> Developing with Spring Boot

Externalized Configuration

Task Execution and Scheduling

Development-time Services

> Packaging Spring Boot Applications

> Production-ready Features

Creating Your Own Auto-

configuration

SSL

> Messaging

> Build Tool Plugins > Spring Boot CLI

> Rest APIs > Java APIs > Kotlin APIs

> Specifications

> Testing

> Data

> 10

SpringApplication

Internationalization Aspect-Oriented Programming

 ✓ Core Features

Profiles

JSON

JSON

Spring Boot provides integration with three JSON mapping libraries:

- Gson
- Jackson

 $If you use Jackson to serialize \ and \ deserialize \ JSON \ data, you might want to write your own \ JsonSerializer \ and \ JsonDeserializer \ and \ An$ classes. Custom serializers are usually registered with Jackson through a module, but Spring Boot provides an alternative @JsonComponent annotation that makes it easier to directly register Spring Beans.

You can use the @JsonComponent annotation directly on JsonSerializer, JsonDeserializer or KeyDeserializer implementations. You can also use it on classes that contain serializers/deserializers as inner classes, as shown in the following example:

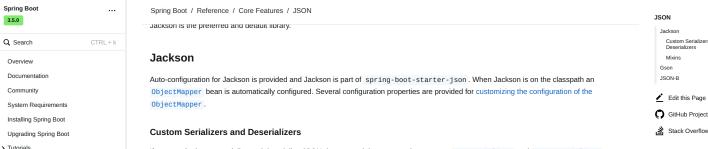


All @JsonComponent beans in the ApplicationContext are automatically registered with Jackson. Because @JsonComponent is meta-annotated with @Component, the usual component-scanning rules apply.

Spring Boot also provides JsonObjectSerializer and JsonObjectDeserializer base classes that provide useful alternatives to the standard Jackson versions when serializing objects. See JsonObjectSerializer and JsonObjectDeserializer in the API documentation for

The example above can be rewritten to use <code>JsonObjectSerializer</code> and <code>JsonObjectDeserializer</code> as follows:

```
Java Kotlin
@JsonComponent
public class MvJsonComponent {
    public static class Serializer extends JsonObjectSerializer<MyObject> {
        protected void serializeObject(MyObject value, JsonGenerator jgen, SerializerProvider provider)
                throws IOException {
            jgen.writeStringField("name", value.getName());
            jgen.writeNumberField("age", value.getAge());
       }
    public static class Deserializer extends JsonObjectDeserializer<MyObject> {
        protected MyObject deserializeObject(JsonParser jsonParser, DeserializationContext context, ObjectCodec co
                JsonNode tree) throws IOException {
            String name = nullSafeValue(tree.get("name"), String.class);
            int age = nullSafeValue(tree.get("age"), Integer.class);
            return new MyObject(name, age);
```







Mixins

Jackson has support for mixins that can be used to mix additional annotations into those already declared on a target class. Spring Boot's Jackson $auto-configuration\ will\ scan\ your\ application's\ packages\ for\ classes\ annotated\ with\ \ \underline{\texttt{@JsonMixin}}\ \ and\ register\ them\ with\ the\ auto-configured$ ${\tt ObjectMapper}\ .\ {\tt The}\ registration\ is\ performed\ by\ Spring\ {\tt Boot's}\ {\tt JsonMixinModule}\ .$

Gson

Auto-configuration for Gson is provided. When Gson is on the classpath a Gson bean is automatically configured. Several spring.gson.* con $figuration\ properties\ are\ provided\ for\ customizing\ the\ configuration.\ To\ take\ more\ control,\ one\ or\ more\ \ {\tt GsonBuilderCustomizer}\ beans\ can\ be$

JSON-B

Auto-configuration for JSON-B is provided. When the JSON-B API and an implementation are on the classpath a Jsonb bean will be automatically $configured. \ The \ preferred \ JSON-B \ implementation \ is \ Eclipse \ Yasson \ for \ which \ dependency \ management \ is \ provided.$

Aspect-Oriented Programming

Task Execution and Scheduling >