

json

1 概述

JSON(JavaScript Object Notation), JS 对象表示法, 是一种轻量级的数据交换格式

在数据传递中, 使用XML具有很好的优势, 可以跨平台、跨语言, 还可以对数据进行很详细的描述, 有较好的可读性。但是在XML解析的过程, 会变的较为麻烦, 并且很多时候效率还很低。

使用Json格式的数据进行传输, 就可以很好的解决XML传输中所出现的问题, 因为Json 提供了更好的简单性和灵活性。

注意, JSON在JavaScript中, 只是用来表示对象的一种方法

例如,

```
1 var obj = {name:"tom", age:20};
```

对象中, key的表示不需要双引号, value值的表示需要双引号, 但是value值如果是数字, 则不需要双引号

这种对象可以很容易的解析, 获取其中的key和value

```
1 console.log(obj.name);  
2 console.log(obj.age);
```

2 转换

2.1 JavaScript

在JavaScript中, 将JSON对象转为字符串, 将字符串转为JSON对象

例如, JSON对象转为字符串

```
1 var jsonObj = {name:"tom", age:20};  
2 var jsonStr = JSON.stringify(jsonObj);
```

例如, 字符串转为JSON对象

```
1 var jsonStr = "{ \"name\": \"tom\", \"age\": 20 }";
2 var jsonObj = JSON.parse(jsonStr);
```

注意，JSON格式的字符串中，key和value都需要双引号（一定不能使用单引号），如果value是数组可以不用双引号

例如，JSON数组对象转为字符串

```
1 var jsonArr = [];
2 var obj1 = {name: "tom1", age: 21};
3 var obj2 = {name: "tom2", age: 22};
4 jsonArr.push(obj1);
5 jsonArr.push(obj2);
6 console.log(JSON.stringify(jsonArr));
```

输出的内容为：

```
1 [{"name": "tom1", "age": 21}, {"name": "tom2", "age": 22}]
```

格式化后：

```
1  [
2    {
3      "name": "tom1",
4      "age": 21
5    },
6    {
7      "name": "tom2",
8      "age": 22
9    }
10 ]
```

例如，字符串转换JSON数组对象

```
1 var jsonArrStr = "[{ \"name\": \"tom1\", \"age\": 21 }, { \"name\": \"tom2\", \"age\": 22 }]";
2
3 var jsonArr = JSON.parse(jsonArrStr);
4
5 console.log(jsonArr);
```

输出的结果为：



过滤输出

▼ (2) [...]

```
▶ 0: Object { name: "tom1", age: 21 }
▶ 1: Object { name: "tom2", age: 22 }
  length: 2
▶ <prototype>: Array []
```

注意1，将对象转为字符串的目的是为了，在请求或响应中方便携带数据

注意2，将字符串转为对象的目的是为了，在JavaScript或java中方便解析

2.2 java

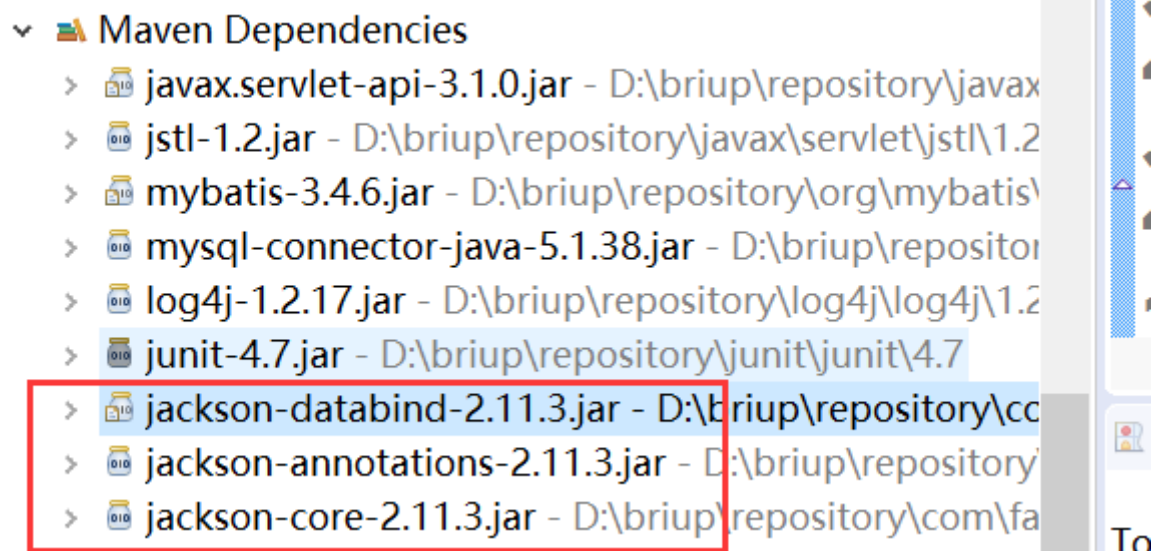
在Java中，将JSON对象转为字符串，将字符串转为JSON对象

在java中，完成这个操作，也很容易，有很多第三方jar包都提供了该功能，例如 jackson

1、项目中引入依赖

```
1 <dependency>
2   <groupId>com.fasterxml.jackson.core</groupId>
3   <artifactId>jackson-databind</artifactId>
4   <version>2.11.3</version>
5 </dependency>
```

如图：



2、代码中使用

以 `User` 类型为例：

```
1 @JsonIgnoreProperties({"password"})
2 public class User{
3
4     private Integer id;
```

```

5     private String name;
6     @JsonFormat(pattern = "yyyy-MM-dd")
7     private Date dob;
8     private String password;
9
10    public User() {}
11
12    public User(Integer id, String name, Date dob, String password) {
13        this.id = id;
14        this.name = name;
15        this.dob = dob;
16        this.password = password;
17    }
18
19    public Integer getId() {
20        return id;
21    }
22    public void setId(Integer id) {
23        this.id = id;
24    }
25    public String getName() {
26        return name;
27    }
28    public void setName(String name) {
29        this.name = name;
30    }
31    public Date getDob() {
32        return dob;
33    }
34    public void setDob(Date dob) {
35        this.dob = dob;
36    }
37
38    public String getPassword() {
39        return password;
40    }
41
42    public void setPassword(String password) {
43        this.password = password;
44    }
45
46    @Override
47    public String toString() {
48        return "User [id=" + id + ", name=" + name + ", dob=" + dob + ",
password=" + password + " ]";
49    }
50
51 }

```

注意，`@JsonFormat(pattern = "yyyy-MM-dd")` 指定该属性转换json格式字符串的时候的日期格式

注意，`@JsonIgnoreProperties({"password"})` 指定改属性在转json格式字符串的时候可以忽略掉

测试代码：

例如，User对象转换json格式字符串

```
1  @Test
2  public void test_obj2json() {
3      ObjectMapper mapper = new ObjectMapper();
4
5      User user = new User(1, "tom", new Date(), "123456");
6
7      try {
8          String jsonStr = mapper.writeValueAsString(user);
9          System.out.println(jsonStr);
10     } catch (JsonProcessingException e) {
11         e.printStackTrace();
12     }
13
14 }
```

输出结果：

```
1  {"id":1,"name":"tom","dob":"2020-10-29"}
```

格式化：

```
1  {
2      "id":1,
3      "name":"tom",
4      "dob":"2020-10-29"
5  }
```

例如，json格式字符串转User对象

```
1  @Test
2  public void test_json2Obj() {
3
4      ObjectMapper mapper = new ObjectMapper();
5
6      String jsonStr = "{\"id\":1,\"name\":\"tom\",\"dob\":\"2022-10-29\"}";
7
8      try {
9          User user = mapper.readValue(jsonStr, User.class);
10         System.out.println(user);
11     } catch (JsonMappingException e) {
12         e.printStackTrace();
13     } catch (JsonProcessingException e) {
14         e.printStackTrace();
15     }
16 }
```

例如，Map<String,String>对象转json格式字符串

```
1  @Test
2  public void test_map2Json_string() {
3      ObjectMapper mapper = new ObjectMapper();
```

```

4
5     Map<String, String> map = new HashMap<String,String>();
6     map.put("name", "tom");
7     map.put("age", "21");
8
9     try {
10         String jsonStr = mapper.writeValueAsString(map);
11         System.out.println(jsonStr);
12     } catch (JsonProcessingException e) {
13         e.printStackTrace();
14     }
15
16 }

```

输出结果：

```

1     {"name":"tom","age":"21"}

```

```

1     {
2         "name":"tom",
3         "age":"21"
4     }

```

例如，json格式字符串转Map<String,String>对象

```

1     @Test
2     public void test_json2Map_string() {
3         ObjectMapper mapper = new ObjectMapper();
4
5         String jsonStr = "{\"name\":\"tom\",\"age\":\"21\"}";
6
7         try {
8             @SuppressWarnings("unchecked")
9             Map<String,String> map = mapper.readValue(jsonStr, Map.class);
10            System.out.println(map);
11
12        } catch (JsonMappingException e) {
13            e.printStackTrace();
14        } catch (JsonProcessingException e) {
15            e.printStackTrace();
16        }
17
18    }

```

例如，map<String,User>对象转Json格式字符串

```

1     @Test
2     public void test_map2Json_user() {
3         ObjectMapper mapper = new ObjectMapper();
4
5         Map<Integer, User> map = new HashMap<>();
6         map.put(1, new User(1, "tom1", new Date(), "123456"));
7         map.put(2, new User(2, "tom2", new Date(), "123456"));

```

```

8      map.put(3, new User(3,"tom3",new Date(),"123456"));
9
10     try {
11         String jsonStr = mapper.writeValueAsString(map);
12         System.out.println(jsonStr);
13     } catch (JsonProcessingException e) {
14         e.printStackTrace();
15     }
16
17 }

```

输出结果：

```

1  {"1":{"id":1,"name":"tom1","dob":"2020-10-29"},"2":
   {"id":2,"name":"tom2","dob":"2020-10-29"},"3":{"id":3,"name":"tom3","dob":"2020-
   10-29"}}

```

格式化：

```

1  {
2      "1":{
3          "id":1,
4          "name":"tom1",
5          "dob":"2020-10-29"
6      },
7      "2":{
8          "id":2,
9          "name":"tom2",
10         "dob":"2020-10-29"
11     },
12     "3":{
13         "id":3,
14         "name":"tom3",
15         "dob":"2020-10-29"
16     }
17 }

```

例如，json格式字符串转Map<String,User>对象

```

1  @Test
2  public void test_json2Map_user() {
3
4      ObjectMapper mapper = new ObjectMapper();
5
6      String jsonStr = "{ \"1\": { \"id\": 1, \"name\": \"tom1\", \"dob\": \"2022-10-29\" }, \"2\": { \"id\": 2, \"name\": \"tom2\", \"dob\": \"2022-10-29\" }, \"3\": { \"id\": 3, \"name\": \"tom3\", \"dob\": \"2022-10-29\" } }";
7
8      try {
9          //指定要解析转换java类型是Map，已经Map中的俩个泛型的类型分别是String和用户

```

```

10         JavaType javaType =
            mapper.getTypeFactory().constructParametricType(Map.class,
String.class, User.class);
11         Map<String, User> map = mapper.readValue(jsonStr, javaType);
12         for(User user:map.values()) {
13             System.out.println(user);
14         }
15     } catch (JsonMappingException e) {
16         e.printStackTrace();
17     } catch (JsonProcessingException e) {
18         e.printStackTrace();
19     }
20
21 }

```

例如，List<String> 对象转Json格式字符串

```

1  @Test
2  public void test_list2Json_string() {
3
4      ObjectMapper mapper = new ObjectMapper();
5
6      List<String> list = new ArrayList<>();
7
8      list.add("hello");
9      list.add("world");
10     list.add("briup");
11
12     try {
13         String jsonStr = mapper.writeValueAsString(list);
14         System.out.println(jsonStr);
15     } catch (JsonProcessingException e) {
16         e.printStackTrace();
17     }
18
19 }

```

输出结果：

```

1  ["hello","world","briup"]

```

例如，Json格式字符串转List<String>对象

```

1  @Test
2  public void test_json2List_string() {
3
4      ObjectMapper mapper = new ObjectMapper();
5
6      String jsonStr = "[\"hello\",\"world\",\"briup\"]";
7
8      try {
9          @SuppressWarnings("unchecked")
10         List<String> list = mapper.readValue(jsonStr, List.class);

```



```

11     System.out.println(list);
12 } catch (JsonProcessingException e) {
13     e.printStackTrace();
14 }
15
16 }

```

例如，List<User>对象转Json格式字符串

```

1  @Test
2  public void test_list2Json_user() {
3      ObjectMapper mapper = new ObjectMapper();
4
5      List<User> list = new ArrayList<>();
6
7      list.add(new User(1, "tom1", new Date(), "123456"));
8      list.add(new User(2, "tom2", new Date(), "123456"));
9      list.add(new User(3, "tom3", new Date(), "123456"));
10
11     try {
12         String jsonStr = mapper.writeValueAsString(list);
13         System.out.println(jsonStr);
14     } catch (JsonProcessingException e) {
15         e.printStackTrace();
16     }
17
18 }

```

输出结果：

```

1  [{ "id":1, "name":"tom1", "dob":"2020-10-29"}, {"id":2, "name":"tom2", "dob":"2020-10-29"}, {"id":3, "name":"tom3", "dob":"2020-10-29"}]

```

格式化：

```

1  [
2      {
3          "id":1,
4          "name":"tom1",
5          "dob":"2020-10-29"
6      },
7      {
8          "id":2,
9          "name":"tom2",
10         "dob":"2020-10-29"
11     },
12     {
13         "id":3,
14         "name":"tom3",
15         "dob":"2020-10-29"
16     }
17 ]

```

例如，json格式字符串转List<User>对象

```
1  @Test
2  public void test_json2List_user() {
3
4      ObjectMapper mapper = new ObjectMapper();
5
6      String jsonStr = "[{\"id\":1,\"name\":\"tom1\",\"dob\":\"2022-10-29\"},
7      {\"id\":2,\"name\":\"tom2\",\"dob\":\"2022-10-29\"},
8      {\"id\":3,\"name\":\"tom3\",\"dob\":\"2022-10-29\"}]";
9
10     try {
11         JavaType javaType =
12             mapper.getTypeFactory().constructParametricType(List.class, User.class);
13
14         List<String> list = mapper.readValue(jsonStr, javaType);
15
16         System.out.println(list);
17     } catch (JsonProcessingException e) {
18         e.printStackTrace();
19     }
20 }
```