关于JSON的处理

01、JSON是什么？

JSON (JavaScript Object Notation, JS 对象标记) 是一种轻量级的数据交换格式。（就告诉它不是数据类型）它基于 ECMAScript (w3c制定的js规范)的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。 易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

> 总结：*json*它不是数据类型，它是一种有格式化数据结构。

在久远的项目数据交互中：

String传输

存在问题，不方便解析和分割。因为不具有结构性，如果要用户信息的时候：

```
1#yykk#35#广州
```

上面存在的问题：# 是一个分隔符，但是难免你主题内容刚好有# ，这样就会出问题。

xml传输

```xml
<user>
    <id>1</id>
    <name>yykk</name>
    <age>35</age>
</user>
```

好处可想而知：有数据格式的，可以方便的解析。

缺点：xml格式体积比较大，如果xml层级很深，解析真就一个非常痛苦的事情。

json传输

```json
{
    id:1,
    name:"yykk",
    age:35,
    address:"广州"
}
```

但是不论是用那种，网络上数据传递全部都是：本质都是字符串。只不过xml和json是一种有格式的字符串。

02、JSON疑问？

比如：表单注册

```
# 参数
var params = {
    id:1,
    name:"yykk",
    age:35,
    address:"广州"
}

# jquery ajax
$.post("url",params);

```

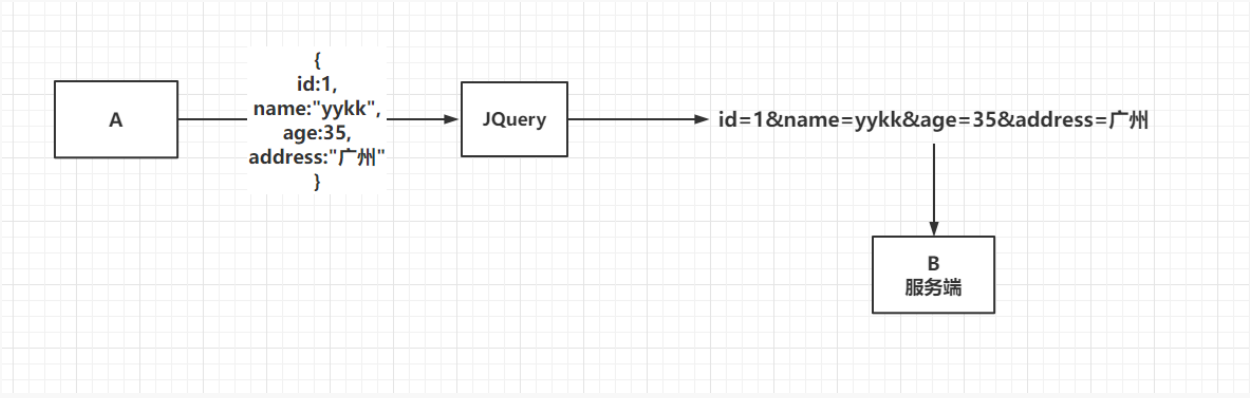
后端

```
public String saveUser(@RequestBody User user){
    System.out.println(user);// params
}
```

本质：

- 服务端根本就不认识上面json数据结构。也就输送传递数据参数不是传递的json格式，
- json数据格式，只是为了让我们程序开发人员比较方便进行参数的传递以及方便js库方便进行解析。

原理



03、为什么要这样做呢？

- 因为如果直接让你写参数，其实很容易写错，经常漏写& 或者=号。各种问题。
- 因为json数据具有结构性，方便控制和解析。
- 就因为前端js已经对这种json数据格式做了支持，所以服务端的语音开始层出不穷推出各种工具将其自身语音的数据结构转化JSON字符串进行返回，方便js进行解析和处理。

04、为什么java要将对象转成json字符串呢？

- 原因1：json本身就js提出来，可以方便解析
- 原因2：java的数据类型是不可能直接传递js取调用认识，毕竟是两门语言，语言与语言之间，内存结构，数据类型不同，存储的方式不同，执行引擎不同，自然而然是不可能互相认识。所以语言之间通信只能通过一个数据格式进行通信：json或者xml或者string
- 原因3：因为前后端的开发中，其实都是语言 ---js进行交互。那么为什么不这样做呢？

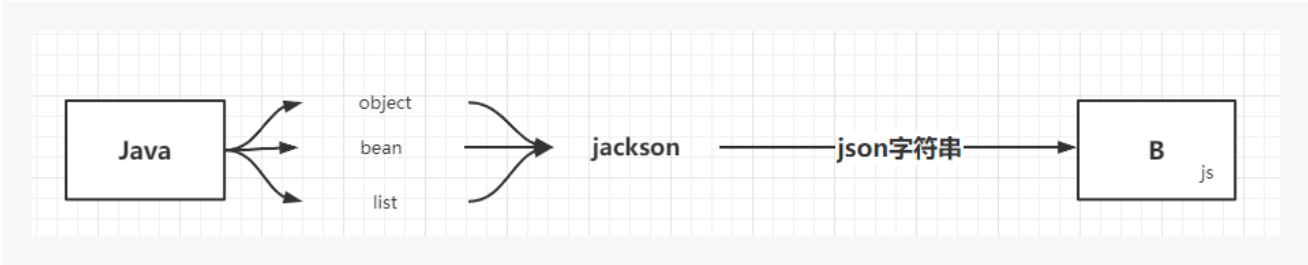
05、JS（ajax） + 语言Java

springmvc框架提供了如下json工具支持：

- jackson(spring)
- fastjson(阿里)
- gson (谷歌)

06、jackson

springmvc框架默认就使用jackson进行数据类型转化：



请注意：json处理不包括基础数据类型，封装数据类型 和 String。

实现步骤：

首先定义一个User实体

```
package com.kuangstudy.bean;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;

/**
 * @author 飞哥
 * @Title: 学相伴出品
 * @Description: 飞哥B站地址：https://space.bilibili.com/490711252
 * 记得关注和三连哦!
 * @Description: 我们有一个学习网站：https://www.kuangstudy.com
 * @date 2021/12/22 10:51
 */
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User {
    private Long id;
    private String nickname;
    private String password;
    private Integer age;
    private String address;
```

```
    private Integer status; // 1 发布 0未发布
    private Date creataTime;
}

```

1：引入依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

2：配置

```
spring:
  jackson:
    date-format: yyyy-MM-dd HH:mm
    time-zone: GMT+8
    locale: zh_CN
    generator:
      write-numbers-as-strings: true
      write-bigdecimal-as-plain: true
```

底层有一个配置类：JacksonAutoConfiguration 和 JacksonProperties 进行ObjectMapper的初始化。

> *ObjectMapper是jackson专门来讲对象转化成json字符串的一个类。*
>
> 比如案例：
>
> ```
> package com.kuangstudy.util;
>
> import com.fasterxml.jackson.databind.ObjectMapper;
> import com.kuangstudy.bean.User;
> ```

```
5
6  import java.util.Date;
7
8  /**
9   * @author 飞哥
10  * @Title: 学相伴出品
11  * @Description: 飞哥B站地址:
    https://space.bilibili.com/490711252
12  * 记得关注和三连哦!
13  * @Description: 我们有一个学习网站:
    https://www.kuangstudy.com
14  * @date 2021/12/23 22:25
15  */
16 public class TestJson {
17
18     public static void main(String[] args) throws
    Exception{
19
20         User user = new User();
21         user.setId(1317503462556848129L);
22         user.setAges(23);
23         user.setPassword("123456");
24         user.setNickname("geely@happymmall.com");
25         user.setCreataTime(new Date());
26
27         // jackson转化器ObjectMapper
28         ObjectMapper objectMapper = new ObjectMapper();
29         String userJson =
    objectMapper.writeValueAsString(user);
30
31         System.out.println(userJson);
32
33
34     }
35 }
36
```

结果:

> *{"id":1317503462556848129,"nickname":"geely@happymmall.com","password":"123456","ages":23,"address":null,"creataTime":1640269602171}*

只不过springmvc和springboot已经将这个初始化ObjectMapper和转换对象成json字符串的过程全部进行封装。让程序看上去是一个自动的过程。

分析

```
{
    "nickname": "geely@happymmall.com",
    "password": "123456",
    "ages": 23,
    "address": null,
    "id": 1317503462556848000,
    "creataTime": "2021-12-23T14:31:19.950+00:00"
}
```

- 第一个问题：时间没格式化
- 第二个问题：id因为是一个long型，而jackson默认是以integer来进行转换，会造成精度丢失。

3：定义访问接口进行测试

```
package com.kuangstudy.web;

import com.kuangstudy.bean.User;
import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RestController;

import java.util.Date;

/**
 * @author 飞哥
 * @Title: 学相伴出品
```

```
 * @Description: 飞哥B站地址：https://space.bilibili.com/490711252
 * 记得关注和三连哦!
 * @Description: 我们有一个学习网站：https://www.kuangstudy.com
 * @date 2021/12/22 10:56
 */
@RestController
public class JSONController {

    @GetMapping("/user/json1")
    public User json1() {
        User user = new User();
        user.setId(1317503462556848129L);
        user.setAges(23);
        user.setPassword("123456");
        user.setNickname("geely@happymmall.com");
        user.setCreataTime(new Date());
        return user;
    }
}

```

然后访问：http://localhost:8088/user/json1

4：结果

```
{
    "nickname": "geely@happymmall.com",
    "password": "123456",
    "ages": "23",
    "address": null,
    "id": "1317503462556848129",
    "creataTime": "2021-12-23 22:35"
}
```

得到最终json的格式，没有问题。

5：观察json格式

```
{
    "id": "1317503462556848129",
    "nickname": "geely@happymmall.com",
    "password": "123456",
    "age": "23",
    "address": null,
    "status": "1",
    "creataTime": "2021-12-23 22:39"
}
```

- age : "23" status:"1" 都编程了字符串。其实问题不大，但是在未来的vue开发中，vue的开关和判断处理对数据类型是很敏感的。vue的组件如果需要一个数字是1是开，如果你这个时候传递是 :"1"就不会生效。

6：解决long的转字符串影响其他数据类型的问题

单独对long处理，覆盖内部的配置：

```
package com.kuangstudy.config;

import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.module.SimpleModule;
import
com.fasterxml.jackson.databind.ser.std.ToStringSerializer;
import org.springframework.boot.jackson.JsonComponent;
import org.springframework.context.annotation.Bean;
import
org.springframework.http.converter.json.Jackson2ObjectMapperB
uilder;

import java.text.SimpleDateFormat;
import java.util.Locale;
import java.util.SimpleTimeZone;

```

```java
@JsonComponent
public class JsonSerializeConfiguration {

    @Bean
    public ObjectMapper jacksonObjectMapper(Jackson2ObjectMapperBuilder builder) {
        ObjectMapper objectMapper = builder.createXmlMapper(false).build();
        //忽略value为null 时 key的输出
        // JsonInclude.Include.ALWAYS 所以属性都必须在
        // JsonInclude.Include.NON_NULL 如果你属性是null，就会自动剔除
        // JsonInclude.Include.NON_EMPTY 如果你属性是""，就会自动剔除

 objectMapper.setSerializationInclusion(JsonInclude.Include.ALWAYS);
        objectMapper.setDateFormat(new SimpleDateFormat("yyyy-MM-dd HH:mm:ss"));

 objectMapper.setTimeZone(SimpleTimeZone.getDefault());
        objectMapper.setLocale(new Locale("zh_CN"));
        SimpleModule module = new SimpleModule();
        // 对long单独处理
        module.addSerializer(Long.class, ToStringSerializer.instance);
        module.addSerializer(Long.TYPE, ToStringSerializer.instance);
        objectMapper.registerModule(module);
        return objectMapper;
    }
}


```

注意：覆盖以后全局配置文件依然可以使用。 建议做集中管理，全部用配置类。

07、关于json的字段为null到底要不要返回的问题？

建议是一定返回：保证数据格式的完整性。

都统一使用：

```
objectMapper.setSerializationInclusion(JsonInclude.Include.ALWAYS);
```

08、如果返回的字段null太多了怎么办?

在开发中，有时候你们写SQL查询就所有。但是往往我们在开发中，页面获取数据列就那么几列。如何按需处理

- 每个业务写自己的SQL语句，写需要明确的字段。不允许使用(推荐)

  ```
  select id,title,description,nickname,avatar from bbs
  ```

  如果是mp(mybatis-plus)。可以过来查询

  ```
  QueryWrapper<bbs> queryWrapper = new QueryWrapper<>();
  queryWrapper.select("id","title","description","nickname","avatar");
  ```

- 可以先查询所有，然后用stream做筛选--然后用新的vo进行封装返回

```
select * from bbs
List<User> users = [];

# stream流
List<UserVo> userVos = users.stream().map(user->{
    UserVo uservo = new UserVo();
    BeanUtils.copy(user,uservo);
    return uservo;
}).collect();

List<UserVo> userVos =
copyWithCollection(users,UserVo.class);
```

09、A服务 + B服务

未来的程序开发，可能会脱离springmvc 。

A 架构：dubbo ----------JSONUtil.java----User---UserJson---------- B架构：Go服务

```
package com.kuangstudy.util;

import com.fasterxml.jackson.annotation.JsonInclude;
import
com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.JavaType;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.kuangstudy.bean.User;
import lombok.extern.slf4j.Slf4j;
import org.springframework.util.StringUtils;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
```

```
15 import java.util.List;
16 import java.util.Map;
17
18
19 @Slf4j
20 public class JsonUtil {
21
22     private static ObjectMapper objectMapper = new
   ObjectMapper();
23
24     static {
25         // 对象的所有字段全部列入
26
    objectMapper.setSerializationInclusion(JsonInclude.Include
   .ALWAYS);
27         // 取消默认转换timestamps形式
28
    objectMapper.configure(SerializationFeature.WRITE_DATE_KEY
   S_AS_TIMESTAMPS, false);
29         // 设置long类型的精度问题
30
    objectMapper.configure(SerializationFeature.WRITE_BIGDECIM
   AL_AS_PLAIN, false);
31         // 忽略空Bean转json的错误
32
    objectMapper.configure(SerializationFeature.FAIL_ON_EMPTY_
   BEANS, false);
33         // 所有的日期格式都统一为以下的样式，即yyyy-MM-dd HH:mm:ss
34         objectMapper.setDateFormat(new
   SimpleDateFormat("yyyy-MM-dd HH:mm:ss"));
35         // 忽略 在json字符串中存在，但是在java对象中不存在对应属性的
   情况。防止错误
36
    objectMapper.configure(DeserializationFeature.FAIL_ON_UNKN
   OWN_PROPERTIES, false);
37         // 精度的转换问题
```

```
38
    objectMapper.configure(DeserializationFeature.USE_BIG_DECIMAL_FOR_FLOATS, true);
39
40
    objectMapper.configure(DeserializationFeature.ACCEPT_SINGLE_VALUE_AS_ARRAY, true);
41      }
42
43      public static <T> String obj2String(T obj) {
44          if (obj == null) {
45              return null;
46          }
47          try {
48              return obj instanceof String ? (String) obj : objectMapper.writeValueAsString(obj);
49          } catch (Exception e) {
50              log.warn("Parse Object to String error", e);
51              return null;
52          }
53      }
54
55      // 这个在开发中不允许使用，只能在平时的测试和学习上使用。
56      // 为什么不用：因为格式化会增加换行符，会增加json体积和大小，
57      public static <T> String obj2StringPretty(T obj) {
58          if (obj == null) {
59              return null;
60          }
61          try {
62              return obj instanceof String ? (String) obj
63                      : objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(obj);
64          } catch (Exception e) {
65              log.warn("Parse Object to String error", e);
66              return null;
67          }
68      }
```

```
69
70      public static <T> T string2Obj(String str, Class<T>
    clazz) {
71          if (StringUtils.isEmpty(str) || clazz == null) {
72              return null;
73          }
74
75          try {
76              return clazz.equals(String.class) ? (T) str :
    objectMapper.readValue(str, clazz);
77          } catch (Exception e) {
78              log.warn("Parse String to Object error", e);
79              return null;
80          }
81      }
82
83
84      public static <T> T string2Obj(String str, Class<?>
    collectionClass, Class<?>... elementClasses) {
85          JavaType javaType =
    objectMapper.getTypeFactory().constructParametricType(colle
    ctionClass, elementClasses);
86          try {
87              return objectMapper.readValue(str, javaType);
88          } catch (Exception e) {
89              log.warn("Parse String to Object error", e);
90              return null;
91          }
92      }
93
94      public static void main(String[] args) {
95
96          User user = new User();
97          user.setId(13175048173422059 54L);
98          user.setAge(23);
99          user.setPassword("123456");
100         user.setNickname("geely@happymmall.com");
101         user.setCreataTime(new Date());
```

```
102
103
104 //        String userJsonPretty =
    JsonUtil.obj2String(user);
105 //        log.info("userJson:{}", userJsonPretty);
106 //        User user2 = JsonUtil.string2Obj(userJsonPretty,
    User.class);
107 //        log.info("user2:{}", user2);
108
109
110         List<User> userList = new ArrayList<>();
111         userList.add(user);
112         userList.add(user);
113         userList.add(user);
114         String obj2String =
    JsonUtil.obj2StringPretty(userList);
115         System.out.println(obj2String);
116
117
118         List<User> userList1 =
    JsonUtil.string2Obj(obj2String, ArrayList.class,
    User.class);
119         System.out.println(userList1);
120
121         List<Map<String,Object>> userList2 =
    JsonUtil.string2Obj(obj2String, ArrayList.class,
    Map.class);
122         System.out.println(userList2);
123
124
125     }
126
127 }
```