[线程和线程池 1](#_Toc501539738)

[数据库连接池原理 1](#_Toc501539739)

[四类线程池 2](#_Toc501539740)

[线程安全，预防脏读 3](#_Toc501539741)

[加密算法 5](#_Toc501539742)

[类的属性反射 7](#_Toc501539743)

[文件读取 8](#_Toc501539744)

[FastJson 13](#_Toc501539745)

[Json字符串转 map 对象 13](#_Toc501539746)

[Json字符串转 List 集合 13](#_Toc501539747)

[复杂JSON字符串 转 List集合 14](#_Toc501539748)

[解析JSON字符串到指定对象class 15](#_Toc501539749)

[Map对象转化成json字符串 16](#_Toc501539750)

[JackJson 16](#_Toc501539751)

[指定对象Class转成 json字符串 17](#_Toc501539752)

[List集合转化成json字符串 17](#_Toc501539753)

[Json字符串转化成指定Class类 18](#_Toc501539754)

[Json字符串转化成集合List 18](#_Toc501539755)

## 线程和线程池

C3PO数据库连接池

<http://blog.csdn.net/suzunshou/article/details/47014545>

Dbutils基础知识介绍

<https://www.cnblogs.com/CQY1183344265/p/5854418.html>

## 数据库连接池原理

连接池基本的思想是在系统初始化的时候，将数据库连接作为对象存储在内存中，当用户需要访问数据库时，并非建立一个新的连接，而是从连接池中取出一个已建立的空闲连接对象。

使用完毕后，用户也并非将连接关闭，而是将连接放回连接池中，以供下一个请求访问使用。而连接的建立、断开都由连接池自身来管理。

同时，还可以通过设置连接池的参数来控制连接池中的初始连接数、连接的上下限数以及每个连接的最大使用次数、最大空闲时间等等。也可以通过其自身的管理机制来监视数据库连接的数量、使用情况等。

线程的三种实现方式：

1. Thread类，可extends
2. Runable接口，需要实现run方法，通过new Thread类调用
3. callable接口，需要实现call方法，通过 FutureTask 调用，并获取返回值，可结合线程池使用。

## 四类线程池

1. newCachedThreadPool

创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。

这种类型的线程池特点是：

工作线程的创建数量几乎没有限制(其实也有限制的,数目为Interger. MAX\_VALUE), 这样可灵活的往线程池中添加线程。

如果长时间没有往线程池中提交任务，即如果工作线程空闲了指定的时间(默认为1分钟)，则该工作线程将自动终止。终止后，如果你又提交了新的任务，则线程池重新创建一个工作线程。

在使用CachedThreadPool时，一定要注意控制任务的数量，否则，由于大量线程同时运行，很有会造成系统瘫痪。

2、newFixedThreadPool

创建一个指定工作线程数量的线程池。每当提交一个任务就创建一个工作线程，如果工作线程数量达到线程池初始的最大数，则将提交的任务存入到池队列中。

FixedThreadPool是一个典型且优秀的线程池，它具有线程池提高程序效率和节省创建线程时所耗的开销的优点。但是，在线程池空闲时，即线程池中没有可运行任务时，它不会释放工作线程，还会占用一定的系统资源。

3、newSingleThreadExecutor

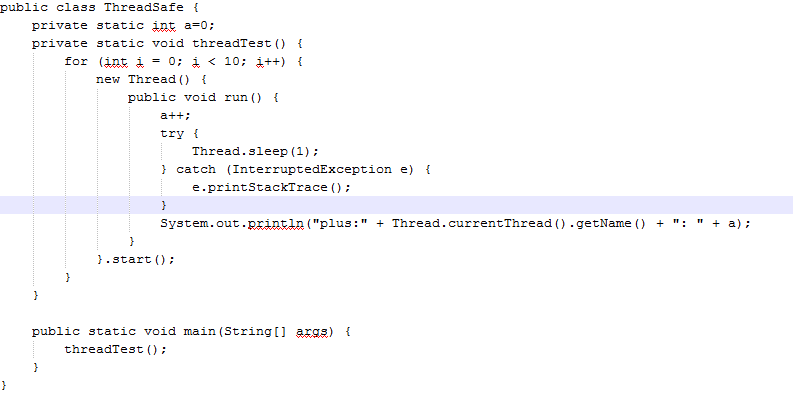
创建一个单线程化的Executor，即只创建唯一的工作者线程来执行任务，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。如果这个线程异常结束，会有另一个取代它，保证顺序执行。单工作线程最大的特点是可保证顺序地执行各个任务，并且在任意给定的时间不会有多个线程是活动的。

4、newScheduleThreadPool

创建一个定长的线程池，而且支持定时的以及周期性的任务执行，支持定时及周期性任务执行。

## 线程安全，预防脏读

线程脏读：



输出：

plus:Thread-5: 4

plus:Thread-3: 4

plus:Thread-0: 4

plus:Thread-4: 5

plus:Thread-8: 5

plus:Thread-9: 6

plus:Thread-7: 9

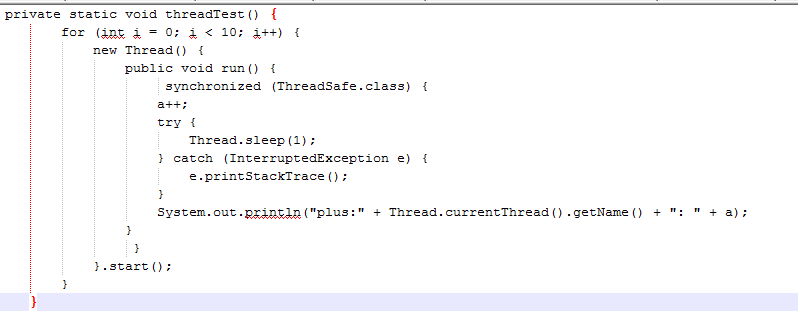
plus:Thread-2: 10

plus:Thread-6: 10

plus:Thread-1: 10

为了防止线程并发时的脏读情况，可采用以下方法避免

1. 对代码块使用synchronized



输出：

plus:Thread-3: 1

plus:Thread-7: 2

plus:Thread-0: 3

plus:Thread-6: 4

plus:Thread-9: 5

plus:Thread-8: 6

plus:Thread-2: 7

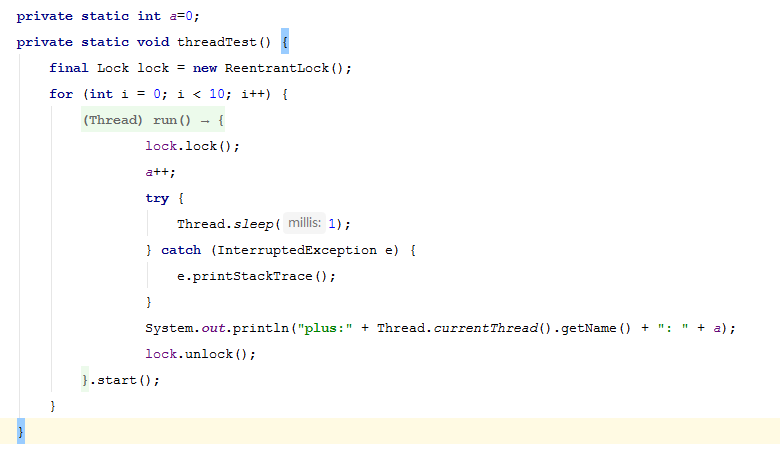
plus:Thread-5: 8

plus:Thread-4: 9

plus:Thread-1: 10

1. 使用ReentrantLock类

Lock()--Unlock()



tryLock()—unlock()



输出：

plus:Thread-0: 5

plus:Thread-6: 8

plus:Thread-1: 13

plus:Thread-2: 14

plus:Thread-4: 15

plus:Thread-8: 16

plus:Thread-5: 17

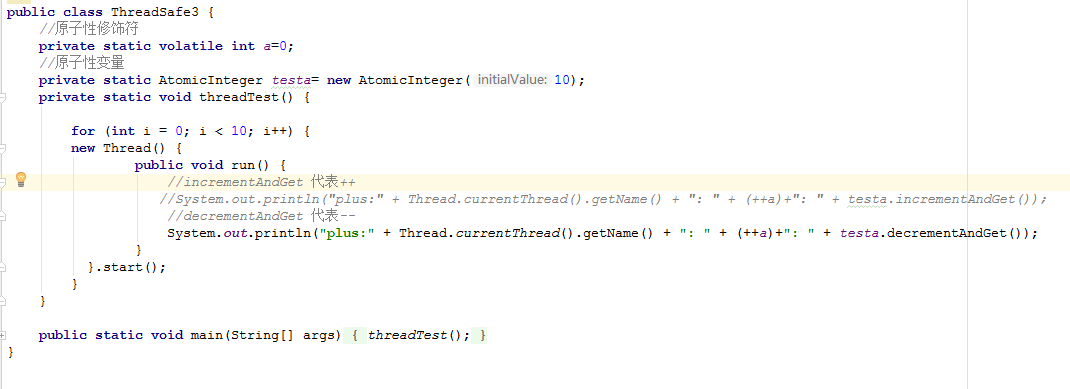
plus:Thread-9: 18

plus:Thread-3: 19

plus:Thread-7: 20

lockInterruptibly()

1. 原子变量内部锁，AtomicInteger只能同步一个值，不能同步代码块



输出：

plus:Thread-1: 1: 9

plus:Thread-0: 2: 8

plus:Thread-3: 4: 6

plus:Thread-5: 3: 7

plus:Thread-2: 5: 5

plus:Thread-8: 6: 4

plus:Thread-9: 7: 3

plus:Thread-7: 8: 2

plus:Thread-4: 9: 1

plus:Thread-6: 10: 0

## 加密算法

1. 直接使用写好的加密算法

public final static String MD5(String pwd) {

//用于加密的字符

char md5String[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9','A', 'B', 'C', 'D', 'E', 'F' };

try {

//使用平台的默认字符集将此 String 编码为 byte序列，并将结果存储到一个新的 byte数组中

byte[] btInput = pwd.getBytes();

//信息摘要是安全的单向哈希函数，它接收任意大小的数据，并输出固定长度的哈希值。

MessageDigest mdInst = MessageDigest.getInstance("MD5");

//MessageDigest对象通过使用 update方法处理数据， 使用指定的byte数组更新摘要

mdInst.update(btInput);

// 摘要更新之后，通过调用digest（）执行哈希计算，获得密文

byte[] md = mdInst.digest();

// 把密文转换成十六进制的字符串形式

int j = md.length;

char str[] = new char[j \* 2];

int k = 0;

for (int i = 0; i < j; i++) { // i = 0

byte byte0 = md[i]; //95

str[k++] = md5String[byte0 >>> 4 & 0xf]; // 5

str[k++] = md5String[byte0 & 0xf]; // F

}

//返回经过加密后的字符串

return new String(str);

} catch (Exception e) {

return null;

}

}

在main方法中调用：

String str4 = "123456";

System.out.println(encodeByMd5(str4));

//输出：4QrcOUm6Wau+VuBX8g+IPg==

1. 使用commons-codec 包下的md5

//使用commons.codec包下DigestUtils的md5Hex方法，对abc进行加密

String str1 = DigestUtils.md5Hex(str);

System.out.println(str1.toUpperCase());

输出：abc转化成大写字母：900150983CD24FB0D6963F7D28E17F72

1. 使用比MD5更安全的 SHA-1加密

String str11 = DigestUtils.sha1Hex(str);

System.out.println(str11);

输出： a9993e364706816aba3e25717850c26c9cd0d89d

1. 使用Base64的decodeBase64、encodeBase64进行加解密

byte[] b = Base64.decodeBase64("YWJj".getBytes());

System.out.println(new String(b));//输出abc

byte[] code = Base64.encodeBase64("123456".getBytes());

System.out.println(new String(code)+"--");//输出：MTIzNDU2

1. 使用安全性更高的Md5Crypt，Md5Crypt基于算子的MD5，以$开头，安全性高，每次加密的结果都不一样。

String str2 = Md5Crypt.md5Crypt("abc".getBytes());

System.out.println(str2);//abc

## 类的属性反射

前提条件：

存在两个person类，有相同的属性

1. 第一种方法：

读取person类的各个属性，及值

public static void getFile(Object object) {

Class clz=object.getClass();

Field[] fields = clz.getDeclaredFields();

for (Field field : fields) {

try {

field.setAccessible(true);

System.out.println(field.getName()+ " value "+ field.get(object));

} catch (IllegalArgumentException | IllegalAccessException e) {

e.printStackTrace();

}

}

}

将反射类属性的方法封装成copy，使用时直接传参：

public static void copy(Object from, Object to) {

Class fromClass= from.getClass();

Class toClass= to.getClass();

Field[] fields = toClass.getDeclaredFields();

for (Field field : fields) {

try {

field.setAccessible(true);

System.out.println(field.getName());

Field fromfield = fromClass.getDeclaredField(field.getName());

fromfield.setAccessible(true);

field.set(to, fromfield.get(from));

} catch (Exception e) {

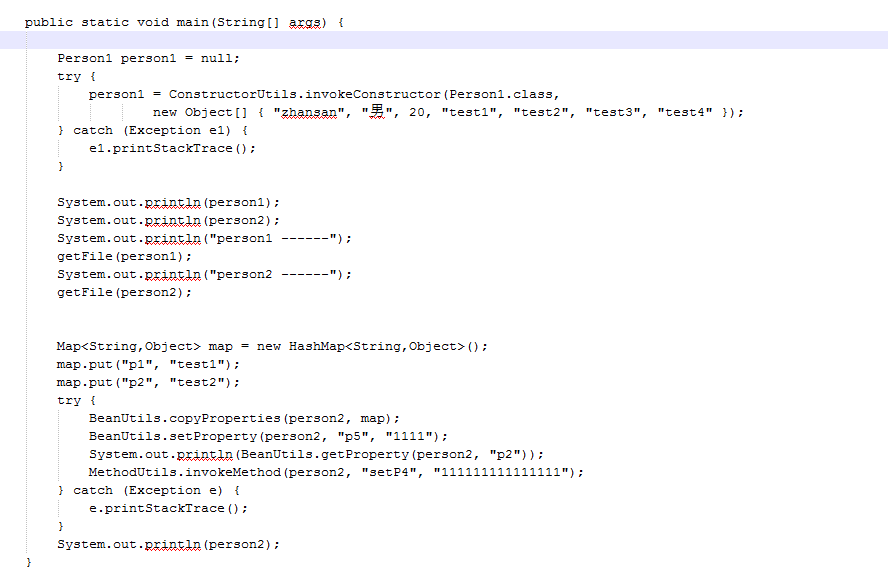
e.printStackTrace();

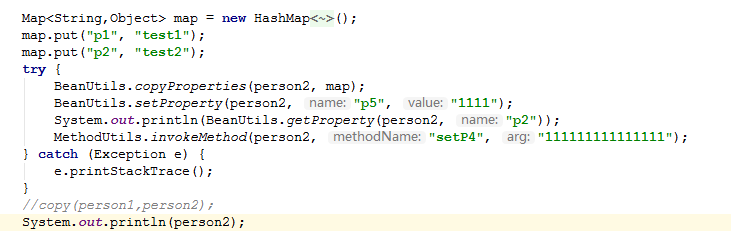
}

}

}

1. 使用commons-beanutils包下的BeanUtils工具类





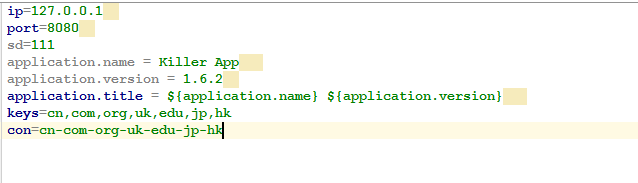
## 文件读取

1. 使用JDK自带的的类读取，Properties工具类



1. 使用commons.configuration包下的PropertiesConfiguration类读取properties文件

Properties文件内容：





*/\*\*   
 \*   
 \* 依次输出结果如下   
 \* 127.0.0.1   
 \* 8080   
 \* Killer App 1.6.2   
 \* defaultid   
 \* [cn, com, org, uk, edu, jp, hk]   
 \* [cn, com, org, uk, edu, jp, hk]   
 \* [cn, com, org, uk, edu, jp, hk]   
 \*/*

1. 使用commons.configuration包下的XMLConfiguration类读取xml文件



*/\*\* 依次输出结果如下：  
 boy: tom  
 student.name: [lily, lucy]  
 title: [abc, cbc, bbc, bbs]  
 ball[@size]: 20  
 student(1)[@id]: 2  
 student(0).name[@go]: common1  
 student.name(0)[@go]: common1  
 \*/*

1. 将对象转化成xml文件，并输出

先创建对象person, @XmlRootElement

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement

public class Person1 {

private String name;

private String sex;

private int age;

private String p1;

private String p2;

private String p3;

private String p4;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getSex() {

return sex;

}

public void setSex(String sex) {

this.sex = sex;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public String getP1() {

return p1;

}

public void setP1(String p1) {

this.p1 = p1;

}

public String getP2() {

return p2;

}

public void setP2(String p2) {

this.p2 = p2;

}

public String getP3() {

return p3;

}

public void setP3(String p3) {

this.p3 = p3;

}

public String getP4() {

return p4;

}

public void setP4(String p4) {

this.p4 = p4;

}

public Person1() {

}

public Person1(String name, String sex, int age, String p1, String p2, String p3, String p4) {

super();

this.name = name;

this.sex = sex;

this.age = age;

this.p1 = p1;

this.p2 = p2;

this.p3 = p3;

this.p4 = p4;

}

@Override

public String toString() {

return "Person1 [name=" + name + ", sex=" + sex + ", age=" + age + ", p1=" + p1 + ", p2=" + p2 + ", p3=" + p3

+ ", p4=" + p4 + "]";

}

}



第5节：

1. HttpUtils 封装 httpget,httppost
2. FastJson，解析JSON
3. JackJson

## Httpclient模拟get请求

**private static** CloseableHttpClient *httpclient*;  
  
**static** {  
 PoolingHttpClientConnectionManager manager = **new** PoolingHttpClientConnectionManager();  
 manager.setMaxTotal(200); *//连接池最大并发连接数* manager.setDefaultMaxPerRoute(200);*//单路由最大并发数,路由是对maxTotal的细分  
 httpclient* = HttpClients.*custom*().setConnectionManager(manager).build();  
}  
  
*/\* ConnectionRequestTimeout httpclient使用连接池来管理连接，这个时间就是从连接池获取连接的超时时间，可以想象下数据库连接池  
 ConnectTimeout 建立连接最大时间  
 SocketTimeout 数据传输过程中数据包之间间隔的最大时间  
 HttpHost 代理  
 \*/***private static** RequestConfig *config* =RequestConfig.*copy*(RequestConfig.***DEFAULT***)  
 .setSocketTimeout(10000)  
 .setConnectTimeout(5000)  
 .setConnectionRequestTimeout(100).build();  
 *// .setProxy(new HttpHost("127.0.0.1",8888,"http")).build();***public static** String doGet(String url, Map<String, Object> header)  
 **throws** HttpClientException {  
 String ret = **""**;  
 HttpGet get = **new** HttpGet(url);  
 get.setConfig(*config*);  
 get.addHeader(HTTP.***CONTENT\_ENCODING***, **"UTF-8"**);  
 CloseableHttpResponse closeableHttpResponse = **null**;  
 **try** {  
 **if** (header != **null**) {  
 **for** (Map.Entry<String, Object> entry : header.**entrySet**()) {  
 get.setHeader(entry.**getKey**(), entry.**getValue**().toString());  
 }  
 }  
 closeableHttpResponse = *httpclient*.execute(get);  
 **if** (closeableHttpResponse.**getStatusLine**().**getStatusCode**() == 200) {  
 ret = EntityUtils.*toString*(closeableHttpResponse.**getEntity**(), **"UTF-8"**);  
 } **else** {  
 **throw new** HttpClientException(  
 **"System level error, Code=["** + closeableHttpResponse.**getStatusLine**().**getStatusCode**() + **"]."**);  
 }  
 } **catch** (ClientProtocolException e) {  
 **throw new** HttpClientException(**"HttpClient error,"** + e.getMessage());  
 } **catch** (IOException e) {  
 **throw new** HttpClientException(**"IO error,"** + e.getMessage());  
 } **finally** {  
 **if** (closeableHttpResponse != **null**) {  
 **try** {  
 closeableHttpResponse.**close**();  
 } **catch** (IOException e) {  
 }  
 }  
 }  
 **return** ret;  
}

**public static** String doGet(String url) **throws** HttpClientException {  
 **return** *doGet*(url,**null**);  
}

测试：

String result = **null**;  
String url =**"http://api.superepc.com/vtm/DataFunc?grant\_code=XF9JKY0R&is\_car\_config=0&isCN=1&vinCode=LGBG22E22AY081092"**;  
**try** {  
 result = *doGet*(**"http://123.58.251.183:8080/goods/UserServlet?method=loginMobile&loginname=test1&loginpass=test1"**);  
 String result1 = *doGet*(url);  
 System.***out***.println(result);  
} **catch** (HttpClientException e) {  
 e.printStackTrace();  
}

输出：

## Httpclient模拟post请求

**public static** String doPost(String url, Map<String, Object> params, Map<String, Object> header)  
 **throws** HttpClientException {  
 String ret = **""**;  
 HttpPost post = **new** HttpPost(url);  
 post.setConfig(*config*);  
 post.addHeader(HTTP.***CONTENT\_ENCODING***, **"UTF-8"**);  
 CloseableHttpResponse closeableHttpResponse = **null**;  
 HttpEntity postEntity = **null**;  
 **try** {  
 **if**(params!=**null**) {  
 List<NameValuePair> list = **new** ArrayList<NameValuePair>();  
 **for** (Map.Entry<String, Object> entry : params.**entrySet**()) {  
 list.**add**(**new** BasicNameValuePair(entry.**getKey**(), entry.**getValue**().toString()));  
 }  
 postEntity = **new** UrlEncodedFormEntity(list);  
 post.setEntity(postEntity);  
 }  
  
 **if** (header != **null**) {  
 **for** (Map.Entry<String, Object> entry : header.**entrySet**()) {  
 post.setHeader(entry.**getKey**(), entry.**getValue**().toString());  
 }  
 }  
 closeableHttpResponse = *httpclient*.execute(post);  
 **if** (closeableHttpResponse.**getStatusLine**().**getStatusCode**() == 200) {  
 ret = EntityUtils.*toString*(closeableHttpResponse.**getEntity**(), **"UTF-8"**);  
 } **else** {  
 **throw new** HttpClientException(  
 **"System level error, Code=["** + closeableHttpResponse.**getStatusLine**().**getStatusCode**() + **"]."**);  
 }  
 } **catch** (ClientProtocolException e) {  
 **throw new** HttpClientException(**"HttpClient error,"** + e.getMessage());  
 } **catch** (IOException e) {  
 **throw new** HttpClientException(**"IO error,"** + e.getMessage());  
 } **finally** {  
 **if**(postEntity!=**null**) {  
 **try** {  
 EntityUtils.*consume*(postEntity);  
 } **catch** (IOException e) {  
 }  
 }  
 **if** (closeableHttpResponse != **null**) {  
 **try** {  
 closeableHttpResponse.**close**();  
 } **catch** (IOException e) {  
 }  
 }  
 }  
 **return** ret;  
}

**public static** String doPost(String url, Map<String, Object> params) **throws** HttpClientException {  
 **return** *doPost*(url,params,**null**);  
}

## FastJson

需要包：

<**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>fastjson</**artifactId**>  
 <**version**>1.2.43</**version**>  
</**dependency**>

### Json字符串转 map 对象

String jsonString1 = **"{\"param5\":\"value5\",\"param3\":\"value3\",\"param4\":\"value4\",\"param1\":\"value1\",\"param2\":\"value2\"}"**;  
System.***out***.println(jsonString1);  
Map<String,String> stringStringMap = (Map<String,String>) JSON.*parse*(jsonString1);  
**for** (String s : stringStringMap.**keySet**()) {  
 System.***out***.println(s + **"==>"** +stringStringMap.**get**(s));  
}

输出：

{"param5":"value5","param3":"value3","param4":"value4","param1":"value1","param2":"value2"}

param5==>value5

param3==>value3

param4==>value4

param1==>value1

param2==>value2

### Json字符串转 List 集合

String jsonString2 = **"[{\"param5\":\"value5\",\"param3\":\"value3\",\"param4\":\"value4\",\"param1\":\"value1\",\"param2\":\"value2\"},{\"p1\":\"v1\",\"p2\":\"v2\",\"p3\":\"v3\",\"p4\":\"v4\",\"p5\":\"v5\"}]"**;  
System.***out***.println(jsonString2);  
List<Map<String,String>> mapList = JSON.*parseObject*(jsonString2, **new** TypeReference<List<Map<String,String>>>(){});  
**for** (Map<String, String> stringObjectMap : mapList) {  
 **for** (String s : stringObjectMap.**keySet**()) {  
 System.***out***.println(s + **"==>"** + stringObjectMap.**get**(s));  
 }  
}

输出：

[{"param5":"value5","param3":"value3","param4":"value4","param1":"value1","param2":"value2"},{"p1":"v1","p2":"v2","p3":"v3","p4":"v4","p5":"v5"}]

param5==>value5

param3==>value3

param4==>value4

param1==>value1

param2==>value2

p1==>v1

p2==>v2

p3==>v3

p4==>v4

p5==>v5

### 复杂JSON字符串 转 List集合

String jsonString3 = **"{\"count\":2,\"list\":[{\"param5\":\"value5\",\"param3\":\"value3\",\"param4\":\"value4\",\"param1\":\"value1\",\"param2\":\"value2\"},{\"p1\":\"v1\",\"p2\":\"v2\",\"p3\":\"v3\",\"p4\":\"v4\",\"p5\":\"v5\"}]}"**;  
System.***out***.println(jsonString3);  
Map<String,Object> map = JSON.*parseObject*(jsonString3);  
System.***out***.println(map.**get**(**"count"**));  
String tempjsonString3 = map.**get**(**"list"**).toString();  
System.***out***.println(tempjsonString3);  
List<Map<String,String>> mapList2 = JSON.*parseObject*(tempjsonString3, **new** TypeReference<List<Map<String,String>>>(){});  
**for** (Map<String, String> stringObjectMap : mapList2) {  
 **for** (String s : stringObjectMap.**keySet**()) {  
 System.***out***.println(s + **"==>"** + stringObjectMap.**get**(s));  
 }  
}

输出：

{"count":2,"list":[{"param5":"value5","param3":"value3","param4":"value4","param1":"value1","param2":"value2"},{"p1":"v1","p2":"v2","p3":"v3","p4":"v4","p5":"v5"}]}

2

[{"param5":"value5","param3":"value3","param4":"value4","param1":"value1","param2":"value2"},{"p1":"v1","p2":"v2","p3":"v3","p4":"v4","p5":"v5"}]

param5==>value5

param3==>value3

param4==>value4

param1==>value1

param2==>value2

p1==>v1

p2==>v2

p3==>v3

p4==>v4

p5==>v5

### 解析JSON字符串到指定对象class

第一种方式：

String jsonString4 = **"[{\"age\":12,\"date\":1465475917155,\"name\":\"s1\"},{\"age\":12,\"date\":1465475917175,\"name\":\"s2\"},{\"age\":12,\"date\":1465475917175,\"name\":\"s3\"},{\"age\":12,\"date\":1465475917175,\"name\":\"s4\"},{\"age\":12,\"date\":1465475917175,\"name\":\"s5\"},{\"age\":12,\"date\":1465475917175,\"name\":\"s6\"}]"**;  
System.***out***.println(jsonString4);  
List<Student> studentList = JSON.*parseArray*(jsonString4,Student.**class**);  
**for** (Student student : studentList) {  
 System.***out***.println(student);  
}

第二种方式：

System.***out***.println(jsonString4);  
List<Student> studentList2 = JSON.*parseObject*(jsonString4,**new** TypeReference<List<Student>>(){});  
**for** (Student student : studentList2) {  
 System.***out***.println(student);  
}

输出：

[{"age":12,"date":1465475917155,"name":"s1"},{"age":12,"date":1465475917175,"name":"s2"},{"age":12,"date":1465475917175,"name":"s3"},{"age":12,"date":1465475917175,"name":"s4"},{"age":12,"date":1465475917175,"name":"s5"},{"age":12,"date":1465475917175,"name":"s6"}]

Student{name='s1', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s2', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s3', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s4', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s5', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s6', age=12, date=Thu Jun 09 20:38:37 CST 2016}

### Map对象转化成json字符串

Map<String, Object> maps = **new** HashMap<String,Object>();  
maps.**put**(**"grant\_code"**,**"XF9JKY0R"**);  
maps.**put**(**"is\_car\_config"**,0);  
maps.**put**(**"isCN"**,1);  
maps.**put**(**"vinCode"**,**"LGBG22E22AY081092"**);  
String jsonParam = JSON.*toJSONString*(maps);  
System.***out***.println(jsonParam);

输出：

{"grant\_code":"XF9JKY0R","is\_car\_config":0,"vinCode":"LGBG22E22AY081092","isCN":1}

## JackJson

需要包：

jackson-core-2.2.3.jar（核心jar包）

jackson-annotations-2.2.3.jar（该包提供Json注解支持）

jackson-databind-2.2.3.jar

<**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-core</**artifactId**>  
 <**version**>2.9.1</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-annotations</**artifactId**>  
 <**version**>2.9.1</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-databind</**artifactId**>  
 <**version**>2.9.1</**version**>  
</**dependency**>

### 指定对象Class转成 json字符串

User user = **new** User();  
user.setName(**"小民"**);  
user.setEmail(**"xiaomin@sina.com"**);  
user.setAge(20);  
  
SimpleDateFormat dateformat = **new** SimpleDateFormat(**"yyyy-MM-dd"**);  
user.setBirthday(dateformat.parse(**"1996-10-01"**));  
  
*/\*\*  
 \* ObjectMapper是JSON操作的核心，Jackson的所有JSON操作都是在ObjectMapper中实现。  
 \* ObjectMapper有多个JSON序列化的方法，可以把JSON字符串保存File、OutputStream等不同的介质中。  
 \* writeValue(File arg0, Object arg1)把arg1转成json序列，并保存到arg0文件中。  
 \* writeValue(OutputStream arg0, Object arg1)把arg1转成json序列，并保存到arg0输出流中。  
 \* writeValueAsBytes(Object arg0)把arg0转成json序列，并把结果输出成字节数组。  
 \* writeValueAsString(Object arg0)把arg0转成json序列，并把结果输出成字符串。  
 \*/*ObjectMapper mapper = **new** ObjectMapper();  
  
*//User类转JSON  
//输出结果：{"name":"小民","age":20,"birthday":844099200000,"email":"xiaomin@sina.com"}*String json = mapper.writeValueAsString(user);  
System.***out***.println(json);

输出：

{"name":"小民","age":20,"birthday":844099200000,"email":"xiaomin@sina.com"}

### List集合转化成json字符串

List<User> users = **new** ArrayList<User>();  
users.**add**(user);  
String jsonlist = mapper.writeValueAsString(users);  
System.***out***.println(jsonlist);

输出：

[{"name":"小民","age":20,"birthday":844099200000,"email":"xiaomin@sina.com"}]

### Json字符串转化成指定Class类

String json = **"{\"name\":\"小民\",\"age\":20,\"birthday\":844099200000,\"email\":\"xiaomin@sina.com\"}"**;  
*/\*\*  
 \* ObjectMapper支持从byte[]、File、InputStream、字符串等数据的JSON反序列化。  
 \*/*ObjectMapper mapper = **new** ObjectMapper();  
User user = mapper.readValue(json, User.**class**);  
System.***out***.println(user);

输出：

User{name='小民aa', age=25, birthday=Tue Oct 01 00:00:00 CST 1996, email='xiaomin@sina.com'}

### Json字符串转化成集合List

方法一：

String jsonString=**"[{'id':'1'},{'id':'2'}]"**;  
ObjectMapper mapper = **new** ObjectMapper();  
JavaType javaType = mapper.getTypeFactory().constructParametricType(List.**class**, Bean.**class**);  
*//如果是Map类型 mapper.getTypeFactory().constructParametricType(HashMap.class,String.class, Bean.class);*List<Bean> lst = (List<Bean>)mapper.readValue(jsonString, javaType);

输出：

Student{name='s1', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s2', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s3', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s4', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s5', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s6', age=12, date=Thu Jun 09 20:38:37 CST 2016}

方法二：

String jsonString=**"[{'id':'1'},{'id':'2'}]"**;  
ObjectMapper mapper = **new** ObjectMapper();  
List<Bean> beanList = mapper.readValue(jsonString, **new** TypeReference<List<Bean>>() {});

输出：

Student{name='s1', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s2', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s3', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s4', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s5', age=12, date=Thu Jun 09 20:38:37 CST 2016}

Student{name='s6', age=12, date=Thu Jun 09 20:38:37 CST 2016}

## httpclient模拟post请求发送json数据

### 传头部信息

**public static** String doPostByType(String contentType,String url,String jsonParam,Map<String,Object> header) **throws** HttpClientException {  
 String ret =**""**;  
 HttpPost post = **new** HttpPost(url);  
 post.setConfig(*config*);  
 post.addHeader(HTTP.***CONTENT\_ENCODING***,**"utf-8"**);  
 CloseableHttpResponse closeableHttpResponse =**null**;  
 StringEntity postEntity =**null**;  
 **try** {  
 **if** (jsonParam != **null**) {  
 postEntity = **new** StringEntity(jsonParam);  
 postEntity.setContentEncoding(**"UTF-8"**);  
 **if**(contentType.equals(**"json"**)){  
 postEntity.setContentType(**"application/json"**);  
 }**else if**(contentType.equals(**"soap+xml"**)){  
 postEntity.setContentType(**"application/soap+xml"**);  
 }  
  
 post.setEntity(postEntity);  
 }  
 **if** (header != **null**) {  
 **for** (Map.Entry<String, Object> entry : header.**entrySet**()) {  
 post.setHeader(entry.**getKey**(), entry.**getValue**().toString());  
 }  
 }  
 closeableHttpResponse = *httpclient*.execute(post);  
 **if** (closeableHttpResponse.**getStatusLine**().**getStatusCode**() == 200) {  
 ret = EntityUtils.*toString*(closeableHttpResponse.**getEntity**(), **"UTF-8"**);  
 } **else** {  
 **throw new** HttpClientException(**"System level error, Code=["** + closeableHttpResponse.**getStatusLine**().**getStatusCode**() + **"]."**);  
 }  
 } **catch** (ClientProtocolException e) {  
 **throw new** HttpClientException(**"HttpClient error,"** + e.getMessage());  
 }**catch** (IOException e) {  
 **throw new** HttpClientException(**"IO error,"** + e.getMessage());  
 } **finally** {  
 **try** {  
 **if**(postEntity!=**null**) {  
 EntityUtils.*consume*(postEntity);  
 }  
 **if**(closeableHttpResponse !=**null**){  
 closeableHttpResponse.**close**();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 **return** ret;  
}

### 不传头部信息

**public static** String doPostByType(String contentType,String url,String Param) **throws** HttpClientException {  
 **return** *doPostByType*(contentType,url,Param,**null**);  
}

测试类：

**public static void** doPostJsonTest(){  
 String url =**"http://123.58.251.183:8080/goods/json2"**;  
 Map<String, Object> map = **new** HashMap<String,Object>();  
 map.**put**(**"count"**, 2);  
 String jsonParam = JSON.*toJSONString*(map);  
 Map<String, Object> header = **new** HashMap<String,Object>();  
 header.**put**(**"token"**, **"61b3590090982a0185dda9d3bd793b46"**);  
 **try** {  
 String reString = HttpUtils.*doPostJson*(url, jsonParam, header);  
 System.***out***.println(reString);  
 } **catch** (HttpClientException e) {  
 e.printStackTrace();  
 }  
}

## httpclient模拟上传文件操作

### 传头部信息

**public static** String doUpload(String url, File file,Map<String,Object> params,Map<String,Object> header){  
 String ret =**""**;  
 HttpPost post = **new** HttpPost(url);  
 post.setConfig(*config*);  
 post.addHeader(HTTP.***CONTENT\_ENCODING***,**"UTF-8"**);  
 CloseableHttpResponse response = **null**;  
 **try** {  
 MultipartEntityBuilder entityBuilder = MultipartEntityBuilder.*create*();  
 entityBuilder.addBinaryBody(**"file"**,file);  
 **if**(params!=**null**){  
 **for**(Map.Entry<String,Object> entry:params.**entrySet**()){  
 entityBuilder.addTextBody(entry.**getKey**(),entry.**getValue**().toString());  
 }  
 }  
 post.setEntity(entityBuilder.build());  
 **if**(header!=**null**){  
 **for**(Map.Entry<String,Object> entry:header.**entrySet**()){  
 post.addHeader(entry.**getKey**(),entry.**getValue**().toString());  
 }  
 }  
  
 response = *httpclient*.execute(post);  
 **if**(response.**getStatusLine**().**getStatusCode**()==200){  
 ret = EntityUtils.*toString*(response.**getEntity**(),**"utf-8"**);  
 }**else**{  
 **throw new** HttpClientException(**"System level error, Code=["** + response.**getStatusLine**().**getStatusCode**() + **"]."**);  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **catch** (HttpClientException e) {  
 e.printStackTrace();  
 }**finally** {  
 **if**(response!=**null**){  
 **try** {  
 response.**close**();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 **return** ret;  
}

### 不传头部信息

**public static** String doUpload(String url,File file){  
 **return** *doUpload*(url,file,**null**,**null**);  
}

测试类：

String url=**"http://123.58.251.183:8080/FileSever/upload.do"**;  
String filePath = System.*getProperty*(**"user.dir"**)+ File.***separator***+**"testdata"**+File.***separator***;  
File file = **new** File(filePath+**"test1.xml"**);  
String result = HttpUtils.*doUpload*(url,file);  
System.***out***.println(result);

## httpclient模拟下载文件操作

### 传头部信息

**public static void** doDownload(String url, File descfile,Map<String,Object> header){  
 HttpPost post = **new** HttpPost(url);  
 post.setConfig(*config*);  
 post.addHeader(HTTP.***CONTENT\_ENCODING***,**"UTF-8"**);  
 CloseableHttpResponse response = **null**;  
 **try** {  
 **if**(header!=**null**){  
 **for**(Map.Entry<String,Object> entry:header.**entrySet**()){  
 post.addHeader(entry.**getKey**(),entry.**getValue**().toString());  
 }  
 }  
 response = *httpclient*.execute(post);  
 **if**(response.**getStatusLine**().**getStatusCode**()==200){  
 FileUtils.*copyToFile*(response.**getEntity**().**getContent**(),descfile);  
 }**else**{  
 **throw new** HttpClientException(**"System level error, Code=["** + response.**getStatusLine**().**getStatusCode**() + **"]."**);  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **catch** (HttpClientException e) {  
 e.printStackTrace();  
 }**finally** {  
 **if**(response!=**null**){  
 **try** {  
 response.**close**();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

### 不传头部信息

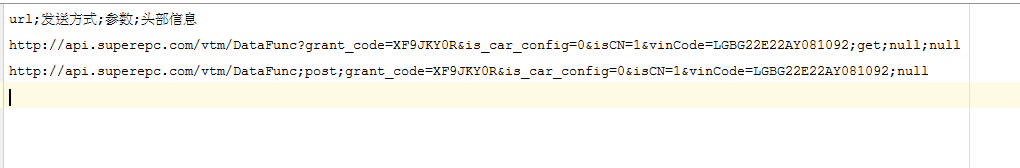
**public static void** doDownload(String url,File descfile){  
 *doDownload*(url,descfile,**null**);  
}

测试类：

String url=**"http://123.58.251.183:8080/FileSever/upload.do"**;  
String filePath = System.*getProperty*(**"user.dir"**)+ File.***separator***+**"testdata"**+File.***separator***;  
File file = **new** File(filePath+**"test1.xml"**);  
String result = HttpUtils.*doUpload*(url,file);  
System.***out***.println(result);  
  
JSONObject object =JSON.*parseObject*(result);  
String fileId = object.getString(**"fileId"**);  
**try** {  
 String result2 = HttpUtils.*doGet*(**"http://123.58.251.183:8080/FileSever/url.do?fileId="**+fileId);  
 JSONObject object1 = JSON.*parseObject*(result2);  
 String downloadUrl = object1.getString(**"url"**);  
 File file1 = **new** File(filePath+**"test2.xml"**);  
 HttpUtils.*doDownload*(downloadUrl,file1);  
} **catch** (HttpClientException e) {  
 e.printStackTrace();  
}

## Httpclient模拟从文件中获取的请求

Data.txt内容如下：



将字符串转化成Map,重载doPost方法：

**public static** String doPost(String url, String params,String regex) **throws** HttpClientException {  
 String[] param =params.split(regex);  
 Map<String,Object> map = **new** HashMap<String,Object>();  
 **for**(**int** i=0;i<param.**length**;i++){  
 String[] pp = param[i].split(**"="**);  
 **for**(String p:pp){  
 map.**put**(pp[0],pp[1]);  
 }  
 }  
 **return** *doPost*(url,map,**null**);  
}

测试类：

**public static void** main(String[] args) **throws** HttpClientException, IOException {  
 String filePath = System.*getProperty*(**"user.dir"**) + File.***separator*** + **"testdata"** + File.***separator***;  
 File file = **new** File(filePath + **"data.txt"**);  
 File file1 = **new** File(filePath + **"result.txt"**);  
 String ret = **""**;  
 List<String> lines = FileUtils.*readLines*(file, **"UTF-8"**);  
 **for** (**int** m = 0; m < lines.**size**(); m++) {  
 **if** (m > 0) {  
 String[] ss = lines.**get**(m).split(**";"**);  
 String url = ss[0];  
 String method = ss[1];  
 **if** (**"get"**.equals(method)) {  
 ret = HttpUtils.*doGet*(url);  
 *writeTofile*(file1, **"\nget ------>"** + ret);  
 System.***out***.println(**"get------>"** + ret);  
 } **else if** (**"post"**.equals(method)) {  
 ret = HttpUtils.*doPost*(url, ss[2], **"&"**);  
 *writeTofile*(file1, **"\npost ------>"** + ret);  
 System.***out***.println(**"post------>"** + ret);  
 }  
 }  
 }  
}

**public static void** writeTofile(File file,String str) **throws** IOException {  
 FileUtils.*writeStringToFile*(file,str,**"utf-8"**,**true**);  
  
}

输出：

get------>DataFunc([{"TID":"8519","retCode":"1"}])

post------>DataFunc([{"TID":"8519","retCode":"1"}])