JSON

Spring Boot provides integration with three JSON mapping libraries:

- Gson
- Jackson

Jackson is the preferred and default library.

Edit this Page
GitHub Project
Stack Overflow

Jackson

Auto-configuration for Jackson is provided and Jackson is part of `spring-boot-starter-json`. When Jackson is on the classpath an `ObjectMapper` bean is automatically configured. Several configuration properties are provided for customizing the configuration of the `ObjectMapper`.

Custom Serializers and Deserializers

If you use Jackson to serialize and deserialize JSON data, you might want to write your own `JsonSerializer` and `JsonDeserializer` 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:

Java Kotlin

```java
@JsonComponent
public class MyJsonComponent {

    public static class Serializer extends JsonSerializer<MyObject> {

        @Override
        public void serialize(MyObject value, JsonGenerator jgen, SerializerProvider serializers) throws IOExcepti
            jgen.writeStartObject();
            jgen.writeStringField("name", value.getName());
            jgen.writeNumberField("age", value.getAge());
            jgen.writeEndObject();
        }

    }

    public static class Deserializer extends JsonDeserializer<MyObject> {

        @Override
        public MyObject deserialize(JsonParser jsonParser, DeserializationContext ctxt) throws IOException {
            ObjectCodec codec = jsonParser.getCodec();
            JsonNode tree = codec.readTree(jsonParser);
            String name = tree.get("name").textValue();
            int age = tree.get("age").intValue();
            return new MyObject(name, age);
        }

    }

}
```

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 details.

The example above can be rewritten to use `JsonObjectSerializer` and `JsonObjectDeserializer` as follows:

Java Kotlin

```java
@JsonComponent
public class MyJsonComponent {

    public static class Serializer extends JsonObjectSerializer<MyObject> {

        @Override
        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> {

        @Override
        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 `@JsonMixin` and register them with the auto-configured `ObjectMapper`. The registration is performed by Spring Boot’s `JsonMixinModule`.

Gson

Auto-configuration for Gson is provided. When Gson is on the classpath a `Gson` bean is automatically configured. Several `spring.gson.*` configuration properties are provided for customizing the configuration. To take more control, one or more `GsonBuilderCustomizer` beans can be used.

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.

Prev
Aspect-Oriented Programming

Next
Task Execution and Scheduling