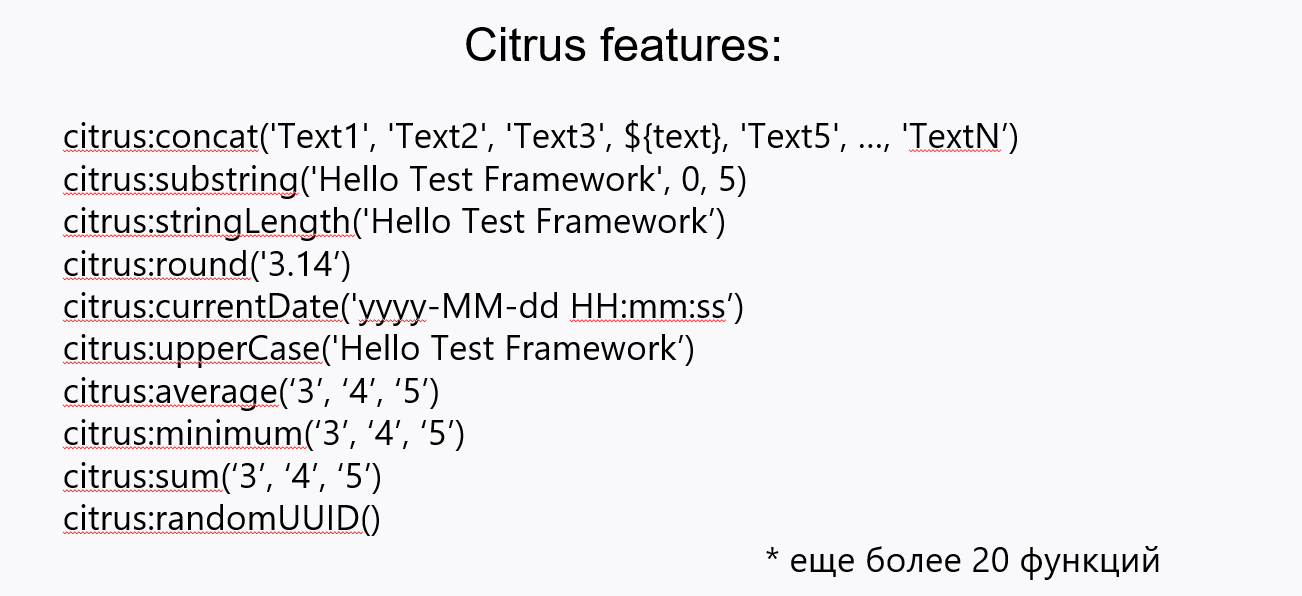
**Endpoints** (некие конечные точки, на схеме плитки с надписью citrus, они же mock, они же заглушки или симуляторы). Какие endpoints мы можем создавать:



**Действия** доступные в citrus framework:

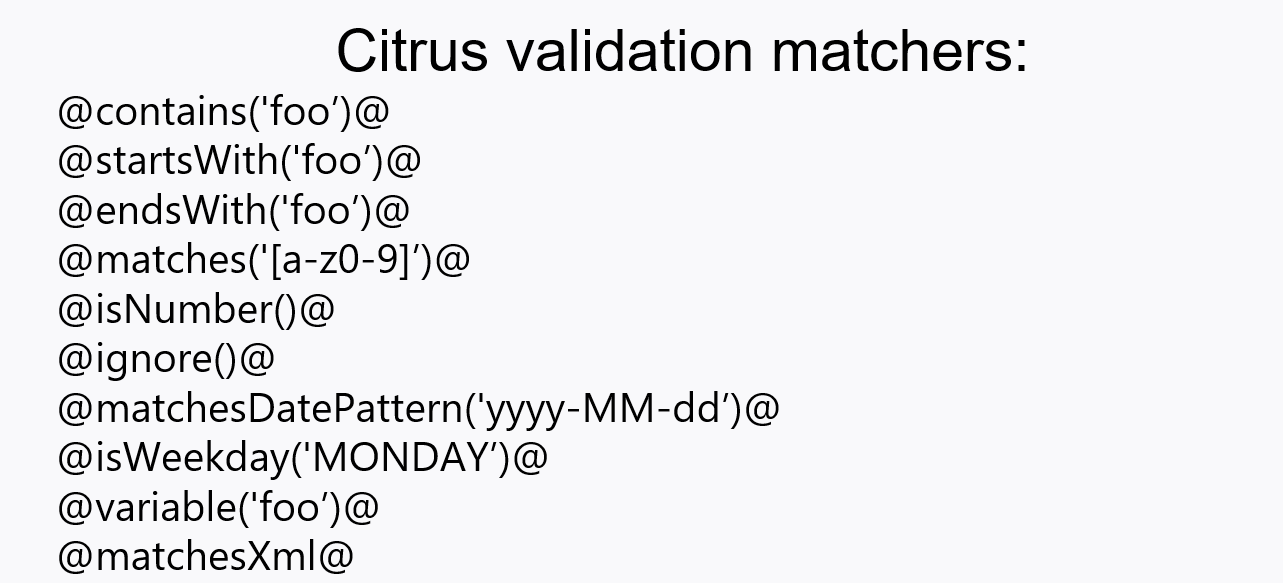


**Внутренние фичи** citrus framework:



<https://citrusframework.org/citrus/reference/3.2.1/html/index.html#actions>

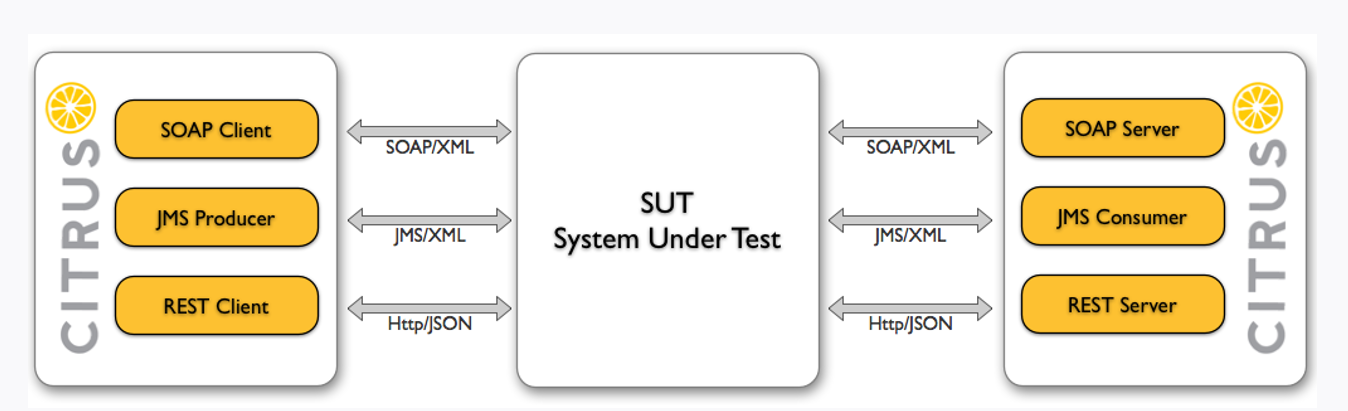
**Валидаторы** для проверок соответствия (в дополнении к assertions):



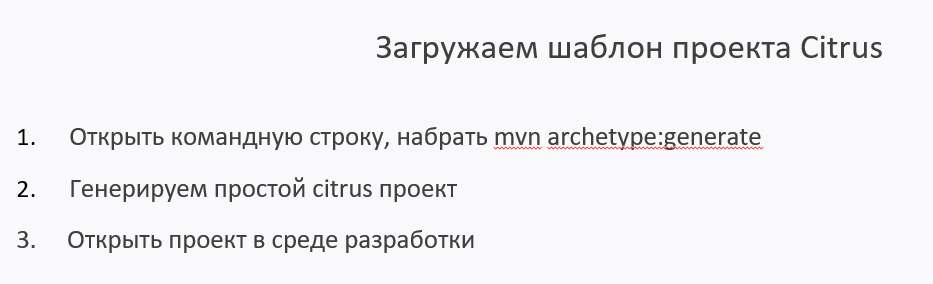
<https://citrusframework.org/citrus/reference/3.2.1/html/index.html#validation-matcher>

**SUT (System Under Test)** – система под тестированием.

Citrus framework позволяет осуществить любую комбинацию для входа/выхода SUT, например, получаем на вход данные по протоколу SOAP, а на выходе отдаем REST и т.п. Это встроенные библиотеки citrus.

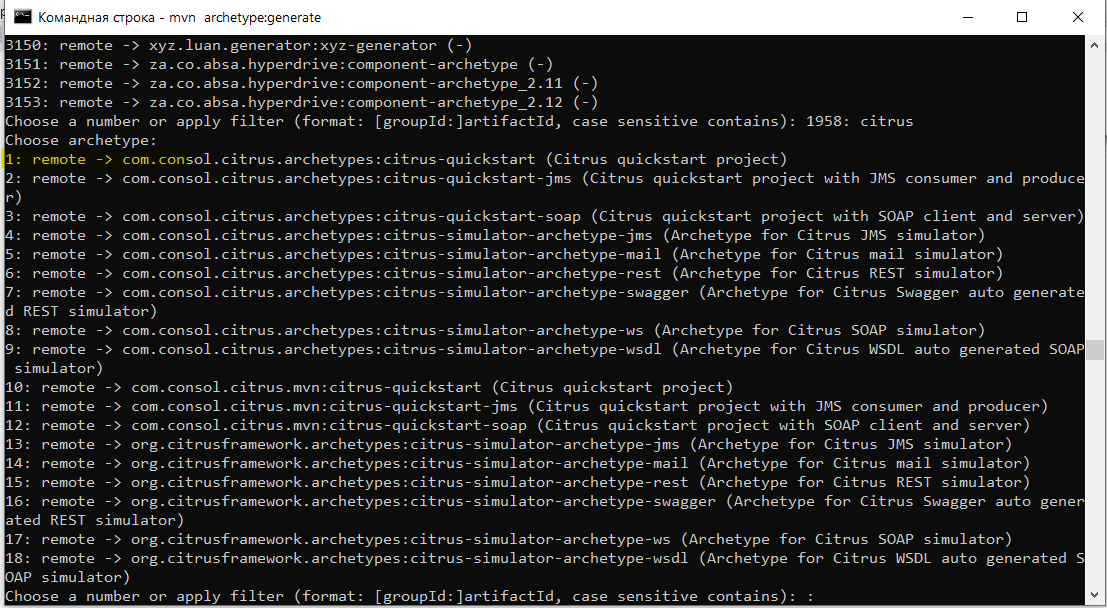


**Генерация проекта Citrus через архитипы Maven:**



Последняя стабильная версия Citrus Framework 3.2.1, предыдущая стабильная версия 2.8.0

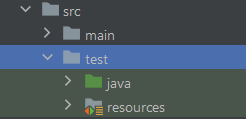
1. Заходим в директорию, в которой будем генерить проект.
2. Набираем команду mvn archetype:generate
3. Забиваем фильтр citrus. Получаем 18 вариантов. Выбираем 1



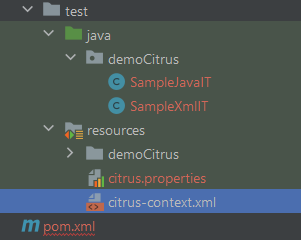
1. Выбираем версию 3.2.1
2. groupId = demoCitrus
3. artifactId = newLesson
4. Snapshot, package – пропускаем
5. Подтверждаем Y
6. Открываем IDEA сгенеренный проект

Под копотом citrus является Spring framework, но с возможностями для тестирования.

Src/main пользуются разработчики, когда разрабатывают сервисы. Поэтому мы работаем только в папке src/test



**Структура проекта:**

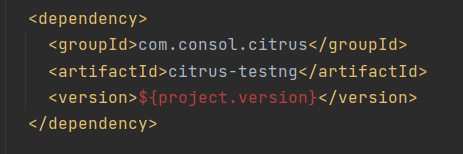


В папке demoCitrus – тесты на xml (они нам не нужны)

**Citrus.properties** – файл с настройками

**Citrus-context.xml** – у spring-проекта есть три варианта конфигурирования (через конфиг файл, через xml-файл и через аннотации). В данном случае мы имеем варирант конфигурирования через xml-файл.

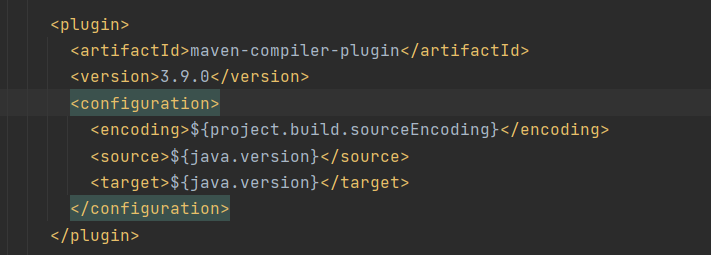
Правим pom.xml:

 меняем на ${citrus.version}

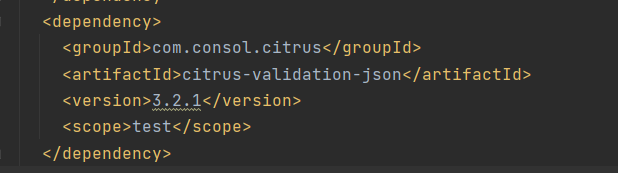
Это коммитем:



Версия 3.9.0



Добавляем зависимость для валидации json-сообщений (для soap понадобится citrus-validation-xml)



В итоге имеем в pom.xml

1. Citrus-core
2. Citrus-jms (java message service)
3. Citrus-ws (soap)
4. Citrus-http (для создания http-клиентов)
5. Citrus-testng
6. Citrus-validation-json (валидация json)
7. Логирование
8. Плагины, которые связывают citrus с maven

Учебный проект: <https://github.com/reviol/QA-JAVA/tree/main/lesson_Citrus_Beginner_v3.2.1>

Проект, который будем тестировать: <https://reqres.in/>

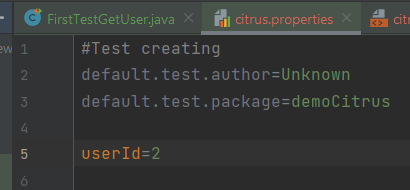
**Citrus-context.xml**

Т.к. у нас xml-конфигурация, то нужно подключение namespaces. Это аналог import только в xml-представлении.

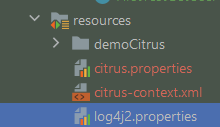
В файле также задается глобальные переменные в блоке. Добавим настройку path, чтобы глобально из всего проекта будут доступны наши переменные из файла citrus.properties

<citrus:global-variables>  
 <citrus:variable name="project.name" value="Citrus Integration Tests"/>  
 <citrus:file path="classpath:citrus.properties"/>   
</citrus:global-variables>

Например userId



Для вывода логов в папке resources необходим файл log4j2.properties (его нет в первочанальном проекте)



#  
# The logging properties used  
# Extra logging related to initialization of Log4j  
# Set to debug or trace if log4j initialization is failing  
status = warn  
# Name of the configuration  
name = Log4j2PropertiesConfig  
  
# Root logger level  
rootLogger.level = debug  
rootLogger.appenderRefs = stdout  
rootLogger.appenderRef.stdout.ref = STDOUT  
  
# Console appender configuration  
appender.console.type = Console  
appender.console.name = STDOUT  
appender.console.layout.type = PatternLayout  
appender.console.layout.pattern = %d{HH:mm:ss,SSS} %-5.5p %20.20c{2}| %m%n

В логах каждый test action (в нашем тесте каждое echo) citrus помечает как TEST STEP.

Конфигурируем клиента.

Настраиваем bean клиента в **Citrus-context.xml**

<citrus-http:client id="restClientReqres"  
 request-url="${url}"  
 content-type="application/json"  
 timeout="15000"/>

Подробней здесь: <https://citrusframework.org/samples/http/>

Для конвертации из pojo в json нужен ObjectMapper. Конфигурим bean на основе библиотеки jackson для него:

<bean id="objectMapper" class="com.fasterxml.jackson.databind.ObjectMapper">  
 <property name="serializationInclusion" value="NON\_NULL"/>  
</bean>

**ApplyBehavior**

Поведенческие классы, в котором будем эмулировать некие действия. Основная цель: избавить от дублирования кода.

Пример: создание пользователя, как предусловия к какому-то тесту.

**Soap**

Настроить в pom.xml

<dependency>  
 <groupId>com.consol.citrus</groupId>  
 <artifactId>citrus-ws</artifactId>  
 <version>${citrus.version}</version>  
</dependency>

<dependency>  
 <groupId>jakarta.xml.ws</groupId>  
 <artifactId>jakarta.xml.ws-api</artifactId>  
 <version>2.3.2</version>  
 <scope>test</scope>  
</dependency>

Используем сервис <http://www.dataaccess.com/webservicesserver/>

Для него wsdl



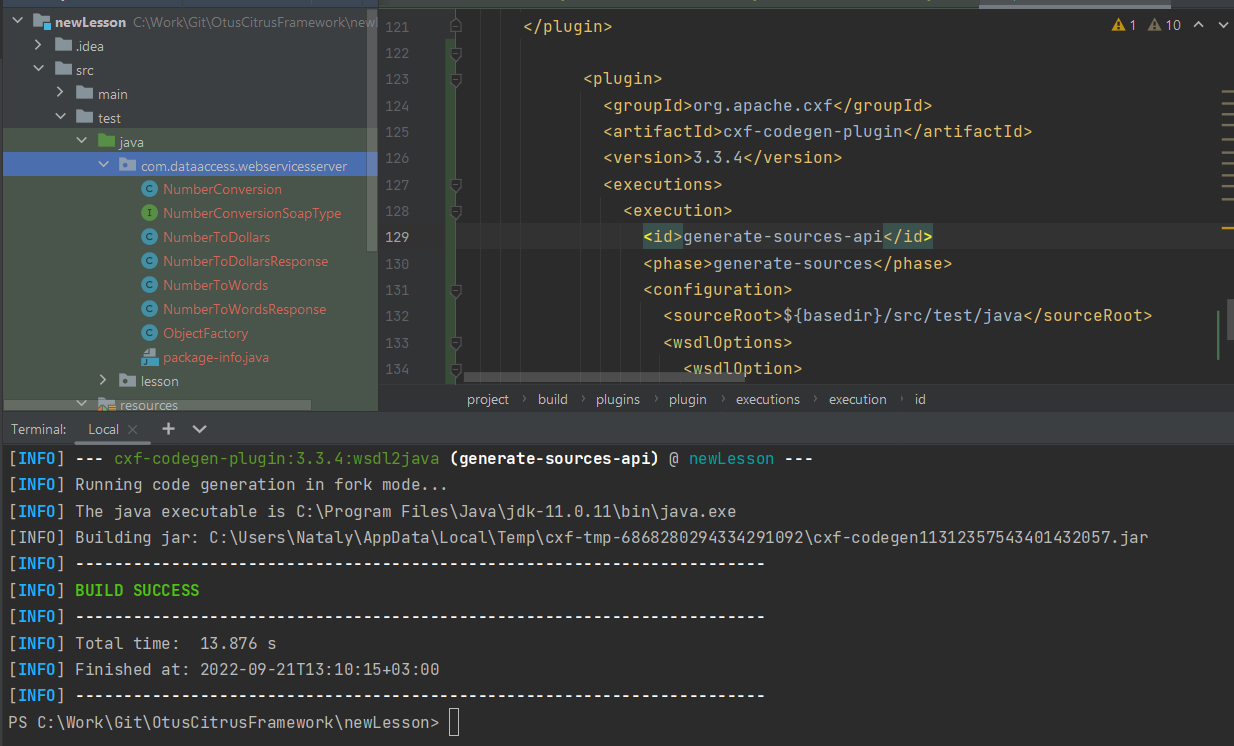
Для создание pojo прописываем в pom.xml плагин:

<plugin>  
 <groupId>org.apache.cxf</groupId>  
 <artifactId>cxf-codegen-plugin</artifactId>  
 <version>3.3.4</version>  
 <executions>  
 <execution>  
 <id>generate-sources-api</id>  
 <phase>generate-sources</phase>  
 <configuration>  
 <sourceRoot>${basedir}/src/test/java</sourceRoot>  
 <wsdlOptions>  
 <wsdlOption>  
 <wsdl>https://www.dataaccess.com/webservicesserver/numberconversion.wso?WSDL</wsdl>  
 </wsdlOption>  
 </wsdlOptions>  
 </configuration>  
 <goals>  
 <goal>wsdl2java</goal>  
 </goals>  
 </execution>  
 </executions>  
</plugin>

Набираем в teminal команду:

**mvn generate-source**

После BUILD SUCCESS видим, что **java** появилась новая папочка **com.dataaccess.webservicesserver** c **pojo :**



Для SOAP нужно настроить собственного SOAP-клиента

<citrus-ws:client id="soapClient"  
 request-url="${urlSOAP}"  
 timeout="15000"/>  
  
<!--For SOAP 1.1-->  
<bean id="messageFactory" class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory">  
 <property name="soapVersion">  
 <util:constant static-field="org.springframework.ws.soap.SoapVersion.SOAP\_11"/>  
 </property>  
</bean>  
  
<!--For SOAP 1.2-->  
<!-- <bean id="messageFactory" class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory">-->  
<!-- <property name="soapVersion">-->  
<!-- <util:constant static-field="org.springframework.ws.soap.SoapVersion.SOAP\_12"/>-->  
<!-- </property>-->  
<!-- </bean>-->  
  
<citrus:schema-repository id="schemaRepositoryService">  
 <citrus:schemas>  
 <citrus:schema id="DATAACCESS" location="classpath:wsdl/numberconversion.wsdl" />  
 </citrus:schemas>  
</citrus:schema-repository>

+ в секцию beans добавить:

xmlns:util="http://www.springframework.org/schema/util"

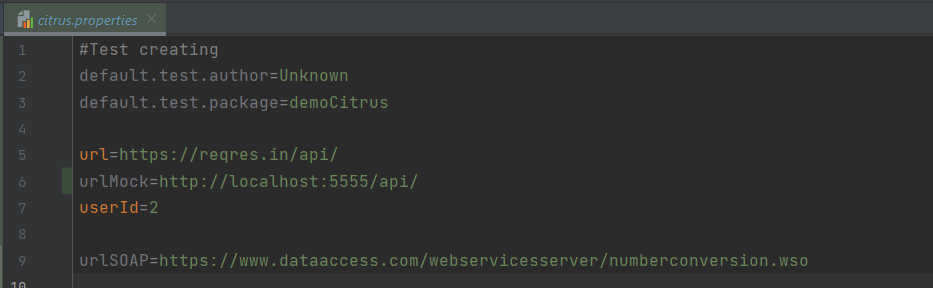
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd

**Mock**

Фраеморк Wiremock

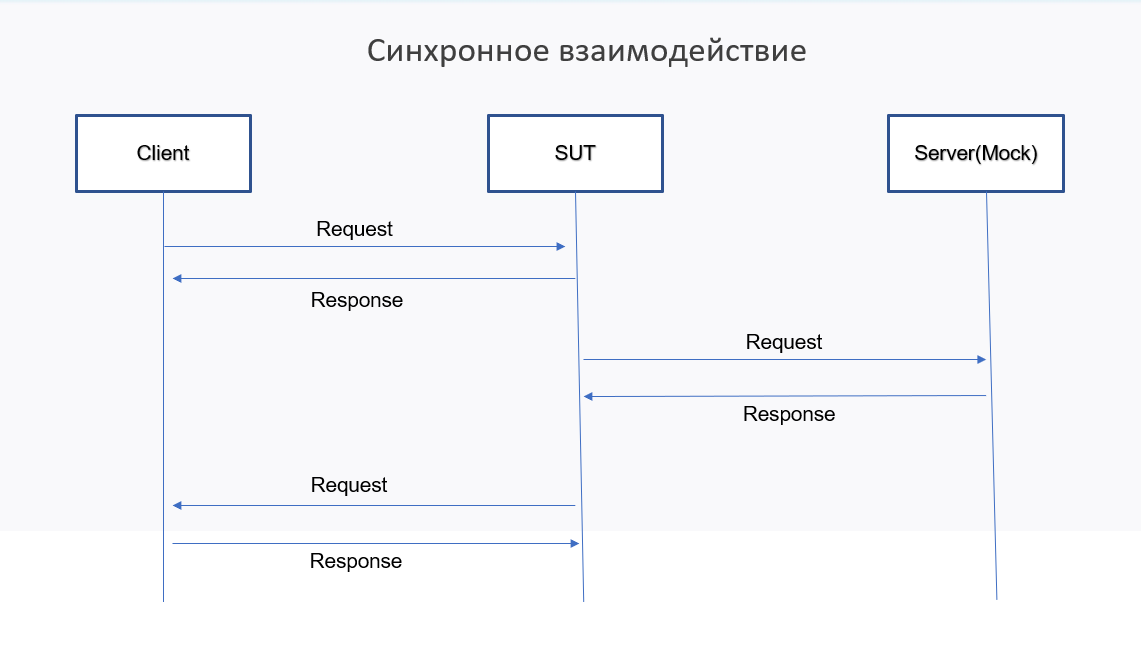
1. Создаем REST-клиента (мокируем reqres), который будет отправлять запросы на порт localhost:5555/api

<!--REST Client (Mock reqres)-->  
<citrus-http:client id="restClient"  
 request-url="${urlMock}"  
 content-type="application/json"  
 timeout="15000"/>



1. Конфигурируем сервер

<!--REST Server (Mock reqres)-->  
<citrus-http:server id="restServer"  
 port="5555"  
 auto-start="true"  
 timeout="15000"/>



fork

