Luxoft
Компания
рейтинг 106,62

Профиль | 144 Блог | 0 Вакансии | 1,5k Подписчики

26 апреля в 17:17

Разработка → Шпаргалка Java программиста 8. Библиотеки для работы с Json (Gson, Fastjson, LoganSquare, Jackson, JsonPath и другие)

tutorial

Разработка веб-сайтов*, Программирование*, Open source*, Java*, Блог компании Luxoft

В одной из моих прошлых статей я рассказывал о своем opensorce pet проекте useful-java-links, идея которого собрать как можно больше ссылок на полезные Java библиотеки и фреймворки. У него так же есть подпроект Hello World project идея которого для каждой библиотеки собрать несколько простых примеров её использования.

Проблема программистов в Java мире в том что кроме стандартной библиотеки JDK есть огромное других полезных библиотек, причем переход от одной библиотеки к другой может вызывать проблемы из-за неполной документации, отсутствия простых примеров или даже сложности понять какие зависимости нужно добавить в maven чтобы все запустилось. А на новой работе вполне могут использовать вместо твоей любимой библиотеки ту которую ты не знаешь. Идея моего проекта облегчить изучение и выбор разных библиотек.

Общее оглавление 'Шпаргалок'

1. JPA и Hibernate в вопросах и ответах
2. Триста пятьдесят самых популярных не мобильных Java opensource проектов на github
3. Коллекции в Java (стандартные, guava, apache, trove, gs-collections и другие)
4. Java Stream API
5. Двести пятьдесят русскоязычных обучающих видео докладов и лекций о Java
6. Список полезных ссылок для Java программиста
7 Типовые задачи
  7.1 Оптимальный путь преобразования InputStream в строку
  7.2 Самый производительный способ обхода Map'ы, подсчет количества вхождений подстроки
8. Библиотеки для работы с Json (Gson, Fastjson, LoganSquare, Jackson, JsonPath и другие)

Итак, давайте посмотрим какие известные библиотеки есть для работы с JSON в Java…

Цитата из useful-java-links:

8. Работа с Json...

8. Работа с Json

JSON парсеры

1. Alibaba Fastjson Быстрый JSON обработчик, рейтинг github'a — 4851. User guide и Hello World examples. Лицензия: Apache 2. **Лицензия совместима с закрытым ПО**
2. Gson — Простая сериализации объектов в JSON и обратно. Хорошая производительность и легкость в использовании, рейтинг github'a — 4120. User guide и Hello World examples. Лицензия: Apache 2. **Лицензия совместима с закрытым ПО**
3. LoganSquare -Библиотека парсинга и сериализации JSON, основанная на Jackson's streaming API. По словам разработчиков, превосходит по производительности GSON и Jackson библиотеки, рейтинг github'a — 2188. User guide и Hello World examples. Лицензия: Apache 2. **Лицензия совместима с закрытым ПО**
4. JSON java Реализация работы с JSON в Java от разработчиков JSON стандарта, рейтинг github'a — 1937. User guide и Hello World examples. Лицензия: Crockford's license (MIT License + "Good, not Evil").
5. Square Moshi JSON библиотека для Android и Java, служит для упрощения парсинга Json в объекты Java, рейтинг github'a — 1732. User guide и Hello World examples. Лицензия: Apache 2 **Лицензия совместима с закрытым ПО**
6. Instagram Ig json parser Быстрый JSON парсер для java проектов, рейтинг github'a — 921. User guide и Hello World examples. Лицензия: BSD 3. **Лицензия совместима с закрытым ПО**
7. Jackson — Похоже на GSON, но более производительна, если вам нужно часто создавать экземпляр библиотеки. Подпроекты: Jackson core Базовая часть функционала, Jackson databind Базовая реализация databind'a, рейтинг github'a — 881. User guide и Hello World examples. Лицензия: Apache 2. **Лицензия совместима с закрытым ПО**
8. Genson — Мощная и простая в использовании Java библиотека для преобразования в/из JSON, рейтинг github'a — 108. User guide и Hello World examples. Лицензия: Apache 2. **Лицензия совместима с закрытым ПО**

Аналог XPath для JSON

1. Jayway JsonPath Java JsonPath — реализация аналога XPATH только для Json, а не XML, рейтинг github'a — 849. User guide и Hello World examples. Лицензия: Apache 2. **Лицензия совместима с закрытым ПО**
2. Alibaba Fastjson Быстрый JSON обработчик, рейтинг github'a — 4851. User guide и Hello World examples. Лицензия: Apache 2. **Лицензия совместима с закрытым ПО**

Генерация Java классов из JSON или JSON схемы и JSON валидация

1. Jsonschema2pojo Генерация Java классов из JSON схемы (или примера JSON) с аннотациями для data-binding для Jackson 1.x or 2.x, Gson и т. п.., рейтинг github'a — 1664. User guide и Hello World examples. Лицензия: Apache 2. **Лицензия совместима с закрытым ПО**
2. Json schema validator Валидация JSON схемы, реализована на чистой Java, создана с целью проверки Json файлов, используя Json схемы., так же может генерировать Java классы из схемы и наоборот, рейтинг github'a — 547. User guide и Hello World examples. Лицензия: GNU Lesser 3/Apache 2. **Лицензия совместима с закрытым ПО**

Итак, у нас восемь библиотек для сериализации и десериализации в json, две библиотеки для генерации Java классов по схеме или json файлу, одна библиотека для валидации схемы и два аналога XPath, но для json. Давайте рассмотрим каждую из них.

1. JSON парсеры

Существует три основных способа сериализации и десериализации среди указанных библиотек (от самого простого к самому сложному) и один дополнительный:

1. Data bind,
2. Tree Model,
3. Streaming API,
4. (И дополнительный способ) Аналоги XPath,

Давайте рассмотрим с чем их едят:

1. **Data bind** самый популярный и простой способ, вы просто указываете класс, который нужно преобразовать в json, может быть часть полей отмечаете аннотациями (а зачастую даже это необязательно), а библиотека сама превращает этот класс и всю его иерархию классов в json. Аналогом при работе с xml будет JAXB (Java Architecture for XML Binding)
   Плюсы: наиболее простой из всех, по сути главное реализовать только Java классы, более того можно просто сгенерировать Java классы из json'a или json схемы.
   Минусы: скорость и память. Большинство библиотек использует рефлексию и т.п. методы работы с Java классами (хотя не все), что очевидно не очень быстро. К тому же, весь json файл сразу превращается в Java объекты, что может просто исчерпать всю доступную память, если вы попытаетесь обработать очень большой json.
   Вывод: если нет проблем с производительностью, памятью и вы не собираетесь обрабатывать многогигабайтные json'ы скорее всего самый лучший способ.

2. **Tree Model** — данный парсер представляет json в виде Java классов таких как Node или JsonElement с иерархической структурой, а уже сам программист их обходит и получает из них информацию. Данный способ похож на DOM парсеры в xml.
   Плюсы: обычно быстрее первого способа и проще третьего,
   Минусы: уступает **Data bind** по простоте, плюс ряд библиотек способен генерить классы при **Data bind**, а не использовать

рефлексию, в этом случае то что **Tree Model** будет быстрее не очевидно, к тому же не решается проблема огромных файлов и ограничения памяти.

3. **Streaming API** — самый низкоуровневый способ, по сути программист сам вручную разбирает токены json'а. Зато никаких ограничений по памяти и в теории максимальная производительность.
   Плюсы: производительность и минимальное потребление памяти,
   Минусы: сложность использования,

4. **Аналоги XPath** — дополнительный способ, не очень подходит, если нужно получит всю информацию из json'а, зато позволяет написав выражение `$.store.book[*].author` и получить список всех авторов всех книг из json'а магазина. То есть легко получать часть информации из json'а.
   Плюсы: позволяет быстро получить информацию из json'а по сложным критериям,
   Минусы: не очень подходит, когда нужна все информация из json'а, не работает в обратную сторону на формирования json'ов,

1.1 Обзор библиотек

Способ	Fastjson	Gson	LoganSquare	JSON java	Moshi	Ig json parser	Jackson	Genson	JsonPath
1. Data bind	Да	Да	Да	-	Да	Да	Да	Да	-
2. Tree Model	-	Да	-	Да	-	-	Да	-	-
3. Streaming API	-	Да	-	-	-	-	Да	-	-
4. Аналоги XPath	Да	-	-	-	-	-	-	-	Да
5. Генерация классов для Data bind*	-	-	Да	-	-	Да	-	-	-
6. Github's star	4851	4120	2188	1937	1732	921	881	108	849
7. Работает со static inner class**	Да	Да	Нет	-	Да	Нет	Да	Да	-
8. Обязательность аннотаций***	Нет	Нет	Да	-	Нет	Да	Нет	Нет	-

По ссылкам на **Да** можно найти примеры использования.
`*` — Генерация классов для Data bind позволяет сгенерировать классы на стадии компиляции, что в теории должно давать значительный прирост производительности библиотеки,
`**` — Работает со static inner class имеет смысл только для случая Data bind, возможно ли сериализация и десериализация для случая статических внутренних классов (не статические внутренние классы сериализовать не рекомендуется),
`***` — тоже только для случая Data bind можно ли не использовать аннотации или их использование крайне рекомендуется,

1.2 Простейшие примеры использование Data bind

Для демонстрации работы библиотек будем использовать следующий json:

```
jsonString =
{
  "message": "Hi",
  "place": {
    "name": "World"
  }
}
```

И следующие Java классы (в разных примерах могут слегка отличаться наличием аннотаций, если они обязательны):

▾ Java классы

```
class Human {
    private String message;
    private Place place;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
```

```
        }

        public Place getPlace() {
            return place;
        }

        public void setPlace(Place place) {
            this.place = place;
        }

        public void say() {
            System.out.println();
            System.out.println(getMessage() + " , " + getPlace().getName() + "!");
        }
    }

    class Place {
        private String name;

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }
    }


    // init class
    Place place = new Place();
    place.setName("World");

    Human human = new Human();
    human.setMessage("Hi");
    human.setPlace(place);
```

Как можно увидеть, Java классы всего лишь состоять из двух классов Human и Place, в которых храниться сообщение Hi World!.. Json тоже содержит эти два вложенных объекта.

Примеры использования (Data bind): Способ	Fastjson	Gson	LoganSquare	Moshi	Ig json parser	Jackson	Genson
Инициализация	---	`Gson gson = new Gson()`	---	`Moshi moshi = new Moshi.Builder().build(); JsonAdapter<Human> jsonAdapter = moshi.adapter(Human.class)`	---	`ObjectMapper mapper = new ObjectMapper()`	`Genson genson = new Genson()`
Из Java в json	`JSON.toJSONString(human)`	`gson.toJson(human)`	`LoganSquare.serialize(human)`	`jsonAdapter.toJson(human)`	`Human__JsonHelper.serializeToJson(human)`	`mapper.writeValueAsString(human)`	`genson.serialize(human)`
Из json в Java	`JSON.parseObject(jsonString, Human.class)`	`gson.fromJson(jsonString, Human.class)`	`LoganSquare.parse(jsonString, Human.class)`	`jsonAdapter.fromJson(jsonString)`	`Human__JsonHelper.parseFromJson(jsonString)`	`mapper.readValue(jsonString, Human.class)`	`genson.deserialize(jsonString, Human.class)`

Human__JsonHelper — это класс который Ig json parser сгенерировал на этапе компиляции, у LoganSquare так же есть генерации на этапе компиляции, но там классы подключаются "под капотом" внутри LoganSquare.

Давайте рассмотрим примеры подробнее.

Fastjson

```
// convert to json
String jsonString = JSON.toJSONString(human);
System.out.println("json " + jsonString); //  напечатает  "json {"message":"Hi","place":{"name":"World"}}"

// convert from json
Human newHuman = JSON.parseObject(jsonString, Human.class);
newHuman.say(); //  напечатает  "Hi , World!"
```

Подробный пример

Gson

```
// convert to json
Gson gson = new Gson();
String jsonString = gson.toJson(human);
System.out.println("json " + jsonString); //  напечатает  "json {"message":"Hi","place":{"name":"World"}}"

// convert from json
Human newHuman = gson.fromJson(jsonString, Human.class);
newHuman.say(); //  напечатает  "Hi , World!"
```

Подробный пример

LoganSquare

```
@JsonObject
public class Human {
    @JsonField(name="message")
    public String message;
    @JsonField(name="place")
    public Place place;

       ....

        // convert to json
        String jsonString = LoganSquare.serialize(human);
        System.out.println("json " + jsonString); // напечатает "json {"place":{"name":"World"},"message":"Hi"}"

        // convert from json
        Human newHuman = LoganSquare.parse(jsonString, Human.class);
        newHuman.say(); // напечатает "Hi , World!"
```

Подробный пример

Moshi

```
// convert to json
Moshi moshi = new Moshi.Builder().build();
JsonAdapter<Human> jsonAdapter = moshi.adapter(Human.class);

String jsonString = jsonAdapter.toJson(human);
System.out.println("json " + jsonString); // напечатает  "json {"message":"Hi","place":{"name":"World"}}"

// convert from json
Human newHuman = jsonAdapter.fromJson(jsonString);
newHuman.say(); //  напечатает "Hi , World!"
```

Подробный пример

Ig json parser

```
@JsonType
public class Human {
    @JsonField(fieldName="message")
    public String message;
    @JsonField(fieldName="place")
    public Place place;

        ...

        // convert to json
        String jsonString = Human__JsonHelper.serializeToJson(human);
        System.out.println("json " + jsonString); // напечатает  "json {"place":{"name":"World"},"message":"Hi"}"

        // convert from json
        Human newHuman = Human__JsonHelper.parseFromJson(jsonString);
        newHuman.say(); //  напечатает "Hi , World!"
```

Подробный пример

Jackson

```
        // convert to json
        ObjectMapper mapper = new ObjectMapper();
        String jsonString = mapper.writeValueAsString(human);
        System.out.println("json " + jsonString); //  напечатает  "json {"message":"Hi","place":{"name":"World"}}"

        // convert from json
        Human newHuman = mapper.readValue(jsonString, Human.class);
        newHuman.say(); //  напечатает "Hi , World!"
```

Подробный пример

Genson

```
        // convert to json
        String jsonString = new Genson().serialize(human);

        System.out.println("json " + jsonString); // напечатает  "json {"message":"Hi","place":{"name":"World"}}"

        // convert from json
        Human newHuman =  new Genson().deserialize(jsonString, Human.class);
        newHuman.say(); //  напечатает  "Hi , World!"
```

Подробный пример

Для изучения более сложных примеров библиотек см. раздел документация и генерация java классов из json. Используя генерацию можно быстро получить нужные java классы со всеми аннотациями для библиотек jackson или gson.

1.3 Простейшие примеры использование Tree Model

Использование Tree Model есть у трех библиотек: Gson, Jackson и Json Java. Давайте посмотрим их реализацию.

Для демонстрации работы библиотек будем использовать тот же json:

```
jsonString =
{
  "message": "Hi",
  "place": {
```

```
    "name": "World"
  }
}
```

Методы парсинга json'а:

Действие	Gson	Jackson	JSON java
Инициализация	`JsonParser parser = new JsonParser()`	`new ObjectMapper()`	-
Парсинг json'а	`parser.parse(<строка>)`	`mapper.readValue(<строка>, JsonNode.class)`	`new JSONObject(<строка>)`
Получение главного объекта	`root.getAsJsonObject()`	-	-
Получение строки	`root.get(<имя>).getAsString()`	`root.get(<имя>).asText()`	`root.getString(<имя>)`
Получение дочернего объекта	`root.getAsJsonObject(<имя>)`	`root.get(<имя>)`	`root.getJSONObject(<имя>)`

Методы генерации json'а:

Действие	Gson	Jackson	JSON java
Инициализация	-	`new ObjectMapper()`	-
Создание главного объекта	`new JsonObject()`	`mapper.createObjectNode()`	`new JSONObject()`
Добавить строковое поле	`root.addProperty(<имя>, <строка>)`	`root.put(<имя>, <строка>)`	`root.put(<имя>, <строка>)`
Добавить дочерний объект	`root.add(<имя>, <объект>);`	`root.putObject(<имя>)`	`root.put(<имя>, <объект>)`

Примеры:

1) Чтение Gson

Чтение json с помощью Gson

```
JsonParser parser = new JsonParser();
JsonElement jsonElement = parser.parse("{\"message\":\"Hi\",\"place\":{\"name\":\"World!\"}}");

JsonObject rootObject = jsonElement.getAsJsonObject(); // чтение главного объекта
String message = rootObject.get("message").getAsString(); // получить поле "message" как строку
JsonObject childObject = rootObject.getAsJsonObject("place"); // получить объект Place
String place = childObject.get("name").getAsString(); // получить поле "name"
System.out.println(message + " " + place); // напечатает "Hi World!"*/
```

Подробный пример

2) Генерация Gson

Генерация json с помощью Gson

```
JsonObject rootObject = new JsonObject(); // создаем главный объект
rootObject.addProperty("message", "Hi"); // записываем текст в поле "message"
JsonObject childObject = new JsonObject(); // создаем объект Place
childObject.addProperty("name", "World!"); // записываем текст в поле "name" у объект Place
rootObject.add("place", childObject); // сохраняем дочерний объект в поле "place"

Gson gson = new Gson();
String json = gson.toJson(rootObject); // генерация json строки
System.out.println(json); // напечатает "{"message":"Hi","place":{"name":"World!"}}"
```

Подробный пример

3) Чтение Jackson

Чтение json с помощью Jackson

```
        ObjectMapper mapper = new ObjectMapper();
        JsonNode rootNode = mapper.readValue("{\"message\":\"Hi\",\"place\":{\"name\":\"World!\"}}", JsonNode.class); // парсинг т
екста
        String message = rootNode.get("message").asText(); // получение строки из поля "message"
        JsonNode childNode =  rootNode.get("place"); // получаем объект Place
        String place = childNode.get("name").asText(); // получаем строку из поля "name"
        System.out.println(message + " " + place); // напечатает "Hi World!"
```

Подробный пример

4) Генерация Jackson

Генерация json с помощью Jackson

```
        OutputStream outputStream = new ByteArrayOutputStream();

        ObjectMapper mapper = new ObjectMapper();
        ObjectNode rootNode = mapper.createObjectNode(); // создание главного объекта
        rootNode.put("message", "Hi");
        ObjectNode childNode = rootNode.putObject("place"); // создание дочернего объекта Place
        childNode.put("name", "World!");
        mapper.writeValue(outputStream, childNode); // запись json строки в стрим

        System.out.println(outputStream.toString()); // напечатает "{"message":"Hi","place":{"name":"World!"}}"
```

Подробный пример

5) Чтение и генерация Json Java
И Json Java (эталонная реализация от разработчиков стандарта json), который использует JSONObject

Чтение и генерация json с помощью Json Java

```
        // convert Java to json
        JSONObject root = new JSONObject(); // создаем главный объект
        root.put("message", "Hi");
        JSONObject place = new JSONObject(); // создаем объект Place
        place.put("name", "World!");
        root.put("place", place); // сохраняем объект Place в поле place
        String json = root.toString();
        System.out.println(json); // напечатает "{"message":"Hi","place":{"name":"World!"}}"

        System.out.println();
        // convert json to Java
        JSONObject jsonObject = new JSONObject(json);  // парсинг json
        String message = jsonObject.getString("message");
        String name = jsonObject.getJSONObject("place").getString("name");
        System.out.println(message + " " + name); // напечатает "Hi World!"
```

Подробный пример

В общем, можно заметить что во всех библиотеках выполняются примерно те же действия, отличаются только названия классов.

1.4 Простейшие примеры использование Streaming API

Для демонстрации работы библиотек будем использовать все тот же json:

```
jsonString =
{
  "message": "Hi",
  "place": {
    "name": "World"
  }
}
```

Обычно Streaming API используется крайне редко, только в задачах требующих очень высокой производительности или при очень больших файлах.

Методы парсинга json'а:

Действие	Gson	Jackson
Инициализация	-	`new JsonFactory()`
Парсинг json'а	`reader = new JsonReader((<input_stream>)`	`parser = jsonFactory.createParser(<строка>)`
Проверка есть ли ещё токены	`reader.hasNext()`	`parser.hasCurrentToken()`
Получение типа токена	`reader.peek()`	`parser.nextToken()`

Получение следующего токена	`reader.nextString()`
	`reader.beginObject()`
	`reader.endObject()` и т.п.
		`parser.nextToken()`

Пропуск токена	`reader.skipValue()`	`parser.nextToken()`
Получение строки	`reader.nextString()`	`parser.getText()`

Методы генерации json'а:

Действие	Gson	Jackson
Инициализация	writer = new JsonWriter(<output_stream>)	`generator = new JsonFactory().createGenerator(<output_stream>, <кодировка>)`
Токен начала объекта	`writer.beginObject()`	`generator.writeStartObject()`
Токен окончания объекта	`writer.endObject()`	`generator.writeEndObject()`
Токен имени поля	`writer.name(<имя>)`	`generator.writeFieldName(<имя>)`
Токен строкового значения	`writer.value(<строка>)`	`generator.writeStringField(<имя>, <строка>)`

Примеры:

1) Чтение в Gson

Чтение json с помощью Gson

```
String str = "{\"message\":\"Hi\",\"place\":{\"name\":\"World!\"}}";
InputStream in = new ByteArrayInputStream(str.getBytes(Charset.forName("UTF-8")));
JsonReader reader = new JsonReader(new InputStreamReader(in, "UTF-8"));
while (reader.hasNext()) { // обходим все токены
    JsonToken jsonToken = reader.peek(); // получаем тип следующего токена
    if(jsonToken == JsonToken.BEGIN_OBJECT) { // если начало объекта
        reader.beginObject();
    } else if(jsonToken == JsonToken.END_OBJECT) { // если конец объекта
        reader.endObject();
    } if(jsonToken == JsonToken.STRING) { // в случае если токен строковое знание - выводим на экран
        System.out.print(reader.nextString() + " "); // напечатает Hi World!
    } else {
        reader.skipValue(); // пропускаем все прочие токены
    }
}
reader.close();
```

Подробный пример

2) Запись в Gson

Генерация json с помощью Gson

```
OutputStream outputStream = new ByteArrayOutputStream();
JsonWriter writer = new JsonWriter(new OutputStreamWriter(outputStream, "UTF-8"));
writer.beginObject(); // создаем токен начала главного объекта
writer.name("message"); // записываем поле message
writer.value("Hi");
writer.name("place"); // сохраняем объект Place в поле place
writer.beginObject(); // начинаем объект Place
writer.name("name");
writer.value("World!");
writer.endObject(); // закрываем объект Place
writer.endObject(); // закрываем главный объект
writer.close();
System.out.println(outputStream.toString()); // напечатает "{"message":"Hi","place":{"name":"World!"}}"
```

Подробный пример

3) Чтение в Jackson

Чтение json с помощью Jackson

```
JsonFactory jsonFactory = new JsonFactory();
JsonParser jsonParser = jsonFactory.createParser("{\"message\":\"Hi\",\"place\":{\"name\":\"World!\"}}");
JsonToken jsonToken = jsonParser.nextToken();
while(jsonParser.hasCurrentToken()) { // обходим токены
    if(jsonToken == VALUE_STRING) { // в случае если токен строковое знание - выводим на экран
        System.out.print(jsonParser.getText() + " "); // напечатает "Hi World!"
    }
    jsonToken = jsonParser.nextToken();
}
```

Подробный пример

2) Запись в Jackson

Генерация json с помощью Jackson

```
JsonFactory jsonFactory = new JsonFactory();
OutputStream outputStream = new ByteArrayOutputStream();
JsonGenerator jsonGenerator = jsonFactory.createGenerator(outputStream, JsonEncoding.UTF8);
jsonGenerator.writeStartObject(); // создаем токен начала главного объекта
jsonGenerator.writeStringField("message", "Hi"); // создаем поле message
jsonGenerator.writeFieldName("place");
jsonGenerator.writeStartObject(); // начинаем объект Place
jsonGenerator.writeStringField("name", "World!");
jsonGenerator.writeEndObject(); // закрываем объект Place
jsonGenerator.writeEndObject(); // закрываем главный объект
jsonGenerator.close();
System.out.println(outputStream.toString()); // напечатает "{"message":"Hi","place":{"name":"World!"}}"
```

Подробный пример

1.4 Использование аналогов XPath для json

Методы:

Действие	JsonPath	FastJson
Получение значения по фильтру	`JsonPath.read(<json>, <шаблон>)`	`JSONPath.eval(<java_объект>, <шаблон>)`
Получение коллекции по фильтру	`JsonPath.read(<json>, <шаблон>)`	`JSONPath.eval(<java_объект>, <шаблон>)`

Давайте посмотрим примеры, будем использовать все тот же json

```
jsonString =
{
  "message": "Hi",
  "place": {
    "name": "World"
  }
}
```

▾ С помощью JsonPath

```
        String jsonHiWorld = "{\"message\":\"Hi\",\"place\":{\"name\":\"World!\"}}\"";

        String message = JsonPath.read(jsonHiWorld, "$.message");
        String place = JsonPath.read(jsonHiWorld, "$.place.name");
        System.out.println(message + " " + place); // напечатает "Hi World!"
```

Подробный пример

▾ С помощью FastJson

```
        // преобразования из json'а в Java объекты
        String jsonString = "{\"message\":\"Hi\",\"place\":{\"name\":\"World!\"}}\"";
        Human newHuman = JSON.parseObject(jsonString, Human.class);

        // поиск информации в Java объектах используя eval
        Object message = JSONPath.eval(newHuman, "$.message");
        Object world = JSONPath.eval(newHuman, "$.place.name");
        System.out.println(message + " " + world); // print Hi World
```

Подробный пример

▾ Более сложный пример с JsonPath

```
        List<String> authors = JsonPath.read(json, "$.store.book[*].author");
        System.out.println("authors: " + authors); // print ["Nigel Rees","Evelyn Waugh","Herman Melville","J. R. R. Tolkien"]

        List<Map<String, Object>> expensiveBooks = JsonPath
                .using(Configuration.defaultConfiguration())
                .parse(json)
                .read("$.store.book[?(@.price > 22)].title", List.class);

        System.out.println(expensiveBooks); // print ["Hello, Middle-earth! "]
```

где json это

▸ json =

Подробный пример

2. Генерация Java классов по json схеме и валидация json

Осталось рассмотреть вопросы генерации Java классов и валидации json. Советую посмотреть следующие два online ресурса:

1. jsonschema2pojo.org — ресурс от разработчиков библиотеки jsonschema2pojo, он позволяет из json'а или json схемы сгенерировать соответствующие классы для библиотек Jackson (первой и второй версии) и Gson со всеми аннотациями. Очень удобный ресурс для быстрого использования этих библиотек, достаточно только иметь пример json'а или json схемы.
2. json-schema-validator.herokuapp.com — ресурс от разработчиков json-schema-validator. Он позволяет проверить json схему, сгенерировать java классы по схеме и т.д.

Давайте рассмотрим варианты использование этих библиотек в Java коде.

Пример генерации Java классов из json'а (используя jsonschema2pojo)

```
        // Init json
        String source = "{\n" +
                "  \"type\":\"object\",\n" +
                "  \"properties\": {\n" +
                "    \"messageHiWorld\": {\n" +
                "      \"type\": \"string\"\n" +
                "    },\n" +
                "    \"bar\": {\n" +
                "      \"type\": \"integer\"\n" +
                "    },\n" +
                "    \"baz\": {\n" +
                "      \"type\": \"boolean\"\n" +
                "    }\n" +
                "  }\n" +
                "}";

        // Init config
        JCodeModel codeModel = new JCodeModel();

        GenerationConfig config = new DefaultGenerationConfig() {
            @Override
            public boolean isGenerateBuilders() { // set config option by overriding method
                return true;
            }
        };

        // Generate Java POJO from json
        SchemaMapper mapper = new SchemaMapper(new RuleFactory(config, new Jackson2Annotator(), new SchemaStore()), new SchemaGene
rator());
        mapper.generate(codeModel, "HelloWorldClass", "com.github.vedenin", source);

        // Save generated class to file
        File directory = new File("helloworlds/3.8-json/jsonschema2pojo/output");
        directory.mkdirs();
        codeModel.build(directory);

        // Show generated class
        File cls = new File("helloworlds/3.8-json/jsonschema2pojo/output/com/github/vedenin/HelloWorldClass.java");
        String codeHelloWorld = Files.toString(cls, Charsets.UTF_8);
        System.out.println(codeHelloWorld);
```

Подробный пример

Пример валидации json файлов соответственно схеме (используя json-schema-validator)

```
        final JsonNode fstabSchema = Utils.loadResource("/fstab.json");
        final JsonNode good = Utils.loadResource("/fstab-good.json");
        final JsonNode bad = Utils.loadResource("/fstab-bad.json");
        final JsonNode bad2 = Utils.loadResource("/fstab-bad2.json");

        final JsonSchemaFactory factory = JsonSchemaFactory.byDefault();

        final JsonSchema schema = factory.getJsonSchema(fstabSchema);

        ProcessingReport report;
```

```
        report = schema.validate(good);
        System.out.println(report);

        report = schema.validate(bad);
        System.out.println(report);

        report = schema.validate(bad2);
        System.out.println(report);
```

Подробный пример

Пример использования maven plugin для генерации классов по схеме json (используя jsonschema2pojo)

1) В maven добавляем следующий код, меняя sourceDirectory (где лежат схемы json) и targetPackage (пакет у сгенерированых классов)

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.jsonschema2pojo</groupId>
                <artifactId>jsonschema2pojo-maven-plugin</artifactId>
                <version>0.4.22</version>
                <configuration>
```
```${basedir}/src/main/resources</sourceDirectory>
```
```
                    <targetPackage>com.github.vedenin</targetPackage>
                </configuration>
                <executions>
                    <execution>
                        <goals>
                            <goal>generate</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
     </build>
```

2) Положить нужные схемы json в sourceDirectory
3) После запуска install maven'a по всем схемам будут сгенерированы Java классы.

Подробный пример

3. Документация

Документация всех библиотек:

JSON парсеры

1. Alibaba Fastjson
2. Gson
3. LoganSquare
4. JSON java
5. Square Moshi
6. Instagram Ig json parser
7. Jackson
8. Genson

Аналог XPath для JSON

1. Jayway JsonPath
2. Alibaba Fastjson

Генерация Java классов из JSON или JSON схемы и JSON валидация

1. Jsonschema2pojo
2. Json schema validator

Все примеры:

1. Alibaba Fastjson
2. Gson
3. LoganSquare
4. JSON java
5. Square Moshi
6. Instagram Ig json parser
7. Jackson
8. Genson
9. Jayway JsonPath
10. Jsonschema2pojo
11. Json schema validator

4. Заключение

Надеюсь вам понравилась эта статья, более подробную информацию о библиотеках и примерах кода можно найти на github'e. Версию на английском языке можно найти здесь, обновляемая версия на русском будет на github'e.

Помощь проекту:
Буду благодарен как за добавления новых полезных ссылок в список библиотек, так за добавления Hello world примеров, и за исправления русской и английской грамматики в статьях (см. подробнее тут). Буду благодарен так же за любые замечания и добавления.

Общее оглавление 'Шпаргалок'

1. JPA и Hibernate в вопросах и ответах
2. Триста пятьдесят самых популярных не мобильных Java opensource проектов на github
3. Коллекции в Java (стандартные, guava, apache, trove, gs-collections и другие)
4. Java Stream API
5. Двести пятьдесят русскоязычных обучающих видео докладов и лекций о Java
6. Список полезных ссылок для Java программиста
7 Типовые задачи
   7.1 Оптимальный путь преобразования InputStream в строку
   7.2 Самый производительный способ обхода Map'ы, подсчет количества вхождений подстроки
8. Библиотеки для работы с Json (Gson, Fastjson, LoganSquare, Jackson, JsonPath и другие)

Следующую статью мне хотелось бы по теме...

- [ ] Bean Mapping
- [ ] Functional Programming
- [ ] Reactive Programming
- [ ] Code generation and changing byte code
- [ ] Machine Learning
- [ ] Natural Language Processing (NLP)
- [ ] Другую тему (в комментариях)
- [ ] Не имеет значения
- [ ] Все плохо, не нужно больше статей

Проголосовало 166 человек. Воздержалось 52 человека.

Только зарегистрированные пользователи могут участвовать в опросе. Войдите, пожалуйста.

java, json, jackson, fastjson, genson, gson, ig json parser, json java, json path, json schema validator, jsonschema2pojo, logansquare, moshi

+17 26,8k 334

Автор: @vedenin1980

рейтинг
Luxoft 106,62
Сайт Twitter

Похожие публикации

+29 Шпаргалка Java-программиста 5. Двести пятьдесят русскоязычных обучающих видео докладов и лекций о Java
96,8k 1205 26

+25 Шпаргалка Java программиста 4. Java Stream API
116k 727 17

+57 Шпаргалка Java программиста 3. Коллекции в Java (стандартные, guava, apache, trove, gs-collections и другие)
105k 1126 39

Комментарии (14)

jamakasi666 27 апреля 2016 в 00:51 +2

Статья хорошая, но лично мне больше понравился json-simple от гугла, действительно маленький и без свистоперделок.

sdwvit 27 апреля 2016 в 01:36 +1

Кстати да, json-simple действительно легкий, но судя по мейвену давно не обновлялся

jamakasi666 27 апреля 2016 в 07:57 +1

Скажу честно, и не надо обновлять, не встречал в нем багов, работает стабильно как стальной лом без сюрпризов.

vedenin1980 27 апреля 2016 в 12:17 +1

Ок, чуть позже добавлю json-simple

f1avalanche 27 апреля 2016 в 09:42 +2

Возможно вам ещё больше понравится minimal-json github.com/ralfstx/minimal-json

delargo 27 апреля 2016 в 09:42 +1

В статье сильно не хватает сравнения по производительности библиотек.

vedenin1980 27 апреля 2016 в 09:45 *(комментарий был изменён)* +1

Да, но с производительностью все сложнее, нужно придумать какие-то усредненные примеры и попытаться их замерить. Учитывая, что в зависимости от входных данных производительность библиотек может сильно отличаться это не так просто, можно получить ситуацию когда в замерах будет одни результаты, а в реальном использовании совсем другие.

JuliaRakitina 27 апреля 2016 в 09:42 +2

Отличный проэкт, спасибо. Как раз то, чего так не хвататает: коротко и понятно.

Throwable 27 апреля 2016 в 11:34 +2

Забыли упомянуть Eclipselink MOXy. Реализует стандарт JAXB, но может вместо XML сериализовать в JSON.
http://www.eclipse.org/eclipselink/documentation/2.4/moxy/json003.htm

vedenin1980 27 апреля 2016 в 12:17 +1

Спасибо добавлю

tamakio 27 апреля 2016 в 19:46 0

А есть где-то толковое сравнение производительности Jackson and json?

jamakasi666 27 апреля 2016 в 21:34 0

частично можно глянуть на гитхабе minimal-json, там есть пара сравнений.

vedenin1980 27 апреля 2016 в 21:39 *(комментарий был изменён)* 0

Ну, например в LoganSquare есть сравнение производительности на главной странице. Понятно что в пользу LoganSquare (впрочем как библиотека генерирующая классы вместе рефлексии она все равно должна была выиграть по-идее), но производительность Jackson и Gson относительно друг друга наверное показана более-менее верно. В целом, Gson вроде бы чуть побыстрее. Но на самом деле, если нужна производительность можно взять тот же LoganSquare или использовать не data bind.

basnopisets 8 июля 2016 в 14:48 0

Не хватает небольшого уточнения: хотя LoganSquare использует Data Bind с кодогенерацией, но внутри используется Jackson Streaming API

Только зарегистрированные пользователи могут оставлять комментарии. Войдите, пожалуйста.

Самое читаемое

Разработка

Сейчас Сутки Неделя Месяц

+242 Как Skype уязвимости чинил
23,4k 100 134

+20 Улучшение производительности PHP 7
2,4k 25 4

+77 Здравствуй, дорогой Мегафон
9k 15 79

+9 Прототип RFC HTTP-кодов состояния для ошибок разработчиков (диапазон 7XX)
1,5k 4 6

+12 Компания Google представила набор тестов Wycheproof
1,3k 7 0

Интересные публикации

- Гейзенбаг: Версия 1.0 0
- Компания Google представила набор тестов Wycheproof 0
- Улучшение производительности PHP 7 3
- Защищенный Dell 4
- Преобразование формы представления данных при помощи Excel+PowerQuery 0