

(//)

# Crear un servicio web SOAP con Spring



por baeldung (<https://www.baeldung.com/author/baeldung/>)  
(<https://www.baeldung.com/author/baeldung/>)

Bota de primavera (<https://www.baeldung.com/category/spring/spring-boot/>)

Acabo de anunciar el nuevo curso *Learn Spring*, centrado en los fundamentos de Spring 5 y Spring Boot 2:

>> VER EL CURSO (/ls-course-start)

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay



En este tutorial, veremos cómo crear un servicio web basado en SOAP con Spring Boot Starter Web Services.

## 2. Servicios web SOAP

En resumen, un servicio web es un servicio de plataforma a máquina independiente de la máquina que permite la comunicación a través de una red.



SOAP es un protocolo de mensajería. Los mensajes (solicitudes y respuestas) son documentos XML sobre HTTP . El contrato XML está definido por el WSDL (lenguaje de descripción de servicios web). Proporciona un conjunto de reglas para definir los mensajes, enlaces, operaciones y ubicación del servicio.

El XML utilizado en SOAP puede volverse extremadamente complejo. Por esta razón, es mejor usar SOAP con un marco como JAX-WS (<https://www.baeldung.com/jax-ws>) o Spring, como veremos en este tutorial.

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay



## 3. Estilo de desarrollo del primer contrato

Existen dos enfoques posibles al crear un servicio web: Contract-Last y Contract-First (<https://docs.spring.io/spring-ws/sites/1.5/reference/html/why-contract-first.html>) . Cuando usamos un enfoque de último contrato, comenzamos con el código Java y generamos el contrato de servicio web ( WSDL ) a partir de las clases. Cuando usamos contract-first, comenzamos con el contrato WSDL, desde el cual generamos las clases Java.



## 4. Configuración del proyecto Spring Boot

Vamos a crear un proyecto Spring Boot (<https://www.baeldung.com/spring-boot>) donde definiremos nuestro servidor SOAP WS.

### 4.1. Dependencias de Maven

Comencemos agregando *spring-boot-starter-parent* (<https://search.maven.org/search?q=g:org.springframework.boot%20a:spring-boot-starter-parent>) a nuestro proyecto:

```
1 <parent>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-parent</artifactId>
4
5 </parent>
```

Okay



A continuación, agreguemos las (<https://search.maven.org/search?q=g:wsdl4j%20%20a:wsdl4j>) dependencias `spring-boot-starter-web-services` (<https://search.maven.org/search?q=g:org.springframework.boot%20a:spring-boot-starter-web-services>) y `wsdl4j` (<https://search.maven.org/search?q=g:wsdl4j%20%20a:wsdl4j>):

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-web-services</artifactId>
4 </dependency>
```

## 4.2. El archivo XSD

El enfoque de primer contrato requiere que primero creamos el dominio (métodos y parámetros) para nuestro servicio. Vamos a utilizar un archivo de esquema XML (XSD) que Spring-WS exportará automáticamente como WSDL:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" (http://www.w3.org/2001/XMLSchema)" xmlns:tns="ht
3   targetNamespace="http://www.baeldung.com/springsoap/gen" (http://www.baeldung.com/springsoap/
4
5   <xs:element name="getCountryRequest">
6     <xs:complexType>
7       <xs:sequence>
8         <xs:element name="name" type="xs:string"/>
Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la Política de privacidad y cookies completa \(/privacy-policy\).
9       </xs:sequence>
10      </xs:complexType>                                Okay
11    </xs:element>
```

```
12<xs:element name="getCountryResponse">
13    <xs:complexType>
14        <xs:sequence>
15            <xs:element name="country" type="tns:country"/>
16        </xs:sequence>
17    </xs:complexType>
18</xs:element>
19</xs:element>
20
21<xs:complexType name="country">
22
23    <xs:sequence>
24        <xs:element name="name" type="xs:string"/>
25        <xs:element name="capital" type="xs:string"/>
26        <xs:element name="currency" type="tns:currency"/>
27    </xs:sequence>
28</xs:complexType>
29
30<xs:simpleType name="currency">
31    <xs:restriction base="xs:string">
32        <xs:enumeration value="GBP"/>
33        <xs:enumeration value="EUR"/>
34        <xs:enumeration value="PLN"/>
35    </xs:restriction>
36</xs:simpleType>
37</xs:schema>
```

En este archivo, vemos el formato de la solicitud del servicio web `getCountryRequest`. Lo definimos para aceptar un parámetro de la *cadena de tipo*.

A continuación, definimos el formato de la respuesta, que contiene un objeto del tipo *país*. Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Por fin, vemos el objeto de *moneda*, utilizado dentro del objeto de *país*.

Okay



## 4.3. Generar las clases Java de dominio

Ahora vamos a generar las clases Java a partir del archivo XSD definido en la sección anterior. El complemento `jaxb2-maven` (<https://search.maven.org/search?q=g:org.codehaus.mojo%20a:jaxb2-maven-plugin>) hará esto automáticamente durante el tiempo de compilación. El complemento utiliza la herramienta XJC como motor de generación de código. XJC compila el archivo de esquema XSD en clases Java totalmente anotadas.



Agreguemos y configuremos el complemento en nuestro pom.xml:

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay

```
1 <plugin>
2   <groupId>org.codehaus.mojo</groupId>
3   <artifactId>jaxb2-maven-plugin</artifactId>
4   <version>1.6</version>
5   <executions>
6     <execution>
7       <id>xjc</id>
8       <goals>
9         <goal>xjc</goal>
10      </goals>
11
12      <configuration>
13        <schemaDirectory>$\{project.basedir}/src/main/resources</schemaDirectory>
14        <outputDirectory>$\{project.basedir}/src/main/java</outputDirectory>
15      </configuration>
16    </execution>
17  </executions>
18 </plugin>
```

Aquí notamos dos configuraciones importantes:

- `<schemaDirectory> ${project.basedir} / src / main / resources </schemaDirectory>` - La ubicación del archivo XSD
- `<outputDirectory> ${project.basedir} / src / main / java </outputDirectory>` - Donde queremos que se genere nuestro código Java

Para generar las clases Java, simplemente podríamos usar la herramienta xjc de nuestra instalación Java. Aunque en nuestro proyecto Maven las cosas son aún más simples, ya que las clases se generarán automáticamente durante la construcción habitual de Maven:

1 mvn compile  
Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay



## 4.4. Agregar el punto final del servicio web SOAP

La clase de punto final del servicio web SOAP gestionará todas las solicitudes entrantes para el servicio. Iniciará el procesamiento y enviará la respuesta de regreso.

Antes de definir esto, creamos un repositorio de *país* para proporcionar datos al servicio web.

```
1 @Component
2 public class CountryRepository {
3
4     private static final Map<String, Country> countries = new HashMap<>();
5
6     @PostConstruct
7     public void initData() {
8         // initialize countries map
9     }
10
11    public Country findCountry(String name) {
12        return countries.get(name);
13    }
14 }
```

A continuación, configuremos el punto final:

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay



```
1 @Endpoint
2 public class CountryEndpoint {
3
4     private static final String NAMESPACE_URI = "http://www.baeldung.com/springsoap/gen (http://www.bae
5
6     private CountryRepository countryRepository;
7
8     @Autowired
9     public CountryEndpoint(CountryRepository countryRepository) {
10         this.countryRepository = countryRepository;
11     }
12
13     @PayloadRoot(namespace = NAMESPACE_URI, localPart = "getCountryRequest")
14     @ResponsePayload
15     public GetCountryResponse getCountry(@RequestPayload GetCountryRequest request) {
16         GetCountryResponse response = new GetCountryResponse();
17         response.setCountry(countryRepository.findCountry(request.getName()));
18
19         return response;
20     }
21 }
```

Aquí hay algunos detalles para notar:

- `@Endpoint`: registra la clase con Spring WS como punto final de servicio web
- `@PayloadRoot`: define el método del controlador de acuerdo con el *espacio de nombres* y los atributos *localPart*
- `@ResponsePayload`: indica que este método devuelve un valor que se asignará a la carga útil de respuesta
- `@RequestPayload`: indica que este método acepta un parámetro que se asignará desde la solicitud entrante

Okay



## 4.5. Los beans de configuración del servicio web SOAP

Ahora creemos una clase para configurar el servlet del despachador de mensajes Spring para recibir la solicitud:

```
1 @EnableWs  
2 @Configuration  
3 public class WebServiceConfig extends WsConfigurerAdapter {  
4     // bean definitions  
5 }
```

`@EnableWs` habilita las características del servicio web SOAP en esta aplicación Spring Boot. La clase `WebServiceConfig` amplía la clase base `WsConfigurerAdapter`, que configura el modelo de programación Spring-WS basado en anotaciones.

Creemos un `MessageDispatcherServlet` que se usa para manejar solicitudes SOAP:

```
1 @Bean  
2 public ServletRegistrationBean messageDispatcherServlet(ApplicationContext applicationContext) {  
3     MessageDispatcherServlet servlet = new MessageDispatcherServlet();  
4     servlet.setApplicationContext(applicationContext);  
5     servlet.setTransformWSDLLocations(true);  
6     return new ServletRegistrationBean(servlet, "/ws/*");  
7 }
```

Establecemos el objeto injectado `ApplicationContext` del `servlet` para que Spring-WS pueda encontrar otros Spring beans.

También habilitamos la transformación de servlet de ubicación WSDL. Esto transforma el atributo de ubicación Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay



## 5. Prueba del proyecto SOAP

Una vez que se haya completado la configuración del proyecto, estamos listos para probarlo.

### 5.1. Construye y ejecuta el proyecto

Sería posible crear un archivo WAR e implementarlo en un servidor de aplicaciones externo. En su lugar, utilizaremos Spring Boot, que es una forma más rápida y fácil de poner en funcionamiento la aplicación.

Primero, agregamos la siguiente clase para hacer que la aplicación sea ejecutable:

```
1 @SpringBootApplication
2 public class Application {
3     public static void main(String[] args) {
4         SpringApplication.run(Application.class, args);
5     }
6 }
```

Tenga en cuenta que no estamos utilizando ningún archivo XML (como web.xml) para crear esta aplicación. Todo es puro Java.

Ahora estamos listos para compilar y ejecutar la aplicación:

```
1 mvn spring-boot:run
```

Para verificar si la aplicación se está ejecutando correctamente, podemos abrir el WSDL a través de la URL:

<http://localhost:8080/ws/countrieswsdl>

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay



Finalmente, creamos un objeto `DefaultWsdl11Definition`. Esto expone un WSDL 1.1 estándar usando un `XsdSchema`. El nombre WSDL será el mismo que el nombre del bean.

```
1 @Bean(name = "countries")
2 public DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema countriesSchema) {
3     DefaultWsdl11Definition wsdl11Definition = new DefaultWsdl11Definition();
4     wsdl11Definition.setPortTypeName("CountriesPort");
5     wsdl11Definition.setLocationUri("/ws");
6     wsdl11Definition.setTargetNamespace("http://www.baeldung.com/springsoap/gen (http://www.baeldung.co
7     wsdl11Definition.setSchema(countriesSchema);
8     return wsdl11Definition;
9 }
10
11 @Bean
12 public XsdSchema countriesSchema() {
13     return new SimpleXsdSchema(new ClassPathResource("countries.xsd"));
14 }
```

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay



## 5.2. Probar una solicitud SOAP

Para probar una solicitud, creamos el siguiente archivo y lo llamamos request.xml:

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" (http://schemas.xmlsoap.org/s
2           xmlns:gs="http://www.baeldung.com/springsoap/gen" (http://www.baeldung.com/springsoap/g
3   <soapenv:Header/>
4   <soapenv:Body>
5     <gs:getCountryRequest>
6       <gs:name>Spain</gs:name>
7     </gs:getCountryRequest>
8   </soapenv:Body>
9 </soapenv:Envelope>
```

Para enviar la solicitud a nuestro servidor de prueba, podríamos usar herramientas externas como SoapUI o la extensión Google Chrome Wizdler. Otra forma es ejecutar el siguiente comando en nuestro shell:

```
1 curl --header "content-type: text/xml" -d @request.xml http://localhost:8080/ws
```

La respuesta resultante podría no ser fácil de leer sin sangría o saltos de línea.

Para verlo formateado, podemos copiarlo y pegarlo en nuestro IDE u otra herramienta. Si hemos instalado xmllint, podemos canalizar la salida de nuestro comando curl a *xmllint*:

```
1 curl [command-line-options] | xmllint --format -
```

La respuesta debe contener información sobre España:

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay



```
1 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" (http://schemas.xmlsoap.or
2 <SOAP-ENV:Header/>
3 <SOAP-ENV:Body>
4   <ns2:getCountryResponse xmlns:ns2="http://www.baeldung.com/springsoap/gen (http://www.baeldung.com/
5     <ns2:country>
6       <ns2:name>Spain</ns2:name>
7       <ns2:population>46704314</ns2:population>
8       <ns2:capital>Madrid</ns2:capital>
9       <ns2:currency>EUR</ns2:currency>
10      </ns2:country>
11    </ns2:getCountryResponse>
12  </SOAP-ENV:Body>
13 </SOAP-ENV:Envelope>
```

## 6. Conclusión

En este artículo, aprendimos cómo crear un servicio web SOAP usando Spring Boot. También aprendimos cómo generar código Java a partir de un archivo XSD, y vimos cómo configurar los beans Spring necesarios para procesar las solicitudes SOAP.

El código fuente completo está disponible en GitHub (<https://github.com/eugenp/tutorials/tree/master/spring-soap>) .

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay



Acabo de anunciar el nuevo curso *Learn Spring*, centrado en los fundamentos de Spring 5 y Spring Boot 2:

>> VER EL CURSO (/ls-course-end)

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay



¿Estás aprendiendo a construir tu  
API  
con Spring ?

Ingrese su dirección de correo electrónico

>> Obtenga el libro electrónico

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay



iLos comentarios están cerrados en este artículo!

## CATEGORIAS

PRIMAVERA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/](https://www.baeldung.com/category/spring/))  
DESCANSO ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/](https://www.baeldung.com/category/rest/))  
JAVA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/](https://www.baeldung.com/category/java/))  
SEGURIDAD ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/](https://www.baeldung.com/category/security-2/))  
PERSISTENCIA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/](https://www.baeldung.com/category/persistence/))  
JACKSON ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/](https://www.baeldung.com/category/json/jackson/))  
HTTP DEL LADO DEL CLIENTE ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/](https://www.baeldung.com/category/http/))  
KOTLIN ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/](https://www.baeldung.com/category/kotlin/))

## SERIE

TUTORIAL DE JAVA 'VOLVER A LO BÁSICO' (/JAVA-TUTORIAL)  
JACKSON JSON TUTORIAL (/JACKSON)  
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)  
RESTO CON SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)  
TUTORIAL SPRING PERSISTENCE (/PERSISTENCE-WITH-SPRING-SERIES)  
SEGURIDAD CON PRIMAVERA (/SECURITY-SPRING)

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay



## ACERCA DE

SOBRE BAELDUNG (/ABOUT)

LOS CURSOS (HTTPS://COURSES.BAELDUNG.COM)

TRABAJOS (/TAG/ACTIVE-JOB/)

EL ARCHIVO COMPLETO (/FULL\_ARCHIVE)

ESCRIBIR PARA BAELDUNG (/CONTRIBUTION-GUIDELINES)

EDITORES (/EDITORS)

NUESTROS COMPAÑEROS (/PARTNERS)

ANUNCIE EN BAELDUNG (/ADVERTISE)

TÉRMINOS DE SERVICIO (/TERMS-OF-SERVICE)

POLÍTICA DE PRIVACIDAD (/PRIVACY-POLICY)

INFORMACIÓN DE LA COMPAÑÍA (/BAELDUNG-COMPANY-INFO)

CONTACTO (/CONTACT)

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#).

Okay