|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **Json API** | |  | |  | |
| |  | | --- | |  | | |  | | --- | | view plaincopy to clipboardprint? 01.package com.baiyyy.polabs.util.json;  02.  03.import java.text.ParseException;  04.import java.util.ArrayList;  05.import java.util.Date;  06.import java.util.HashMap;  07.import java.util.Iterator;  08.import java.util.List;  09.import java.util.Map;  10.  11.import com.baiyyy.polabs.util.DateUtil;  12.  13.import net.sf.json.JSONArray;  14.import net.sf.json.JSONObject;  15.import net.sf.json.JsonConfig;  16.import net.sf.json.util.CycleDetectionStrategy;  17.  18./\*\*  19. \* 处理json的工具类，负责json数据转换成java对象和java对象转换成json  20. \*  21. \* @author yongtree  22. \* @date 2008-11-22 上午10:47:13  23. \* @version 1.0  24. \*/  25.public class JsonUtil {  26.  27. /\*\*  28. \* 从一个JSON 对象字符格式中得到一个java对象  29. \*  30. \* @param jsonString  31. \* @param pojoCalss  32. \* @return  33. \*/  34. public static Object getObject4JsonString(String jsonString, Class pojoCalss) {  35. Object pojo;  36. JSONObject jsonObject = JSONObject.fromObject(jsonString);  37. pojo = JSONObject.toBean(jsonObject, pojoCalss);  38. return pojo;  39. }  40.  41.  42. /\*\*  43. \* 从json HASH表达式中获取一个map，改map支持嵌套功能  44. \*  45. \* @param jsonString  46. \* @return  47. \*/  48. public static Map getMap4Json(String jsonString) {  49. JSONObject jsonObject = JSONObject.fromObject(jsonString);  50. Iterator keyIter = jsonObject.keys();  51. String key;  52. Object value;  53. Map valueMap = new HashMap();  54.  55. while (keyIter.hasNext()) {  56. key = (String) keyIter.next();  57. value = jsonObject.get(key);  58. valueMap.put(key, value);  59. }  60.  61. return valueMap;  62. }  63.  64.  65. /\*\*  66. \* 从json数组中得到相应java数组  67. \*  68. \* @param jsonString  69. \* @return  70. \*/  71. public static Object[] getObjectArray4Json(String jsonString) {  72. JSONArray jsonArray = JSONArray.fromObject(jsonString);  73. return jsonArray.toArray();  74.  75. }  76.  77.  78. /\*\*  79. \* 从json对象集合表达式中得到一个java对象列表  80. \*  81. \* @param jsonString  82. \* @param pojoClass  83. \* @return  84. \*/  85. public static List getList4Json(String jsonString, Class pojoClass) {  86.  87. JSONArray jsonArray = JSONArray.fromObject(jsonString);  88. JSONObject jsonObject;  89. Object pojoValue;  90.  91. List list = new ArrayList();  92. for (int i = 0; i < jsonArray.size(); i++) {  93.  94. jsonObject = jsonArray.getJSONObject(i);  95. pojoValue = JSONObject.toBean(jsonObject, pojoClass);  96. list.add(pojoValue);  97.  98. }  99. return list;  100.  101. }  102.  103.  104. /\*\*  105. \* 从json数组中解析出java字符串数组  106. \*  107. \* @param jsonString  108. \* @return  109. \*/  110. public static String[] getStringArray4Json(String jsonString) {  111.  112. JSONArray jsonArray = JSONArray.fromObject(jsonString);  113. String[] stringArray = new String[jsonArray.size()];  114. for (int i = 0; i < jsonArray.size(); i++) {  115. stringArray[i] = jsonArray.getString(i);  116.  117. }  118.  119. return stringArray;  120. }  121.  122.  123. /\*\*  124. \* 从json数组中解析出javaLong型对象数组  125. \*  126. \* @param jsonString  127. \* @return  128. \*/  129. public static Long[] getLongArray4Json(String jsonString) {  130.  131. JSONArray jsonArray = JSONArray.fromObject(jsonString);  132. Long[] longArray = new Long[jsonArray.size()];  133. for (int i = 0; i < jsonArray.size(); i++) {  134. longArray[i] = jsonArray.getLong(i);  135.  136. }  137. return longArray;  138. }  139.  140.  141. /\*\*  142. \* 从json数组中解析出java Integer型对象数组  143. \*  144. \* @param jsonString  145. \* @return  146. \*/  147. public static Integer[] getIntegerArray4Json(String jsonString) {  148.  149. JSONArray jsonArray = JSONArray.fromObject(jsonString);  150. Integer[] integerArray = new Integer[jsonArray.size()];  151. for (int i = 0; i < jsonArray.size(); i++) {  152. integerArray[i] = jsonArray.getInt(i);  153.  154. }  155. return integerArray;  156. }  157.  158. /\*\*  159. \* 从json数组中解析出java Date 型对象数组，使用本方法必须保证  160. \*  161. \* @param jsonString  162. \* @return  163. \* @throws ParseException  164. \*/  165. public static Date[] getDateArray4Json(String jsonString, String DataFormat)  166. throws ParseException {  167.  168. JSONArray jsonArray = JSONArray.fromObject(jsonString);  169. Date[] dateArray = new Date[jsonArray.size()];  170. String dateString;  171. Date date;  172.  173. for (int i = 0; i < jsonArray.size(); i++) {  174. dateString = jsonArray.getString(i);  175. date = DateUtil.parseDate(dateString, DataFormat);  176. dateArray[i] = date;  177.  178. }  179. return dateArray;  180. }  181.  182.  183. /\*\*  184. \* 从json数组中解析出java Integer型对象数组  185. \*  186. \* @param jsonString  187. \* @return  188. \*/  189. public static Double[] getDoubleArray4Json(String jsonString) {  190.  191. JSONArray jsonArray = JSONArray.fromObject(jsonString);  192. Double[] doubleArray = new Double[jsonArray.size()];  193. for (int i = 0; i < jsonArray.size(); i++) {  194. doubleArray[i] = jsonArray.getDouble(i);  195.  196. }  197. return doubleArray;  198. }  199.  200.  201. /\*\*  202. \* 将java对象转换成json字符串  203. \*  204. \* @param javaObj  205. \* @return  206. \*/  207. public static String getJsonString4JavaPOJO(Object javaObj) {  208.  209. JSONObject json;  210. json = JSONObject.fromObject(javaObj);  211. return json.toString();  212.  213. }  214.  215. /\*\*  216. \* 将java对象转换成json字符串,并设定日期格式  217. \*  218. \* @param javaObj  219. \* @param dataFormat  220. \* @return  221. \*/  222. public static String getJsonString4JavaPOJO(Object javaObj,  223. String dataFormat) {  224.  225. JSONObject json;  226. JsonConfig jsonConfig = configJson(dataFormat);  227. json = JSONObject.fromObject(javaObj, jsonConfig);  228. return json.toString();  229.  230. }  231.  232. /\*\*  233. \* JSON 时间解析器具  234. \*  235. \* @param datePattern  236. \* @return  237. \*/  238. public static JsonConfig configJson(String datePattern) {  239. JsonConfig jsonConfig = new JsonConfig();  240. jsonConfig.setExcludes(new String[] { "" });  241. jsonConfig.setIgnoreDefaultExcludes(false);  242. jsonConfig.setCycleDetectionStrategy(CycleDetectionStrategy.LENIENT);  243. jsonConfig.registerJsonValueProcessor(Date.class,  244. new JsonDateValueProcessor(datePattern));  245.  246. return jsonConfig;  247. }  248.  249.  250. /\*\*  251. \* 除去不想生成的字段（特别适合去掉级联的对象）+时间转换  252. \* @param excludes 除去不想生成的字段  253. \* @param datePattern  254. \* @return  255. \*/  256. public static JsonConfig configJson(String[] excludes, String datePattern) {  257. JsonConfig jsonConfig = new JsonConfig();  258. jsonConfig.setExcludes(excludes);  259. jsonConfig.setIgnoreDefaultExcludes(true);  260. jsonConfig.setCycleDetectionStrategy(CycleDetectionStrategy.LENIENT);  261. jsonConfig.registerJsonValueProcessor(Date.class,  262. new JsonDateValueProcessor(datePattern));  263.  264. return jsonConfig;  265. }  266.  267.}  268.  269.  270.package com.baiyyy.polabs.util.json;  271.  272.import java.text.SimpleDateFormat;  273.import java.util.Date;  274.  275.import net.sf.json.JsonConfig;  276.import net.sf.json.processors.JsonValueProcessor;  277.  278./\*\*  279. \*  280. \* @author yongtree  281. \* @date 2008-11-22 上午10:54:19  282. \* @version 1.0  283. \*/  284.public class JsonDateValueProcessor implements JsonValueProcessor {  285.  286. private String format = "yyyy-MM-dd HH:mm:ss";  287.  288. public JsonDateValueProcessor() {  289.  290. }  291.  292. public JsonDateValueProcessor(String format) {  293. this.format = format;  294. }  295.  296. public Object processArrayValue(Object value, JsonConfig jsonConfig) {  297. String[] obj = {};  298. if (value instanceof Date[]) {  299. SimpleDateFormat sf = new SimpleDateFormat(format);  300. Date[] dates = (Date[]) value;  301. obj = new String[dates.length];  302. for (int i = 0; i < dates.length; i++) {  303. obj[i] = sf.format(dates[i]);  304. }  305. }  306. return obj;  307. }  308.  309. public Object processObjectValue(String key, Object value, JsonConfig jsonConfig) {  310. if (value instanceof Date) {  311. String str = new SimpleDateFormat(format).format((Date) value);  312. return str;  313. }  314. return value == null ? null : value.toString();  315. }  316.  317. public String getFormat() {  318. return format;  319. }  320.  321. public void setFormat(String format) {  322. this.format = format;  323. }  324.  325.} | | |