fastjson循环依赖

循环依赖例子

```
HashMap map = new HashMap();
map.put("foo", map);
System.out.println(map);
```

输出是：{foo=(this Map)}，this Map代表map自己引用自己
如果用fastjson序列换成json会怎么样呢?
fastjson处理json序列化的方法都放在JSON类中，主要有3个方法:

- toJSON()
- toJSONString()
- toJSONByteds()

toJSON()

```
HashMap map = new HashMap();
map.put("foo", map);
System.out.println(JSON.toJSON(map));
```

汇报异常Exception in thread "main" java.lang.StackOverflowError，原因跟toJSON实现方式有关

```
JSONObject json = new JSONObject(innerMap);

for (Map.Entry<Object, Object> entry : map.entrySet()) {
    Object key = entry.getKey();
    String jsonKey = TypeUtils.castToString(key);
    Object jsonValue = toJSON(entry.getValue());//<--这里
    json.put(jsonKey, jsonValue);
}
```

可以看到对Map的value还会调用toJSON方法，造成了死循环，且对value处理前任何配置都没生效，所以toJSON方法对循环引用无解。

尝试一下toJSONString和toJSONBytes方法

```
HashMap map = new HashMap();
map.put("foo", map);
System.out.println(JSON.toJSONString(map));
System.out.println(new String(JSON.toJSONBytes(map)));
```

输出：{"foo":{"$ref":"@"}}，跟直接print输出一致。这两个方法会依据配置SerializerFeature.DisableCircularReferenceDetect来控制是否开启重复引用检查，在处理value时判断上下文中是否已经存在对象。如果开启配置，且上下文中value对象已存在则用引用形式表示。

```
//JavaBeanSerializer.class
public boolean writeReference(JSONSerializer serializer, Object object, int
fieldFeatures) {
    SerialContext context = serializer.context;
    int mask = SerializerFeature.DisableCircularReferenceDetect.mask;
    if (context == null || (context.features & mask) != 0 || (fieldFeatures
& mask) != 0) {//<--这里
        return false;
    }

    if (serializer.references != null && serializer.references.containsKey(o
bject)) {
        serializer.writeReference(object);
        return true;
    } else {
        return false;
    }
}
```

不同层级的引用会以不同的符号代替，具体可以参考JSONSerializer.writeReference()方法