Content negotiation in Spring Boot refers to the mechanism that allows a client and server to agree on the format of data exchange for a given resource. This enables a single endpoint to serve responses in various formats (e.g., JSON, XML, plain text) based on the client's preference and the server's capabilities.

How Content Negotiation Works in Spring Boot:

Spring's ContentNegotiationManager handles content negotiation, determining the appropriate media type (MIME type) for the response based on a hierarchical strategy:

* **Path Extension (Highest Preference):**

The client can request a specific format by including an extension in the URL, such as .json or .xml. For example, http://localhost:8080/products/1.json would request a JSON representation.

* **URL Parameter:**

The client can specify the desired format using a URL parameter, like format=json or format=xml. For example, http://localhost:8080/products?format=xml.

* **Accept Header (HTTP Header):**

The client sends an Accept header in the HTTP request, indicating the preferred media types in order of preference (using "q" values for quality). For example, Accept: application/json, application/xml;q=0.9.

Enabling Content Negotiation:

To enable content negotiation for formats like XML in addition to the default JSON, you typically need to add the appropriate data format dependencies to your pom.xml (for Maven) or build.gradle (for Gradle). For XML, this usually involves adding the jackson-dataformat-xml dependency.

Code

<dependency>  
 <groupId>com.fasterxml.jackson.dataformat</groupId>  
 <artifactId>jackson-dataformat-xml</artifactId>  
</dependency>

Configuration:

While Spring Boot often auto-configures content negotiation, you can further customize it using the ContentNegotiationConfigurer within a @Configuration class that extends WebMvcConfigurer. This allows you to:

* Specify the order of content negotiation strategies.
* Disable path extension or parameter-based negotiation if desired.
* Register default content types.

Example:

Consider a REST endpoint that returns product details. With content negotiation enabled, the same URL can return JSON or XML based on the client's request:

* **JSON Request:** GET /products/1 with Accept: application/json or GET /products/1.json
* **XML Request:** GET /products/1 with Accept: application/xml or GET /products/1.xml

Spring Boot will then serialize the product object into the requested format if the necessary converters are available.