**深入浅出：Spring Boot 自定义消息转换器的实现与应用**

Spring Boot 作为当前最流行的 Java Web 开发框架之一，广泛应用于微服务架构、企业级应用等多个场景。Spring Boot 提供了灵活且易于扩展的架构，其中消息转换器（Message Converter）是其重要组成部分。消息转换器在 Spring Boot 中负责将 HTTP 请求体中的数据转换为 Java 对象，或将 Java 对象转换为 HTTP 响应体中的数据。

Spring Boot 默认提供了多种消息转换器，如 Jackson 用于 JSON 格式转换、JAXB 用于 XML 格式转换等。然而，实际开发中往往需要根据业务需求进行自定义的消息转换器。自定义消息转换器不仅可以满足特定格式的数据处理需求，还能提升系统的灵活性和可扩展性。

本文将详细介绍如何在 Spring Boot 中实现自定义消息转换器，探讨其工作原理、配置方式以及实际应用场景。通过这篇文章，读者将能清晰理解如何在 Spring Boot 项目中灵活使用消息转换器来实现数据的转换。

**一、消息转换器的工作原理**

Spring Boot 中的消息转换器（HttpMessageConverter）用于将 HTTP 请求和响应的内容转换为 Java 对象，或将 Java 对象转换为 HTTP 请求和响应的内容。它是通过 RestController、@RequestBody、@ResponseBody 等注解进行自动绑定的。

**1.1 消息转换器的基础概念**

消息转换器接口 HttpMessageConverter 负责将 HTTP 请求体的数据转换成 Java 对象，或将 Java 对象转换成 HTTP 响应体的数据。常见的消息转换器包括：

* MappingJackson2HttpMessageConverter：用于将 JSON 数据与 Java 对象相互转换。
* Jaxb2RootElementHttpMessageConverter：用于将 XML 数据与 Java 对象相互转换。
* StringHttpMessageConverter：用于处理文本数据，如 text/plain。

在 Spring Boot 中，HttpMessageConverter 主要依赖于 @RequestBody 和 @ResponseBody 来自动绑定请求体与 Java 对象，或者将响应数据转换为指定的格式。

**1.2 HttpMessageConverter 接口**

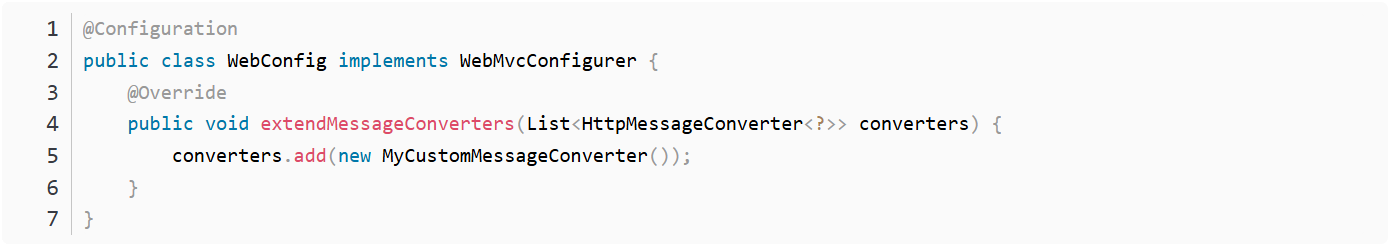
HttpMessageConverter 是 Spring 中进行消息转换的核心接口，其定义了以下主要方法：

* boolean canRead(Class<?> clazz, MediaType mediaType)：判断当前消息转换器是否能够将指定的媒体类型转换为目标类型。
* boolean canWrite(Class<?> clazz, MediaType mediaType)：判断当前消息转换器是否能够将指定的对象转换为指定的媒体类型。
* List<MediaType> getSupportedMediaTypes()：返回当前消息转换器支持的媒体类型。
* T read(Class<? extends T> clazz, HttpInputMessage inputMessage)：将输入消息体转换为 Java 对象。
* void write(T t, MediaType contentType, HttpOutputMessage outputMessage)：将 Java 对象转换为输出消息体。

这些方法提供了自定义消息转换器时必须实现的基本框架。

**1.3 HttpMessageConverter 与 Spring Boot 的集成**

在 Spring Boot 中，HttpMessageConverter 被自动注册到 HttpMessageConverters 中，Spring Boot 会根据请求头（如 Content-Type 和 Accept）自动选择合适的消息转换器进行数据转换。例如，当客户端发送 JSON 数据时，Spring 会自动选择 MappingJackson2HttpMessageConverter 进行转换。



上述代码中，我们可以通过 WebMvcConfigurer 的 extendMessageConverters 方法来添加自定义的消息转换器。

**二、如何实现自定义消息转换器**

在 Spring Boot 中实现自定义消息转换器并不复杂，基本步骤是继承 AbstractHttpMessageConverter 类，重写必要的方法，并注册到 Spring 的消息转换器列表中。

**2.1 自定义消息转换器的基本步骤**

1.继承 AbstractHttpMessageConverter：创建一个新的类，继承 AbstractHttpMessageConverter 或实现 HttpMessageConverter 接口。

2.实现 canRead 和 canWrite 方法：这两个方法用于判断当前消息转换器是否支持读取和写入指定类型的数据。

3.实现 read 和 write 方法：分别负责从输入流读取数据并转换成 Java 对象，以及将 Java 对象写入输出流。

**2.2 示例：实现一个自定义 JSON 格式的消息转换器**

假设我们需要实现一个自定义的 JSON 消息转换器，来处理一些特殊格式的 JSON 数据。我们可以按以下步骤进行实现。

**2.2.1 继承 AbstractHttpMessageConverter**





在这个例子中，我们创建了一个 MyCustomJsonMessageConverter 类，它继承自 AbstractHttpMessageConverter。我们重写了 supports 方法来判断该转换器是否支持 MyCustomObject 类型，readInternal 方法来从输入流读取 JSON 数据并转换为 MyCustomObject，以及 writeInternal 方法将 MyCustomObject 转换为 JSON 格式并写入输出流。

**2.2.2 注册自定义消息转换器**

接下来，我们需要将自定义的消息转换器注册到 Spring Boot 中。可以通过实现 WebMvcConfigurer 接口并覆盖 extendMessageConverters 方法来实现。



在WebConfig 配置类中，我们通过 extendMessageConverters 方法将 MyCustomJsonMessageConverter 注册到 Spring Boot 的消息转换器列表中。这样，当 Spring Boot 遇到 MyCustomObject 类型的请求时，就会使用我们自定义的转换器。

**三、实际应用场景**

**3.1 处理特殊数据格式**

自定义消息转换器常常用于处理一些特殊的数据格式，如自定义的 JSON、XML 格式，或第三方服务返回的数据格式。例如，某些 API 返回的数据不完全符合标准的 JSON 格式，或者需要对请求或响应的数据进行加密解密处理。这时，通过自定义消息转换器，我们可以在数据转换的过程中进行特殊处理。

**3.2 提升系统灵活性与可扩展性**

自定义消息转换器可以使得系统具备更高的灵活性和可扩展性。例如，系统中可能有多个不同的数据源，每个数据源返回的格式不同。通过自定义消息转换器，我们可以灵活地扩展支持不同格式的消息处理，而无需修改现有的业务逻辑代码。

**3.3 实现统一的格式化输出**

在实际开发中，有时需要将不同的数据源统一成一种输出格式。通过自定义消息转换器，可以将各种不同的数据格式统一转化为 JSON 或 XML 格式，并进行统一处理。例如，数据输出时可以统一添加时间戳、状态码等元信息，确保输出格式一致。

**四、总结**

Spring Boot 提供了强大的消息转换器机制，允许开发者根据业务需求灵活地对请求和响应数据进行格式转换。自定义消息转换器不仅可以帮助我们处理特殊格式的数据，还能提升系统的灵活性和可扩展性。

通过本文的讲解，我们了解了如何实现自定义消息转换器的基本步骤，包括继承AbstractHttpMessage

Converter` 类，重写核心方法，注册到 Spring Boot 的消息转换器列表中，并在实际开发中应用。希望通过这些示例和分析，能帮助开发者在实际项目中更加高效地使用 Spring Boot 的消息转换器来处理各种复杂的数据格式。

原文链接：https://blog.csdn.net/fudaihb/article/details/144816534