Библиотеки для работы с Json (Gson, Fastjson, LoganSquare, Jackson(вроде встроен в SpringBoot), JsonPath и другие)

+ **json-simple** от гугла, действительно маленький и без свистоперделок. Давно не обновлялся но скажу честно, и не надо обновлять, не встречал в нем багов, работает стабильно как стальной лом без сюрпризов.

Поставил ссылку так как там очень много скрытых примеров

<https://habr.com/ru/company/luxoft/blog/280782/>

Глобально выходит вот это:

Существует три основных способа сериализации и десериализации среди указанных библиотек (*от самого простого к самому сложному*) и один дополнительный(*очень легко получить часть информации из json, но нельзя сформировать json*):

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.

1. **Tree Model** — данный парсер представляет json в виде Java классов таких как Node или JsonElement c 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.

1. **Streaming API** — самый низкоуровневый способ, по сути программист сам вручную разбирает токены json'a. Зато никаких ограничений по памяти и в теории максимальная производительность.  
   **Плюсы**: производительность и минимальное потребление памяти,  
   **Минусы**: сложность использования,
2. **Аналоги XPath** — позволяет очень легко получать часть информации из json'а ( позволяет написав выражение $.store.book[\*].author и получить список всех авторов всех книг из json'a магазина). Не очень подходит, если нужно получит всю информацию из json'a + нельзя сформировать json.

**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.** Часто возникает потребность работы с Json, в частности его чтения и парсинга(анализа). В Java обычно ты знаешь с каким типом переменных работаешь, а при парсинге Json смущает то, что тип полей может быть любой.

Какие есть способы разбора Json? Как это делать?

Вот, допустим, как достать данные из Json, представленного ниже?

{

"firstName": "Json",

"lastName": "Smith",

"age": 30,

"address": {

"streetAddress": "666 1nd Street",

"city": "New York",

"state": "NY",

"postalCode": 10021

},

"phoneNumbers": [

{

"type": "home",

"number": "542 666-1234"

},

{

"type": "fax",

"number": "653 666-4567"

}

],

"friends": [

{

"firstName": "Test",

"lastName": "Snow",

"age": 20,

"phoneNumbers": [

{

"type": "home",

"number": "141 111-1234"

}

],

"friends": [

{

"firstName": "UnknownFirstName",

"lastName": "UnknownLastName",

"age": 999,

"phoneNumbers": [

{

"type": "home",

"number": "000 000-0000"

}

]

}

]

},

{

"firstName": "Flash",

"lastName": "Tompson",

"age": 23,

"phoneNumbers": [

{

"type": "home",

"number": "999 111-1234"

}

]

}

]

}

## 

## 1 ответ

Достать данные можно разными способами и, конечно, зависит от задач. Попробую рассмотреть некоторые варианты разбора Json.

**Заметка:** для каждого из примеров для парсинга будет взят Json из вопроса, чтобы зря не копировать в ответ.

# Simple Json

**Где взять:** [здесь](http://www.java2s.com/Code/Jar/j/json-simple.htm) / [репозиторий на github](https://github.com/fangyidong/json-simple) / или через Maven и пр.

Это самый примитивный способ. По сути, всё, что тут есть - это JSONObject и JSONArray.

* JSONArray может включать в себя несколько объектов JSONObject, его можно обходить циклом на каждой итерации получая объект JSONObject.
* JSONObject - объект, из которого можно доставать его отдельные свойства.

Я бы использовал его для небольших Json строк, где не надо сильно заморачиваться или если не лень писать свой класс-обработчик на основе кода, который продемонстрирован ниже:

// Считываем json

Object obj = new JSONParser().parse(jsonString); // Object obj = new JSONParser().parse(new FileReader("JSONExample.json"));

// Кастим obj в JSONObject

JSONObject jo = (JSONObject) obj;

// Достаём firstName and lastName

String firstName = (String) jo.get("firstName");

String lastName = (String) jo.get("lastName");

System.out.println("fio: " + firstName + " " + lastName);

// Достаем массив номеров

JSONArray phoneNumbersArr = (JSONArray) jo.get("phoneNumbers");

Iterator phonesItr = phoneNumbersArr.iterator();

System.out.println("phoneNumbers:");

// Выводим в цикле данные массива

while (phonesItr.hasNext()) {

JSONObject test = (JSONObject) phonesItr.next();

System.out.println("- type: " + test.get("type") + ", phone: " + test.get("number"));

}

Остальная работа с вложенными массивами аналогична. Можно складывать в List, Map и пр.

## GSON

**Где взять:** [здесь](http://www.java2s.com/Code/Jar/g/gson.htm) / [репозиторий на github](https://github.com/google/gson) / или через Maven и пр.

**Документация:** <http://www.studytrails.com/java/json/java-google-json-introduction/>

Позволяет парсить Json также, как и Json-simple, т.е. используя  JSONObject  и  JSONArray (см. [документацию](http://www.studytrails.com/java/json/java-google-json-introduction/)), но имеет более мощный инструмент парсинга. Достаточно создать классы, которые повторяют структуру Json'а. Для парсинга Json из вопроса создадим классы:

class Person {

public String firstName;

public String lastName;

public int age;

public Address address;

public List<Phones> phoneNumbers;

public List<Person> friends;

}

class Address {

public String streetAddress;

public String city;

public String state;

public int postalCode;

}

class Phones {

public String type;

public String number;

}

Теперь достаточно написать:

Gson g = new Gson();

Person person = g.fromJson(jsonString, Person.class);

Всё! Магия! Чудо! Теперь в person лежит объект с типом Person, в котором находятся данные именно с теми типами, которые были указаны в созданных классах! Теперь можно работать с любым типом, как это привыкли всегда делать: String, Integer, List, Map и всё остальное.

// Выведет фамилии всех друзей с их телефонами

for (Person friend : person.friends) {

System.out.print(friend.lastName);

for (Phones phone : friend.phoneNumbers) {

System.out.println(" - phone type: " + phone.type + ", phone number : " + phone.number);

}

}

// output:

// Snow - phone type: home, phone number : 141 111-1234

// Tompson - phone type: home, phone number : 999 111-1234

**Пример парсинга в Map:**

 JSON для разбора:

{

"2":{

"sessions":[

{

"time":"13:00",

"price":"410"

},

{

"time":"06:40",

"price":"340"

},

{

"time":"16:50",

"price":"370"

}

],

"name":"Кинокис-L",

"locate":"Москва, Садовая-Спасская ул., 21, 56",

"metro":"Красные ворота"

},

"7":{

"sessions":[

{

"time":"06:35",

"price":"190"

},

{

"time":"00:05",

"price":"410"

}

],

"name":"Кинокис-V",

"locate":"Павелецкая пл., 2, строение 1",

"metro":"Павелецкая"

},

"8":{

"sessions":[

{

"time":"15:10",

"price":"330"

}

],

"name":"Кинокис-J",

"locate":"ул. Пречистенка, 40/2",

"metro":"Кропоткинская"

},

"9":{

"sessions":[

{

"time":"13:00",

"price":"600"

},

{

"time":"08:30",

"price":"300"

},

{

"time":"04:00",

"price":"510"

},

{

"time":"13:15",

"price":"340"

}

],

"name":"Кинокис-U",

"locate":"Шарикоподшипниковская ул., 24",

"metro":"Дубровка"

}

}

...... Классы (POJO):

class Seance {

public String name;

public String locate

public String metro;

public List<Sessions> sessions;

}

class Sessions {

public String time;

public double price;

}

...... Сам разбор выглядит так:

Gson g = new Gson();

Type type = new TypeToken<Map<String, Seance>>(){}.getType();

Map<String, Seance> myMap = g.fromJson(json, type);

Всё.

Дополнительно в GSON можно использовать аннотации, например: исключить указанные поля при парсинге, поменять имя переменной (например не personFirstName, а fName) и многое другое. Подробнее см. в [документации](https://github.com/google/gson/blob/master/UserGuide.md).

## Jackson(вроде встроен в SpringBoot)

**Где взять:** [здесь](http://www.java2s.com/Code/Jar/j/jackson-all.htm) / [репозиторий на github](https://github.com/FasterXML/jackson-databind/) / или через Maven и пр.

**Документация и примеры:** <https://github.com/FasterXML/jackson-docs>

Как и GSON он также позволяет работать используя JSONObject и JSONArray если это требуется, и тоже умеет парсить на основе предоставленных классов (см. пример ниже).

Аналогично в нем можно указывать дополнительные требования за счет аннотаций, например: не парсить указанные поля, использовать кастомный конструктор класса, поменять имя переменной (например не firstName, а fName) и многое другое. Подробнее см. в [документации](https://github.com/FasterXML/jackson-docs).

ObjectMapper mapper = new ObjectMapper();

Person person = mapper.readValue(jsonString, Person.class);

System.out.println("My fio: " + person.firstName + " " + person.lastName + " and my friends are: ");

for (Person friend : person.friends) {

System.out.print(friend.lastName);

for (Phones phone : friend.phoneNumbers) {

System.out.println(" - phone type: " + phone.type + ", phone number : " + phone.number);

}

}

// output:

// My fio: Json Smith and my friends are:

// Snow - phone type: home, phone number : 141 111-1234

// Tompson - phone type: home, phone number : 999 111-1234

## JsonPath

**Где взять:** через Maven и другие сборщики / [репозиторий на github](https://github.com/json-path/JsonPath)

Относится к так называемым **XPath** библиотекам. Её суть аналогична xpath в xml, то есть **очень легко получать часть информации из json'а,** по указанному пути (см. *https://testerslittlehelper.wordpress.com/2019/01/20/jsonpath-in-rest-assured/*). А также позволяет фильтровать по условию.

// Выведет все фамилии друзей

List<String> friendsLastnames = JsonPath.read(jsonString, "$.friends[\*].lastName");

for (String lastname : friendsLastnames) {

System.out.println(lastname);

}

// output:

// Snow

// Tompson

Пример с выборкой по условию:

// Поиск друга, которому больше 22 лет

List<String> friendsWithAges = JsonPath

.using(Configuration.defaultConfiguration())

.parse(jsonString)

.read("$.friends[?(@.age > 22)].lastName", List.class);

for (String lastname : friendsWithAges) {

System.out.println(lastname);

}

// output:

// Tompson

**2.** Привет, у меня есть константы класса, которые содержат только статические константы класса variables.public

public class Constant

{

public static class A

{

public static class B

{

public static final int COLUMN = 0;

public static final String TYPE = ColumnType.INPUT;

}

}

}

Есть ли способ преобразовать этот класс в JSON ? Я использовал GSON, но, по-видимому, он игнорирует статические переменные. Так как же я могу это сделать ?

Ответ.

Вы можете использовать класс GsonBuilder с методом [excludeFieldsWithModifiers.](http://google-gson.googlecode.com/svn/tags/1.2.3/docs/javadocs/com/google/gson/GsonBuilder.html" \l "excludeFieldsWithModifiers%28int...%29)

GsonBuilder gsonBuilder = new GsonBuilder();

// Разрешение сериализации статических полей

gsonBuilder.excludeFieldsWithModifiers(java.lang.reflect.Modifier.TRANSIENT);

// Создает экземпляр Gson на основе текущей конфигурации

Gson gson = gsonBuilder.create();

String json = gson.toJson(objectToSerialize);

System.out.println(json);