基于原生 JSON 封装解析与生成工具

作者：pencilso
原文链接：https://ld246.com/article/1493036721796
来源网站：链滴

将JavaBean 封装成JSON格式的字符串

封装JSON数据的代码片段

```
    List listUser = new ArrayList<>(); //创建一个List集合

    Map mapUser = new HashMap<>(); //创建一个Map集合

    mapUser.put("map-key", new UserBean(4, "map-1", 10, 2.456789, null,null, null)); //添加一map的元素

    listUser.add(new UserBean(3, "list-1", 17, 1.456, null, null, null));//添加一个list的元素

    UserBean[] arrayUser = { new UserBean(2, "array-1", 16, 10.0123456,null, null, null) }; //创一个对象数组并添加一个元素

    UserBean us = new UserBean(1, "大白", 18, 3.230, listUser, mapUser,arrayUser); //将list map 对象数组 都设置到JavaBean当中

    Pson pson = Pson.Instance(); //初始化工具类
    /**
     * 使用工具进行封装JavaBean中的所有字段
     */
    String jsonStr = pson.objectToJson(us);
    System.out.println(jsonStr);
    /**
     * 使用工具进行封装JavaBean中的ID 与名字
     */
    jsonStr = pson.objectToJson(us,"id","name");
    System.out.println(jsonStr);
```

封装JSON格式后的数据

```
//封装所有的字段
{"id":1,"name":"大白","age":18,"money":3.23,"listUser":[{"id":3,"name":"list-1","age":17,"money"1.456}],"mapUser":{"map-key":{"id":4,"name":"map-1","age":10,"money":2.456789}},"arrayUser"[{"id":2,"name":"array-1","age":16,"money":10.0123456}]}

//封装指定的字段
{"id":1,"name":"大白"}
```

将JSON格式字符串解析成Object实体类

首先准备两个JSON格式的数据 一个为对象 一个为数组

JSON-1: String jsonStr = {"id":1,"name":"大白","age":18,"money":1.0453}

JSON-2: String jsonArray = [{"id":1,"name":"大白","age":18,"money":2.0453},{"id":1,"name":"大","age":18,"money":1.0453}]

解析对象

```
    Pson pson = Pson.Instance();
    /**
     * 调用工具类进行解析 一参为被解析的类的class对象 二参为JSON格式的字符串数据
     */
    UserBean ub = pson.jsonToObject(UserBean.class, jsonStr);
    System.out.println(ub);

    //解析完毕后打印的ub对象
    UserBean [id=1, name=大白, age=18, money=1.0453, listUser=null, mapUser=null, arrayUs
r=null]
```

解析数组

```
    UserBean users[] = {}; // 首先先初始化一个空的数组
    users = pson.jsonToObject(users.getClass(), jsonArray);
    for (UserBean u : users) {
        System.out.println(u);
    }

    //解析完毕后打印的数组元素
    UserBean [id=1, name=大白1, age=18, money=1.0453, listUser=null, mapUser=null, arrayU
er=null]
    UserBean [id=2, name=大白2, age=18, money=2.0453, listUser=null, mapUser=null, arrayU
er=null]
```

JSON工具类

使用工具类前需要有原生org.json包 前往下载

之前因为不太喜欢第三方的东西 虽然知道有Gson这类的第三方工具

所以自己动手写了一个工具

```
public class Pson {
    private String pattern = "yyyy-MM-dd HH:mm:ss";//时间格式

    public void setPattern(String pattern) {
        this.pattern = pattern;
    }

    public static final Pson pson = new Pson();

    public static Pson Instance() {
        return pson;
    }

    private Pson() {

    }

    /**
```

```
     * 将对象 转换成JSON格式的字符串
     *
     * @param object    需要封装的对象
     * @param fieldName 需要封装的指定字段 如果为空 则封装所有字段
     * @return 返回封装后的JSON格式的字符串
     */
    public Object objectToJson(Object object, String... fieldName) {
        return ObjectToString(object, fieldName);
    }

    /**
     * 将对象 转换成JSON格式的字符串
     *
     * @param object
     * @param fieldName 需要封装指定的字段
     * @return 接收到之后 调用toString即可
     */
    private Object ObjectToString(Object object, String... fieldName) {
        if (object == null) {
            return null;
        } else if (object instanceof List) { // 判断是否是List集合
            List list = (List) object;
            JSONArray array = new JSONArray();
            if (list.size() != 0) {
                Object oneObject = list.get(0);// 先取出第一个元素 然后判断是否是基本数据类型
                // 或者String类型
                if (toBasicData(oneObject)) { // 元素是基本数据类型
                    for (Object element : list) {
                        array.put(element);
                    }
                } else {
                    for (Object element : list) {
                        array.put(ObjectToString(element, fieldName));
                    }
                }
            }
            return array;
        } else if (object.getClass().isArray()) {
            Object oArray[] = (Object[]) object;
            JSONArray jsonArray = new JSONArray();
            /**
             * 判断数组长度
             */
            if (oArray.length != 0) {
                if (toBasicData(oArray[0])) { // 判断数组的元素是基本数据 还是对象
                    for (Object elementObject : oArray) {
                        jsonArray.put(elementObject);
                    }
                } else {
                    for (Object elementObject : oArray) {
                        jsonArray.put(ObjectToString(elementObject, fieldName));
                    }
                }
            }
```

```
    return jsonArray;
} else if (object instanceof Map) {
    Map map = (Map) object;
    Set> entries = map.entrySet();
    JSONObject jsonObject = new JSONObject();
    if (fieldName.length != 0) {//判断是否需要解析指定的字段
        for (String keyName :
                fieldName) {
            jsonPutMap(jsonObject, keyName, map.get(keyName), fieldName);
        }
    } else {
        /**
         * 遍历MAP集合
         */
        for (Map.Entry entry : entries) {
            jsonPutMap(jsonObject, entry.getKey(), entry.getValue(), fieldName);
        }
    }
    return jsonObject;
} else if (object instanceof Date) { //时间类
    return new SimpleDateFormat(pattern).format((Date) object);
}
JSONObject json = new JSONObject();
Class c = object.getClass();
Field fs[] = null;
if (fieldName != null && fieldName.length != 0) { // 判断是否需要 取出指定的字段
    int len = fieldName.length;
    fs = new Field[len];
    int i = 0;
    for (String s : fieldName) {
        try {
            fs[i] = c.getDeclaredField(s);
        } catch (NoSuchFieldException | SecurityException e) {
            e.printStackTrace();
        }
        i++;
    }
} else {
    fs = c.getDeclaredFields();
}

for (Field f : fs) {
    if (f == null)
        continue;
    f.setAccessible(true);
    try {
        Object o = f.get(object);
        if (o != null) { // 判断属性内容是否为空
            if (toBasicData(o)) { // 判断是否是基本数据类型 或者是String类型
                json.put(f.getName(), o);
            } else { // 可能是一个JavaBean 或者是集合
                json.put(f.getName(), ObjectToString(o, fieldName));
            }
        } else {
```

```
                json.put(f.getName(), o);
            }
        } catch (IllegalArgumentException | IllegalAccessException e) {
            e.printStackTrace();
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
    return json;
}

/**
 * 判断是否是字符串 或者是 基本数据类型
 *
 * @param object
 * @return 基本数据类型|字符串类型 返回true
 */
private boolean toBasicData(Object object) {
    if (object == null) {
        return false;
    } else if (object instanceof String || object instanceof Integer || object instanceof Boolean
            || object instanceof Character || object instanceof Byte || object instanceof Short
            || object instanceof Long || object instanceof Float || object instanceof Double) {
        return true;
    }
    return false;
}

/**
 * 判断是否是字符串 或者是 基本数据类型
 *
 * @param oc
 * @return 基本数据类型|字符串类型 返回true
 */
private boolean toBasicData(Class oc) {
    if (oc.isPrimitive() || oc.isAssignableFrom(String.class) || oc.isAssignableFrom(Integer.class)
            || oc.isAssignableFrom(Boolean.class) || oc.isAssignableFrom(Character.class)
            || oc.isAssignableFrom(Byte.class) || oc.isAssignableFrom(Short.class)
            || oc.isAssignableFrom(Long.class) || oc.isAssignableFrom(Float.class)
            || oc.isAssignableFrom(Double.class)) {
        return true;
    }
    return false;
}

/**
 * 将JSON转JavaBean对象实例
 *
 * @param oClass  JavaBean的Class对象
 * @param jsonStr JSON对象
 * @param
 * @return
 */
public  T jsonToObject(Class oClass, String jsonStr) {
```

```
        T objectData = null;
        try {
            if (oClass.isArray()) {// 该class为数组的时候
                JSONArray array = new JSONArray(jsonStr);
                int len = array.length();
                // 获取到数组元素的class
                Class ComponentType = oClass.getComponentType();
                Object createArray = Array.newInstance(ComponentType, array.length());// 创建一
与JSON数组长度相同的数组
                if (toBasicData(ComponentType.newInstance())) { // 进行判断是否是基本数据类型或
符串类型
                    for (int i = 0; i < len; i++) {
                        Array.set(createArray, i, array.get(i));
                    }
                } else {
                    for (int i = 0; i < len; i++) {
                        Array.set(createArray, i, jsonToObject(ComponentType, array.get(i).toString()));
                    }
                }
                objectData = (T) createArray;
            } else if (oClass.isAssignableFrom(Date.class)) {//时间类型
                Date parse = new SimpleDateFormat(pattern).parse(jsonStr);
                objectData = (T) parse;
            } else {// 如果不是数组的话 则默认以JavaBean的形式进行解析
                objectData = (T) oClass.newInstance(); // 创建该class的对象的实例
                JSONObject json = new JSONObject(jsonStr);
                Field[] fields = oClass.getDeclaredFields();// 获取到所有的属性
                for (Field fs : fields) {// 开始遍历属性
                    fs.setAccessible(true);
                    Class cl = fs.getType(); // 取出该属性代表的class
                    String fieldName = fs.getName(); // 取出该属性的名称
                    /**
                     * 判断该属性名字在json中不为空
                     */
                    if (!json.isNull(fieldName)) {
                        String jsonData = json.get(fieldName).toString();

                        /**
                         * 判断 属性字段是否是数组类型
                         */
                        if (cl.isArray()) {
                            fs.set(objectData, analyticArray(cl, jsonData));
                        } else if (toBasicData(cl)) { // 基本数据类型
                            fs.set(objectData, json.get(fieldName));
                        } else if (cl.isInterface()) { // 接口
                            Object oj = analyticInterface(cl, (ParameterizedType) fs.getGenericType(), jso
Data);
                            fs.set(objectData, oj);
                        } else if (cl.isAssignableFrom(ArrayList.class)) { // 判断该成员属性是否为ArrayLis
集合
                            Object oj = analyticInterface(List.class, (ParameterizedType) fs.getGenericTy
e(),
                                    jsonData);
                            fs.set(objectData, oj);
```

```
                } else if (cl.isAssignableFrom(HashMap.class)) { // 判断该成员属性是否为HashMa
集合
                    Object oj = analyticInterface(Map.class, (ParameterizedType) fs.getGenericT
pe(), jsonData);
                    fs.set(objectData, oj);
                } else { // 如果以上条件都没有成立 则当对象来解析
                    fs.set(objectData, jsonToObject(cl, jsonData));
                }
            }
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
return objectData;
}

/**
 * 解析数组数据
 *
 * @param arrayClass 数组的class对象
 * @param jsonStr    Json格式的字符串数据
 * @return
 */
private Object analyticArray(Class arrayClass, String jsonStr) {
    try {
        // 首先 先获取到数组的子元素的类型class
        Class componentType = arrayClass.getComponentType();
        // 判断子元素类型 是否是基本类型 或者是String类型
        boolean is = toBasicData(componentType.newInstance());
        JSONArray jsonArray = new JSONArray(jsonStr);
        int len = jsonArray.length();
        /**
         * 创建一个与该数组类型一至 长度与JSONArray长度一致的数组
         */
        Object elementObject = Array.newInstance(componentType, len);
        if (is) {
            for (int i = 0; i < len; i++) {
                Array.set(elementObject, i, jsonArray.get(i));
            }
        } else { // 如果不是基本数据类型的话 则调用解析对象的方法
            for (int i = 0; i < len; i++) {
                Array.set(elementObject, i, jsonToObject(componentType, jsonArray.getString(i)));
            }
        }
        return elementObject;
    } catch (Exception e) {
        e.printStackTrace();
    }

    return null;
}

/**
```

```
    * 解析成员属性为接口的方法 只支持Map List 集合接口 其他的 均返回null
    *
    * @param interfaceClass    接口Class对象
    * @param jsonStr           JSON格式的文本
    * @param parameterizedType 泛型的管理
    * @return 返回解析后的实例
    */
   private Object analyticInterface(Class interfaceClass, ParameterizedType parameterizedType
String jsonStr) {
       Object breakObject = null;
       try {
           if (interfaceClass.isAssignableFrom(List.class)) { // List集合
               JSONArray array = new JSONArray(jsonStr);
               int len = array.length();
               Class c = (Class) parameterizedType.getActualTypeArguments()[0];// 获取到List集合
泛型
               List list = new ArrayList<>();
               if (toBasicData(c)) {// 判断List的泛型是否为String 或基本类型
                   for (int i = 0; i < len; i++) {
                       list.add(array.get(i));
                   }
               } else {// 调用解析对象方法
                   for (int i = 0; i < len; i++) {
                       list.add(jsonToObject(c, array.getString(i)));
                   }
               }
               breakObject = list;
           } else if (interfaceClass.isAssignableFrom(Map.class)) {
               /**
                * 如果是map集合的话 是会有两个泛型的 对应的为 key value
                */
               Class k = (Class) parameterizedType.getActualTypeArguments()[0];
               Class v = (Class) parameterizedType.getActualTypeArguments()[1];
               if (!k.isAssignableFrom(String.class)) {
                   throw new ClassCastException("key 请用String类型");
               } else {
                   Map map = new HashMap<>();
                   JSONObject json = new JSONObject(jsonStr);
                   Iterator iterator = json.keys();
                   if (toBasicData(v)) {// 判断value是否为基本类型或String类型
                       while (iterator.hasNext()) {
                           String key = iterator.next();
                           map.put(key, json.get(key));
                       }
                   } else {
                       while (iterator.hasNext()) {
                           String key = iterator.next();
                           map.put(key, jsonToObject(v, json.getString(key)));
                       }
                   }
                   breakObject = map;
               }
           }
       } catch (Exception e) {
```

```
            e.printStackTrace();
        }
        return breakObject;
    }

    /**
     * JSON put map的key values
     *
     * @param jsonObject
     * @param key
     * @param values
     * @param fieldName
     */
    private void jsonPutMap(JSONObject jsonObject, String key, Object values, String... fieldN
me) {
        try {
            if (toBasicData(values)) {// 判断是否是基本数据类型
                jsonObject.put(key, values);
            } else {
                jsonObject.put(key, ObjectToString(values, fieldName));
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}
```