tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

About the Tutorial

Boon is a simple Java based toolkit for JSON. You can use Boon JSON to encode or decode JSON data in an efficient and faster way.

Audience

This tutorial is designed for Software Professionals who are willing to encode/decode JSON data in Java in simple and easy steps. This tutorial will give you an understanding of the Boon concepts and after completing the tutorial, you will be at an intermediate level of expertise from where you can take yourself to a higher level of expertise.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of Java, JSON, JavaScript, text editor, and execution of programs, etc.

Copyright & Disclaimer

Table of Contents

1. Boon — Overview

Boon is a simple Java based toolkit for JSON. You can use Boon JSON to encode or decode JSON data in an efficient and faster way.

Features of Boon

The features of Boon are explained below:

- **Fast** – Boon JSON is faster at Object Serialization, enabling JSON Expression and JSON Parsing as compared to Jackson.
- **Lightweight** – It has very few classes and provides the necessary functionalities like encode/decode Object mapping.
- **Data Binding** – Most of the operations in Boon are done using data binding and index overlay.
- **No public tree model** – End user view is data binding view.
- **Supports simple data binding** – Provides data binding with primitives as well with auto boxing.
- **High performance** – Heap based parser is used which provides high performance.
- **No dependency** – No external library dependency and can be independently included.
- **JDK1.2 compatible** – Source code and the binary are JDK1.2 compatible.

2. Boon — Environment Setup

In this chapter, we will learn about the local environment setup of Boon and how to set up the path of Boon for Windows 2000/XP, Windows 95/98/ME etc. We will also understand about some popular java editors and how to download Boon archive.

Local Environment Setup

If you are still willing to set up your environment for Java programming language, then this chapter will guide you on how to download and set up Java on your machine. Please follow the steps mentioned below to set up the environment.

Java SE is freely available from the link https://www.oracle.com/java/technologies/oracle-java-archive-downloads.html. So, you can download a version based on your operating system.

Follow the instructions to download Java and run the **.exe** to install Java on your machine. Once you have installed Java on your machine, you would need to set environment variables to point to correct installation directories.

Path for Windows 2000/XP

We are assuming that you have installed Java in **c:\Program Files\java\jdk** directory.

- Right-click on **'My Computer'** and select **'Properties'**.
- Click on the **'Environment variables'** button under the **'Advanced'** tab.
- Now, alter the **'Path'** variable so that it also contains the path to the Java executable. Example, if the path is currently set to **'C:\WINDOWS\SYSTEM32'**, then change your path to read **'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.**

Path for Windows 95/98/ME

We are assuming that you have installed Java in **c:\Program Files\java\jdk** directory.

- Edit the 'C:\autoexec.bat' file and add the following line at the end – 'SET PATH=%PATH%;C:\Program Files\java\jdk\bin'.

Path for Linux, UNIX, Solaris, FreeBSD

Environment variable PATH should be set to point, where the Java binaries have been installed. Refer to your shell documentation if you have trouble doing this.

Example, if you use bash as your shell, then you would add the following line to the end of your '.bashrc: export PATH=/path/to/java:$PATH'.

Popular Java Editors

To write your Java programs, you need a text editor. There are many sophisticated IDEs available in the market. But for now, you can consider one of the following:

- **Notepad** – On Windows machine you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.
- **Netbeans** – It is a Java IDE that is open-source and free which can be downloaded from https://www.netbeans.org/index.html.
- **Eclipse** – It is also a Java IDE developed by the eclipse open-source community and can be downloaded from https://www.eclipse.org/.

Download Boon Archive

Download the latest version of Boon jar file from **Maven Repository - Boon**, which is available at https://mvnrepository.com/artifact/io.fastjson/boon. In this tutorial, boon-0.34.jar is downloaded and copied into C:\> boon folder.

OS	Archive name
Windows	boon-0.34.jar
Linux	boon-0.34.jar
Mac	boon-0.34.jar

Set Boon Environment

Set the **BOON** environment variable to point to the base directory location where, Boon jar is stored on your machine. Assume that we have extracted **boon-0.34.jar** in Boon folder on various Operating Systems as follows:

OS	Output
Windows	Set the environment variable BOON to C:\Boon
Linux	export BOON=/usr/local/Boon
Mac	export BOON=/Library/Boon

Set CLASSPATH Variable

Set the **CLASSPATH** environment variable to point to the Boon jar location. Assume that you have stored boon-0.34.jar in Boon folder on various Operating Systems as follows:

OS	Output
Windows	Set the environment variable CLASSPATH to %CLASSPATH%;%Boon%\boon-0.34.jar;.;
Linux	export CLASSPATH=$CLASSPATH:$BOON/boon-0.34.jar:.
Mac	export CLASSPATH=$CLASSPATH:$BOON/boon-0.34.jar:.

3. Boon — To Object

ObjectMapper is the main actor class of Boon library. ObjectMapper class provides functionality for reading and writing JSON, either to and from basic POJOs (Plain Old Java Objects), or to and from a general-purpose JSON Tree Model (JsonNode), as well as related functionality for performing conversions.

It is also highly customisable to work both with different styles of JSON content, and to support more advanced Object concepts such as polymorphism and Object identity.

Example

Following example is using ObjectMapper class to parse a JSON string to a Student Object.

```
import org.boon.json.JsonFactory;
import org.boon.json.ObjectMapper;

public class BoonTester {
   public static void main(String args[]){
      ObjectMapper mapper = JsonFactory.create();
      String jsonString = "{\"name\":\"Mahesh\", \"age\":21}";

      Student student = mapper.readValue(jsonString, Student.class);
      System.out.println(student);
   }
}
class Student {
   private String name;
   private int age;
   public Student(){}
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public int getAge() {
      return age;
   }
```

```
   public void setAge(int age) {
      this.age = age;
   }
   public String toString(){
      return "Student [ name: "+name+", age: "+ age+ " ]";
   }
}
```

Output

The output is mentioned below:

```
Student [ name: Mahesh, age: 21 ]
```

4. Boon — To Map

ObjectMapper class can also be used to parse a json to Map object instead of a POJO object.

Example

Following example is using ObjectMapper class to parse a JSON string to a Map Object.

```
import java.util.Map;
import org.boon.json.JsonFactory;
import org.boon.json.ObjectMapper;

public class BoonTester {
   public static void main(String args[]) {
      ObjectMapper mapper = JsonFactory.create();
      String jsonString = "{\"name\":\"Mahesh\", \"age\":21}";
      Map studentMap = mapper.readValue(jsonString, Map.class);
      System.out.println("Name: " + studentMap.get("name"));
      System.out.println("Age: " + studentMap.get("age"));
   }
}
```

Output

The output is given below:

```
Name: Mahesh
Age: 21
```

5. Boon — Sources

ObjectMapper class can be used to parse a json from varying sources. It can use following sources to parse JSON.

- byte Array
- char Array
- File
- Reader classes
- Input Stream classes
- String

Example

Following example is using ObjectMapper class to parse a JSON char array to a Map Object.

```
import java.util.Map;
import org.boon.json.JsonFactory;
import org.boon.json.ObjectMapper;

public class BoonTester {
   public static void main(String args[]){
      ObjectMapper mapper = JsonFactory.create();
      String jsonString = "{\"name\":\"Mahesh\", \"age\":21}";
      char[] jsonCharAray = jsonString.toCharArray();

      Map studentMap = mapper.readValue(jsonCharAray, Map.class);
      System.out.println("Name: " + studentMap.get("name"));
      System.out.println("Age: " + studentMap.get("age"));
   }
}
```

Output

You will see the following output:

```
Name: Mahesh
Age: 21
```

6. Boon — From Object

ObjectMapper class can be used to generate a json string from an Object.

Example

Following example is using ObjectMapper class to generate a JSON string from a Student Object.

```
import org.boon.json.JsonFactory;
import org.boon.json.ObjectMapper;

public class BoonTester {
   public static void main(String args[]){
      ObjectMapper mapper = JsonFactory.create();
      Student student = new Student("Mahesh", 21);
      String jsonString = mapper.writeValueAsString(student);
      System.out.println(jsonString);
   }
}
class Student {
   public String name;
   public int age;
   public Student(String name, int age) {
      this.name = name;
      this.age = age;
   }
}
```

Output

This produces the following output:

```
{"name":"Mahesh","age":21}
```

7. Boon — From Map

ObjectMapper class can be used to generate a json string from a Map.

Example

Following example is using ObjectMapper class to generate a JSON string from a Map Object.

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.boon.json.JsonFactory;
import org.boon.json.ObjectMapper;

public class BoonTester {
   public static void main(String args[]){
      ObjectMapper mapper = JsonFactory.create();
      Map<String, String> student = new HashMap<>();
      student.put("Name", "Mahesh");
      student.put("RollNo", "21");

      Map<String, String> student1 = new HashMap<>();
      student1.put("Name", "Suresh");
      student1.put("RollNo", "22");

      List<Map<String,String>> studentList = new ArrayList<>();
      studentList.add(student);
      studentList.add(student1);

      Map<String, List> studentMap = new HashMap<String, List>();
      studentMap.put("students", studentList);

      String jsonString = mapper.writeValueAsString(studentMap);
      System.out.println(jsonString);
   }
```

```
}
```

Output

When you execute the above code, you should see the following output:

```
{"students":[{"RollNo":"21","Name":"Mahesh"},{"RollNo":"22","Name":"Suresh"}]}
```

8. Boon — Long to Date

ObjectMapper class can be used to work with different date formats in JSON. It can be used to parse/generate long version of date.

Example

Following example is using ObjectMapper class to generate a Date string from a long version.

```
import java.util.Date;
import org.boon.json.JsonFactory;
import org.boon.json.ObjectMapper;

public class BoonTester {
   public static void main(String args[]) {
      ObjectMapper mapper = JsonFactory.create();
      String jsonString = "{\"name\":\"Mahesh\", \"age\":21,
\"dateOfBirth\":976559400000}";

      //mapper converts long to date automatically
      Student student = mapper.readValue(jsonString, Student.class);
      System.out.println(student.dateOfBirth);

      //by default mapper converts date to long
      jsonString = mapper.writeValueAsString(student);
      System.out.println(jsonString);
   }
}
class Student {
   public String name;
   public int age;
   public Date dateOfBirth;

   public Student(String name, int age, Date dateOfBirth) {
      this.name = name;
      this.age = age;
      this.dateOfBirth = dateOfBirth;
```

```
    }
}
```

Output

Given below is the output of the code:

```
Tue Dec 12 00:00:00 IST 2000
{"name":"Mahesh","age":21,"dateOfBirth":976559400000}
```

9. Boon — String to Date

ObjectMapper class can be used to work with different date formats in JSON. It can be used to parse/generate String version of date.

Example

Following example is using ObjectMapper class to generate a Date string from a String version.

```
import java.util.Date;
import org.boon.json.JsonFactory;
import org.boon.json.ObjectMapper;

public class BoonTester {
   public static void main(String args[]) {
      ObjectMapper mapper = JsonFactory.create();
      String jsonString = "{\"name\":\"Mahesh\", \"age\":21,
\"dateOfBirth\":\"1998-08-11T11:31:00.034Z\" }";

      //mapper converts String to date automatically
      Student student = mapper.readValue(jsonString, Student.class);
      System.out.println(student.dateOfBirth);

      //by default mapper converts date to long
      jsonString = mapper.writeValueAsString(student);
      System.out.println(jsonString);
   }
}
class Student {
   public String name;
   public int age;
   public Date dateOfBirth;

   public Student(String name, int age, Date dateOfBirth) {
      this.name = name;
      this.age = age;
      this.dateOfBirth = dateOfBirth;
```

```
    }
}
```

Output

When you execute the above code, you should see the following output:

```
Tue Aug 11 17:01:00 IST 1998
{"name":"Mahesh","age":21,"dateOfBirth":902835060034}
```

10. Boon — Generating Date

ObjectMapper class can be used to work with different date formats in JSON. It can be used to generate date object as well. By default, ObjectMapper generates Date in long milliseconds version. Using ObjectMapper returned by JsonFactory.createUseJSONDates() method, we can get a string version of date during parsing.

Example

Following example is using ObjectMapper class to generate a Date string by parsing JSON.

```
import java.util.Date;
import org.boon.json.JsonFactory;
import org.boon.json.ObjectMapper;

public class BoonTester {
   public static void main(String args[]) {
      ObjectMapper mapper = JsonFactory.createUseJSONDates();
      String jsonString = "{\"name\":\"Mahesh\", \"age\":21,
\"dateOfBirth\":\"1998-08-11T11:31:00.034Z\" }";

      //mapper converts String to date automatically
      Student student = mapper.readValue(jsonString, Student.class);
      System.out.println(student.dateOfBirth);

      //Mapper converts date to date string now
      jsonString = mapper.writeValueAsString(student);
      System.out.println(jsonString);
   }
}
class Student {
   public String name;
   public int age;
   public Date dateOfBirth;
   public Student(String name, int age, Date dateOfBirth) {
      this.name = name;
      this.age = age;
      this.dateOfBirth = dateOfBirth;
```

```
    }
}
```

Output

You will receive the following output:

```
Tue Aug 11 17:01:00 IST 1998
{"name":"Mahesh","age":21,"dateOfBirth":"1998-08-11T11:31:00.034Z"}
```

11. Boon — @JsonIgnore

@JsonIgnore is used at field level to mark a property or list of properties to be ignored.

Example - @JsonIgnore

Following example is for @JsonIgnore:

```
import org.boon.json.JsonFactory;
import org.boon.json.ObjectMapper;
import org.boon.json.annotations.JsonIgnore;

public class BoonTester {
   public static void main(String args[]) {
      ObjectMapper mapper = JsonFactory.create();
      Student student = new Student(1,11,"1ab","Mark");
      String jsonString = mapper.writeValueAsString(student);
      System.out.println(jsonString);
   }
}
class Student {
   public int id;
   @JsonIgnore
   public String systemId;
   public int rollNo;
   public String name;

   Student(int id, int rollNo, String systemId, String name) {
      this.id = id;
      this.systemId = systemId;
      this.rollNo = rollNo;
      this.name = name;
   }
}
```

Output

You will see the following output:

```
{"id":1,"rollNo":11,"name":"Mark"}
```

12. Boon — @JsonInclude

@JsonInclude is used to include properties having null/empty or default values. By default, Boon ignore such properties during serialization/de-serialization.

Example - @JsonInclude

Following example is for @JsonInclude:

```
import org.boon.json.JsonFactory;
import org.boon.json.ObjectMapper;
import org.boon.json.annotations.JsonInclude;

public class BoonTester {
   public static void main(String args[]) {
      ObjectMapper mapper = JsonFactory.createUseAnnotations( true );
      Student student = new Student(1,null);
      String jsonString = mapper.writeValueAsString(student);
      System.out.println(jsonString);
   }
}
class Student {
   public int id;
   @JsonInclude
   public String name;

   Student(int id, String name) {
      this.id = id;
      this.name = name;
   }
}
```

Output

When the script runs successfully, you will see the following output:

```
{"id":1,"name":null}
```

13. Boon — @JsonViews

@JsonViews is used to control the values which are to be serialized or not.

Example - @JsonView

Following example is for @JsonView:

```
import org.boon.json.JsonSerializer;
import org.boon.json.JsonSerializerFactory;
import org.boon.json.annotations.JsonViews;

public class BoonTester {
   public static void main(String args[]) {
      JsonSerializer serializerPublic = new JsonSerializerFactory()
         .useAnnotations()
         .setView( "public" )
         .create();

      JsonSerializer serializerInternal = new JsonSerializerFactory()
         .useAnnotations()
         .setView( "internal" )
         .create();

      Student student = new Student(1,"Mark", 20);
      String jsonString = serializerPublic.serialize( student ).toString();
      System.out.println(jsonString);
      jsonString = serializerInternal.serialize( student ).toString();
      System.out.println(jsonString);
   }
}
class Student {
   public int id;
   public String name;
   @JsonViews( ignoreWithViews = {"public"}, includeWithViews = {"internal"})
   public int age;
```

```
   Student(int id, String name, int age) {
      this.id = id;
      this.name = name;
      this.age = age;
   }
}
```

Output

We will get the output similar as follows:

```
{"id":1,"name":"Mark"}
{"id":1,"name":"Mark","age":20}
```

14. Boon — @JsonProperty

@JsonProperty is used to mark non-standard getter/setter method to be used with respect to json property.

Example - @JsonProperty

Following example is for @JsonProperty:

```
import org.boon.json.JsonFactory;
import org.boon.json.ObjectMapper;
import org.boon.json.annotations.JsonProperty;

public class BoonTester {
   public static void main(String args[]) {
      ObjectMapper mapper = JsonFactory.create();
      Student student = new Student(1);
      String jsonString = mapper.writeValueAsString(student);
      System.out.println(jsonString);
   }
}
class Student {
   private int id;
   Student(){}
   Student(int id){
      this.id = id;
   }
   @JsonProperty("id")
   public int getTheId() {
      return id;
   }
   @JsonProperty("id")
   public void setTheId(int id) {
      this.id = id;
   }
}
```

Output

Upon execution, you will receive the following output:

```
{"id":1}
```