中国矿业大学计算机学院

2019 级本科生课程报告

课程名称	《软件测试 A》
报告时间	2022 年 7 月
学生姓名	胡钧耀
学　　号	06192081
专　　业	计算机科学与技术
任课教师	薛猛

《软件测试 A》课程评分表

序号	指标点	课程教学目标	占比	得分
1	3.4 在充分理解计算机软硬件及系统的基础上，能够设计针对计算机领域复杂工程问题的解决方案，设计或开发满足特定需求和约束条件的软硬件系统、模块或算法流程，并能够进行模块和系统级优化。	**目标 1：**能够充分理解认识软件测试的重要性，掌握各种软件测试方法和技术。能够针对待测系统选择使用合适的测试方法和技术，设计合理的测试方案。	40%	
2	4.2 针对计算机领域复杂工程问题，具有根据解决方案进行工程设计与实施的能力，具有系统的工程研究与实践经历。	**目标 2：**能够根据实际复杂的待测系统与测试方案编写测试代码，实施测试任务，分析测试结果。	30%	
3	5.2 能够在计算机领域复杂工程问题的预测、建模、模拟或解决过程中，开发、选择与使用恰当的技术、软硬件及系统资源、现代工程研发工具，提高解决复杂工程问题的能力和效率。	**目标 3：**能够在实际软件测试任务中选择使用自动化的测试工具，如：CheckStyle 、Junit、Jmeter、Monkey、Appium 等，提高软件测试的效率。	30%	
			总得分	

摘　　要

软件测试指的是在规定的条件下对程序进行操作，以发现程序的错误，衡量软件质量，并对其是否能满足用户需求进行评估的过程。本文按照《软件测试 A》课程标准要求和作业要求，完成了各项软件测试子任务，并进行了相应地深入分析。

首先在绪论部分，分析了软件测试的研究意义与发展现状，了解到软件测试在当前人工智能背景下的开发活动中仍有重要意义。接着，使用白盒测试中的逻辑覆盖方法测试了工业代码 Json，同时利用自动化测试工具 EvoSuite 辅助编码设计，最终代码覆盖率达到 89%。此外，采用黑盒测试方法中的等价类方法、组合测试等方法，对 Excel 中的公式设置页面进行了测试用例的设计。最后，结合 Web 开发课程的网站作业，使用 JMeter 对网站进行性能测试与分析，尝试使用软件进行脚本的录制、优化和回放。

全文从理论到实践，从基础到深入，对软件测试有较为全面的认识和理解，提升了自己的代码编程水平，也对计算机学科整体性和统一性有了更深入的认知。

关键词：软件测试，白盒测试，黑盒测试，性能测试，Json，EvoSuite，JMeter

ABSTRACT

Software testing is the process of operating the program under specified conditions to find the program's errors, measure the quality of the software, and evaluate whether it can meet the needs of users. In this paper, according to *Software testing A* curriculum standard requirements and homework requirements, completed the software testing sub-tasks.

Firstly, in the introduction, the research significance and development status of software testing are analyzed, and software testing is still of great significance in the current development activities under the background of artificial intelligence. Next, industrial code *Json* was tested using the logical coverage method in white box testing, while *EvoSuite*, an automated testing tool, was used to assist with coding design, resulting in 89% code coverage. In addition, the formula setting page in *Excel* is designed by using equivalence class method and combination test method in black box test method. Finally, combined with the website assignments of the Web course, *JMeter* was used to test and analyze the performance of the website, and the software was used to recording, optimization and replay the script.

From theory to practice, from basic to in-depth, I have a more comprehensive understanding and understanding of software testing, improve my code programming level, and also have a deeper understanding of the integrity and unity of computer science.

Key Words: Software testing, White box testing, Black box testing, performance testing, *Json*, *EvoSuite*, *JMeter*

目　　录

1 绪论

1.1 研究背景

对于软件测试，笔者在大三上半学期的《软件工程》课程中已经略有了解：在理论上，了解了等价类、黑盒测试、白盒测试的基本概念；在实践上，学习编写了简单的测试四则运算的 Junit 单元测试。以上对软件测试有了初步的认识。本学期的《软件测试》课程中，对软件测试的研究现状、测试技术等进行了更深入、全方面的学习，对软件工程的意义和地位有了全新的、更深入的认识。

当前的时代是信息化时代，计算机科学与技术的应用几乎覆盖了生活和工作的各个方面，人们越来越离不开软件：小到疫情期间的核酸检测健康码的设计，要注意满足高并发需求以及注重隐私问题，需要开发者不断测试调试后才能上线运作；大到近期的神舟十四号成功发射，航天员顺利进驻空间站天和核心舱，也需要科研人员专心细致，反复验证，确保航天工程的安全和顺利。软件测试其实就是通过多种测试手段验证软件开发是否达到开发预期结果的过程，其意义和地位不容忽视，它是软件开发过程中一个重要的环节。

1.2 研究意义

1.2.1 提高用户使用软件的舒适度

提高用户体验并非只是前端设计工程师需要思考的内容，软件并非只需要拥有交互性好、简洁大方的界面即可，如果软件有错误却不给用户给予提示，或者返回了错误的结果，或者遇到缺陷却以代码形式反馈用户，或者程序直接崩溃报错，这些结果都是不合理的。注重软件测试，从用户使用软件的角度出发，了解用户的需求和使用方法，可以使软件更加人性化，更具有可操作性，从而提高用户使用软件的舒适度与体验感，也能增加软件的潜在用户，利用软件给开发者不断带来新的价值。

1.2.2 降低或避免软件缺陷造成的损失

注重软件测试，可以不断减少软件缺陷，做到防范于未然，也就降低或者避免软件缺陷可能造成的各种损失，这包括但不限于财产、人身等各个方面。如出现在各种网络销售平台的“羊毛党”有组织有目的性地追踪平台优惠漏洞，常常集体连夜薅羊毛，致使网购平台损失严重。[1]

1.2.3 软件缺陷数量高于硬件缺陷

尽管在计算机早期时代，纸带等工具损耗率较高，硬件条件极大限制了计算机的发展，但随着时代不断发展，硬件损耗速度不断降低，运算处理能力也不断提升，硬件发生问题的几率越来越小，往往问题是开发者的编码失误造成的。

1.2.4 软件顶层设计并非一劳永逸

为了对软件设计、开发、编码等过程进行规范化，相关专家一致在寻找合理统一的形式化方法与软件规约，提出一系列的软件标准，力求保证软件质量，但软件本身是极其复杂的，也拥有不确定的因素，没有一套方法可以一劳永逸地设计好一款软件，有好

的顶层设计也并不意味着就有完全正确的编码。无论是从软件开发过程还是软件测试自身的好处看，软件测试在今后较长时间内仍将是保证软件质量的重要手段。

1.2.5 小结

测试人员水平越高，找到软件问题的时间就越早，软件就越容易更正，越有利于软件开发者。随着人们对软件测试重要性的认识越来越深刻，软件测试阶段在整个软件开发周期中所占的比重日益增大。总之，有软件的地方就需要软件测试进行质量保证，想要做出设计优秀、用户喜爱、功能完备的好软件，就必须经过严谨细致的软件测试。

1.3 研究现状

1.3.1 自动化测试

随着21世纪的到来，手工软件测试的占比不断减小，更多自动化的新测试方法不断出现，使软件测试的面貌焕然一新。通过大量的自动化测试框架，从不同驱动方式来看有基于数据驱动的[2]，也有基于关键字驱动的[3]，从面向平台来看，有基于嵌入式的测试方法[4]，也有面向移动端如安卓的自动化测试[5]，都使得测试人员能够以更高的效率执行其任务，从而使测试人员能把重心放在更重要的地方，例如分析软件架构，对错误缺陷进行分析复盘等。当前，服务上云，企业上云的趋势越来越明显，云测试的方法也被提出并且实施，例如华为云测系统，同它的 DevOps 敏捷开发平台构成统一的整体，这可以帮助企业以更快的速度和更少的资金管理产品的测试，提高效率，节约成本。

1.3.2 人工智能背景下的软件测试

人工智能不再是完全虚无飘渺，捉摸不定的名词，现在很多领域，AI 可以实际落地，成为用到生活和工业中的应用技术，对于软件测试来说，人工智能背景下的软件测试主要考虑两方面：一种情况是需要对使用了人工智能技术的软件进行软件测试；另一种情况指的是需要应用人工智能技术对各种软件进行测试，类似自动化测试。

1.3.2.1 对使用人工智能技术的软件进行测试

深度神经网络学习算法通常被认为是黑盒模型，但是随着训练的数据集不同，其测试结果具有不稳定的特性，并不是给定一个输入，每一次输出就是相同的（例如手写数字识别就是要判断十个数字的相似比率，谁最高就认为是哪个数字，但对于同一个输入，不一定每次都是一个数字的相似比率最高）。通常在神经网络的构建中，需要使用训练集进行训练，使用验证集进行验证，都是需要使用大量的大规模数据集进行模型的验证，很难说要划分出一个等价类或者边界值，测试往往是测试的模型训练的优质程度，对于分类模型来说就是总体分类正确率等，回归模型就是误差之和的大小。总之，这些模型必须依赖于大量的数据才可以达到测试的效果。典型的例子有自动驾驶软件[6]、电缆隧道移动巡检机器人系统[7]。

1.3.2.2 使用人工智能技术对软件进行测试

人工智能是机器通过感知，理解和学习模仿人类行为的能力。对于一般的经典的程序设计是通过编程去输入一些算法规则和数据，然后再通过程序的算法去输出答案，但对人工智能来说是先输入数据和答案，中间的规则是由 AI 系统的训练过程所得到的，最后通过数据来训练数据、训练规则，得到自己的一套软件测试方法。数据驱动的 AI

系统相对于传统软件测试更复杂，更具有挑战性。机器学习还可以用来识别可疑语句，如测试间观察到的相关故障，可在调试期间帮助故障定位。[8]

1.3.3 小结

软件测试尽管是一个很老的话题，但是随着时代的变化，技术也在不管更新迭代，当前人工智能时代下，软件测试的发展与应用的有新的变化，软件测试也正在面临着新的挑战，软件测试的发展也将受到重大影响。大数据和人工智能背景下的软件测试，并不会因为时代的发展而被抛弃，而是相互结合，相辅相成，软件测试为人工智能发展找到新方向，人工智能为软件测试赋予新动力。

1.4 课程总结

1.4.1 消除对软件测试的误解和偏见

软件测试并不意味着就是发现缺陷，解决缺陷本身就可以，软件测试需要望见更长远的事情，不仅要能细致入微，找出错误，更要有大局观，举一反三，去考虑有没有其他类似的错误，能不能复现，有没有可借鉴到别的漏洞的地方，这都是需要去思考的，而不是简简单单的做好修补工作即可。

1.4.2 进行了软件测试的代码实践

进一步学习了 JUnit 进行软件测试的方法实践，在面对较难的工程代码时能抽丝剥茧，理清类之间的关系，同时学习了 Evosuite 自动化测试软件协助进行软件测试，提高测试覆盖率。

1.4.3 计算机学科专业素养的全面提升

在进行软件测试的实践中，不仅学习了软件测试的知识，还加深了数据结构的学习，红黑树、线段树这些数据结构都是《数据结构》课程中没有接触过的工程代码，这相当于在提高了软件测试的编码能力的同时也学习到了新的知识。同时，还使用了 JMeter 对《Web 程序设计》中搭建的网站进行了性能测试，这将看似割裂的专业课程结合在了一起，构成了一个有机的整体。

2 白盒测试

2.1 选题介绍

2.1.1 Json 简介

Json（JavaScript Object Notation）是一种轻量级的数据交换格式。其采用完全独立于编程语言的文本格式来存储和表示数据。简单来说，其就是一个序列化的对象或数组，简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。不仅易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

2.1.2 Json 构成

Json 这套标记符包含六个构造字符（"(", ")", "[", "]", ":", ","）、字符串、数字和三个字面名（true, false, null）。

Json 共有 15 个类文件，分析其中的组织结构，其类图如图 2.1-1 所示。

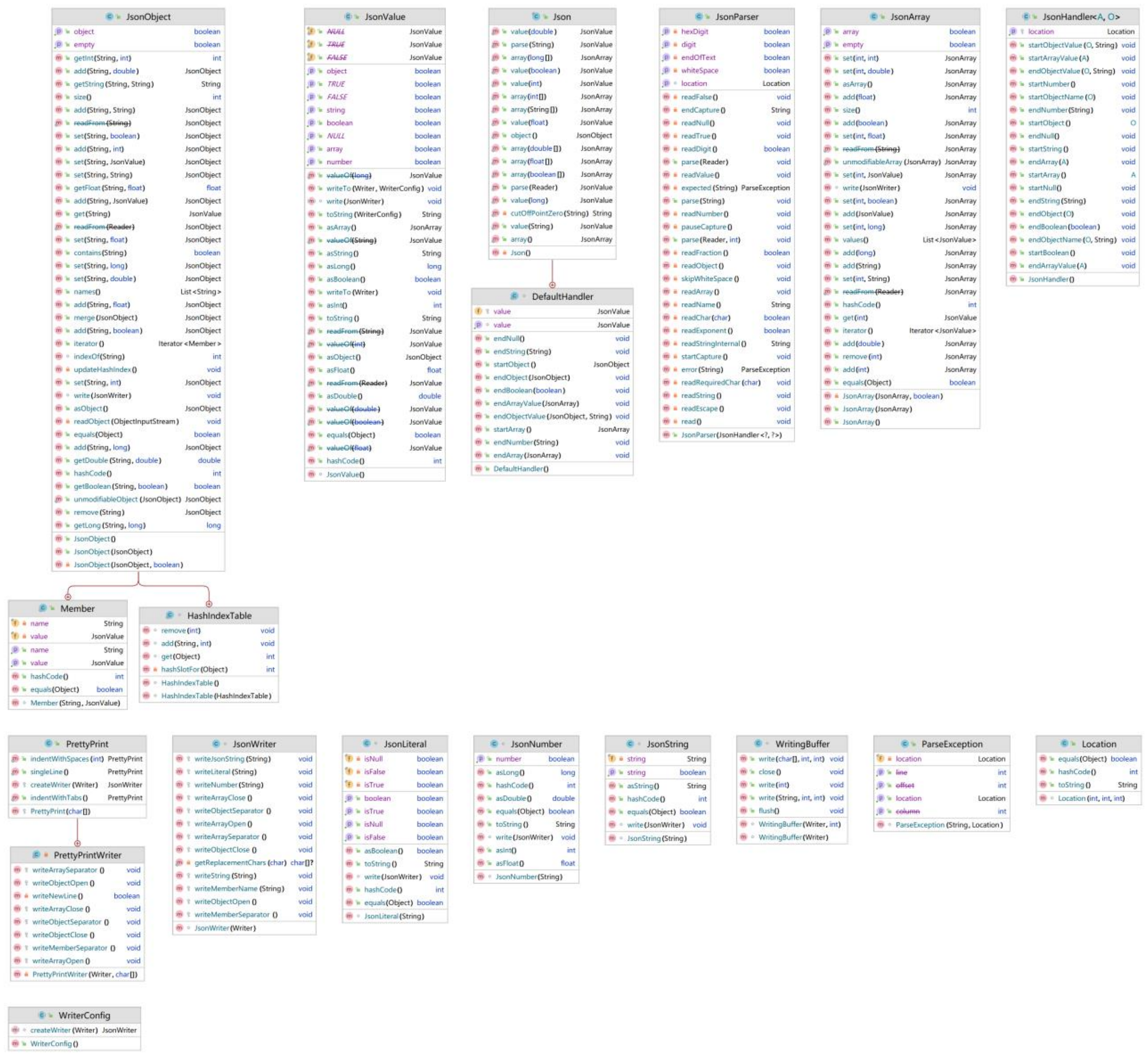


图 2.1-1 Json 类图

2.2 控制流图与覆盖设计

2.2.1 Json

2.2.1.1 类的作用

这个类用作最小 Json API 的入口点。要解析给定的 Json 输入，可以使用下面的 parse()方法：JsonObject 对象= Json.parse(字符串).asObject(); 要创建要序列化的 Json 数据结构，使用方法 value()、array()和 object()。要从给定的 Java 数组创建一个 Json 数组，可以使用带参数的 array()方法。

这里大部分代码主要是直接 return 返回的，大多数连分支结构都没有，测试时直接输出即可判断正确性，因此选取 public static JsonValue value(double value) 函数和 public static JsonArray array(int... values)函数为代表绘制控制流图和设计测试样例，下面其他的类的测试分析同样如此，完整测试源代码参考附录部分。

2.2.1.2 public static JsonValue value(double value)

value 函数是一系列函数，重载了不同的输入类型，其中，浮点数作为输入的 value 函数有一些判定条件，如果这个数是无穷大的，或者不是数，就需要抛出异常进入异常流，否则才能进行下面的正常操作。

该函数代码如代码 2.2-1 所示，已标出简单分支的标号。

代码 2.2-1 JsonValue value(double value)代码

JsonValue value(double value)

```
public static JsonValue value(double value) {
  if (Double.isInfinite(value) 【1】|| Double.isNaN(value)【2】) {
    throw new IllegalArgumentException("…"); 【3】
  }
  return new JsonNumber(cutOffPointZero(Double.toString(value));【4】
}
```

按照上述标号，对该函数进行分析，其控制流图如图 2.2-1 所示（笔者使用黄色圆圈表示进入异常流，蓝色表示正常程序流输出，下同）。

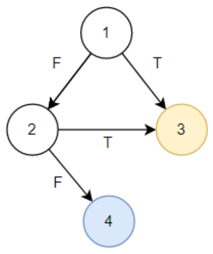


图 2.2-1 JsonValue value(double value)控制流图

该程序模块有 3 条不同的路径：P1(1-3), P2(1-2-4), P3(1-2-3)，基本判定条件有 M（Double.isInfinite）和 N（Double.isNaN）。根据分析，设计基本路径覆盖测试样例如表 2.2-1 所示。

表 2.2-1 JsonValue value(double value)测试样例

样例 value	条件（M、N）	结果	路径
1.2	FF	1.2	P2(1-2-4)
POSITIVE_INFINITY	TF	出错	P1(1-3)
Nan	FT	出错	P3(1-2-3)

2.2.1.3 public static JsonArray array(int... values)

array 函数创建一个新的 JsonArray，array(int... values)函数其中包含给定的 int 值。如果列表为空，则抛出异常，否则对列表中的数据一个一个进行增加 add 操作，从而延长 JsonArray。

该函数代码如代码 2.2-2 所示，已标出简单分支的标号。

代码 2.2-2 JsonArray array(int... values)代码

JsonArray array(int... values)

```
public static JsonArray array(int... values) {
  if (values == null) 【1】 {
    throw new NullPointerException("values is null"); 【2】
  }
  JsonArray array = new JsonArray(); 【3】
  for (int value : values) 【4】 {
    array.add(value);【5】
  }
  return array;【6】
}
```

按照上述标号，对该函数进行分析，其控制流图如图 2.2-2 所示。

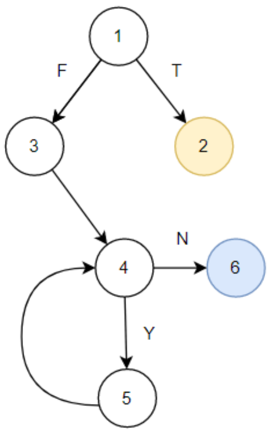


图 2.2-2 JsonArray array(int... values)控制流图

该程序模块有 2 条不同的路径能覆盖所有边：P1(1-2), P2(1-3-4-5-4-6)，基本判定条件有 M（values == null）和 N（for）。根据分析，设计基本路径覆盖测试样例如表 2.2-2 所示。

表 2.2-2 JsonArray array(int... values)测试样例

样例 value	条件（M、N）	循环次数	结果	路径
null	F-	0	出错	P1(1-2)
[-1, 0.5]	TT、TF	2	正常	P2(1-3-4-5-4-6)

2.2.2 JsonArray

2.2.2.1 类的作用

JsonArray，顾名思义是由各种 JsonObject 构成的数组，用 "[{ } , { } , ... , { }]" 来表示。可以使用 add 方法添加元素，该方法接受 JsonValue、字符串、原始数字和布尔值的实例。要替换数组的元素，可以使用 set 方法。可以使用 get(int)通过它们的索引访问元素。还支持使用 iterator()或增强的 for 循环按文档顺序遍历元素。

2.2.2.2 void write(JsonWriter writer)

该函数代码如代码 2.2-3 所示，已标出简单分支的标号。

代码 2.2-3 void write(JsonWriter writer)代码

void write(JsonWriter writer)

```
public void write(JsonWriter writer) throws IOException {
  writer.writeArrayOpen();【1】
  Iterator<JsonValue> iterator = iterator();【2】
  if (iterator.hasNext()) 【3】{
    iterator.next().write(writer);【4】
    while (iterator.hasNext()) 【5】{
      writer.writeArraySeparator(); 【6】
      iterator.next().write(writer); 【7】
    }
  }
  writer.writeArrayClose();【8】
}
```

按照上述标号，对该函数进行分析，其控制流图如图 2.2-3 所示。

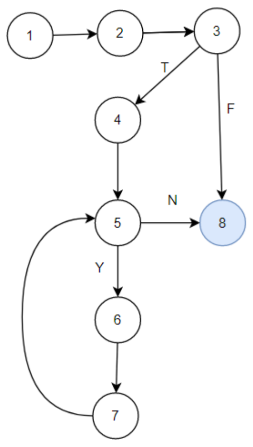


图 2.2-3 void write(JsonWriter writer)控制流图

该程序模块有 2 条不同的路径能覆盖所有边：P1(1-2-3-8), P2(1-2-3-4-5-6-7-5-8)，基本判定条件有 M（iterator.hasNext()）且【3】和【5】都是在判断 M。根据分析，设计基本路径覆盖测试样例如表 2.2-3 所示。

表 2.2-3 void write(JsonWriter writer)测试样例

样例	条件（M）	循环次数	结果	路径
空 JsonArray	F-	0	正常	P1(1-2-3-8)
含两项的 JsonArray	TT、TF	2	正常	P2(1-2-3-4-5-6-7-5-8)

根据测试样例，设计空 JsonArray 和含两项的 JsonArray 分别做测试，可知测试代码如代码 2.2-4。

代码 2.2-4 void testWrite()测试代码

void testWrite()

```
@Test
public void testWrite01() throws Exception {
    JsonArray jsonArray0 = new JsonArray();
    StringWriter stringWriter0 = new StringWriter();
    JsonWriter jsonWriter0 = new JsonWriter(stringWriter0);
    jsonArray0.write(jsonWriter0);
    Assert.assertEquals("[]", stringWriter0.toString());
}

@Test
public void testWrite02() throws Exception {
    JsonArray jsonArray0 = new JsonArray();
    JsonArray jsonArray1 = jsonArray0.add((long) 0);
    jsonArray1 = jsonArray0.add(10L);
    StringWriter stringWriter0 = new StringWriter();
    JsonWriter jsonWriter0 = new JsonWriter(stringWriter0);
    jsonArray0.write(jsonWriter0);
    Assert.assertEquals("[0,10]", stringWriter0.toString());
}
```

2.2.3 JsonNumber

2.2.3.1 类的作用

JsonNumber 继承了 JsonValue，是 Json 的一种值的类型，在该类中定义了构造函数，写方法，转换字符串方法、判断数字类型方法，转换数字类型方法，以及 equals 方法。JsonString 类、JsonLiteral 类与 JsonNumber 类的设计是基本同理的，后面不再具体展开介绍。

2.2.3.2 boolean equals(Object object)

这是经典的需要重写的方法，该函数代码如代码 2.2-5 所示，已标出简单分支的标号。

代码 2.2-5 void write(JsonWriter writer)代码

void write(JsonWriter writer)

```
@Override
public boolean equals(Object object) {
  if (this == object)【1】 {
    return true;【2】
  }
  if (object == null) 【3】 {
    return false;【4】
  }
  if (getClass() != object.getClass()) 【5】{
    return false;【6】
  }
  JsonNumber other = (JsonNumber)object;【7】
  return string.equals(other.string);【8】
}
```

按照上述标号，对该函数进行分析，【1】【3】【5】共出现了三次判断结构，采用判定覆盖对上述代码进行测试样例设计。

表 2.2-4 void write(JsonWriter writer)测试样例

样例 JsonNumber115114	条件（1、3、5）	结果
自己	T--	true
新的 JsonNumber 115114	FFF	true
JsonNumber 1414810	FFF	false
空	FT-	false
string 类 115114	FFT	false

对上述测试样例设计进行编码设计，该部分代码设计如代码 2.2-6。

代码 2.2-6 void testEquals()测试代码

void testEquals()

```
@Test
public void testEquals() throws Exception {
    JsonNumber jsonNumber = new JsonNumber("115114");
    JsonNumber jsonNumber1 = jsonNumber;
    JsonNumber jsonNumber2 = new JsonNumber("115114");
    JsonNumber jsonNumber3 = new JsonNumber("1414810");
    JsonNumber jsonNumber4 = null;
    String string = "115114";
    Assert.assertEquals(jsonNumber,jsonNumber1);
    Assert.assertEquals(jsonNumber,jsonNumber2);
    Assert.assertNotEquals(jsonNumber,jsonNumber3);
    Assert.assertNotEquals(jsonNumber,jsonNumber4);
    Assert.assertNotEquals(jsonNumber,string);
}
```

2.2.4 JsonObject

2.2.4.1 类的作用

JsonObject 是一种数据结构，可以理解为 Json 格式的数据结构（key-value 结构），可以使用 add 方法给 json 对象添加元素，也有 get 和 set，以及 remove 函数。JsonObject 可以很方便的转换成字符串，也可以很方便的把其他对象转换成 JsonObject 对象。此外，Json 不鼓励但不禁止重复名称。

2.2.4.2 JsonObject set(String name, JsonValue value)

该 set 函数将具有指定名称的成员的值设置为指定的 Json 值。如果该对象不包含具有此名称的成员，则在该对象的末尾添加一个新成员。如果此对象包含多个具有此名称的成员，则只更改最后一个成员。此方法只能用于修改现有对象。要用成员填充一个新对象，应该首选方法 add(name, value)，它要快得多（因为它不需要搜索现有成员）。

该函数代码如代码 2.2-7 所示，已标出简单分支的标号。

代码 2.2-7 JsonObject set(String name, JsonValue value)代码

JsonObject set(String name, JsonValue value)

```
public JsonObject set(String name, JsonValue value) {
  if (name == null) 【1】{
    throw new NullPointerException("name is null");【2】
  }
  if (value == null) 【3】{
    throw new NullPointerException("value is null");【4】
  }
  int index = indexOf(name);【5】
  if (index != -1) 【6】{
    values.set(index, value);【7】
  } else {
    table.add(name, names.size());【8】
    names.add(name);【9】
    values.add(value);【10】
  }
  return this;【11】
}
```

按照上述标号，对该函数进行分析，【1】【3】【5】共出现了三次判断结构，采用判定覆盖对上述代码进行测试样例设计，如表 2.2-5 所示。

表 2.2-5 JsonObject set(String name, JsonValue value)测试样例

样例	条件（1、3、5）	结果
在表里 name, value	FFF	set
不在表里 name, value	FFT	add
null, value	T--	出错 name
name, null	FT-	出错 value

对上述测试样例设计进行编码设计，该部分代码设计如代码 2.2-8。

代码 2.2-8 void testSet()测试代码

void testSet()

```java
@Test(timeout = 4000)
public void testSet01() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1;
    jsonObject1 = jsonObject0.set("111", 20);
    Assert.assertEquals("20", jsonObject0.get("111").toString());
    jsonObject1 = jsonObject0.set("111", "1414");
    Assert.assertEquals("\"1414\"", jsonObject0.get("111").toString());
}

@Test(timeout = 4000, expected = NullPointerException.class)
public void testSet02() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1;
    jsonObject1 = jsonObject0.set(null, 20);
}

@Test(timeout = 4000, expected = NullPointerException.class)
public void testSet03() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1;
    jsonObject1 = jsonObject0.set("111", (String) null);
```

2.2.5 JsonWriter

2.2.5.1 类的作用

JsonWriter 包装了 Writer，进行数据的写入，用于序列化操作。主要对于括号、引号、冒号、逗号、以及换行等进行处理，PrettyPrint 类也有类似的功能，后者对 Json 格式化的处理效果更好。

2.2.5.2 char[] getReplacementChars(char ch)

该函数具有较多的分支，需要仔细考虑。getReplacementChars 是为了将某个字符转换成一系列字符数组，比如引号转换为反引号和引号的组合，还转换了一些换行符号和 unicode 编码。该函数代码如代码 2.2-9 所示，已标出简单分支的标号。

代码 2.2-9 char[] getReplacementChars(char ch)代码

char[] getReplacementChars(char ch)

```java
public static char[] getReplacementChars(char ch) {
  if (ch > '\\') 【1】{
    if (ch < '\u2028' 【2a】|| ch > '\u2029') 【2b】{
      // The lower range contains 'a' .. 'z'. Only 2 checks required.
      return null; 【3】
    }
    return ch == '\u2028' ? UNICODE_2028_CHARS : UNICODE_2029_CHARS;
【4】
  }
```

```
    if (ch == '\\') 【5】 {
      return BS_CHARS; 【6】
    }
    if (ch > '"') 【7】{
      // This range contains '0' .. '9' and 'A' .. 'Z'. Need 3 checks to
  get here.
      return null; 【8】
    }
    if (ch == '"') 【9】 {
      return QUOT_CHARS; 【10】
    }
    if (ch > CONTROL_CHARACTERS_END) 【11】{
      return null; 【12】
    }
    if (ch == '\n') 【13】 {
      return LF_CHARS; 【14】
    }
    if (ch == '\r') 【15】 {
      return CR_CHARS; 【16】
    }
    if (ch == '\t') 【17】 {
      return TAB_CHARS; 【18】
    }
    return new char[] {'\\', 'u', '0', '0', HEX_DIGITS[ch >> 4 & 0x000f],
  HEX_DIGITS[ch & 0x000f]}; 【19】
  }
```

按照上述标号，对该函数进行分析，共出现了【1】【2a】【2b】【4】【5】【7】【9】【11】【13】【15】【17】这 11 次判断结构，采用条件覆盖对上述代码进行测试样例设计。此外 CONTROL_CHARACTERS_END = 0x001f。样例如表 2.2-6 所示。

表 2.2-6 char[] getReplacementChars(char ch)测试样例

样例	满足的条件	结果
a	1、2a	null
A	7、2a	null
1	7、2a	null
✅（\u2705）	2b	null
\u0028	1、4	{'\\', 'u', '2', '0', '2', '8'}
\u0029	1	{'\\', 'u', '2', '0', '2', '9'}
\	5	{'\\','\\'}
"	9	{'\\','"'}
#（\u0023）	11	null
\n	13	{'\\','n'}
\r	15	{'\\','r'}
\t	17	{'\\','t'}
\u001a	都不满足	{'\\', 'u', '0', '0', '1', 'a'}

对上述测试样例设计进行编码设计，该部分代码设计如代码 2.2-10。

代码 2.2-10 void testGetReplacementChars()测试代码

void testGetReplacementChars()
```
                  @Test
public void testGetReplacementChars() throws Exception {
    Assert.assertNull(JsonWriter.getReplacementChars('a'));
    Assert.assertNull(JsonWriter.getReplacementChars('A'));
    Assert.assertNull(JsonWriter.getReplacementChars('1'));
    Assert.assertNull(JsonWriter.getReplacementChars('#'));
    Assert.assertNull(JsonWriter.getReplacementChars('✅'));
    Assert.assertEquals(Arrays.toString(new char[]{'\\', 'u', '2', '0', '2', '8'}),
                        Arrays.toString(JsonWriter.getReplacementChars('\u2028')));
    Assert.assertEquals(Arrays.toString(new char[]{'\\', 'u', '2', '0', '2', '9'}),
                        Arrays.toString(JsonWriter.getReplacementChars('\u2029')));
    Assert.assertEquals(Arrays.toString(new char[]{'\\','\\'}),
                        Arrays.toString(JsonWriter.getReplacementChars('\\')));
    Assert.assertEquals(Arrays.toString(new char[]{'\\','"'}),
                        Arrays.toString(JsonWriter.getReplacementChars('\"')));
    Assert.assertEquals(Arrays.toString(new char[]{'\\','n'}),
                        Arrays.toString(JsonWriter.getReplacementChars('\n')));
    Assert.assertEquals(Arrays.toString(new char[]{'\\','r'}),
                        Arrays.toString(JsonWriter.getReplacementChars('\r')));
    Assert.assertEquals(Arrays.toString(new char[]{'\\','t'}),
                        Arrays.toString(JsonWriter.getReplacementChars('\t')));
    Assert.assertEquals(Arrays.toString(new char[]{'\\', 'u', '0', '0', '1', 'a'}),
                        Arrays.toString(JsonWriter.getReplacementChars('\u001a')));
}
```

2.2.6 WritingBuffer

2.2.6.1 类的作用

WritingBuffer 主要作为一个存储器，将输入的字符进行拼接构成完整的 Json 文件。

2.2.6.2 public void write(String str, int off, int len)

该函数是对 write 函数的重写，实现了将缓存中的字符串存入存储器中。当 buffer 剩下的长度不够，就清除 buffer，如果输入字符小于 buffer 长度，那么只输入长度前的部分，否则不做处理。当 buffer 剩下的长度足够 len 字符串放下时，输入 len 长度的字符串，fill 加上 len。该函数代码如代码 2.2-11 所示，已标出简单分支的标号。

代码 2.2-11 public void write(String str, int off, int len)代码

public void write(String str, int off, int len)
```
@Override
public void write(String str, int off, int len) throws IOException {
  if (fill > buffer.length - len)【1】 {
    flush();【2】
    if (len > buffer.length) 【3】{
      writer.write(str, off, len);【4】
      return;【5】
    }
  }
  str.getChars(off, off + len, buffer, fill);【6】
  fill += len;【7】
}
```

按照上述标号，对该函数进行分析，共出现了【1】【3】这 2 次判断结构，采用判定覆盖对上述代码进行测试样例设计。如表 2.2-7 所示。

表 2.2-7 public void write(String str, int off, int len)测试样例

样例	满足的条件（1、3）	结果
buffer 16 fill 0 str 16	F-	追加写，str 都输入，fill 16
buffer 16 fill 16 str 2	TF	覆盖写，str 都输入，fill 2
buffer 16 fill 2 str 18	TT	覆盖写，str 截断写 16 位，fill 0

对上述测试样例设计进行编码设计，该部分代码设计如代码 2.2-12。

代码 2.2-12 void testWrite()测试代码

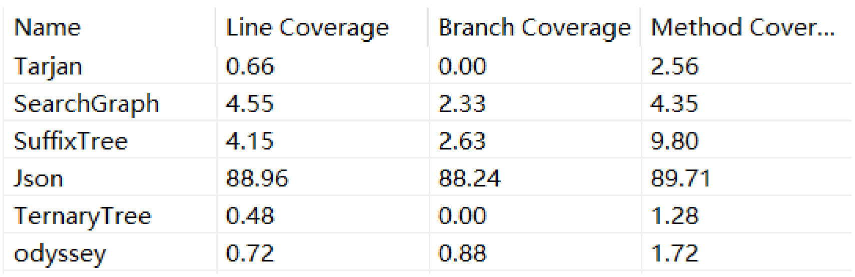
void testWrite()

```
@Test
public void testWriteForCbufOffLen() throws Exception {
    writingBuffer.write(new char[100], 0, 16);        //F-
    Assert.assertEquals(16, writingBuffer.fill);

    writingBuffer.write(new char[]{'a','b'}, 0, 2);   //TF
    Assert.assertEquals(2, writingBuffer.fill);

    writingBuffer.write(new char[100], 0, 18);        //TT
    Assert.assertEquals(0, writingBuffer.fill);

}
```

2.3 结果

2.3.1 测试结果

图 2.3-1 是 Json 文件在 MoocTest 平台递交测试返回的测试覆盖率结果，行覆盖率为 88.96%，分支覆盖率为 88.24%，方法覆盖率为 89.71%。



Name	Line Coverage	Branch Coverage	Method Cover...
Tarjan	0.66	0.00	2.56
SearchGraph	4.55	2.33	4.35
SuffixTree	4.15	2.63	9.80
Json	88.96	88.24	89.71
TernaryTree	0.48	0.00	1.28
odyssey	0.72	0.88	1.72

图 2.3-1 MoocTest 网站测试结果

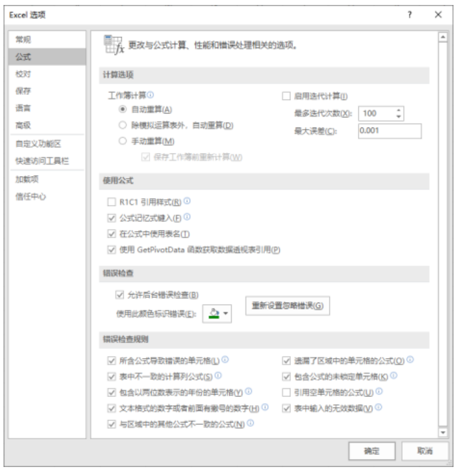
2.3.2 不可达分支说明

根据网站信息反馈可知，分支覆盖率未能达到 100%，通过分析，一方面是由于只重视了每个类单独的测试，如果结合不同类组合起来调用，进行组合测试的话，测试性能将进一步提升；另一方面是有一些类和属性是私有类型，给提升覆盖率带来一定难度，如 JsonParser；以及 Json 工程项目确实比较大，代码跳跃性强，读懂源码仍有一定的困难存在，由于时间有限，未能再提高项目的覆盖率。

3 黑盒测试

3.1 选题介绍

系统参数设置是常见的一类软件模块。图 3.1-1 是 Excel2016 的“选项”中“公式”有关的设置。请综合使用黑盒测试用例设计方法（等价类划分、边界值分析、组合测试等）生成相应的测试用例。


Excel 选项

常规
公式
校对
保存
语言
高级
自定义功能区
快速访问工具栏
加载项
信任中心

更改与公式计算、性能和错误处理相关的选项。

计算选项
工作簿计算
● 自动重算(A)
○ 除模拟运算表外，自动重算(D)
○ 手动重算(M)
☑ 保存工作簿前重新计算(W)
☐ 启用迭代计算(I)
最多迭代次数(X): 100
最大误差(C): 0.001

使用公式
☐ R1C1 引用样式(R)
☑ 公式记忆式键入(F)
☑ 在公式中使用表名(T)
☑ 使用 GetPivotData 函数获取数据透视表引用(P)

错误检查
☑ 允许后台错误检查(B)
使用此颜色标识错误(E):
重新设置忽略错误(G)

错误检查规则
☑ 所含公式导致错误的单元格(L)
☑ 表中不一致的计算列公式(S)
☑ 包含以两位数表示的年份的单元格(Y)
☑ 文本格式的数字或者前面有撇号的数字(H)
☑ 与区域中的其他公式不一致的公式(N)
☑ 遗漏了区域中的单元格的公式(O)
☑ 包含公式的未锁定单元格(K)
☐ 引用空单元格的公式(U)
☑ 表中输入的无效数据(V)

图 3.1-1 白盒测试题目

3.2 用例设计方法

3.2.1 等价类划分

等价类划分法将程序所有可能的输入数据（有效的和无效的）划分成若干个等价类。然后从每个部分中选取具有代表性的数据当做测试用例，从而进行合理的分类，对于等价类划分，主要是对于有多种类型出现的情况，或者是多种数值出现的情况而言的，观察选项窗口，在启用自动重算功能后，在计算选项右侧需要设置最多迭代次数和最大误差，是数字，因此可以考虑使用等价类划分进行测试。

其等价类划分情况如表 3.2-1 所示。

表 3.2-1 等价类划分测试设计

输入	有效等价类	无效等价类
最多迭代次数	1. 0＜最多迭代次数≤32767，整数	2. 最多迭代运算次数＜0 3. 最多迭代运算次数＞37267 4. 为空 5. 为非数字 6. 为非整数数字
最大误差	7. 最大误差≥0	8. 最大误差＜0 9. 为空 10. 为非数字

启用自动重算后，根据等价类划分，设计测试用例及其运行结果如表 3.2-2 所示。

表 3.2-2 等价类测试用例设计

测试用例	覆盖等价类	结果
最多迭代次数=10，最大误差=1	1，7	正常保存
最多迭代次数=0，最大误差=1	2，7	仅保存上一个合法结果
最多迭代次数=37268，最大误差=1	3，7	直接限制输入
最多迭代次数=，最大误差=1	4，7	仅保存上一个合法结果
最多迭代次数=h，最大误差=1	5，7	直接限制输入
最多迭代次数=1.2，最大误差=1	6，7	直接限制输入
最多迭代次数=10，最大误差=-10	1，8	仅保存上一个合法结果
最多迭代次数=10，最大误差=h	1，9	提示“输入不正确。要求输入内容为整数或小数。”
最多迭代次数=10，最大误差=	1，10	提示“输入不正确。要求输入内容为整数或小数。”

3.2.2 边界值分析

这里既可以对具有数据格式的输入框进行测试，也可以对具有多个多选框的模块的选择数量进行分析，参照等价类分类结果分析如下。

3.2.2.1 输入框

对于最多迭代次数：

- 有效等价类：1、2、16000、32766、32767
- 无效等价类：-1、-0.5、0、19.09、32768

对于最大误差：

- 有效等价类：0、1、19.09、200、1000000
- 无效等价类：-1

对于数据格式的边界值分析，将结合下面的组合测试，统一生成测试样例。

3.2.2.2 颜色选择器

点击错误检查模块中的颜色选择器，共有 1 种默认颜色和 49 种给定颜色，而不是用户输入 RGB 代码，想对此进行测试的话，可以认为这个模块内部是一个数组或者字典，标号为 1-50 号，保存了固定的颜色代码，这样也能对颜色代表的序列号进行边界值分析，可以选定第 1、2、25、49、50 个颜色进行测试，对于颜色序列号的边界值分析，将结合下面的组合测试，统一生成测试样例。

3.2.2.3 多选框

以使用错误检查规则模块为例，共有 9 个多选框，根据五点法边界值分析，可以对有 0 个、1 个、4 个、8 个、9 个选项勾选的情况进行分析，从而对该模块功能进行测试，给出一种测试样例情况如表 3.2-3。

表 3.2-3 规则检查模块多选框边界值测试样例

个数	L	S	Y	H	N	O	K	U	V
0									
1	√								
4	√	√			√				√
8	√		√	√	√	√	√	√	√
9	√	√	√	√	√	√	√	√	√

3.2.3 组合测试

为了实现组合测试，采用了 PICT 测试软件辅助自己生成测试样例。在使用 PICT 时，需要输入与测试用例相关的所有参数，以达到全面覆盖的效果。

编写的 PICT 输入代码如代码 3.2-1 所示，这里使用到了条件判断，结合上文的边界值的一些分析，从而实现输出更准确的具有实际意义的测试样例。

对于保存工作前重新计算这个（W）功能，笔者认为内部需要考虑 check 状态和 clickable 状态都要进行判断，也就是当没有实现手动重算（M）功能时，这个地方应该是不能点击的，但是可以保留勾选和不勾选的状态。对于迭代计算（I）情况，前面已经测试过不合法的情况，这里将 I 关闭的情况对应地迭代数量（maxIter）和误差（maxError）设置为负值便于统一处理。因此 PICT 的测试输入脚本中最后有两个条件。

代码 3.2-1 PICT 测试输入

input.txt

```
Cal: A, D, M
W-check: on, off
W-clickable: on, off
Auto: on, off
I: on, off
maxIter: -1, 1, 2, 16000, 32766, 32767
maxError: -1, 0, 1, 19.09, 200, 1000000
R: on, off
F: on, off
T: on, off
P: on, off
B: on, off
E: 1, 2, 25, 49, 50
G-click: yes, no
L: on, off
S: on, off
Y: on, off
H: on, off
N: on, off
Q: on, off
K: on, off
U: on, off
V: on, off
IF [Cal] <> "M" THEN [W-clickable] = "off";
IF [I] = "off" THEN [maxIter] = -1 AND [maxError] = -1;
```

在 PICT 软件所在文件夹打开 CMD 命令行，输入 `pict input.txt >`

`output.xls`，可以在 *output.xls* 文件中获得最后的组合测试结果，共给出了 44 个测试样例，展示如表 3.2-4（为方便查看，参数为 off 的地方以空格展示）。

表 3.2-4 PICT 软件根据输入文件得出的测试样例结果

Cal	W-check	W-click-able	Auto	I	max Iter	max Error	R	F	T	P	B	E	L	S	Y	H	N	Q	K	U	V
D			on	on	16000	200	on	on		on	on	1	on	on					on		on
M	on	on		on	2	0			on			49			on	on	on	on		on	
A	on		on	on	16000	-1		on	on	on		50	on			on		on			
M		on		on	16000	1	on				on	25		on	on		on	on	on	on	on
A			on	on	1	19.09	on		on	on		2		on		on	on		on	on	
M	on	on		on	32766	0		on			on	2	on		on						on
D					-1	-1	on				on	50	on		on	on	on		on	on	on
M		on	on	on	2	1E6		on		on		49		on					on		on
D	on			on	16000	0			on			1		on			on	on	on		
M	on	on	on	on	-1	200		on	on	on		25		on	on			on		on	
A	on			on	-1	1E6	on	on	on		on	1	on		on	on	on	on			
A				on	2	200	on		on		on	25	on			on	on	on			on
M	on	on		on	32767	200				on		50		on				on			
D	on		on	on	-1	1		on	on	on		49	on			on					
M			on	on	32766	1E6	on		on	on		1		on		on	on	on	on	on	
M	on	on		on	2	1		on		on		2		on				on			
M	on	on	on	on	2	-1		on	on	on		1		on							
M	on	on	on	on	32766	19.09		on			on	49	on		on			on			on
M	on	on		on	1	200		on			on	49	on		on			on			on
M	on	on		on	16000	19.09		on	on	on		25		on							
M		on		on	32766	1		on	on	on		50		on				on			
A			on	on	1	0	on	on	on	on		25		on				on			
A			on	on	32767	-1	on	on	on		on	49	on		on	on	on		on	on	on
M	on	on	on	on	1	-1		on	on	on		25		on				on			
M	on	on		on	-1	19.09		on		on		2		on				on			
D			on	on	32767	1E6		on	on	on		2		on				on			
M	on	on	on	on	32766	200		on	on	on		25		on				on			
M	on	on		on	1	1E6		on		on		50		on				on			
M	on	on	on	on	32767	19.09		on	on	on		1		on				on			
D			on	on	16000	1E6		on		on		25		on				on			
M		on		on	32767	0		on		on		25		on				on			
M	on	on	on	on	32766	-1		on	on	on		2		on				on			
D			on	on	2	19.09		on		on		50		on				on			
M	on	on	on		-1	-1		on	on	on		49		on				on			
A				on	16000	200	on			on	on	2		on		on				on	
D	on		on	on	1	1				on	on	1	on	on			on			on	on
M		on			-1	-1	on		on		on	1	on				on			on	
A	on		on	on	32766	1		on		on		2		on				on	on	on	on
A	on				-1	-1		on				25	on	on			on				on
A				on	32767	1			on	on		2	on	on				on		on	
D	on				-1	-1	on				on	2	on				on	on	on		
A				on	-1	0	on		on		on	50	on			on				on	on
D	on		on	on	32766	200	on	on		on		2			on	on			on	on	on
D	on		on	on	16000	1	on			on	on	49			on		on	on		on	

3.2.4 其他（基于用户使用或者经验）

3.2.4.1 快捷键

除了使用鼠标测试点击多选框之外进行选择之外，还应该测试每条文本对应地快捷键能否正常使用，使用快捷键对选项卡进行测试。

3.2.4.2 左侧栏

左侧栏的测试首先要考虑是否能跳转到其他页签，若能跳转，要测试跳转前后信息是否一致，特别关注是否切换后前面未保存的数据直接变为空白，而不是临时保存。

3.2.4.3 帮助

在每个选框的提示词附近悬浮光标，一段时间后将出现提示，需要测试能否正常弹出悬浮框，或者正确提示问题信息。选项卡右上角同样也有问号需要进行测试。

3.2.4.4 撤销、取消、确定

撤销是通过右上角直接退出、取消是当前做出的选择清空退出，确定是对当前选择的事物进行应用，三种退出类型都需要进行测试，查看选项是否生效。

4 性能测试

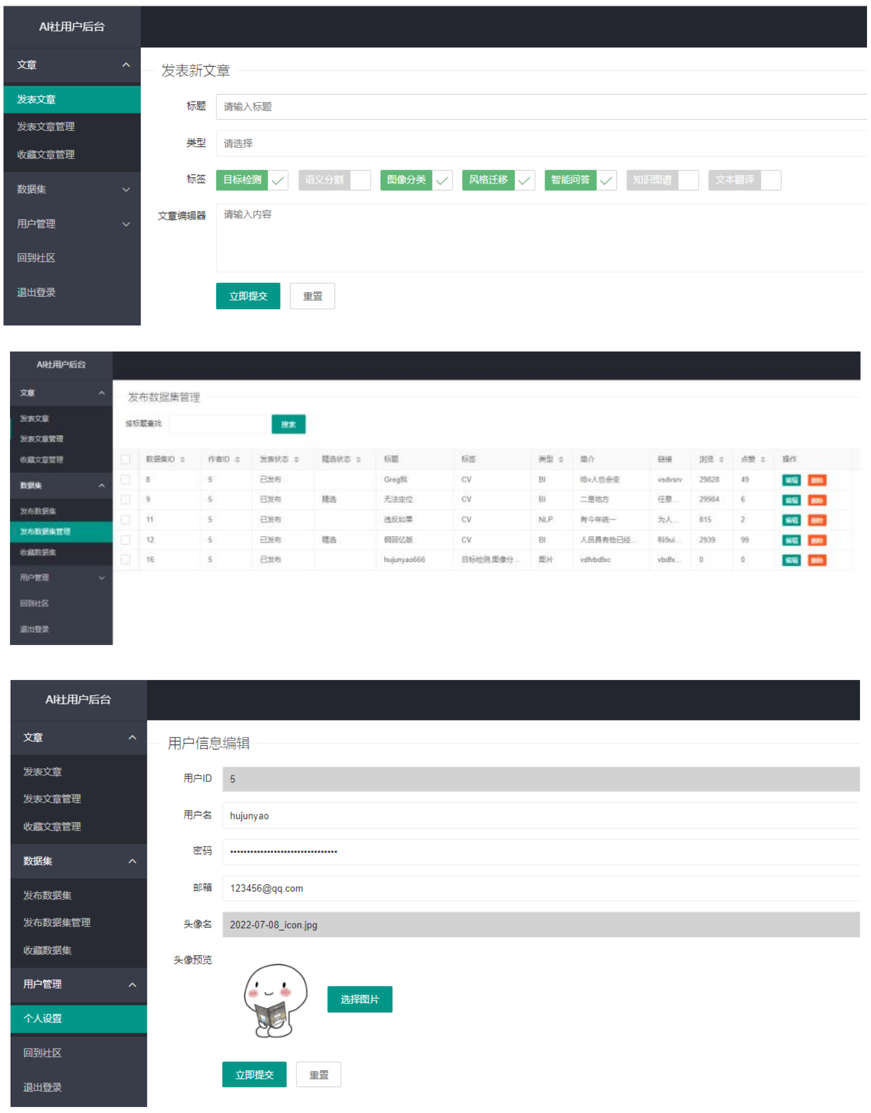
4.1 选题介绍

4.1.1 JMeter 介绍

Apache JMeter 是 Apache 组织基于 Java 开发的压力测试工具，用于对软件做压力测试，被设计用于 Web 应用测试，可对服务器、网络或对象模拟巨大的负载，在不同压力类别下测试它们的强度和分析整体性能。另外，JMeter 能够对应用程序做功能/回归测试，通过创建带有断言的脚本来验证程序是否返回了期望结果。

4.1.2 被测试 Web 网站介绍

在《Web 应用开发技术》这门课程中，笔者设计了“AI 社”人工智能技术交流论坛后台管理系统，网站的主要功能是用户发布、编辑、发布自己的文章和数据集，和用户进行分享交流，管理系统主要模块展示如图 4.1-1 所示（创建文章、管理数据集、用户信息修改）。



数据集ID	作者ID	发表状态	精选状态	标题	标签	类型	简介	链接	浏览	点赞	操作
8	5	已发布		Greg我	CV	BI	给v人也会变	vsdvsrv	29828	49	编辑 删除
9	5	已发布	精选	无法定位	CV	BI	二是地方	任意...	29984	6	编辑 删除
11	5	已发布		违反如果	CV	NLP	有今年统一	为人...	815	2	编辑 删除
12	5	已发布	精选	钢回亿版	CV	BI	人员具有他已经...	科9ui...	2939	99	编辑 删除
16	5	已发布		hujunyao666	目标检测,图像分...	图片	vdfvbdfxc	vbdfx...	0	0	编辑 删除

用户ID	5
用户名	hujunyao
密码	································
邮箱	123456@qq.com
头像名	2022-07-08_icon.jpg

图 4.1-1 管理系统主要模块（创建文章、管理数据集、用户信息修改）

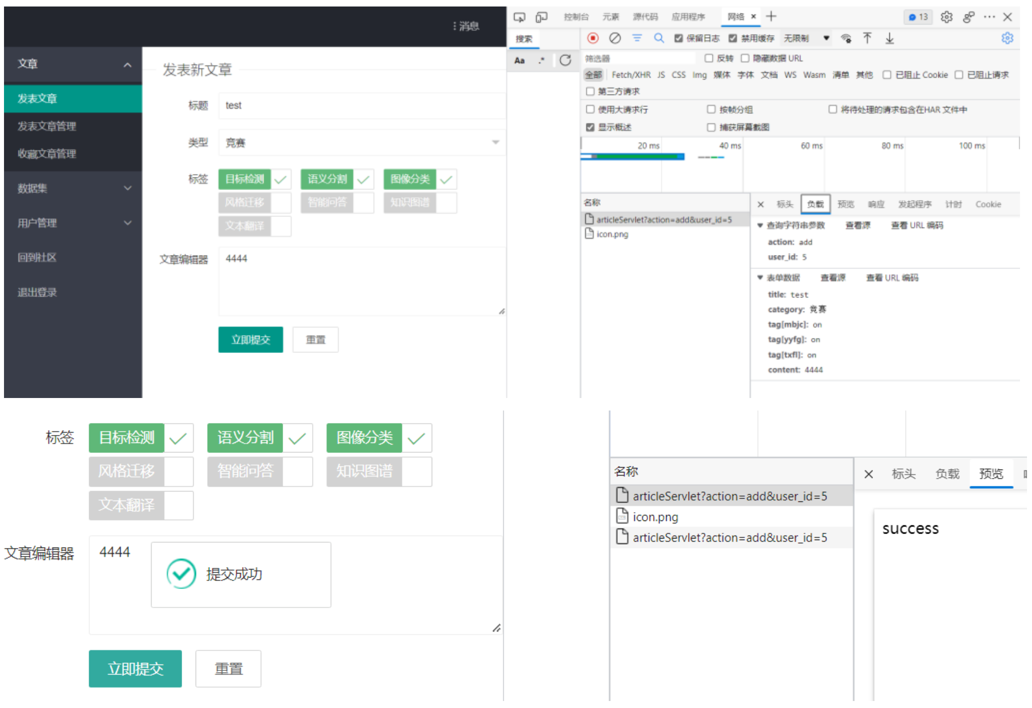
4.2 压力测试

下面将利用 JMeter 对本网站的该业务进行性能测试。

4.2.1 被测试模块（创建新文章）介绍

创建新文章是系统的一个重要模块，因此选择文章创建作为网站的性能测试的目标。创建文章页面主要有六个部分构成：文章标题（单行文本框）、文章类型（含有三项和默认空项的下拉列表）、标签（一组七个多选框标签）、正文（多行文本框）、提交（提交到服务器保存的按钮）、重置（清空当前页面本文的按钮）。

发表新文章的流程如下：当用户将测试数据（编辑好的文章）点击立即提交上传数据库时，页面发起 Post 方法，使用 add 请求修改 user_id 为 5（当前登录用户）的信息，同时在传输过程中附上了网页对应的表单数据，后端接收并成功完成新增数据任务后，正常时应返回 success 信息，当前端接收到消息后，弹出提交成功弹窗，表明已经成功添加新文章，过程中相关发送和接收状态如图 4.2-1 所示。


发表新文章

标题 test

类型 竞赛

标签 目标检测 ✓ 语义分割 ✓ 图像分类 ✓ 风格迁移 智能问答 知识图谱 文本翻译

文章编辑器 4444

提交成功

articleServlet?action=add&user_id=5

icon.png

查询字符串参数

action: add

user_id: 5

表单数据

title: test

category: 竞赛

tag[mbjc]: on

tag[yyfg]: on

tag[txfl]: on

content: 4444

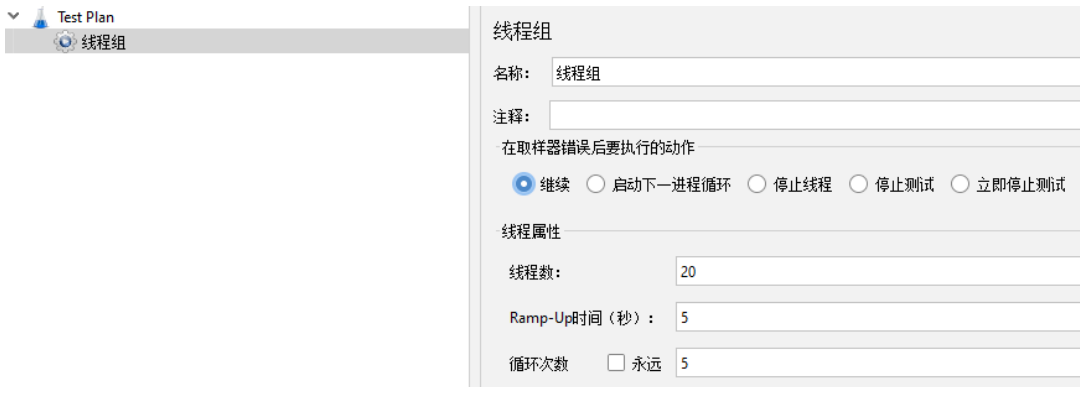
success

图 4.2-1 新增文章过程中的发送和接收的消息

4.2.2 创建过程

4.2.2.1 新建一个线程组

在左侧栏右键 Test Plan，点击添加，线程（用户），线程组，从而创建一个新线程组。在右侧设置线程组相关的参数。线程数即虚拟用户数。Ramp-Up 时间是虚拟用户数全部启动的时长。循环次数是每个线程发送请求的个数。如果勾选了“永远”，那么所有线程会一直发送请求，直到手动停止按钮，或者设置的线程时间结束。对本程序设置 20 个线程（20 个虚拟用户），启动时间设置为 5s，循环次数为 5 次，界面如图 4.2-2 所示。

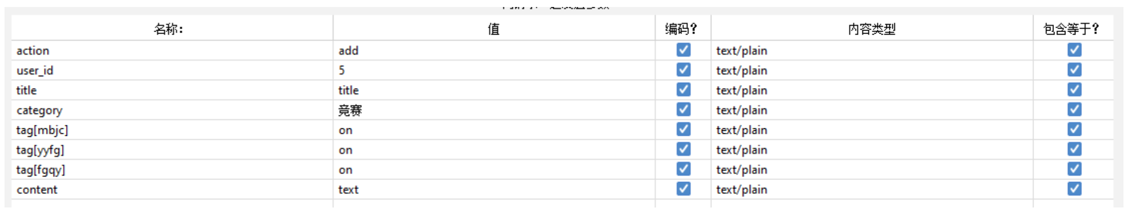

Test Plan
线程组

线程组
名称： 线程组
注释：
在取样器错误后要执行的动作
[✓ 继续] 启动下一进程循环 停止线程 停止测试 立即停止测试
线程属性
线程数： 20
Ramp-Up时间（秒）： 5
循环次数 永远 5

图 4.2-2 新建线程组参数设置

4.2.2.2 新建一个 HTTP 请求

在左侧栏右键刚刚创建的线程组，点击添加，取样器，HTTP 请求，从而创建一个新线程组。设置好协议类型、服务器、端口、请求类型、路径、编码、以及同请求一起发送的各项参数，如图 4.2-3 所示。

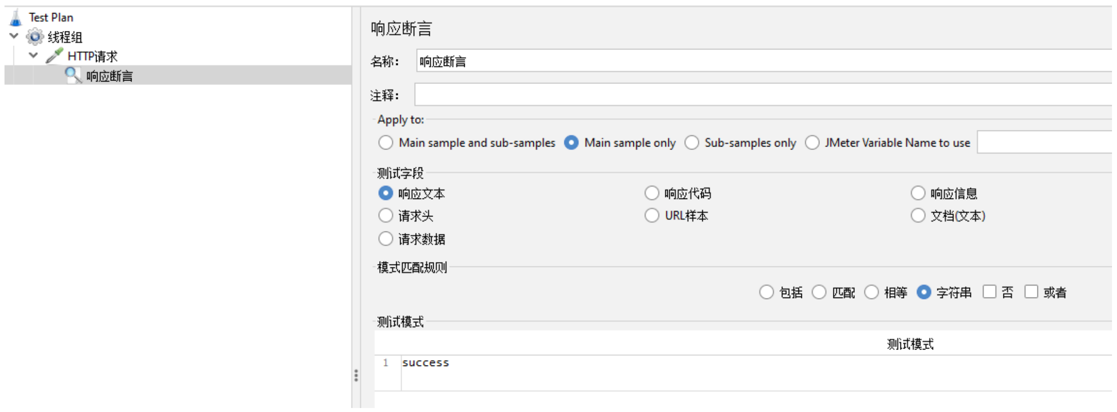


名称:	值	编码?	内容类型	包含等于?
action	add	✓	text/plain	✓
user_id	5	✓	text/plain	✓
title	title	✓	text/plain	✓
category	竞赛	✓	text/plain	✓
tag[mbjc]	on	✓	text/plain	✓
tag[yyfg]	on	✓	text/plain	✓
tag[fgqy]	on	✓	text/plain	✓
content	text	✓	text/plain	✓

图 4.2-3 新建 HTTP 请求参数设置

4.2.2.3 新建断言

在左侧栏右键刚刚创建的 HTTP 请求，点击添加断言，响应断言，从而创建一个新断言。在这次的测试系统中，如果成功响应那么响应结果就是 success 字符串，因此设置如图 4.2-4 所示，只有返回 success 字符串才认为是正确的。

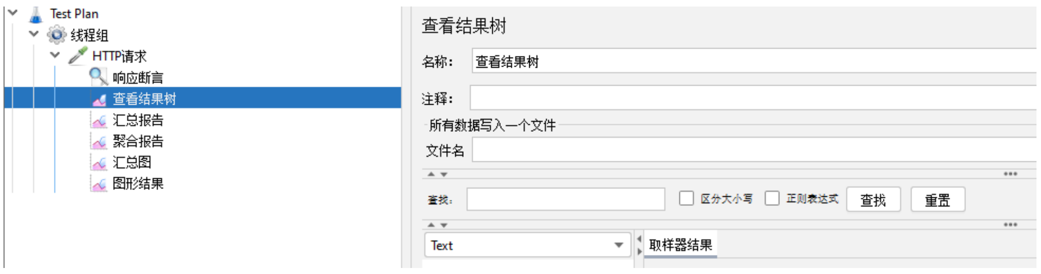

Test Plan
线程组
HTTP请求
响应断言

响应断言
名称： 响应断言
注释：
Apply to:
Main sample and sub-samples [✓ Main sample only] Sub-samples only JMeter Variable Name to use
测试字段
[✓ 响应文本] 响应代码 响应信息
请求头 URL样本 文档(文本)
请求数据
模式匹配规则
包括 匹配 相等 [✓ 字符串] 否 或者
测试模式
测试模式
success

图 4.2-4 断言参数设置

4.2.2.4 添加监听器

JMeter 有许多 UI Listener，可用于直接在 JMeter UI 中查看结果，应选择合适的结果呈现方式，由于笔者是初次使用，因此选择多种呈现方式进行学习研究。在左侧栏右键刚刚创建的 HTTP 请求，点击添加，监听器，选择查看结果树、汇总报告、聚合报告、汇总图、图形结果。添加结果如图 4.2-5 所示。



查看结果树

名称：查看结果树

注释：

所有数据写入一个文件

文件名

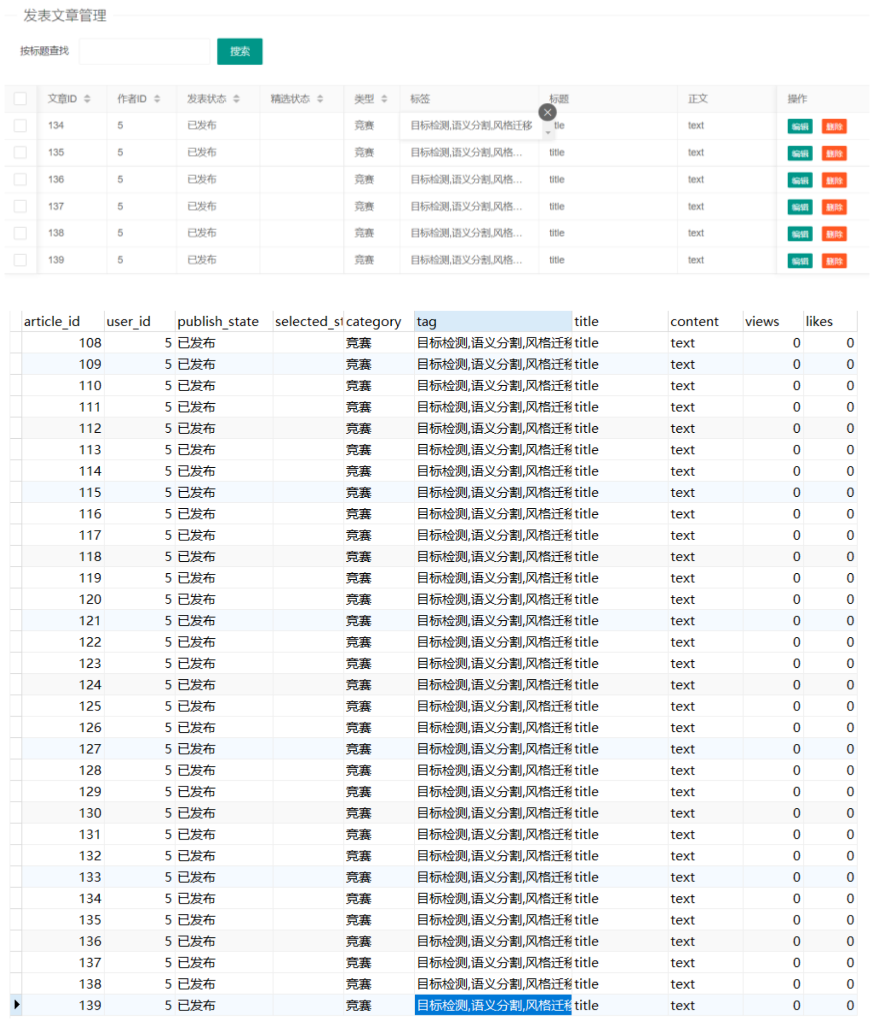
图 4.2-5 添加监听器

4.2.3 结果验证与分析

点击运行图标便可运行测试，等待运行结束，观察运行结果。

4.2.3.1 结果验证

首先查看 JMeter 是否真的插入了设置的数据，在 Web 网站的文章管理界面可以看到已经插入了新的数据，使用 Navicat 查看数据库，也可以发现已经新插入了 100 条数据，也就是多线程测试时添加的数据，验证插入数据有效，页面和数据库展示如图 4.2-6 所示。


发表文章管理

按标题查找 搜索

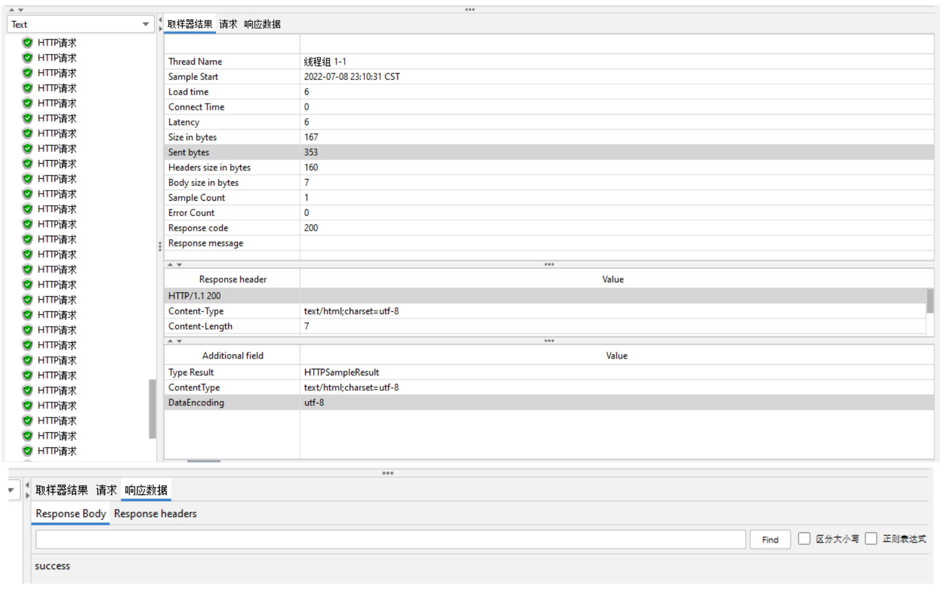
文章ID	作者ID	发表状态	精选状态	类型	标签	标题	正文	操作
134	5	已发布		竞赛	目标检测,语义分割,风格迁移	tle	text	编辑 删除
135	5	已发布		竞赛	目标检测,语义分割,风格...	title	text	编辑 删除
136	5	已发布		竞赛	目标检测,语义分割,风格...	title	text	编辑 删除
137	5	已发布		竞赛	目标检测,语义分割,风格...	title	text	编辑 删除
138	5	已发布		竞赛	目标检测,语义分割,风格...	title	text	编辑 删除
139	5	已发布		竞赛	目标检测,语义分割,风格...	title	text	编辑 删除

article_id	user_id	publish_state	selected_st	category	tag	title	content	views	likes
108	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
109	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
110	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
111	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
112	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
113	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
114	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
115	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
116	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
117	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
118	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
119	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
120	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
121	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
122	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
123	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
124	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
125	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
126	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
127	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
128	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
129	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
130	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
131	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
132	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
133	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
134	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
135	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
136	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
137	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
138	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0
139	5	已发布		竞赛	目标检测,语义分割,风格迁	title	text	0	0

图 4.2-6 文章管理界面和数据库验证展示

4.2.3.2 查看结果树

查看结果树解释为显示所有样本响应的树，允许查看任何样本的响应，但笔者感觉界面其实也没有体现出一个树状的感觉，更多像是按时间顺序的列表，其中可以对类型进行选择和过滤。对于每一条数据可以查看响应的线程名、采样时间、数据大小、响应情况等等信息，界面如图 4.2-7 所示。



Thread Name	线程组 1-1
Sample Start	2022-07-08 23:10:31 CST
Load time	6
Connect Time	0
Latency	6
Size in bytes	167
Sent bytes	353
Headers size in bytes	160
Body size in bytes	7
Sample Count	1
Error Count	0
Response code	200
Response message	

Response header	Value
HTTP/1.1 200	
Content-Type	text/html;charset=utf-8
Content-Length	7

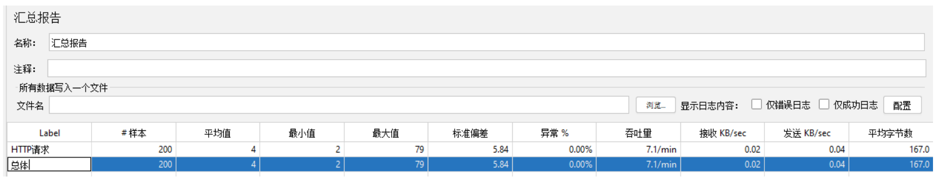
Additional field	Value
Type Result	HTTPSampleResult
ContentType	text/html;charset=utf-8
DataEncoding	utf-8

success

图 4.2-7 查看结果树界面展示

4.2.3.3 汇总报告

汇总报告为测试中的每个不同类型的请求创建一个表行，主要列出了样本数量为 200、平均值为 4、吞吐量 7.1/分、收发速率 0.02KB/秒等消息，其页面如图 4.2-8 所示。



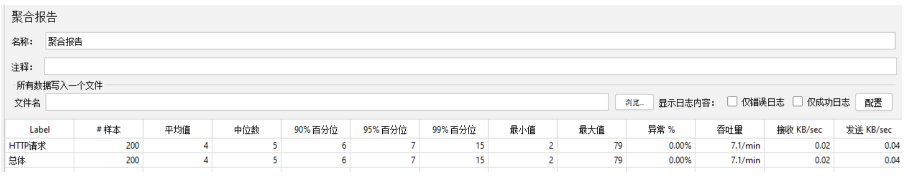
汇总报告
名称： 汇总报告
注释：
所有数据写入一个文件
文件名

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	200	4	2	79	5.84	0.00%	7.1/min	0.02	0.04	167.0
总体	200	4	2	79	5.84	0.00%	7.1/min	0.02	0.04	167.0

图 4.2-8 汇总报告页面展示

4.2.3.4 聚合报告

与汇总报告相似，还多了对百分位的一些统计，不做赘述，界面如图 4.2-9 所示。



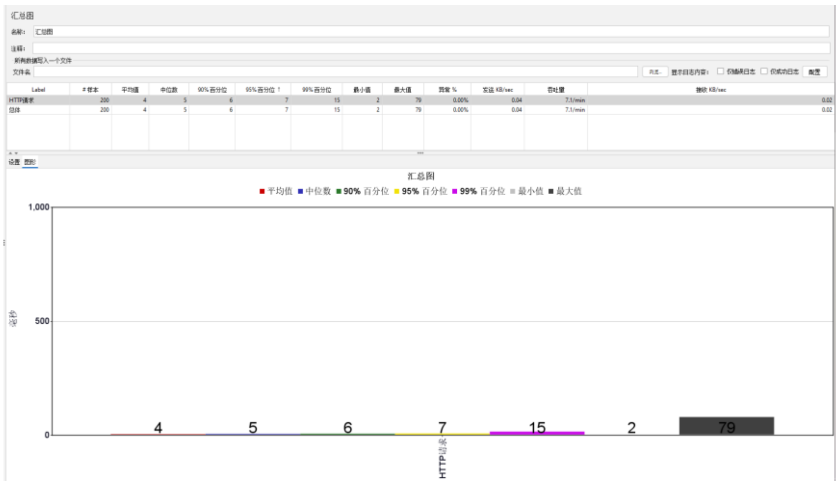
聚合报告
名称： 聚合报告
注释：
所有数据写入一个文件
文件名

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 %	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	200	4	5	6	7	15	2	79	0.00%	7.1/min	0.02	0.04
总体	200	4	5	6	7	15	2	79	0.00%	7.1/min	0.02	0.04

图 4.2-9 聚合报告页面展示

4.2.3.5 汇总图

汇总图除了有类似报告的表格之外，对请求响应的时间进行了绘图，如图 4.2-10 所示。


汇总图

名称： 汇总图

注释：

所有数据写入一个文件

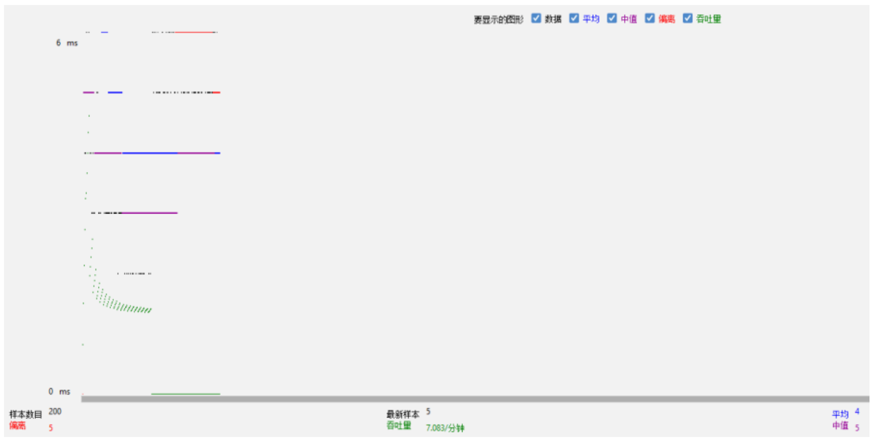
文件名 浏览... 显示日志内容： ☐ 仅错误日志 ☐ 仅成功日志 配置

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 %	发送 KB/sec	吞吐量	接收 KB/sec
HTTP请求	200	4	5	6	7	15	2	79	0.00%	0.04	7.1/min	0.02
总体	200	4	5	6	7	15	2	79	0.00%	0.04	7.1/min	0.02

图 4.2-10 汇总图结果展示

4.2.3.6 图形结果

图形结果生成了一个简单的散点图，其绘制了所有采样时间、吞吐量等数据，以可视化的形式展现数据，页面如图 4.2-11 所示。



样本数目 200
偏离 5
最新样本 5
吞吐量 7.083/分钟
平均 4
中值 5

图 4.2-11 图形结果展示

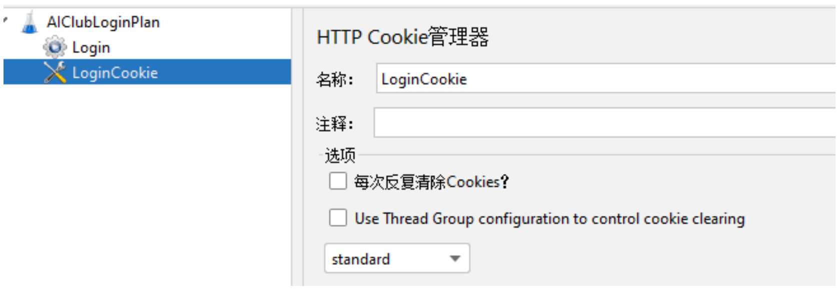
4.3 录制、优化、回放脚本

下面对网站登录的过程进行脚本的录制以及进行其他操作。

4.3.1 录制、优化脚本

4.3.1.1 添加 HTTP Cookie 管理器

首先需要创建测试计划、创建线程组，该步骤与上文大致相同，不再赘述，此时只保留一个线程。然后右键测试计划，添加元件，添加 HTTP Cookie 管理器，从而创建 HTTP Cookie 管理器，添加之后不需要任何配置，录制后，脚本中便保存用户登录的相关信息，界面如图 4.3-1 所示。



HTTP Cookie管理器

名称：LoginCookie

注释：

选项

☐ 每次反复清除Cookies?

☐ Use Thread Group configuration to control cookie clearing

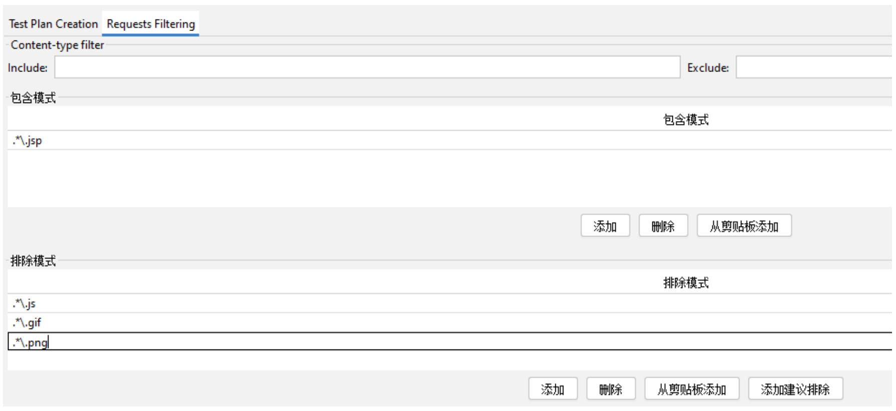
standard

图 4.3-1 Cookie 管理器界面展示

4.3.1.2 HTTP 代理服务器、优化脚本（过滤器）

然后右键测试计划，添加非测试元件，添加 HTTP 代理服务器，从而创建 HTTP 代理服务器。按照下图中进行相应配置，端口为 8002，目标控制器选择登录，分组为每个组放入一个新的控制器。

在测试脚本中有一些信息是没有任何作用的，所以在录制脚本的时候可以通过 HTTP 代理服务器的包含模式和排除模式，保留和排除一些根据需求来确定的请求信息，如 gif、png 等等。例如，在包含模式处填上“.*\.jsp”，请求 jsp 文件时录制脚本。在排除模式处填上".*\.png"，在请求 png 文件时不录制脚本，如图 4.3-2 所示。



Content-type filter

Include:　　Exclude:

包含模式

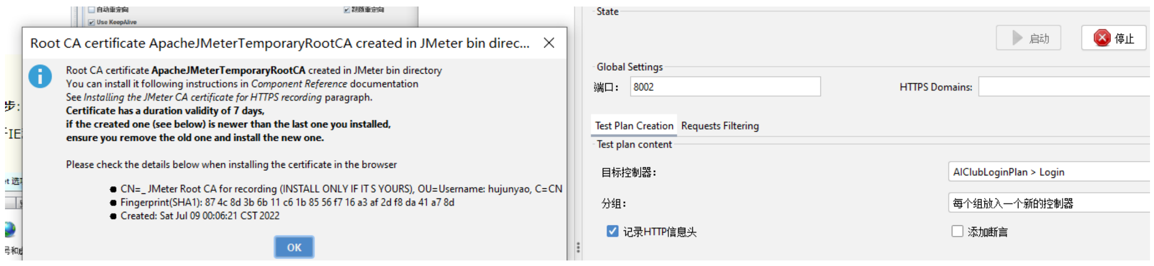
包含模式
.*\.jsp

排除模式

排除模式
.*\.js
.*\.gif
.*\.png

图 4.3-2 包含模式和排除模式

配置完成，点击启动，点击 OK，如图 4.3-3 所示。



Root CA certificate ApacheJMeterTemporaryRootCA created in JMeter bin direc...

Root CA certificate **ApacheJMeterTemporaryRootCA** created in JMeter bin directory
You can install it following instructions in *Component Reference* documentation
See *Installing the JMeter CA certificate for HTTPS recording* paragraph.
Certificate has a duration validity of 7 days,
if the created one (see below) is newer than the last one you installed,
ensure you remove the old one and install the new one.

Please check the details below when installing the certificate in the browser

- CN=_ JMeter Root CA for recording (INSTALL ONLY IF IT S YOURS), OU=Username: hujunyao, C=CN
- Fingerprint(SHA1): 87 4c 8d 3b 6b 11 c6 1b 85 56 f7 16 a3 af 2d f8 da 41 a7 8d
- Created: Sat Jul 09 00:06:21 CST 2022

State

Global Settings

端口：8002　　HTTPS Domains:

Test plan content

目标控制器：AIClubLoginPlan > Login

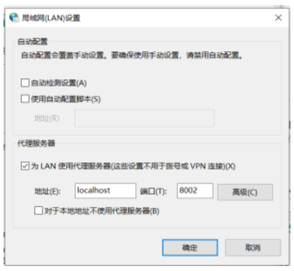
分组：每个组放入一个新的控制器

☑ 记录HTTP信息头　　☐ 添加断言

图 4.3-3 HTTP 代理服务器设置

4.3.1.3 设置代理，录制脚本

打开浏览器，设置 Internet 选项，连接，局域网设置，设置代理服务器为 localhost，端口为上面设置的 8002，设置如图 4.3-4 所示。



局域网(LAN)设置

自动配置

自动配置会覆盖手动设置。要确保使用手动设置，请禁用自动配置。

☐ 自动检测设置(A)

☐ 使用自动配置脚本(S)

地址(R)

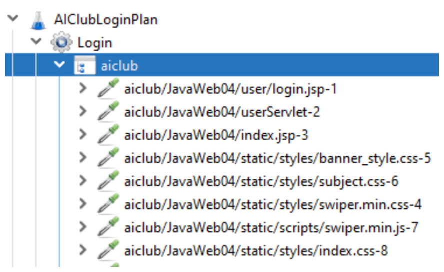
代理服务器

☑ 为 LAN 使用代理服务器(这些设置不用于拨号或 VPN 连接)(X)

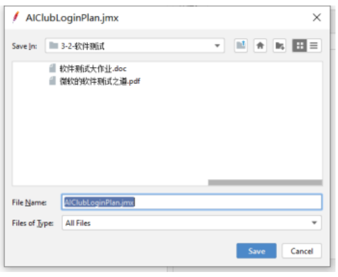
地址(E): localhost 端口(T): 8002 高级(C)

☐ 对于本地地址不使用代理服务器(B)

确定 取消



```
AIClubLoginPlan
  Login
    aiclub
      aiclub/JavaWeb04/user/login.jsp-1
      aiclub/JavaWeb04/userServlet-2
      aiclub/JavaWeb04/index.jsp-3
      aiclub/JavaWeb04/static/styles/banner_style.css-5
      aiclub/JavaWeb04/static/styles/subject.css-6
      aiclub/JavaWeb04/static/styles/swiper.min.css-4
      aiclub/JavaWeb04/static/scripts/swiper.min.js-7
      aiclub/JavaWeb04/static/styles/index.css-8
```



AIClubLoginPlan.jmx

Save In: 3-2-软件测试

软件测试大作业.doc

微软的软件测试之道.pdf

File Name: AIClubLoginPlan.jmx

Files of Type: All Files

Save Cancel

图 4.3-4 脚本局域网选项设置（左）

图 4.3-5 录制线程组中记录的信息（中）

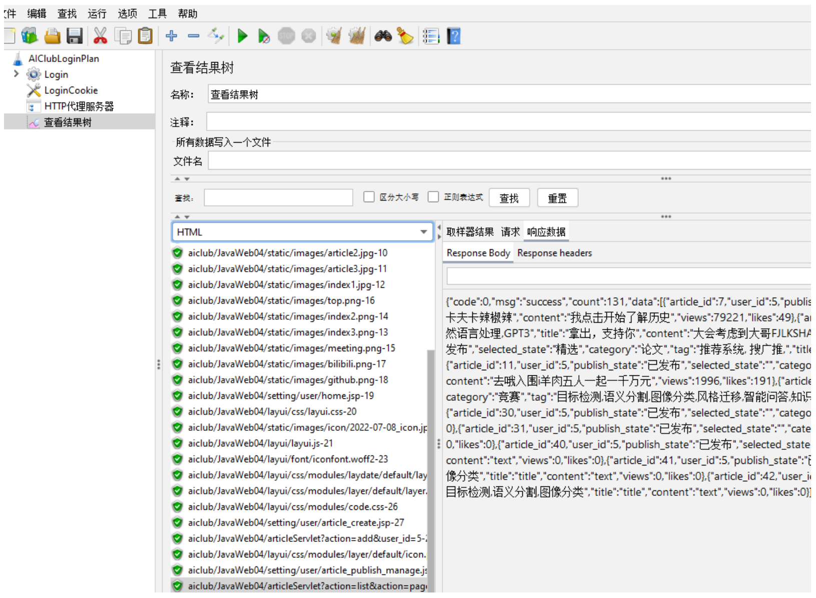
图 4.3-6 保存脚本（右）

4.3.1.4 运行后结束，保存

由于程序在本地进行运行，若一直失败，无法获得脚本，则可尝试通过将网址中的 localhost 修改为本机外网 IP 地址再进入网站，这样就可以获得脚本，适当进行运行后，结束脚本，可以在 JMeter 的线程组中查看各种请求，如图 4.3-5 所示。右键测试计划，点击选中部分保存为，选择制定好的目录即可保存脚本，其缀为 jmx，如图 4.3-6 所示。

4.3.1.5 回放脚本

先添加查看结果树，然后点击运行，即可进行脚本的回放，例如查看 articleServlet 的 list 动作这条请求，返回的响应是当前页面的文章列表的 Json 文件，其中含有响应码，消息，数据总数，以及文章列表子 Json 文件，具体如图 4.3-7 所示。



名称: 查看结果树

注释:

文件名

查找:

HTML

```
aiclub/JavaWeb04/static/images/article2.jpg-10
aiclub/JavaWeb04/static/images/article3.jpg-11
aiclub/JavaWeb04/static/images/index1.jpg-12
aiclub/JavaWeb04/static/images/top.png-16
aiclub/JavaWeb04/static/images/index2.png-14
aiclub/JavaWeb04/static/images/index3.png-13
aiclub/JavaWeb04/static/images/meeting.png-15
aiclub/JavaWeb04/static/images/bilibili.png-17
aiclub/JavaWeb04/static/images/github.png-18
aiclub/JavaWeb04/setting/user/home.jsp-19
aiclub/JavaWeb04/layui/css/layui.css-20
aiclub/JavaWeb04/static/images/icon/2022-07-08_icon.jp
aiclub/JavaWeb04/layui/layui.js-21
aiclub/JavaWeb04/layui/font/iconfont.woff2-23
aiclub/JavaWeb04/layui/css/modules/laydate/default/lay
aiclub/JavaWeb04/layui/css/modules/layer/default/layer.
aiclub/JavaWeb04/layui/css/modules/code.css-26
aiclub/JavaWeb04/setting/user/article_create.jsp-27
aiclub/JavaWeb04/articleServlet?action=add&user_id=5-
aiclub/JavaWeb04/layui/css/modules/layer/default/icon.
aiclub/JavaWeb04/setting/user/article_publish_manage.js
aiclub/JavaWeb04/articleServlet?action=list&action=pag
```

```
{"code":0,"msg":"success","count":131,"data":[{"article_id":7,"user_id":5,"publis
卡夫卡辣椒辣","content":"我点击开始了解历史","views":79221,"likes":49},{"ar
然语言处理,GPT3","title":"拿出，支持你","content":"大会考虑到大哥FJLKSHA
发布","selected_state":"精选","category":"论文","tag":"推荐系统, 搜广推,","title
{"article_id":11,"user_id":5,"publish_state":"已发布","selected_state":"","catego
content":"去哦入围i羊肉五人一起一千万元","views":1996,"likes":191},{"article
category":"竞赛","tag":"目标检测,语义分割,图像分类,风格迁移,智能问答,知识
{"article_id":30,"user_id":5,"publish_state":"已发布","selected_state":"","catego
0},{"article_id":31,"user_id":5,"publish_state":"已发布","selected_state":"","cate
0,"likes":0},{"article_id":40,"user_id":5,"publish_state":"已发布","selected_state
content":"text","views":0,"likes":0},{"article_id":41,"user_id":5,"publish_state":"
像分类","title":"title","content":"text","views":0,"likes":0},{"article_id":42,"user_i
目标检测,语义分割,图像分类","title":"title","content":"text","views":0,"likes":0}
```

图 4.3-7 回放脚本

5 总结感悟

尽管《软件测试 A》已经结课，但是自己仍意犹未尽，感到收获非常丰富：

一是在老师深入浅出的教学之下，自己对于完整的测试体系有了更深入的认识，在实践操作能力方面也有了很大增长，在短时间内对不同的软件测试方法和测试工具快速上手，并且完成效果较好，这是一件很值得骄傲的事情，我们宿舍的同学也往往为了提高一点点覆盖率弄到深更半夜，不改好不罢休，尝试覆盖分支的过程真的其乐无穷；

二是对于学科课程的整体性有了新的认识。这学期所学 JavaWeb 课程有了非常全面的开发实操，其中进行了数据库连接测试和增删改查测试，同时结合所学《软件工程》课程知识，在设计测试样例、模块化测试、整体测试、网站性能测试等方面都得到了或多或少的实践，我真切地感受到了计算机科学与技术作为一门学科，它的课程是具有整体性的，而不是简单的、单一的课程，需要自己找到课程的共性，从而有机结合；

三是解决问题的能力得到了提升，尤其是阅读 Java 项目开发的能力、独立思考解决问题的能力不断提高，通过独立分析、上网查找、同学讨论找到解决问题的办法。特别是了解到了自动化测试软件如 Evosuite 的便利性，但同时也认识到自动化测试的弊端，不能滥用，还需要尽可能理解样例为什么这样生成，否则只为了提高覆盖率毫无意义。

写到这里，仍然觉得有些遗憾，由于这个月压力很大，时间有限，未能尝试老师所说的安卓移动端 Applium 测试等测试软件和方法。但是在整个课程周期中，自己慢慢摸索，也总结出了不少经验，相信自己今后也可以对别的测试软件或者别的项目快速上手，迅速解决问题，我也有信心在之后的学习工作中，不断提高自身技术，不断追求进步。总体上，仍十分难忘整个过程。

参考文献

[1] 金琳.利用平台漏洞恶意注册网络虚拟账号套取优惠券的认定[J].中国检察官,2020(04):8-12.

[2] 朱菊,王志坚,杨雪.基于数据驱动的软件自动化测试框架[J].计算机技术与发展,2006(05):68-70.

[3] 冯玉才,唐艳,周淳.关键字驱动自动化测试的原理和实现[J].计算机应用,2004(08):140-142.

[4] 郭佳,杨涛,贾定.一种嵌入式软件自动化测试技术分析与研究[J].汽车实用技术,2021,46(22):84-87.DOI:10.16638/j.cnki.1671-7988.2021.022.022.

[5] 彭洪超. 基于 Android 系统的自动化测试用例的实现和优化[D].南京邮电大学,2020.DOI:10.27251/d.cnki.gnjdc.2020.000513.

[6] 冯洋,夏志龙,郭安,陈振宇.自动驾驶软件测试技术研究综述[J].中国图象图形学报,2021,26(01):13-27.

[7] 张超永,王振,王鹏,浮明军.人工智能电缆隧道移动巡检软件测试研究及实施[J].工业控制计算机,2021,34(09):34-36.

[8] 侯殿君.人工智能软件测试的研究和应用[J].电子测试,2019(04):117-118+126.DOI:10.16520/j.cnki.1000-8519.2019.04.048.

附　录

第 2 部分 白盒测试源码

代码 4.3-1 JsonTest 类

JsonTest.java

```
package mooctest;

import net.mooctest.Json;
import net.mooctest.JsonArray;
import net.mooctest.JsonObject;
import net.mooctest.JsonValue;
import org.junit.Assert;
import org.junit.Test;

import java.io.IOException;
import java.io.Reader;
import java.io.StringReader;
import java.util.List;

/**
* Json Tester.
*
* @author Hujunyao CUMT CS 2019-4 06192081
* @since <pre>6 月 29, 2022</pre>
* @version 1.0
*/
public class JsonTest {

/**
*
* Method: value(int value)
*
*/
@Test
public void testValueValue() throws Exception {
    int num1 = 1909;
    long num2 = 19092022;
    float num3 = 1909.22f;
    double num4 = -0.6192081;
    JsonValue jsonValue;
    jsonValue = Json.value(num1);
    Assert.assertEquals("1909",jsonValue.toString());
    jsonValue = Json.value(num2);
    Assert.assertEquals("19092022",jsonValue.toString());
    jsonValue = Json.value(num3);
    Assert.assertEquals("1909.22",jsonValue.toString());
    jsonValue = Json.value(num4);
    Assert.assertEquals("-0.6192081",jsonValue.toString());
}

@Test (expected = IllegalArgumentException.class)
public void testValueValue01() throws Exception {
    float num = Float.POSITIVE_INFINITY;
    Json.value(num);
}

@Test (expected = IllegalArgumentException.class)
public void testValueValue02() throws Exception {
    float num = Float.NaN;
```

JsonTest.java

```
    Json.value(num);
}

@Test (expected = IllegalArgumentException.class)
public void testValueValue03() throws Exception {
    double num = Double.NaN;
    Json.value(num);
}

@Test (expected = IllegalArgumentException.class)
public void testValueValue04() throws Exception {
    double num = Double.NaN;
    Json.value(num);
}

    /**
*
* Method: value(String string)
*
*/
@Test
public void testValueString() throws Exception {
    String string;
    string = null;
    Assert.assertEquals("null", Json.value(string).toString());
    string = "hujunyao";
    Assert.assertEquals("\"hujunyao\"", Json.value(string).toString());
}

/**
 *
 * Method: value(Boolean bool)
 *
 */
@Test
public void testValueBool() throws Exception {
    Assert.assertEquals("true", Json.value(true).toString());
    Assert.assertEquals("false", Json.value(false).toString());
}

    /**
*
* Method: array()
*
*/
@Test
public void testArray() throws Exception {
    JsonArray jsonarray = Json.array();
    Assert.assertTrue(jsonarray.isEmpty());
}

/**
*
* Method: array(int... values)
*
*/
@Test
public void testArrayValues01() throws Exception {
    Assert.assertEquals(4, Json.array(-1, 0, 2, 10).size());
```

JsonTest.java

```
    Assert.assertEquals(4, Json.array(-1, 0, 2, 6192081L).size());
    Assert.assertEquals(4, Json.array(-1, 0, 2, 1909.2022f).size());
    Assert.assertEquals(4, Json.array(-1, 0, 2, 0.6192081).size());
    Assert.assertEquals(3, Json.array("cumt", "hujunyao",
"06192081").size());
    Assert.assertEquals(3, Json.array(true, true, false).size());
}

@Test(expected = NullPointerException.class)
public void testArrayValues02() throws Exception {
    int[] array = null;
    Json.array(array);
}

@Test(expected = NullPointerException.class)
public void testArrayValues03() throws Exception {
    float[] array = null;
    Json.array(array);
}

@Test(expected = NullPointerException.class)
public void testArrayValues04() throws Exception {
    double[] array = null;
    Json.array(array);
}

@Test(expected = NullPointerException.class)
public void testArrayValues05() throws Exception {
    boolean[] array = null;
    Json.array(array);
}

@Test(expected = NullPointerException.class)
public void testArrayValues06() throws Exception {
    long[] array = null;
    Json.array(array);
}

@Test(expected = NullPointerException.class)
public void testArrayValues07() throws Exception {
    String[] array = null;
    Json.array(array);
}

/**
*
* Method: object()
*
*/
@Test
public void testObject() throws Exception {
    Json.object();
}

/**
*
* Method: parse(String string)
*
*/
```

JsonTest.java

```
@Test
public void testParseString() throws Exception {
    System.out.println(Json.parse("{\"hello\":[1.2,1.3], \n" +
            "\"hh\":{\n" +
                    "\"123\":666,\n" +
                    "\"789\":[222,101],\n" +
                    "\"66\":[true,false]\n" +
            "}, \n" +
            "\"hh123\": [1230000000000000,4560000000000,999000000000000], \n"
+
            "\"hh1234\": []\n" +
            "}"));
}

@Test(expected = NullPointerException.class)
public void testParseString01() throws Exception {
    String string = null;
    System.out.println(Json.parse(string));
}

    /**
*
* Method: parse(Reader reader)
*
*/
@Test
public void testParseReader() throws Exception {
    Reader reader = new Reader() {
        @Override
        public int read(char[] cbuf, int off, int len) throws IOException {
            try{
                for (int i=off; i<len; i++){
                    System.out.print(cbuf[i]);
                }
                return 1;
            } catch (Exception e ) {
                return -1;
            }
        }

        @Override
        public void close() throws IOException {

        }
    };
    Json.parse(reader);
}

/**
*
* Method: startArray()
*
*/
@Test
public void testStartArray() throws Exception {
    Json.DefaultHandler defaultHandler = new Json.DefaultHandler();
    JsonArray jsonArray = defaultHandler.startArray();
    Assert.assertEquals("[]",jsonArray.toString());
}
```

JsonTest.java

```
/**
*
* Method: startObject()
*
*/
@Test
public void testStartObject() throws Exception {
    Json.DefaultHandler defaultHandler = new Json.DefaultHandler();
    JsonObject jsonObject = defaultHandler.startObject();
    Assert.assertEquals("{}",jsonObject.toString());
}

/**
*
* Method: endNull()
*
*/
@Test
public void testEndNull() throws Exception {
    Json.DefaultHandler defaultHandler = new Json.DefaultHandler();
    defaultHandler.endNull();
    Assert.assertEquals("null",defaultHandler.getValue().toString());
}

/**
*
* Method: endBoolean(boolean bool)
*
*/
@Test
public void testEndBoolean() throws Exception {
    Json.DefaultHandler defaultHandler = new Json.DefaultHandler();
    defaultHandler.endBoolean(true);
    Assert.assertEquals("true",defaultHandler.getValue().toString());
    defaultHandler.endBoolean(false);
    Assert.assertEquals("false",defaultHandler.getValue().toString());
}

/**
*
* Method: endString(String string)
*
*/
@Test
public void testEndString() throws Exception {
    Json.DefaultHandler defaultHandler = new Json.DefaultHandler();
    defaultHandler.endString("hujunyao");

Assert.assertEquals("\"hujunyao\"",defaultHandler.getValue().toString());
}

/**
*
* Method: endNumber(String string)
*
*/
@Test
public void testEndNumber() throws Exception {
```

JsonTest.java

```
    Json.DefaultHandler defaultHandler = new Json.DefaultHandler();
    defaultHandler.endNumber("3.1415926");
    Assert.assertEquals("3.1415926",defaultHandler.getValue().toString());
}

/**
*
* Method: endArray(JsonArray array)
*
*/
@Test
public void testEndArray() throws Exception {
    Json.DefaultHandler defaultHandler = new Json.DefaultHandler();
    JsonArray jsonArray = new JsonArray();
    defaultHandler.endArray(jsonArray);
    Assert.assertEquals("[]",defaultHandler.getValue().toString());
}

/**
*
* Method: endObject(JsonObject object)
*
*/
@Test
public void testEndObject() throws Exception {
    Json.DefaultHandler defaultHandler = new Json.DefaultHandler();
    JsonObject jsonObject = new JsonObject();
    defaultHandler.endObject(jsonObject);
    Assert.assertEquals("{}",defaultHandler.getValue().toString());
}

/**
*
* Method: endArrayValue(JsonArray array)
*
*/
@Test
public void testEndArrayValue() throws Exception {
    Json.DefaultHandler defaultHandler = new Json.DefaultHandler();
    JsonArray jsonArray1 = new JsonArray();
    JsonArray jsonArray2 = new JsonArray();

    jsonArray1.add(1.2);

    jsonArray2.add(100);
    jsonArray2.add("arr2");

    defaultHandler.endArray(jsonArray2);
    defaultHandler.endArrayValue(jsonArray1);

Assert.assertEquals("[100,\"arr2\"]",defaultHandler.getValue().toString());
}

/**
*
* Method: endObjectValue(JsonObject object, String name)
*
*/
@Test
```

JsonTest.java

```
public void testEndObjectValue() throws Exception {
    Json.DefaultHandler defaultHandler = new Json.DefaultHandler();
    JsonObject jsonObject1 = new JsonObject();
    JsonObject jsonObject2 = new JsonObject();

    jsonObject1.add("aa", 1.2);

    jsonObject2.add("cumt",1909);
    jsonObject2.add("hjy",0.6192081);

    defaultHandler.endObject(jsonObject2);
    defaultHandler.endObjectValue(jsonObject1,"obj");

Assert.assertEquals("{\"cumt\":1909,\"hjy\":0.6192081}",defaultHandler.getVa
lue().toString());
}

/**
*
* Method: getValue()
*
*/
@Test
public void testGetValue() throws Exception {
    Json.DefaultHandler defaultHandler = new Json.DefaultHandler();
    Assert.assertNull(defaultHandler.getValue());
}


/**
*
* Method: cutOffPointZero(String string)
*
*/
@Test
public void testCutOffPointZero() throws Exception {
    Assert.assertEquals("123", Json.cutOffPointZero("123"));
    Assert.assertEquals("66", Json.cutOffPointZero("66.0"));

}

@Test(timeout = 4000)
public void test00()  throws Throwable  {
    Json.DefaultHandler json_DefaultHandler0 = new Json.DefaultHandler();
    JsonObject jsonObject0 = json_DefaultHandler0.startObject();
    Assert.assertFalse(jsonObject0.isString());
}

@Test(timeout = 4000)
public void test01()  throws Throwable  {
    Json.DefaultHandler json_DefaultHandler0 = new Json.DefaultHandler();
    JsonObject jsonObject0 = Json.object();
    json_DefaultHandler0.endObject(jsonObject0);
    Assert.assertFalse(jsonObject0.isBoolean());
}

@Test(timeout = 4000)
public void test02()  throws Throwable  {
    Json.DefaultHandler json_DefaultHandler0 = new Json.DefaultHandler();
```

JsonTest.java

```
    JsonObject jsonObject0 = new JsonObject();
    json_DefaultHandler0.endBoolean(false);
    json_DefaultHandler0.endObjectValue(jsonObject0, "{}");
    Assert.assertFalse(jsonObject0.isNull());
}

@Test(timeout = 4000)
public void test03()  throws Throwable  {
    Json.DefaultHandler json_DefaultHandler0 = new Json.DefaultHandler();
    JsonArray jsonArray0 = Json.array();
    json_DefaultHandler0.endNumber("vL4BZ&rOo4_Ht");
    json_DefaultHandler0.endArrayValue(jsonArray0);
    Assert.assertFalse(jsonArray0.isNull());
}

@Test(timeout = 4000)
public void test04()  throws Throwable  {
    JsonValue jsonValue0 = Json.parse("72");
    Assert.assertFalse(jsonValue0.isBoolean());
}

@Test(timeout = 4000)
public void test05()  throws Throwable  {
    JsonValue jsonValue0 = Json.parse("null");
    Assert.assertFalse(jsonValue0.isFalse());
}

@Test(timeout = 4000)
public void test06()  throws Throwable  {
    JsonValue jsonValue0 = Json.parse("false");
    Assert.assertTrue(jsonValue0.isFalse());
}

@Test(timeout = 4000)
public void test07()  throws Throwable  {
    JsonValue jsonValue0 = Json.parse("{}");
    Assert.assertFalse(jsonValue0.isArray());
}

@Test(timeout = 4000)
public void test08()  throws Throwable  {
    StringReader stringReader0 = new StringReader("72");
    JsonValue jsonValue0 = Json.parse((Reader) stringReader0);
    Assert.assertFalse(jsonValue0.isTrue());
}

@Test(timeout = 4000)
public void test09()  throws Throwable  {
    StringReader stringReader0 = new StringReader("null");
    JsonValue jsonValue0 = Json.parse((Reader) stringReader0);
    Assert.assertFalse(jsonValue0.isString());
}

@Test(timeout = 4000)
public void test10()  throws Throwable  {
    StringReader stringReader0 = new StringReader("false");
    JsonValue jsonValue0 = Json.parse((Reader) stringReader0);
    Assert.assertTrue(jsonValue0.isFalse());
}
```

JsonTest.java

```java
@Test(timeout = 4000)
public void test11()  throws Throwable  {
    StringReader stringReader0 = new StringReader("true");
    JsonValue jsonValue0 = Json.parse((Reader) stringReader0);
    Assert.assertFalse(jsonValue0.isFalse());
}

@Test(timeout = 4000)
public void test12()  throws Throwable  {
    String string0 = Json.cutOffPointZero("");
    Assert.assertEquals("", string0);
}

@Test(timeout = 4000)
public void test13()  throws Throwable  {
    boolean[] booleanArray0 = new boolean[0];
    JsonArray jsonArray0 = Json.array(booleanArray0);
    Assert.assertFalse(jsonArray0.isTrue());
}

@Test(timeout = 4000)
public void test14()  throws Throwable  {
    String[] stringArray0 = new String[0];
    JsonArray jsonArray0 = Json.array(stringArray0);
    Assert.assertTrue(jsonArray0.isEmpty());
}

@Test(timeout = 4000)
public void test15()  throws Throwable  {
    long[] longArray0 = new long[0];
    JsonArray jsonArray0 = Json.array(longArray0);
    Assert.assertFalse(jsonArray0.isTrue());
}

@Test(timeout = 4000)
public void test16()  throws Throwable  {
    int[] intArray0 = new int[0];
    JsonArray jsonArray0 = Json.array(intArray0);
    Assert.assertFalse(jsonArray0.isTrue());
}

@Test(timeout = 4000)
public void test17()  throws Throwable  {
    float[] floatArray0 = new float[0];
    JsonArray jsonArray0 = Json.array(floatArray0);
    Assert.assertFalse(jsonArray0.isNumber());
}

@Test(timeout = 4000)
public void test18()  throws Throwable  {
    double[] doubleArray0 = new double[0];
    JsonArray jsonArray0 = Json.array(doubleArray0);
    Assert.assertFalse(jsonArray0.isObject());
}

@Test(timeout = 4000,expected = RuntimeException.class)
public void test19()  throws Throwable  {
      Json.parse("`J]H.");
```

JsonTest.java

```
}

  @Test(timeout = 4000,expected = IOException.class)
public void test20()  throws Throwable  {
    StringReader stringReader0 = new StringReader("");
    stringReader0.close();
      Json.parse((Reader) stringReader0);

}

  @Test(timeout = 4000,expected = NullPointerException.class)
public void test21()  throws Throwable  {

      Json.cutOffPointZero((String) null);

}

@Test(timeout = 4000)
public void test22()  throws Throwable  {
    String string0 = Json.cutOffPointZero("mGh)J7L");
    Assert.assertEquals("mGh)J7L", string0);
}

@Test(timeout = 4000)
public void test23()  throws Throwable  {
    String string0 = Json.cutOffPointZero("E?Cj\"'.0");
    Assert.assertEquals("E?Cj\"'", string0);
}

@Test(timeout = 4000)
public void test24()  throws Throwable  {
    JsonValue jsonValue0 = Json.value(true);
    Assert.assertTrue(jsonValue0.isTrue());
}

@Test(timeout = 4000)
public void test25()  throws Throwable  {
    JsonValue jsonValue0 = Json.value((String) null);
    Assert.assertFalse(jsonValue0.isBoolean());
}

@Test(timeout = 4000)
public void test26()  throws Throwable  {
    JsonValue jsonValue0 = Json.value(4060.0607521795);
    Assert.assertEquals(4060.0607521795, jsonValue0.asDouble(), 0.01);
}

@Test(timeout = 4000)
public void test27()  throws Throwable  {
    JsonValue jsonValue0 = Json.value(0.0F);
    Assert.assertEquals("0", jsonValue0.toString());
}

@Test(timeout = 4000)
public void test28()  throws Throwable  {
    JsonValue jsonValue0 = Json.value(516L);
    Assert.assertEquals("516", jsonValue0.toString());
}
```

JsonTest.java

```
@Test(timeout = 4000)
public void test29()  throws Throwable  {
    Json.DefaultHandler json_DefaultHandler0 = new Json.DefaultHandler();
    JsonValue jsonValue0 = json_DefaultHandler0.getValue();
    Assert.assertNull(jsonValue0);
}

@Test(timeout = 4000)
public void test30()  throws Throwable  {
    JsonValue jsonValue0 = Json.value(2356);
    Assert.assertEquals(2356L, jsonValue0.asLong());
}

@Test(timeout = 4000)
public void test31()  throws Throwable  {
    double[] doubleArray0 = new double[8];
    doubleArray0[1] = (-140.3);
    JsonArray jsonArray0 = Json.array(doubleArray0);
    Assert.assertEquals(8, jsonArray0.size());
}

  @Test(timeout = 4000,expected = NullPointerException.class)
public void test32()  throws Throwable  {

      Json.parse((Reader) null);

}
  @Test(timeout = 4000,expected = NullPointerException.class)
public void test33()  throws Throwable  {
      Json.parse((String) null);

}

@Test(timeout = 4000)
public void test34()  throws Throwable  {
    JsonValue jsonValue0 = Json.parse("true");
    Assert.assertEquals("true", jsonValue0.toString());
}

  @Test(timeout = 4000,expected = NullPointerException.class)
public void test35()  throws Throwable  {

      Json.array((String[]) null);

}

  @Test(timeout = 4000,expected = NullPointerException.class)
public void test36()  throws Throwable  {

      Json.array((boolean[]) null);

}

  @Test(timeout = 4000,expected = NullPointerException.class)
public void test37()  throws Throwable  {

      Json.array((double[]) null);

}
```

JsonTest.java

```
  @Test(timeout = 4000,expected = NullPointerException.class)
public void test38()  throws Throwable  {

      Json.array((float[]) null);

}

  @Test(timeout = 4000,expected = NullPointerException.class)
public void test39()  throws Throwable  {

      Json.array((long[]) null);

}

  @Test(timeout = 4000,expected = NullPointerException.class)
public void test40()  throws Throwable  {

      Json.array((int[]) null);

}

@Test(timeout = 4000)
public void test41()  throws Throwable  {
    boolean[] booleanArray0 = new boolean[23];
    booleanArray0[7] = true;
    JsonArray jsonArray0 = Json.array(booleanArray0);
    Assert.assertEquals(23, jsonArray0.size());
}

@Test(timeout = 4000)
public void test42()  throws Throwable  {
    JsonValue jsonValue0 = Json.value(false);
    Assert.assertEquals("false", jsonValue0.toString());
}

@Test(timeout = 4000)
public void test43()  throws Throwable  {
    String[] stringArray0 = new String[5];
    JsonArray jsonArray0 = Json.array(stringArray0);
    Assert.assertEquals(5, jsonArray0.size());
}

@Test(timeout = 4000)
public void test44()  throws Throwable  {
    JsonValue jsonValue0 = Json.value("true");
    Assert.assertFalse(jsonValue0.isArray());
}

@Test(timeout = 4000)
public void test45()  throws Throwable  {
    long[] longArray0 = new long[7];
    JsonArray jsonArray0 = Json.array(longArray0);
    Assert.assertEquals(7, jsonArray0.size());
}

@Test(timeout = 4000)
public void test46()  throws Throwable  {
    Json.DefaultHandler json_DefaultHandler0 = new Json.DefaultHandler();
```

JsonTest.java

```
    json_DefaultHandler0.endString("yXKpl},");
}

@Test(timeout = 4000)
public void test47()  throws Throwable  {
    Json.DefaultHandler json_DefaultHandler0 = new Json.DefaultHandler();
    json_DefaultHandler0.endNull();
}

@Test(timeout = 4000,expected = RuntimeException.class)
public void test48()  throws Throwable  {
    StringReader stringReader0 = new StringReader("{}");
    Json.parse((Reader) stringReader0);
      Json.parse((Reader) stringReader0);

}

@Test(timeout = 4000)
public void test49()  throws Throwable  {
    Json.DefaultHandler json_DefaultHandler0 = new Json.DefaultHandler();
    JsonArray jsonArray0 = json_DefaultHandler0.startArray();
    Assert.assertTrue(jsonArray0.isArray());
}

  @Test(timeout = 4000,expected = NullPointerException.class)
public void test50()  throws Throwable  {
    int[] intArray0 = new int[27];
    JsonArray jsonArray0 = Json.array(intArray0);
    Json.DefaultHandler json_DefaultHandler0 = new Json.DefaultHandler();
      json_DefaultHandler0.endArrayValue(jsonArray0);

}

@Test(timeout = 4000)
public void test51()  throws Throwable  {
    float[] floatArray0 = new float[15];
    JsonArray jsonArray0 = Json.array(floatArray0);
    Json.DefaultHandler json_DefaultHandler0 = new Json.DefaultHandler();
    json_DefaultHandler0.endArray(jsonArray0);
    Assert.assertEquals(15, jsonArray0.size());
}
}
```

代码 4.3-2 JsonArrayTest 类

JsonArrayTest.java

```java
package mooctest;

import net.mooctest.JsonArray;
import net.mooctest.JsonValue;
import net.mooctest.JsonWriter;
import net.mooctest.PrettyPrint;
import org.junit.Assert;
import org.junit.Test;
import org.junit.Before;
import org.junit.After;

import java.io.IOException;
import java.io.Reader;
import java.io.StringReader;
import java.io.StringWriter;
import java.io.Writer;
import java.util.Iterator;
import java.util.List;

/**
 * JsonArray Tester.
 *
 * @author Hujunyao CUMT CS 2019-4 06192081
 * @since <pre>7 月 4, 2022</pre>
 * @version 1.0
 */
public class JsonArrayTest {

    @Before
    public void before() throws Exception {
    }

    @After
    public void after() throws Exception {
    }

    /**
     *
     * Method: readFrom(Reader reader)
     *
     */
    @Test
    public void testReadFromReader() throws Exception {
        JsonArray jsonArray0 = JsonArray.readFrom("[\n\t\n]");
        Assert.assertEquals(0, jsonArray0.size());
    }

    /**
     *
     * Method: readFrom(String string)
     *
     */
    @Test
    public void testReadFromString() throws Exception {
        StringReader stringReader0 = new StringReader("[1,2,3]");
        JsonArray jsonArray0 = JsonArray.readFrom((Reader) stringReader0);
        Assert.assertEquals(3, jsonArray0.size());
    }
```

JsonArrayTest.java

```
    /**
     *
     * Method: unmodifiableArray(JsonArray array)
     *
     */
    @Test
    public void testUnmodifiableArray() throws Exception {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.add(1530L);
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);
        Assert.assertFalse(jsonArray1.isNull());
    }

    /**
     *
     * Method: add(String value)
     *
     */
    @Test
    public void testAddString() throws Exception {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add((String) null);
        Assert.assertFalse(jsonArray1.isEmpty());
    }


    @Test(timeout = 4000)
    public void testAddBoolean()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add(true);
        Assert.assertFalse(jsonArray1.isNull());
    }

    @Test(timeout = 4000)
    public void testAddFloat()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add(0.0F);
        Assert.assertFalse(jsonArray1.isObject());
    }

    @Test(timeout = 4000)
    public void testAddInt()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add(1909);
        Assert.assertFalse(jsonArray1.isObject());
    }

    @Test(timeout = 4000)
    public void testAddLong()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add(1909L);
        Assert.assertFalse(jsonArray1.isObject());
    }

    @Test(timeout = 4000)
    public void testAddDouble()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add(19.09);
```

JsonArrayTest.java

```
        Assert.assertFalse(jsonArray1.isObject());
    }
    /**
     *
     * Method: set(int index, int value)
     *
     */

    @Test(timeout = 4000)
    public void testSetForIndexJsonValue()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add(0);
        JsonArray jsonArray2 = jsonArray1.set(0, (JsonValue) jsonArray0);
        Assert.assertSame(jsonArray0, jsonArray2);
    }


    @Test(timeout = 4000, expected = NullPointerException.class)
    public void testSetForIndexJsonValue01()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = null;
        jsonArray1.set(0, (JsonValue) jsonArray0);
    }

    @Test(timeout = 4000)
    public void testSetForIndexString()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.add(0);
        JsonArray jsonArray1 = jsonArray0.set(0, (String) null);
        Assert.assertFalse(jsonArray1.isTrue());
    }

    @Test(timeout = 4000)
    public void testSetForIndexLong()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.add((double) 0);
        JsonArray jsonArray1 = jsonArray0.set(0, 1L);
        Assert.assertEquals(1, jsonArray1.size());
    }

    @Test(timeout = 4000)
    public void testSetForIndexInt()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.add(2295L);
        JsonArray jsonArray1 = jsonArray0.set(0, 0);
        Assert.assertSame(jsonArray1, jsonArray0);
    }

    @Test(timeout = 4000)
    public void testSetForIndexFloat()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add((double) 0);
        JsonArray jsonArray2 = jsonArray1.set(0, (float) 0);
        Assert.assertSame(jsonArray2, jsonArray0);
    }

    @Test(timeout = 4000)
    public void testSetForIndexDouble()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
```

JsonArrayTest.java

```
        JsonArray jsonArray1 = jsonArray0.add((long) 0);
        JsonArray jsonArray2 = jsonArray1.set(0, (double) 0);
        Assert.assertFalse(jsonArray2.isNull());
    }

    @Test(timeout = 4000)
    public void testSetForIndexBoolean()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add((long) 0);
        JsonArray jsonArray2 = jsonArray1.set(0, true);
        Assert.assertFalse(jsonArray2.isNull());
    }

    /**
     *
     * Method: remove(int index)
     *
     */
    @Test
    public void testRemove() throws Exception {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add(0);
        JsonArray jsonArray2 = jsonArray0.remove(0);
        Assert.assertSame(jsonArray2, jsonArray1);
    }

    /**
     *
     * Method: size()
     *
     */
    @Test
    public void testSize() throws Exception {
        JsonArray jsonArray0 = JsonArray.readFrom("[\n\t\n]");
        Assert.assertEquals(0, jsonArray0.size());

        JsonArray jsonArray01 = new JsonArray();
        int int0 = jsonArray01.size();
        Assert.assertEquals(0, int0);
    }

    /**
     *
     * Method: isEmpty()
     *
     */
    @Test
    public void testIsEmpty() throws Exception {
        JsonArray jsonArray0 = new JsonArray();
        Assert.assertTrue(jsonArray0.isEmpty());

        jsonArray0.add(2295L);
        boolean boolean0 = jsonArray0.isEmpty();
        Assert.assertFalse(boolean0);
    }

    /**
     *
     * Method: get(int index)
```

JsonArrayTest.java

```
     *
     */
    @Test
    public void testGet() throws Exception {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add((double) 0);
        JsonValue jsonValue0 = jsonArray1.get(0);
        Assert.assertEquals(0.0F, jsonValue0.asFloat(), 0.01F);
    }

    /**
     *
     * Method: values()
     *
     */
    @Test
    public void testValues() throws Exception {
        JsonArray jsonArray0 = new JsonArray();
        List<JsonValue> list0 = jsonArray0.values();
        Assert.assertEquals(0, list0.size());
    }

    /**
     *
     * Method: iterator()
     *
     */
    @Test
    public void testIterator() throws Exception {
        JsonArray jsonArray0 = new JsonArray();
        Iterator<JsonValue> iterator0 = jsonArray0.iterator();
        Assert.assertNotNull(iterator0);
    }

    /**
     *
     * Method: write(JsonWriter writer)
     *
     */
    @Test
    public void testWrite() throws Exception {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add((long) 0);
        StringWriter stringWriter0 = new StringWriter();
        JsonWriter jsonWriter0 = new JsonWriter(stringWriter0);
        jsonArray1.write(jsonWriter0);
        Assert.assertEquals("[0]", stringWriter0.toString());
    }

    /**
     *
     * Method: isArray()
     *
     */
    @Test
    public void testIsArray() throws Exception {
        JsonArray jsonArray0 = new JsonArray();
        boolean boolean0 = jsonArray0.isArray();
        Assert.assertTrue(boolean0);
```

JsonArrayTest.java

```
    }

    /**
     *
     * Method: asArray()
     *
     */
    @Test
    public void testAsArray() throws Exception {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.asArray();
        Assert.assertSame(jsonArray0, jsonArray1);
    }

    /**
     *
     * Method: hashCode()
     *
     */
    @Test
    public void testHashCode() throws Exception {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.hashCode();
    }

    /**
     *
     * Method: equals(Object object)
     *
     */
    @Test
    public void testEquals() throws Exception {
        JsonArray jsonArray0 = new JsonArray();
        Object object0 = new Object();
        boolean boolean0 = jsonArray0.equals(object0);
        Assert.assertFalse(boolean0);

        JsonArray jsonArray1 = new JsonArray();
        boolean boolean1 = jsonArray1.equals((Object) null);
        Assert.assertFalse(boolean1);

        JsonArray jsonArray2 = new JsonArray();
        boolean boolean2 = jsonArray2.equals(jsonArray2);
        Assert.assertTrue(boolean2);


    }

    @Test(timeout = 4000)
    public void test52()  throws Throwable  {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = new JsonArray(jsonArray0);
        Assert.assertTrue(jsonArray1.equals((Object)jsonArray0));

        jsonArray0.add((long) 0);
        boolean boolean0 = jsonArray0.equals(jsonArray1);
        Assert.assertFalse(jsonArray1.equals((Object)jsonArray0));
        Assert.assertFalse(boolean0);
    }
```

JsonArrayTest.java

```java
    @Test(timeout = 4000)
    public void test00() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.add(1530L);
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);
        Assert.assertFalse(jsonArray1.isNull());
    }

    @Test(timeout = 4000)
    public void test01() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add(0);
        JsonArray jsonArray2 = jsonArray1.set(0, (JsonValue) jsonArray0);
        Assert.assertSame(jsonArray0, jsonArray2);
    }

    @Test(timeout = 4000)
    public void test02() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.add(0);
        JsonArray jsonArray1 = jsonArray0.set(0, (String) null);
        Assert.assertFalse(jsonArray1.isTrue());
    }

    @Test(timeout = 4000)
    public void test03() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.add((double) 0);
        JsonArray jsonArray1 = jsonArray0.set(0, 1L);
        Assert.assertEquals(1, jsonArray1.size());
    }

    @Test(timeout = 4000)
    public void test04() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.add(2295L);
        JsonArray jsonArray1 = jsonArray0.set(0, 0);
        Assert.assertSame(jsonArray1, jsonArray0);
    }

    @Test(timeout = 4000)
    public void test05() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add((double) 0);
        JsonArray jsonArray2 = jsonArray1.set(0, (float) 0);
        Assert.assertSame(jsonArray2, jsonArray0);
    }

    @Test(timeout = 4000)
    public void test06() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add((long) 0);
        JsonArray jsonArray2 = jsonArray1.set(0, (double) 0);
        Assert.assertFalse(jsonArray2.isNull());
```

JsonArrayTest.java

```
    }

    @Test(timeout = 4000)
    public void test07() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add(0);
        JsonArray jsonArray2 = jsonArray0.remove(0);
        Assert.assertSame(jsonArray2, jsonArray1);
    }

    @Test(timeout = 4000)
    public void test08() throws Throwable {
        JsonArray jsonArray0 = JsonArray.readFrom("[\n\t\n]");
        Assert.assertEquals(0, jsonArray0.size());
    }

    @Test(timeout = 4000)
    public void test09() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        Iterator<JsonValue> iterator0 = jsonArray0.iterator();
        Assert.assertNotNull(iterator0);
    }

    @Test(timeout = 4000)
    public void test10() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add((double) 0);
        JsonValue jsonValue0 = jsonArray1.get(0);
        Assert.assertEquals(0.0F, jsonValue0.asFloat(), 0.01F);
    }

    @Test(timeout = 4000, expected = NullPointerException.class)
    public void test11() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        PrettyPrint prettyPrint0 = PrettyPrint.indentWithTabs();
        JsonWriter jsonWriter0 = prettyPrint0.createWriter((Writer) null);

        jsonArray0.write(jsonWriter0);
    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test12() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);

        jsonArray1.set(1, true);
    }

    @Test(timeout = 4000, expected = ArrayIndexOutOfBoundsException.class)
    public void test13() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();

        jsonArray0.set((-2), false);

    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test14() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
```

JsonArrayTest.java

```
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);
        jsonArray1.set((-590), (JsonValue) jsonArray0);

    }

    @Test(timeout = 4000, expected = ArrayIndexOutOfBoundsException.class)
    public void test15() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();

        jsonArray0.set((-4901), (JsonValue) jsonArray0);

    }

    @Test(timeout = 4000, expected = ArrayIndexOutOfBoundsException.class)
    public void test16() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.set((-643), "uCu%Y;nNz__Se#s&=");

    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test17() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);

        jsonArray1.set(472, (long) 472);

    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test18() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);
        jsonArray1.set((-1), 1755);
    }

    @Test(timeout = 4000, expected = IndexOutOfBoundsException.class)
    public void test19() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.set(5, 5);
    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test20() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);
        jsonArray1.set((-3585), 965.9589F);

    }

    @Test(timeout = 4000, expected = ArrayIndexOutOfBoundsException.class)
    public void test21() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();

        jsonArray0.set((-3349), (float) (-3349));

    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
```

JsonArrayTest.java

```java
    public void test22() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);

        jsonArray1.set(1532, (double) 1532);

    }

    @Test(timeout = 4000, expected = ArrayIndexOutOfBoundsException.class)
    public void test23() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.set((-1898), (-133.4547883890548));

    }

    @Test(timeout = 4000, expected = ArrayIndexOutOfBoundsException.class)
    public void test24() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();

        jsonArray0.remove((-217));

    }

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test25() throws Throwable {

        JsonArray.readFrom("");

    }

    @Test(timeout = 4000, expected = NullPointerException.class)
    public void test26() throws Throwable {
        JsonArray.readFrom((String) null);
    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test27() throws Throwable {
        StringReader stringReader0 = new StringReader("false");
        JsonArray.readFrom((Reader) stringReader0);

    }

    @Test(timeout = 4000, expected = NullPointerException.class)
    public void test28() throws Throwable {

            JsonArray.readFrom((Reader) null);

    }

    @Test(timeout = 4000,expected = IOException.class)
    public void test29() throws Throwable {
        StringReader stringReader0 = new StringReader(":*<ejk.qXnivNkc");
        stringReader0.close();

            JsonArray.readFrom((Reader) stringReader0);

    }

    @Test(timeout = 4000, expected = IndexOutOfBoundsException.class)
```

JsonArrayTest.java

```
    public void test30() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
            jsonArray0.get(8);

    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test31() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);
        jsonArray1.add(true);

    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test32() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);
        jsonArray1.add((JsonValue) jsonArray0);

    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test33() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);

        jsonArray1.add("7y |Ea");

    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test34() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);

        jsonArray1.add(1L);

    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test35() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);
        jsonArray1.add((-1));

    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test36() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);

        jsonArray1.add(1794.661F);

    }

    @Test(timeout = 4000,expected = NullPointerException.class)
    public void test37() throws Throwable {
```

JsonArrayTest.java

```
        JsonArray jsonArray0 = null;

            jsonArray0 = new JsonArray((JsonArray) null);

    }

    @Test(timeout = 4000)
    public void test38() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        Assert.assertTrue(jsonArray0.isEmpty());

        jsonArray0.add(2295L);
        boolean boolean0 = jsonArray0.isEmpty();
        Assert.assertFalse(boolean0);
    }

    @Test(timeout = 4000)
    public void test39() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        int int0 = jsonArray0.size();
        Assert.assertEquals(0, int0);
    }

    @Test(timeout = 4000,expected = IndexOutOfBoundsException.class)
    public void test40() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();

            jsonArray0.set(756, 199L);

    }

    @Test(timeout = 4000,expected = IndexOutOfBoundsException.class)
    public void test41() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();

            jsonArray0.set(1532, (double) 1532);

    }

    @Test(timeout = 4000)
    public void test42() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        boolean boolean0 = jsonArray0.isArray();
        Assert.assertTrue(boolean0);
    }

    @Test(timeout = 4000)
    public void test43() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add((String) null);
        Assert.assertFalse(jsonArray1.isEmpty());
    }

    @Test(timeout = 4000)
    public void test44() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        Object object0 = new Object();
        boolean boolean0 = jsonArray0.equals(object0);
        Assert.assertFalse(boolean0);
```

JsonArrayTest.java

```
    }

    @Test(timeout = 4000)
    public void test45() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        boolean boolean0 = jsonArray0.equals((Object) null);
        Assert.assertFalse(boolean0);
    }

    @Test(timeout = 4000)
    public void test46() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        boolean boolean0 = jsonArray0.equals(jsonArray0);
        Assert.assertTrue(boolean0);
    }

    @Test(timeout = 4000)
    public void test47() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        boolean boolean0 = jsonArray0.isEmpty();
        Assert.assertTrue(boolean0);
    }

    @Test(timeout = 4000,expected = NullPointerException.class)
    public void test48() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();

            jsonArray0.set(1, (JsonValue) null);

    }

    @Test(timeout = 4000,expected = IndexOutOfBoundsException.class)
    public void test49() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
            jsonArray0.set(0, (JsonValue) jsonArray0);

    }

    @Test(timeout = 4000,expected = NullPointerException.class)
    public void test50() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
            jsonArray0.add((JsonValue) null);

    }

    @Test(timeout = 4000)
    public void test51() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.add((JsonValue) jsonArray0);
        JsonArray jsonArray1 = jsonArray0.set(0, false);
        Assert.assertFalse(jsonArray1.isObject());
    }

    @Test(timeout = 4000,expected = NullPointerException.class)
    public void test53() throws Throwable {
            JsonArray.unmodifiableArray((JsonArray) null);
    }

    @Test(timeout = 4000, expected = IndexOutOfBoundsException.class)
```

JsonArrayTest.java

```
    public void test54() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.set(1, true);

    }

    @Test(timeout = 4000, expected = ArrayIndexOutOfBoundsException.class)
    public void test55() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.set((-1), 1755);
    }

    @Test(timeout = 4000)
    public void test56() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.asArray();
        Assert.assertSame(jsonArray0, jsonArray1);
    }

    @Test(timeout = 4000)
    public void test57() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add((long) 0);
        StringWriter stringWriter0 = new StringWriter();
        JsonWriter jsonWriter0 = new JsonWriter(stringWriter0);
        jsonArray1.write(jsonWriter0);
        Assert.assertEquals("[0]", stringWriter0.toString());
    }

    @Test(timeout = 4000)
    public void test58() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.hashCode();
    }

    @Test(timeout = 4000)
    public void test59() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        List<JsonValue> list0 = jsonArray0.values();
        Assert.assertEquals(0, list0.size());
    }

    @Test(timeout = 4000, expected = IndexOutOfBoundsException.class)
    public void test60() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.set(0, (String) null);

    }

    @Test(timeout = 4000)
    public void test61() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add(true);
        Assert.assertFalse(jsonArray1.isNull());
    }

    @Test(timeout = 4000, expected = IndexOutOfBoundsException.class)
    public void test62() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
```

JsonArrayTest.java

```
        jsonArray0.remove(0);

    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test63() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = JsonArray.unmodifiableArray(jsonArray0);

        jsonArray1.add((double) 0);

    }

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test64() throws Throwable {
        StringReader stringReader0 = new StringReader("");

        JsonArray.readFrom((Reader) stringReader0);

    }

    @Test(timeout = 4000)
    public void test65() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        JsonArray jsonArray1 = jsonArray0.add(0.0F);
        Assert.assertFalse(jsonArray1.isObject());
    }

    @Test(timeout = 4000, expected = IndexOutOfBoundsException.class)
    public void test66() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();

        jsonArray0.set(2, (float) 2);

    }

    @Test(timeout = 4000)
    public void test67() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();
        jsonArray0.add((-16));
        JsonArray jsonArray1 = jsonArray0.asArray();
        Assert.assertFalse(jsonArray1.isObject());
    }

    @Test(timeout = 4000, expected = ArrayIndexOutOfBoundsException.class)
    public void test68() throws Throwable {
        JsonArray jsonArray0 = new JsonArray();

        jsonArray0.get((-16));

    }

    @Test(timeout = 4000, expected = UnsupportedOperationException.class)
    public void test69() throws Throwable {
        JsonArray.readFrom("2");
    }
}
```

代码 4.3-3 JsonLiteralTest 类

JsonLiteralTest.java

```java
package mooctest;

import net.mooctest.JsonLiteral;
import net.mooctest.JsonWriter;
import org.junit.Assert;
import org.junit.Test;
import org.junit.Before;
import org.junit.After;

import java.io.StringWriter;

/**
* JsonLiteral Tester.
*
* @author Hujunyao CUMT CS 2019-4 06192081
* @since <pre>7 月 3, 2022</pre>
* @version 1.0
*/
public class JsonLiteralTest {
    JsonLiteral jsonLiteral;
@Before
public void before() throws Exception {
    jsonLiteral = new JsonLiteral("true");
}

@After
public void after() throws Exception {
}

/**
*
* Method: write(JsonWriter writer)
*
*/
@Test
public void testWrite() throws Exception {
    StringWriter stringWriter0 = new StringWriter();
    JsonWriter jsonWriter = new JsonWriter(stringWriter0);
    jsonLiteral.write(jsonWriter);
}

/**
*
* Method: toString()
*
*/
@Test
public void testToString() throws Exception {
    Assert.assertEquals("true", jsonLiteral.toString());
}

/**
*
* Method: hashCode()
*
*/
@Test
public void testHashCode() throws Exception {
```

JsonLiteralTest.java

```java
    System.out.println(jsonLiteral.hashCode());
}

/**
*
* Method: isNull()
*
*/
@Test
public void testIsNull() throws Exception {
    Assert.assertFalse(jsonLiteral.isNull());
}

/**
*
* Method: isTrue()
*
*/
@Test
public void testIsTrue() throws Exception {
    Assert.assertTrue(jsonLiteral.isTrue());
}

/**
*
* Method: isFalse()
*
*/
@Test
public void testIsFalse() throws Exception {
    Assert.assertFalse(jsonLiteral.isFalse());
}

/**
*
* Method: isBoolean()
*
*/
@Test
public void testIsBoolean() throws Exception {
    Assert.assertTrue(jsonLiteral.isBoolean());

    JsonLiteral jsonLiteral2 = new JsonLiteral("false");
    JsonLiteral jsonLiteral3 = new JsonLiteral("null");
    Assert.assertTrue(jsonLiteral2.isBoolean());
    Assert.assertFalse(jsonLiteral3.isBoolean());
}

/**
*
* Method: asBoolean()
*
*/
@Test (expected = UnsupportedOperationException.class)
public void testAsBoolean() throws Exception {
    Assert.assertTrue(jsonLiteral.asBoolean());

    JsonLiteral jsonLiteral2 = new JsonLiteral("false");
    JsonLiteral jsonLiteral3 = new JsonLiteral("null");
```

JsonLiteralTest.java

```
    Assert.assertFalse(jsonLiteral2.asBoolean());
    jsonLiteral3.asBoolean();
}

/**
*
* Method: equals(Object object)
*
*/
@Test
public void testEquals() throws Exception {
    JsonLiteral jsonLiteral1 = jsonLiteral;
    JsonLiteral jsonLiteral2 = new JsonLiteral("true");
    JsonLiteral jsonLiteral3 = new JsonLiteral("hello1");
    JsonLiteral jsonLiteral4 = null;
    String string = "hello";
    Assert.assertEquals(jsonLiteral,jsonLiteral1);
    Assert.assertEquals(jsonLiteral,jsonLiteral2);
    Assert.assertNotEquals(jsonLiteral,jsonLiteral3);
    Assert.assertNotEquals(jsonLiteral,jsonLiteral4);
    Assert.assertNotEquals(jsonLiteral,string);
}
}
```

代码 4.3-4 JsonNumberTest 类

JsonNumberTest.java

```java
package mooctest;

import net.mooctest.JsonLiteral;
import net.mooctest.JsonWriter;
import org.junit.Assert;
import org.junit.Test;
import org.junit.Before;
import org.junit.After;

import java.io.StringWriter;

/**
* JsonLiteral Tester.
*
* @author Hujunyao CUMT CS 2019-4 06192081
* @since <pre>7 月 3, 2022</pre>
* @version 1.0
*/
public class JsonLiteralTest {
    JsonLiteral jsonLiteral;
@Before
public void before() throws Exception {
    jsonLiteral = new JsonLiteral("true");
}

@After
public void after() throws Exception {
}

/**
*
* Method: write(JsonWriter writer)
*
*/
@Test
public void testWrite() throws Exception {
    StringWriter stringWriter0 = new StringWriter();
    JsonWriter jsonWriter = new JsonWriter(stringWriter0);
    jsonLiteral.write(jsonWriter);
}

/**
*
* Method: toString()
*
*/
@Test
public void testToString() throws Exception {
    Assert.assertEquals("true", jsonLiteral.toString());
}

/**
*
* Method: hashCode()
*
*/
@Test
public void testHashCode() throws Exception {
```

JsonNumberTest.java

```java
    System.out.println(jsonLiteral.hashCode());
}

/**
*
* Method: isNull()
*
*/
@Test
public void testIsNull() throws Exception {
    Assert.assertFalse(jsonLiteral.isNull());
}

/**
*
* Method: isTrue()
*
*/
@Test
public void testIsTrue() throws Exception {
    Assert.assertTrue(jsonLiteral.isTrue());
}

/**
*
* Method: isFalse()
*
*/
@Test
public void testIsFalse() throws Exception {
    Assert.assertFalse(jsonLiteral.isFalse());
}

/**
*
* Method: isBoolean()
*
*/
@Test
public void testIsBoolean() throws Exception {
    Assert.assertTrue(jsonLiteral.isBoolean());

    JsonLiteral jsonLiteral2 = new JsonLiteral("false");
    JsonLiteral jsonLiteral3 = new JsonLiteral("null");
    Assert.assertTrue(jsonLiteral2.isBoolean());
    Assert.assertFalse(jsonLiteral3.isBoolean());
}

/**
*
* Method: asBoolean()
*
*/
@Test (expected = UnsupportedOperationException.class)
public void testAsBoolean() throws Exception {
    Assert.assertTrue(jsonLiteral.asBoolean());

    JsonLiteral jsonLiteral2 = new JsonLiteral("false");
    JsonLiteral jsonLiteral3 = new JsonLiteral("null");
```

JsonNumberTest.java

```
    Assert.assertFalse(jsonLiteral2.asBoolean());
    jsonLiteral3.asBoolean();
}

/**
*
* Method: equals(Object object)
*
*/
@Test
public void testEquals() throws Exception {
    JsonLiteral jsonLiteral1 = jsonLiteral;
    JsonLiteral jsonLiteral2 = new JsonLiteral("true");
    JsonLiteral jsonLiteral3 = new JsonLiteral("hello1");
    JsonLiteral jsonLiteral4 = null;
    String string = "115114";
    Assert.assertEquals(jsonLiteral,jsonLiteral1);
    Assert.assertEquals(jsonLiteral,jsonLiteral2);
    Assert.assertNotEquals(jsonLiteral,jsonLiteral3);
    Assert.assertNotEquals(jsonLiteral,jsonLiteral4);
    Assert.assertNotEquals(jsonLiteral,string);
}
}
```

代码 4.3-5 JsonObjectTest 类

JsonObjectTest.java

```java
package mooctest;

import net.mooctest.JsonObject;
import net.mooctest.JsonString;
import net.mooctest.JsonValue;
import net.mooctest.PrettyPrint;
import org.junit.Assert;
import org.junit.Test;

import java.io.Reader;
import java.io.StringReader;
import java.io.StringWriter;
import java.io.Writer;

/**
* JsonObject Tester.
*
* @author Hujunyao CUMT CS 2019-4 06192081
* @since <pre>7 月 5, 2022</pre>
* @version 1.0
*/
public class JsonObjectTest {



/**
*
* Method: readFrom(Reader reader)
*
*/
@Test(timeout = 4000)
public void testReadFromReader() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    jsonObject0.toString();
    JsonObject jsonObject1 = jsonObject0.add("`#", (-37.28381F));
    String string0 = "n/w\\)";
    jsonObject0.get("n/w)");
    JsonObject jsonObject2 = jsonObject0.set("n/w)", 0L);
    JsonObject jsonObject3 = new JsonObject(jsonObject2);
    JsonObject jsonObject4 = jsonObject3.remove("n/w)");
    jsonObject1.set("`#", 123);
    JsonObject jsonObject5 = jsonObject2.add("config is null", (JsonValue)
jsonObject3);
    StringReader stringReader0 = new StringReader("{\n\n}");
    JsonObject jsonObject6 = JsonObject.readFrom((Reader) stringReader0);
    jsonObject2.equals(jsonObject4);
    String string1 = "Klxx]d6[8b+@7c {}";
    jsonObject6.set("name is null", (float) 0L);
    jsonObject5.set("NsVJ9C{[JU2", (JsonValue) jsonObject4);
    jsonObject0.getInt("`#", 93);
}

/**
*
* Method: readFrom(String string)
*
*/
@Test
```

JsonObjectTest.java

```java
public void testReadFromString() throws Exception {

    JsonObject jsonObject0 = new JsonObject();
    jsonObject0.toString();
    jsonObject0.toString();
    long long0 = 2629L;
    JsonObject.readFrom("{\n\n}");
}

@Test(expected = UnsupportedOperationException.class)
public void testReadFromString1() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    jsonObject0.getDouble("_/\"bG1#J}5Q:18T)#1/", (-1362.19410243));
    jsonObject0.set("null", false);
    StringReader stringReader0 = new StringReader("null");
    JsonObject.readFrom((Reader) stringReader0);

}


/**
*
* Method: unmodifiableObject(JsonObject object)
*
*/
@Test
public void testUnmodifiableObject() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1 = JsonObject.unmodifiableObject(jsonObject0);
}

/**
*
* Method: add(String name, int value)
*
*/
@Test
public void testAddSetForNameValue() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    jsonObject0.set("jN%i:", 456);
    jsonObject0.getLong("", 0L);
    JsonObject jsonObject1 = new JsonObject(jsonObject0);
    JsonObject jsonObject2 = jsonObject0.merge(jsonObject1);
    JsonObject jsonObject3 = jsonObject0.add("net.mooctest.JsonArray",
(double) 0L);
    JsonObject jsonObject4 = jsonObject0.set("net.mooctest.JsonObject",
9.1);
    String string0 = "7see";
    JsonObject jsonObject5 = jsonObject4.set("7see", 1.2);
    jsonObject4.get("net.mooctest.JsonObject");
    JsonObject jsonObject6 = jsonObject5.merge(jsonObject2);
    jsonObject6.getFloat("rkF9bp\"", (-1315.67F));
    JsonObject.HashIndexTable jsonObject_HashIndexTable0 = new
JsonObject.HashIndexTable();
    jsonObject_HashIndexTable0.get(jsonObject3);
    JsonObject jsonObject7 = jsonObject0.set("name is null", 666);
    jsonObject7.getFloat("name is null", (-1));
}
```

JsonObjectTest.java

```java
/**
*
* Method: remove(String name)
*
*/
@Test
public void testRemoveName() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1 = new JsonObject(jsonObject0);
    jsonObject1.add("net.mooctest.JsonObject", (JsonValue) jsonObject0);
    JsonObject jsonObject2 = jsonObject0.add("net.mooctest.JsonObject",
"net.mooctest.JsonObject");
    jsonObject2.remove("VwK6Rq");
    JsonObject.Member jsonObject_Member0 = new JsonObject.Member("=zi",
jsonObject1);
    jsonObject1.equals(jsonObject_Member0);
    String string0 = "Fe.9";
    jsonObject1.add("Fe.9", 0.0);
    jsonObject1.merge(jsonObject0);
    StringReader stringReader0 = new StringReader("{\"hhh\":1909}");
    JsonObject.readFrom((Reader) stringReader0);
}

/**
*
* Method: contains(String name)
*
*/
@Test
public void testContains() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    String string0 = "-.%UaU/Z+S}!gE(+.1h";
    JsonObject jsonObject1 = jsonObject0.set("-.%UaU/Z+S}!gE(+.1h", 0.0);
    JsonObject jsonObject2 = jsonObject1.set("-.%UaU/Z+S}!gE(+.1h", 0.0);
    jsonObject0.getLong("-.%UaU/Z+S}!gE(+.1h", 0L);
    jsonObject0.iterator();
    JsonObject jsonObject3 = JsonObject.unmodifiableObject(jsonObject2);
    JsonObject jsonObject4 = JsonObject.unmodifiableObject(jsonObject1);
    JsonObject jsonObject5 = jsonObject0.merge(jsonObject3);
    jsonObject3.getInt("ys[IF-GJ=", 255);
    jsonObject5.contains("-.%UaU/Z+S}!gE(+.1h");
    JsonObject.Member jsonObject_Member0 = new JsonObject.Member("object is
null", jsonObject4);
    JsonObject.Member jsonObject_Member1 = new JsonObject.Member("object is
null", jsonObject0);
    jsonObject_Member0.equals(jsonObject_Member1);
    jsonObject5.asObject();
}

/**
*
* Method: merge(JsonObject object)
*
*/
@Test
public void testMerge() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    String string0 = "-.%UaU/Z+S}!gE(+.1h";
    JsonObject jsonObject1 = jsonObject0.set("-.%UaU/Z+S}!gE(+.1h", 0.0);
```

JsonObjectTest.java

```
    JsonObject jsonObject2 = jsonObject1.set("-.%UaU/Z+S}!gE(+.1h", 0.0);
    jsonObject0.getLong("-.%UaU/Z+S}!gE(+.1h", 0L);
    jsonObject0.iterator();
    JsonObject jsonObject3 = new JsonObject(jsonObject0);
    jsonObject1.add("-.%UaU/Z+S}!gE(+.1h", false);
    JsonObject jsonObject4 = new JsonObject();
    JsonObject jsonObject5 = jsonObject0.merge(jsonObject3);
    jsonObject5.contains("-.%UaU/Z+S}!gE(+.1h");
    JsonObject.Member jsonObject_Member0 = new JsonObject.Member("object is
null", jsonObject4);
    JsonObject.Member jsonObject_Member1 = new JsonObject.Member("object is
null", jsonObject0);
    jsonObject0.merge(jsonObject5);
    jsonObject2.isEmpty();
    jsonObject_Member0.equals(jsonObject_Member1);
    jsonObject5.asObject();
}

/**
*
* Method: get(String name)
*
*/
@Test
public void testGetName() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    jsonObject0.indexOf("':'");
    JsonString jsonString0 = new JsonString("<!$n5uASK^clg6BU*X");
    JsonObject jsonObject1 = jsonObject0.add(":}7mjBZ -YUX.Dc}S",
(JsonValue) jsonString0);
    JsonObject jsonObject2 = jsonObject0.add("array is null", (long) (-1));
    JsonObject.HashIndexTable jsonObject_HashIndexTable0 = new
JsonObject.HashIndexTable();
    JsonObject.HashIndexTable jsonObject_HashIndexTable1 = new
JsonObject.HashIndexTable(jsonObject_HashIndexTable0);
    jsonObject_HashIndexTable1.remove((-4065));
    JsonObject jsonObject3 = jsonObject2.set("':'", (long) (-4065));
    JsonObject jsonObject4 = jsonObject1.set("<!$n5uASK^clg6BU*X", (-1L));
    jsonObject4.spliterator();
    jsonObject0.set("H4\"WF69a(cz{-[5", (JsonValue) jsonObject2);
    jsonObject3.add("<!$n5uASK^clg6BU*X", (-1.0));
    jsonObject4.set("NsVJ9C{[JU2", 0L);
    jsonObject2.get("H4\"WF69a(cz{-[5");
    jsonObject_HashIndexTable0.remove(648);
    jsonObject3.names();
}

/**
*
* Method: getInt(String name, int defaultValue)
*
*/
@Test
public void testGetInt() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1 = jsonObject0.set("`#", 1);
    jsonObject1.toString();
    JsonObject jsonObject2 = jsonObject0.add("`#", 6666666);
    String string0 = "n/w\\)";
```

JsonObjectTest.java

```
    jsonObject0.get("n/w)");
    JsonObject jsonObject3 = jsonObject0.set("n/w)", 0L);
    JsonObject jsonObject4 = new JsonObject(jsonObject3);
    JsonObject jsonObject5 = jsonObject4.remove("n/w)");
    jsonObject2.set("#L(Y?6XAB", "#L(Y?6XAB");
    JsonObject jsonObject6 = jsonObject3.add("config is null", (JsonValue)
jsonObject4);
    StringReader stringReader0 = new StringReader("{\n\"`#\": 1\n}");
    JsonObject jsonObject7 = JsonObject.readFrom((Reader) stringReader0);
    jsonObject6.set(";/(maJ", (double) 1);
    jsonObject3.equals(jsonObject5);
    String string1 = "Klxx]d6[8b+@7c {}";
    jsonObject7.set("name is null", (float) 0L);
    jsonObject6.set("NsVJ9C{[JU2", (JsonValue) jsonObject5);
    jsonObject0.getInt("`#", 93);
}

/**
*
* Method: getLong(String name, long defaultValue)
*
*/
@Test(expected = NullPointerException.class)
public void testGetLong() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1 = JsonObject.unmodifiableObject(jsonObject0);
    Assert.assertEquals("", jsonObject1.getLong((String) null, 950L));
}
    @Test
public void testGetLong1() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1 = JsonObject.unmodifiableObject(jsonObject0);
    Assert.assertEquals(950, jsonObject1.getLong("(String) null", 950L));
} //8744

    /**
*
* Method: getFloat(String name, float defaultValue)
*
*/
@Test
public void testGetFloat() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    jsonObject0.getFloat("`#", 12f);
}

/**
*
* Method: getDouble(String name, double defaultValue)
*
*/
@Test
public void testGetDouble() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    jsonObject0.getDouble("`#", 1.2);
}

/**
*
```

JsonObjectTest.java

```
* Method: getBoolean(String name, boolean defaultValue)
*
*/
@Test
public void testGetBoolean() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    jsonObject0.getBoolean("`#", false);
}

/**
*
* Method: getString(String name, String defaultValue)
*
*/
@Test
public void testGetString() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1 = new JsonObject();
    jsonObject0.set(":}7mjBZ -YUX.Dc}S", 0L);
    JsonObject jsonObject2 = jsonObject1.set("3Y•~", 0L);
    double double0 = jsonObject2.getDouble("0STB$Bho$8-(>eG[", 1.0);
    Assert.assertEquals(1.0, double0, 0.01);

    jsonObject0.set("3Y•~", 0.0F);
    String string0 = jsonObject0.getString("'XeSq[•@;~pq=iSX", "");
    Assert.assertEquals("", string0);
} //8794

/**
*
* Method: size()
*
*/
@Test
public void testSize() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    jsonObject0.set("", 0L);
    jsonObject0.set("", 0.0F);
    JsonObject jsonObject1 = jsonObject0.set("3Y•~", 3817L);
    jsonObject1.getDouble("0STB$Bho$8-(>eG[", 1.0);
    jsonObject0.set("3Y•~", 0.0F);
    Assert.assertEquals(2, jsonObject0.size());
}

/**
*
* Method: isEmpty()
*
*/
@Test
public void testIsEmpty() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1 = jsonObject0.set("nt.moocest.JsonObject$Member",
0L);
    jsonObject0.getString("value is null", "nt.moocest.JsonObject$Member");
    jsonObject0.asObject();
    jsonObject1.isEmpty();
} //8807
```

JsonObjectTest.java

```
/**
*
* Method: names()
*
*/
@Test
public void testNames() throws Exception {
    JsonObject.HashIndexTable jsonObject_HashIndexTable0 = new
JsonObject.HashIndexTable();
    jsonObject_HashIndexTable0.remove((-3293));
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1 = new JsonObject(jsonObject0);
    JsonObject jsonObject2 = new JsonObject(jsonObject1);
    JsonObject jsonObject3 = jsonObject2.set("8'", (double) (-3293));
    jsonObject0.getFloat("", (-3293));
    jsonObject_HashIndexTable0.remove((-3293));
    StringWriter stringWriter0 = new StringWriter();
    jsonObject2.writeTo((Writer) stringWriter0);
    jsonObject1.add("8'", (-3293.0F));
    jsonObject1.getFloat("8'", (-3293.0F));
    jsonObject1.names();
    jsonObject3.add("8'", "net.mooctest.JsonHandler");
    JsonObject.Member jsonObject_Member0 = new
JsonObject.Member("lYCup1=`K•M~$X%q$3", jsonObject1);
    jsonObject0.getString("8'", "net.mooctest.JsonHandler");
} //8820



/**
*
* Method: isObject()
*
*/
@Test
public void testIsObject() throws Exception {
    JsonObject.HashIndexTable jsonObject_HashIndexTable0 = new
JsonObject.HashIndexTable();
    jsonObject_HashIndexTable0.remove((-4228));
    JsonObject jsonObject0 = new JsonObject();
    jsonObject_HashIndexTable0.remove((-4228));
    JsonObject jsonObject1 = jsonObject0.set("5N(a", 0.0F);
    Assert.assertTrue(jsonObject1.isObject());
} //8832

/**
*
* Method: asObject()
*
*/
@Test
public void testAsObject() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    PrettyPrint.singleLine();
    jsonObject0.hashCode();
    JsonObject jsonObject1 = jsonObject0.add("LwGwuCWy4KhT?;E", 109L);
    JsonObject jsonObject2 = jsonObject0.asObject();
    JsonObject jsonObject3 = jsonObject0.set("LwGwuCWy4KhT?;E", (JsonValue)
jsonObject0);
```

JsonObjectTest.java

```
    JsonObject.Member jsonObject_Member0 = new
JsonObject.Member("LwGwuCWy4KhT?;E", jsonObject0);
    jsonObject3.getDouble("G", 109L);
    jsonObject0.size();
    jsonObject2.isObject();
    jsonObject_Member0.getName();
    jsonObject1.contains("iaHk-{x&G&5%/>>.}8F");
    jsonObject3.add("", "LwGwuCWy4KhT?;E");
    Assert.assertEquals(2, jsonObject0.size());
} //8845

/**
*
* Method: iterator()
*
*/
@Test
public void testIterator() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    jsonObject0.getFloat("", 0.0F);
    JsonObject jsonObject1 = jsonObject0.add("LwGwuCWy4KhT?;E", 109L);
    PrettyPrint prettyPrint0 = PrettyPrint.indentWithSpaces(0);
    StringWriter stringWriter0 = new StringWriter();
    WriterConfig.MINIMAL = (WriterConfig) prettyPrint0;
    JsonWriter jsonWriter0 = prettyPrint0.createWriter(stringWriter0);
    jsonObject0.write(jsonWriter0);
    jsonObject0.hashCode();
    jsonObject0.add("", 109L);
    JsonObject jsonObject2 = jsonObject0.add("", "");
    JsonObject jsonObject3 = jsonObject0.asObject();
    jsonObject2.iterator();
    jsonObject3.set("", false);
    JsonObject jsonObject4 = jsonObject2.set("", (JsonValue) jsonObject1);
    JsonObject.Member jsonObject_Member0 = new JsonObject.Member("",
jsonObject2);
    jsonObject_Member0.getName();
    jsonObject0.size();
    jsonObject3.isObject();
    JsonObject jsonObject5 = jsonObject2.set("LwGwuCWy4KhT?;E", 340.0523F);
    jsonObject5.add("", (float) 0);
    jsonObject_Member0.getName();
    jsonObject5.contains("iaHk-{x&G&5%/>>.}8F");
    jsonObject4.add("", "");
    jsonObject2.get("");
    Assert.assertEquals(5, jsonObject0.size());
}

/**
*
* Method: write(JsonWriter writer)
*
*/
@Test
public void testWrite() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    jsonObject0.getFloat("", 0.0F);
    JsonObject jsonObject1 = jsonObject0.add("LwGwuCWy4KhT?;E", 109L);
    PrettyPrint prettyPrint0 = PrettyPrint.indentWithSpaces(0);
    StringWriter stringWriter0 = new StringWriter();
```

JsonObjectTest.java

```
    WriterConfig.MINIMAL = (WriterConfig) prettyPrint0;
    JsonWriter jsonWriter0 = prettyPrint0.createWriter(stringWriter0);
    jsonObject0.write(jsonWriter0);
    jsonObject0.hashCode();
    jsonObject0.add("", 109L);
    JsonObject jsonObject2 = jsonObject0.add("", "");
    JsonObject jsonObject3 = jsonObject0.asObject();
    jsonObject2.iterator();
    JsonObject jsonObject4 = jsonObject2.set("", (JsonValue) jsonObject1);
    JsonObject.Member jsonObject_Member0 = new JsonObject.Member("",
jsonObject2);
    jsonObject_Member0.getName();
    jsonObject0.size();
    jsonObject3.isObject();
    JsonObject jsonObject5 = jsonObject2.set("LwGwuCWy4KhT?;E", 340.0523F);
    jsonObject5.add("", (float) 0);
    jsonObject_Member0.getName();
    jsonObject5.contains("iaHk-{x&G&5%/>>.}8F");
    jsonObject4.add("", "");
    Assert.assertEquals(5, jsonObject1.size());
    Assert.assertEquals(5, jsonObject0.size());
}




/**
*
* Method: hashCode()
*
*/
@Test
public void testHashCode() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    PrettyPrint prettyPrint0 = PrettyPrint.singleLine();
    StringWriter stringWriter0 = new StringWriter();
    WriterConfig.MINIMAL = (WriterConfig) prettyPrint0;
    JsonWriter jsonWriter0 = prettyPrint0.createWriter(stringWriter0);
    jsonObject0.write(jsonWriter0);
    jsonObject0.hashCode();
}

/**
*
* Method: equals(Object obj)
*
*/
@Test
public void testEqualsObj() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1 = jsonObject0.set("", 1);
    jsonObject1.toString();
    JsonObject jsonObject2 = jsonObject0.add("", 1909);
    String string0 = "n/w\\)";
    jsonObject0.get("n/w)");
    JsonObject jsonObject3 = jsonObject0.set("n/w)", 0L);
    JsonObject jsonObject4 = JsonObject.unmodifiableObject(jsonObject3);
    JsonObject jsonObject5 = JsonObject.unmodifiableObject(jsonObject2);
```

JsonObjectTest.java

```
    jsonObject2.set("#L(Y?6XAB", "#L(Y?6XAB");
    JsonObject jsonObject6 = jsonObject3.add("config is null", (JsonValue)
jsonObject4);
    jsonObject6.set(";/(maJ", (double) 1);
    jsonObject3.equals(jsonObject5);
    String string1 = "Klxx]d6[8b+@7c {}";
    jsonObject0.getInt("", 93);



}
    @Test
    public void testEqualsObj1() throws Exception {
        JsonObject jsonObject0 = new JsonObject();
        JsonObject jsonObject1 = new JsonObject();
        String string0 = "net.mooctest.JsonWriter";
        jsonObject0.getLong("net.mooctest.JsonWriter", 0L);
        JsonObject.Member jsonObject_Member0 = new JsonObject.Member("object
is null", jsonObject0);
        jsonObject_Member0.equals(jsonObject_Member0);
        jsonObject0.asObject();
    }
    @Test
    public void testEqualsObj2() throws Exception {
        JsonObject jsonObject0 = new JsonObject();
        JsonObject.Member jsonObject_Member0 = new JsonObject.Member("",
jsonObject0);
        jsonObject_Member0.equals("object is null");
        jsonObject_Member0.equals((Object) null);
        jsonObject0.add("wWjO<<` rHE;", (-1.0F));
        JsonObject jsonObject1 = jsonObject0.add(">D3{•PdLe$A4", false);
        Assert.assertEquals(2, jsonObject1.size());
    }

    /**
*
* Method: indexOf(String name)
*
*/
@Test
public void testIndexOf() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    jsonObject0.isEmpty();
    String string0 = ">QGj!_9%}2zt5dE>[";
    JsonObject jsonObject1 = jsonObject0.set(">QGj!_9%}2zt5dE>[", 32);
    JsonObject jsonObject2 = jsonObject0.merge(jsonObject1);
    JsonObject jsonObject3 = jsonObject2.set(">QGj!_9%}2zt5dE>[",
">QGj!_9%}2zt5dE>[");
    char[] charArray0 = new char[7];
    charArray0[0] = 'j';
    charArray0[1] = '•';
    charArray0[2] = '!';
    charArray0[3] = 'Q';
    charArray0[4] = '}';
    charArray0[5] = '/';
    charArray0[6] = '.';
    PrettyPrint prettyPrint0 = new PrettyPrint(charArray0);
    jsonObject3.toString((WriterConfig) prettyPrint0);
    StringWriter stringWriter0 = new StringWriter();
```

JsonObjectTest.java

```java
    jsonObject2.writeTo((Writer) stringWriter0);
    jsonObject1.indexOf(">QGj!_9%}2zt5dE>[");
    jsonObject2.contains(">QGj!_9%}2zt5dE>[");
    JsonObject jsonObject4 = jsonObject2.add(">QGj!_9%}2zt5dE>[", (long) 0);
    jsonObject4.names();
    String string1 = "\\";
    jsonObject1.remove("");
}

/**
*
* Method: getValue()
*
*/
@Test
public void testGetValue() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    JsonObject.Member jsonObject_Member0 = new JsonObject.Member("",
jsonObject0);
    jsonObject_Member0.getValue();
    JsonObject.HashIndexTable jsonObject_HashIndexTable0 = new
JsonObject.HashIndexTable();
    jsonObject0.remove("");
    jsonObject_HashIndexTable0.add("", 1);
    JsonObject jsonObject1 = jsonObject0.asObject();
    JsonObject jsonObject2 = jsonObject1.asObject();
    jsonObject2.getFloat("", 0.0F);
    JsonObject jsonObject3 = jsonObject2.add("", false);
    jsonObject3.get("");
    PrettyPrint.singleLine();
}

/**
*
* Method: equals(Object object)
*
*/
@Test
public void testEqualsObject() throws Exception {
    JsonObject jsonObject0 = new JsonObject();
    String string0 = "-.%UaU/Z+S}!gE(+=1h";
    JsonObject jsonObject1 = jsonObject0.set("-.%UaU/Z+S}!gE(+=1h", 0.0);
    jsonObject0.getLong("-.%UaU/Z+S}!gE(+=1h", 0L);
    JsonObject.unmodifiableObject(jsonObject1);
    JsonObject jsonObject2 = new JsonObject();
    jsonObject0.contains("-.%UaU/Z+S}!gE(+=1h");
    JsonObject.Member jsonObject_Member0 = new JsonObject.Member("object is
null", jsonObject2);
    JsonObject.Member jsonObject_Member1 = new JsonObject.Member("object is
null", jsonObject0);
    jsonObject_Member0.equals(jsonObject_Member1);
}

/**
*
* Method: add(String name, int index)
*
*/
@Test
```

JsonObjectTest.java

```java
public void testAddForNameIndex() throws Exception {
    JsonObject.HashIndexTable jsonObject_HashIndexTable0 = new
JsonObject.HashIndexTable();
    jsonObject_HashIndexTable0.add("151556", 2715);
    JsonObject.HashIndexTable jsonObject_HashIndexTable1 = new
JsonObject.HashIndexTable();
    JsonObject.readFrom("{\"cumt\":1909}");
}

/**
*
* Method: remove(int index)
*
*/
@Test
public void testRemoveIndex() throws Exception {
    JsonObject.HashIndexTable jsonObject_HashIndexTable0 = new
JsonObject.HashIndexTable();
    jsonObject_HashIndexTable0.remove((-3293));
    JsonObject jsonObject0 = new JsonObject();
    JsonObject jsonObject1 = new JsonObject(jsonObject0);
    JsonObject jsonObject2 = new JsonObject(jsonObject1);
    JsonObject jsonObject3 = jsonObject2.set("8'", (double) (-3293));
    jsonObject0.getFloat("", (-3293));
    jsonObject_HashIndexTable0.remove((-3293));
    StringWriter stringWriter0 = new StringWriter();
    jsonObject2.writeTo((Writer) stringWriter0);
    jsonObject1.add("8'", (-3293.0F));
    jsonObject1.getFloat("8'", (-3293.0F));
    JsonObject jsonObject4 = new JsonObject(jsonObject2);
    JsonObject.Member jsonObject_Member0 = new
JsonObject.Member("lYCup1=`K•M~$X%q$3", jsonObject1);
    JsonObject jsonObject5 = JsonObject.unmodifiableObject(jsonObject3);
    jsonObject_Member0.equals(jsonObject5);
    jsonObject4.add("9nu{8j| .]NK#)", (long) (-3293));
    Assert.assertTrue(jsonObject2.equals((Object)jsonObject1));
    Assert.assertTrue(jsonObject1.equals((Object)jsonObject3));
}
}
```

代码 4.3-6 JsonParserTest 类

JsonParserTest.java

```java
package mooctest;

import net.mooctest.*;
import org.junit.Assert;
import org.junit.Test;
import org.junit.Before;
import org.junit.After;

import java.io.IOException;
import java.io.Reader;
import java.io.StringReader;
import java.util.Arrays;

/**
* JsonParser Tester.
*
* @author Hujunyao CUMT CS 2019-4 06192081
* @since <pre>7 月 4, 2022</pre>
* @version 1.0
*/
public class JsonParserTest {
    JsonParser jsonParser;
    JsonHandler<JsonArray, JsonObject> jsonHandler;
@Before
public void before() throws Exception {
    jsonHandler = new JsonHandler<JsonArray, JsonObject>() {

    };
    jsonParser = new JsonParser(jsonHandler);
}

@After
public void after() throws Exception {
}

@Test (expected = NullPointerException.class)
public void test1() throws Exception {
    JsonHandler<JsonArray, JsonObject> jsonHandler1 = null;
    new JsonParser(jsonHandler1);

}
    /**
*
* Method: parse(String string)
*
*/
@Test
public void testParseString() throws Exception {
    jsonParser.parse("[1,2,3,\"h\"]");
    Assert.assertEquals("[[, 1, ,, 2, ,, 3, ,, \", h, \", ]]",
Arrays.toString(jsonParser.buffer));
}

@Test(expected = NullPointerException.class)
public void testParseString0() throws Exception {
    String string = null;
    jsonParser.parse(string);
}
```

JsonParserTest.java

```
    /**
*
* Method: parse(Reader reader)
*
*/
@Test
public void testParseReader() throws Exception {
    Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
    JsonParser jsonParser0 = new JsonParser(jsonHandler0);
    StringReader stringReader0 = new StringReader("null");
    jsonParser0.parse((Reader) stringReader0);
    Assert.assertEquals((-1), jsonParser0.fill);

}

/**
*
* Method: parse(Reader reader, int buffersize)
*
*/
@Test
public void testParseForReaderBuffersize() throws Exception {
   Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
    JsonParser jsonParser1 = new JsonParser(jsonHandler0);
    StringReader stringReader1 = new StringReader("true");
    jsonParser1.parse((Reader) stringReader1, 44);
    Assert.assertEquals((-1), jsonParser1.fill);
}

/**
*
* Method: getLocation()
*
*/
@Test
public void testGetLocation() throws Exception {
    Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
    JsonParser jsonParser0 = new JsonParser(jsonHandler0);
    Location location0 = jsonParser0.getLocation();
    Assert.assertEquals((-1), location0.offset);
    Assert.assertEquals(0, location0.column);
    Assert.assertEquals(0, location0.line);
}


    @Test(timeout = 4000)
    public void test18()  throws Throwable  {
        Json.DefaultHandler json_DefaultHandler0 = new
Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(json_DefaultHandler0);
        StringReader stringReader0 = new StringReader("8237e9");
        jsonParser0.parse((Reader) stringReader0);
        jsonParser0.parse("8237e9");
        Assert.assertEquals((-1), jsonParser0.fill);
    }
```

JsonParserTest.java

```
    @Test(timeout = 4000)
    public void test22()  throws Throwable  {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        StringReader stringReader0 = new StringReader("44");
        jsonParser0.parse((Reader) stringReader0);
        Assert.assertEquals((-1), jsonParser0.fill);
    }


    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test00() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse("\" ");
    }



    @Test(timeout = 4000, expected = IllegalArgumentException.class)
    public void test02() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        StringReader stringReader0 = new StringReader("R@B]!>=/f@PIk4(");
        jsonParser0.parse((Reader) stringReader0, (-5177));

    } //7652

    @Test(timeout = 4000)
    public void test07() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        StringReader stringReader0 = new StringReader("8");
        jsonParser0.parse((Reader) stringReader0);

    } //x

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test08() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        StringReader stringReader0 = new StringReader("\"ed,");

        jsonParser0.parse((Reader) stringReader0);
    }

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test09() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse("30CAlu\tMSWz");
    }

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test10() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(null);
        jsonParser0.parse("[0CAlu\tMWz");
    } //7665
```

JsonParserTest.java

```
    @Test(timeout = 4000)
    public void test11() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        StringReader stringReader0 = new StringReader("true");
        jsonParser0.parse((Reader) stringReader0, 44);
        Assert.assertEquals((-1), jsonParser0.fill);
    }

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test12() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        StringReader stringReader0 = new StringReader("30CAlu\tMSWz");
        jsonParser0.parse((Reader) stringReader0, 34);
    }

    @Test(timeout = 4000, expected = NullPointerException.class)
    public void test13() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse((Reader) null, (-701));

    }

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test14() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse(" at ");
    }//7716

    @Test(timeout = 4000, expected = NullPointerException.class)
    public void test15() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(null);
        jsonParser0.parse("{}");
        Assert.assertEquals((-1), jsonParser0.fill);
    }

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test16() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(null);
        jsonParser0.parse("{F");
    }

    @Test(timeout = 4000)
    public void test17() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse("9");
        Assert.assertEquals((-1), jsonParser0.fill);
    }


    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test19() throws Throwable {
```

JsonParserTest.java

```
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        StringReader stringReader0 = new StringReader("7My");
        jsonParser0.parse((Reader) stringReader0);
    }

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test20() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse("6lyD\"t#i(HDVb0-hMZt");
    }

    @Test(timeout = 4000)
    public void test21() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse("5");
        Assert.assertEquals((-1), jsonParser0.fill);
    }


    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test23() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse("2ECTUvxsTR['_");

    }

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test24() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse("1~$0*Z00%7");

    }

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test25() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse("-");

    } //7766

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test26() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse("\"\"de,");
    }

    @Test(timeout = 4000, expected = IllegalArgumentException.class)
    public void test27() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        StringReader stringReader0 = new
StringReader("net.mooctest.Json$DefaultHa@dler");
```

JsonParserTest.java

```
        jsonParser0.parse((Reader) stringReader0, 0);
    }

    @Test(timeout = 4000, expected = NullPointerException.class)
    public void test28() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse((String) null);

    }

    @Test(timeout = 4000, expected = NullPointerException.class)
    public void test29() throws Throwable {
        JsonParser jsonParser0 = null;
        jsonParser0 = new JsonParser((JsonHandler<?, ?>) null);

    }

    @Test(timeout = 4000, expected = RuntimeException.class)
    public void test30() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse("net.mooctest.JsonWriter");

    }

    @Test(timeout = 4000, expected = NullPointerException.class)
    public void test31() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse((Reader) null);
    }

    @Test(timeout = 4000)
    public void test32() throws Throwable {
        Json.DefaultHandler jsonHandler0 = new Json.DefaultHandler();
        JsonParser jsonParser0 = new JsonParser(jsonHandler0);
        jsonParser0.parse("false");
        Assert.assertEquals((-1), jsonParser0.fill);
    } //7779
}
```

代码 4.3-7 JsonStringTest 类

JsonStringTest.java

```java
package mooctest;

import net.mooctest.JsonString;
import net.mooctest.JsonWriter;
import org.junit.Assert;
import org.junit.Test;
import org.junit.Before;

import java.io.IOException;
import java.io.StringWriter;
import java.io.Writer;

/**
* JsonString Tester.
*
* @author Hujunyao CUMT CS 2019-4 06192081
* @since <pre>7 月 2, 2022</pre>
* @version 1.0
*/
public class JsonStringTest {
    JsonString jsonString;
@Before
public void before() throws Exception {
    jsonString = new JsonString("hello");
}



@Test(expected = NullPointerException.class)
public void test1() throws Exception {
    JsonString jsonString1 = new JsonString(null);
}
    /**
*
* Method: write(JsonWriter writer)
*
*/
@Test
public void testWrite() throws Exception {
    Writer writer11 = new Writer() {
        @Override
        public void write(char[] cbuf, int off, int len) throws IOException
{
        }

        @Override
        public void flush() throws IOException {
        }

        @Override
        public void close() throws IOException {
        }
    };
    JsonWriter jsonWriter = new JsonWriter(writer11);
    jsonString.write(jsonWriter);

}
```

JsonStringTest.java

```java
/**
*
* Method: isString()
*
*/
@Test
public void testIsString() throws Exception {
    System.out.println(jsonString.isString());
}

/**
*
* Method: asString()
*
*/
@Test
public void testAsString() throws Exception {
    System.out.println(jsonString.asString());
}

/**
*
* Method: hashCode()
*
*/
@Test
public void testHashCode() throws Exception {
    System.out.println(jsonString.hashCode());
}

/**
*
* Method: equals(Object object)
*
*/
@Test
public void testEquals() throws Exception {
    JsonString jsonString1 = jsonString;
    JsonString jsonString2 = new JsonString("hello");
    JsonString jsonString3 = new JsonString("hello1");
    JsonString jsonString4 = null;
    String string = "hello";
    Assert.assertEquals(jsonString,jsonString1);
    Assert.assertEquals(jsonString,jsonString2);
    Assert.assertNotEquals(jsonString,jsonString3);
    Assert.assertNotEquals(jsonString,jsonString4);
    Assert.assertNotEquals(jsonString,string);


}

@Test(timeout = 4000)
public void test07()  throws Throwable  {
    JsonString jsonString0 = new JsonString("net.mooctest.JsonWriter");
    StringWriter stringWriter0 = new StringWriter();
    JsonWriter jsonWriter0 = new JsonWriter(stringWriter0);
    jsonString0.write(jsonWriter0);
    Assert.assertFalse(jsonString0.isObject());
    Assert.assertFalse(jsonString0.isArray());
```

JsonStringTest.java

```
    Assert.assertFalse(jsonString0.isBoolean());
    Assert.assertFalse(jsonString0.isTrue());
    Assert.assertFalse(jsonString0.isFalse());
    Assert.assertFalse(jsonString0.isNull());
}
}
```

代码 4.3-8 JsonWriterTest 类

```
JsonWriterTest.java
package mooctest;

import net.mooctest.JsonWriter;
import net.mooctest.WritingBuffer;
import org.junit.Assert;
import org.junit.Test;
import org.junit.Before;

import java.io.IOException;
import java.io.StringWriter;
import java.io.Writer;
import java.util.Arrays;

/**
 * JsonWriter Tester.
 *
 * @author Hujunyao CUMT CS 2019-4 06192081
 * @version 1.0
 * @since <pre>7�� 2, 2022</pre>
 */
public class JsonWriterTest {
    JsonWriter jsonWriter;
    WritingBuffer writingBuffer;

    @Before
    public void before() throws Exception {
        writingBuffer= new WritingBuffer(new Writer() {
            @Override
            public void write(char[] cbuf, int off, int len) throws
IOException {System.out.println(1);}

            @Override
            public void flush() throws IOException {
                System.out.println(2);
            }

            @Override
            public void close() throws IOException {
                System.out.println(3);
            }
        });
        jsonWriter = new JsonWriter(writingBuffer);
    }

    /**
     * Method: writeLiteral(String value)
     */
    @Test
    public void testWriteLiteral() throws Exception {
        jsonWriter.writeLiteral("qqq");
    }

    /**
     * Method: writeNumber(String string)
     */
    @Test
    public void testWriteNumber() throws Exception {
```

JsonWriterTest.java

```
        jsonWriter.writeNumber("123");
    }

    /**
     * Method: writeString(String string)
     */
    @Test
    public void testWriteString() throws Exception {
        jsonWriter.writeString("emmm");
    }

    /**
     * Method: writeArrayOpen()
     */
    @Test
    public void testWriteArrayOpen() throws Exception {
        jsonWriter.writeArrayOpen();
    }

    /**
     * Method: writeArrayClose()
     */
    @Test
    public void testWriteArrayClose() throws Exception {
        jsonWriter.writeArrayClose();
    }

    /**
     * Method: writeArraySeparator()
     */
    @Test
    public void testWriteArraySeparator() throws Exception {
        jsonWriter.writeArraySeparator();
    }

    /**
     * Method: writeObjectOpen()
     */
    @Test
    public void testWriteObjectOpen() throws Exception {
        jsonWriter.writeObjectOpen();
    }

    /**
     * Method: writeObjectClose()
     */
    @Test
    public void testWriteObjectClose() throws Exception {
        jsonWriter.writeObjectClose();
    }

    /**
     * Method: writeMemberName(String name)
     */
    @Test
    public void testWriteMemberName() throws Exception {
        jsonWriter.writeMemberName("abcd");
    }
```

JsonWriterTest.java

```
    /**
     * Method: writeMemberSeparator()
     */
    @Test
    public void testWriteMemberSeparator() throws Exception {
        jsonWriter.writeMemberSeparator();
    }

    /**
     * Method: writeObjectSeparator()
     */
    @Test
    public void testWriteObjectSeparator() throws Exception {
        jsonWriter.writeObjectSeparator();
    }

    /**
     * Method: writeJsonString(String string)
     */
    @Test
    public void testWriteJsonString() throws Exception {
        jsonWriter.writeJsonString("hjy[1]\\\"\n\r\t");
        Assert.assertEquals(Arrays.toString(new
char[]{'h','j','y','[','1',']','\\', '\\', '\\', '\"', '\\', 'n', '\\', 'r',
'\\', 't'}), Arrays.toString(writingBuffer.buffer));
    }

    /**
     * Method: getReplacementChars(char ch)
     */
    @Test
    public void testGetReplacementChars() throws Exception {
        Assert.assertNull(JsonWriter.getReplacementChars('a'));
        Assert.assertNull(JsonWriter.getReplacementChars('A'));
        Assert.assertNull(JsonWriter.getReplacementChars('1'));
        Assert.assertEquals(Arrays.toString(new char[]{'\\', 'u', '2', '0',
'2', '8'}), Arrays.toString(JsonWriter.getReplacementChars('\u2028')));
        Assert.assertEquals(Arrays.toString(new char[]{'\\', 'u', '2', '0',
'2', '9'}), Arrays.toString(JsonWriter.getReplacementChars('\u2029')));
        Assert.assertEquals(Arrays.toString(new char[]{'\\','\\'}),
Arrays.toString(JsonWriter.getReplacementChars('\\')));
        Assert.assertEquals(Arrays.toString(new char[]{'\\','"'}),
Arrays.toString(JsonWriter.getReplacementChars('\"')));
        Assert.assertEquals(Arrays.toString(new char[]{'\\','n'}),
Arrays.toString(JsonWriter.getReplacementChars('\n')));
        Assert.assertEquals(Arrays.toString(new char[]{'\\','r'}),
Arrays.toString(JsonWriter.getReplacementChars('\r')));
        Assert.assertEquals(Arrays.toString(new char[]{'\\','t'}),
Arrays.toString(JsonWriter.getReplacementChars('\t')));
        Assert.assertEquals(Arrays.toString(new char[]{'\\', 'u', '0', '0',
'1', 'a'}), Arrays.toString(JsonWriter.getReplacementChars('\u001a')));
    }

    @Test(timeout = 4000)
    public void test19()  throws Throwable  {
        StringWriter stringWriter0 = new StringWriter();
        JsonWriter jsonWriter0 = new JsonWriter(stringWriter0);
        jsonWriter0.writeJsonString("wA2N!m^\"NW{L;K:Ua%");
        Assert.assertEquals("wA2N!m^\\\"NW{L;K:Ua%",
```

JsonWriterTest.java

```
stringWriter0.toString());
    }
}
```

代码 4.3-9 LocationTest 类

LocationTest.java

```java
package mooctest;

import net.mooctest.Location;
import org.junit.Assert;
import org.junit.Test;

/**
* Location Tester.
*
* @author Hujunyao CUMT CS 2019-4 06192081
* @since <pre>6�� 29, 2022</pre>
* @version 1.0
*/
public class LocationTest {

/**
*
* Method: toString()
*
*/
@Test
public void testToString() throws Exception {
    Location location = new Location(1001,1002,1003);
    Assert.assertEquals("1002:1003", location.toString());
}

/**
*
* Method: hashCode()
*
*/
@Test
public void testHashCode() throws Exception {
    Location location = new Location(1001,1002,1003);
    Assert.assertEquals(1001, location.hashCode());
}

/**
*
* Method: equals(Object obj)
*
*/
@Test
public void testEquals() throws Exception {
    Location location = new Location(1001,1002,1003);
    Location location1 = location;
    Location location2 = null;
    Integer integer = 10;
    Location location3 = new Location(1001,1002,1003);
    Location location4 = new Location(1000,1002,1003);
    Location location5 = new Location(1001,1005,1003);
    Location location6 = new Location(1001,1002,1006);

    Assert.assertEquals(location, location1);
    Assert.assertNotEquals(location, location2);
    Assert.assertNotEquals(location, integer);
    Assert.assertEquals(location, location3);
```

LocationTest.java

```
    Assert.assertNotEquals(location, location4);
    Assert.assertNotEquals(location, location5);
    Assert.assertNotEquals(location, location6);
}
}
```

代码 4.3-10 ParseExceptionTest 类

ParseExceptionTest.java

```java
package mooctest;

import net.mooctest.Location;
import net.mooctest.ParseException;
import org.junit.Assert;
import org.junit.Test;

/**
* ParseException Tester.
*
* @author Hujunyao CUMT CS 2019-4 06192081
* @since <pre>6�� 29, 2022</pre>
* @version 1.0
*/
public class ParseExceptionTest {

/**
*
* Method: getLocation()
*
*/
@Test
public void testGetLocation() throws Exception {
    ParseException parseException = new ParseException("test", new
Location(101,102,103));
    Assert.assertEquals("102:103",parseException.getLocation().toString());
}

/**
*
* Method: getOffset()
*
*/
@Test
public void testGetOffset() throws Exception {
    ParseException parseException = new ParseException("test", new
Location(101,102,103));
    Assert.assertEquals(101,parseException.getOffset());
}

/**
*
* Method: getLine()
*
*/
@Test
public void testGetLine() throws Exception {
    ParseException parseException = new ParseException("test", new
Location(101,102,103));
    Assert.assertEquals(102,parseException.getLine());
}

/**
*
* Method: getColumn()
*
*/
```

ParseExceptionTest.java

```
@Test
public void testGetColumn() throws Exception {
    ParseException parseException = new ParseException("test", new
Location(101,102,103));
    Assert.assertEquals(103,parseException.getColumn());
}
}
```

代码 4.3-11 PrettyPrintTest 类

PrettyPrintTest.java

```java
package mooctest;

import net.mooctest.JsonWriter;
import net.mooctest.PrettyPrint;
import net.mooctest.WritingBuffer;
import org.junit.Assert;
import org.junit.Test;
import org.junit.Before;
import org.junit.After;

import java.io.IOException;
import java.io.StringWriter;
import java.io.Writer;

/**
* PrettyPrint Tester.
*
* @author Hujunyao CUMT CS 2019-4 06192081
* @since <pre>6 月 29, 2022</pre>
* @version 1.0
*/
public class PrettyPrintTest {
    PrettyPrint prettyPrint;
    Writer writer = new Writer() {
        @Override
        public void write(char[] cbuf, int off, int len) throws IOException
{
            writingBuffer.write(cbuf, off, len);
        }

        @Override
        public void flush() throws IOException {
            writingBuffer.flush();
        }

        @Override
        public void close() throws IOException {
            writingBuffer.close();
        }
    };
    WritingBuffer writingBuffer = new WritingBuffer(writer, 200);
    PrettyPrint.PrettyPrintWriter prettyPrintWriter;
    PrettyPrint prettyPrint1;
    PrettyPrint.PrettyPrintWriter prettyPrintWriter1;


    @Before
public void before() throws Exception {
    prettyPrint = new PrettyPrint(new char[]{'-','-'});
    prettyPrintWriter = (PrettyPrint.PrettyPrintWriter)
prettyPrint.createWriter(writer);
}

@After
public void after() throws Exception {
}

/**
```

PrettyPrintTest.java

```
*
* Method: singleLine()
*
*/
@Test
public void testSingleLine() throws Exception {
    PrettyPrint.singleLine();
}

/**
*
* Method: indentWithSpaces(int number)
*
*/
@Test(expected = IllegalArgumentException.class)
public void testIndentWithSpaces() throws Exception {
    PrettyPrint.indentWithSpaces(-2);
}

@Test
public void testIndentWithSpaces1() throws Exception {
    PrettyPrint.indentWithSpaces(4);
}

    /**
*
* Method: indentWithTabs()
*
*/
@Test
public void testIndentWithTabs() throws Exception {
    PrettyPrint.indentWithTabs();
}

/**
*
* Method: createWriter(Writer writer)
*
*/
@Test
public void testCreateWriter() throws Exception {
    prettyPrint.createWriter(writer);
}

/**
*
* Method: writeArrayOpen()
*
*/
@Test
public void testWriteArrayOpen() throws Exception {
    String string = "[\n--";
    prettyPrintWriter.writeArrayOpen();
    for ( int i = 0; i < writingBuffer.fill; i++) {
            System.out.print(writingBuffer.buffer[i]);
            Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}
```

PrettyPrintTest.java

```
/**
*
* Method: writeArrayClose()
*
*/
@Test
public void testWriteArrayClose() throws Exception {
    String string = "\n]";
    prettyPrintWriter.writeArrayClose();
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

/**
*
* Method: writeArraySeparator()
*
*/
@Test
public void testWriteArraySeparator() throws Exception {
    String string = ",\n";
    prettyPrintWriter.writeArraySeparator();
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

/**
*
* Method: writeObjectOpen()
*
*/
@Test
public void testWriteObjectOpen() throws Exception {
    String string = "{\n--";
    prettyPrintWriter.writeObjectOpen();
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

/**
*
* Method: writeObjectClose()
*
*/
@Test
public void testWriteObjectClose() throws Exception {
    String string = "\n}";
    prettyPrintWriter.writeObjectClose();
    for ( int i = 0; i < writingBuffer.fill; i++) {
```

PrettyPrintTest.java

```
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

/**
*
* Method: writeMemberSeparator()
*
*/
@Test
public void testWriteMemberSeparator() throws Exception {
    String string = ": ";
    prettyPrintWriter.writeMemberSeparator();
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

/**
*
* Method: writeObjectSeparator()
*
*/
@Test
public void testWriteObjectSeparator() throws Exception {
    String string = ",\n";
    prettyPrintWriter.writeObjectSeparator();
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

/**
*
* Method: writeNewLine()
*
*/
@Test
public void testWriteNewLine() throws Exception {
    String string = "[\n--\n--\n]\n";
    prettyPrintWriter.writeArrayOpen();
    Assert.assertTrue(prettyPrintWriter.writeNewLine());
    prettyPrintWriter.writeArrayClose();
    Assert.assertTrue(prettyPrintWriter.writeNewLine());
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

@Test
public void testWriteNewLine01() throws Exception {
```

PrettyPrintTest.java

```
    prettyPrint1 = new PrettyPrint(null);
    prettyPrintWriter1 = (PrettyPrint.PrettyPrintWriter)
prettyPrint1.createWriter(writer);

    String string = "[]";
    prettyPrintWriter1.writeArrayOpen();
    Assert.assertFalse(prettyPrintWriter1.writeNewLine());
    prettyPrintWriter1.writeArrayClose();
    Assert.assertFalse(prettyPrintWriter1.writeNewLine());
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}


@Test(timeout = 4000)
public void test02()  throws Throwable  {
    PrettyPrint prettyPrint0 = PrettyPrint.singleLine();
    StringWriter stringWriter0 = new StringWriter();
    JsonWriter jsonWriter0 = prettyPrint0.createWriter(stringWriter0);
    jsonWriter0.writeObjectSeparator();
    Assert.assertEquals(", ", stringWriter0.toString());
}

@Test(timeout = 4000)
public void test03()  throws Throwable  {
    char[] charArray0 = new char[5];
    PrettyPrint prettyPrint0 = new PrettyPrint(charArray0);
    StringWriter stringWriter0 = new StringWriter();
    JsonWriter jsonWriter0 = prettyPrint0.createWriter(stringWriter0);
    jsonWriter0.writeObjectSeparator();
    Assert.assertEquals(",\n", stringWriter0.toString());
}

@Test(timeout = 4000)
public void test04()  throws Throwable  {
    PrettyPrint prettyPrint0 = new PrettyPrint((char[]) null);
    StringWriter stringWriter0 = new StringWriter();
    JsonWriter jsonWriter0 = prettyPrint0.createWriter(stringWriter0);
    jsonWriter0.writeArraySeparator();
    Assert.assertEquals(", ", stringWriter0.toString());
}

@Test(timeout = 4000)
public void test05()  throws Throwable  {
    PrettyPrint prettyPrint0 = PrettyPrint.indentWithTabs();
    StringWriter stringWriter0 = new StringWriter(3092);
    JsonWriter jsonWriter0 = prettyPrint0.createWriter(stringWriter0);
    jsonWriter0.writeArraySeparator();
    Assert.assertEquals(",\n", stringWriter0.toString());
}
}
```

代码 4.3-12 WritingBufferTest 类

WritingBufferTest.java

```java
package mooctest;

import net.mooctest.JsonWriter;
import net.mooctest.PrettyPrint;
import net.mooctest.WritingBuffer;
import org.junit.Assert;
import org.junit.Test;
import org.junit.Before;
import org.junit.After;

import java.io.IOException;
import java.io.StringWriter;
import java.io.Writer;

/**
* PrettyPrint Tester.
*
* @author Hujunyao CUMT CS 2019-4 06192081
* @since <pre>6 月 29, 2022</pre>
* @version 1.0
*/
public class PrettyPrintTest {
    PrettyPrint prettyPrint;
    Writer writer = new Writer() {
        @Override
        public void write(char[] cbuf, int off, int len) throws IOException
{
            writingBuffer.write(cbuf, off, len);
        }

        @Override
        public void flush() throws IOException {
            writingBuffer.flush();
//            System.out.println(writingBuffer.buffer);
        }

        @Override
        public void close() throws IOException {
            writingBuffer.close();
        }
    };
    WritingBuffer writingBuffer = new WritingBuffer(writer, 200);
    PrettyPrint.PrettyPrintWriter prettyPrintWriter;
    PrettyPrint prettyPrint1;
    PrettyPrint.PrettyPrintWriter prettyPrintWriter1;

    @Before
public void before() throws Exception {
    prettyPrint = new PrettyPrint(new char[]{'-','-'});
    prettyPrintWriter = (PrettyPrint.PrettyPrintWriter)
prettyPrint.createWriter(writer);
}

@After
public void after() throws Exception {
}

/**
```

WritingBufferTest.java

```
*
* Method: singleLine()
*
*/
@Test
public void testSingleLine() throws Exception {
    PrettyPrint.singleLine();
}

/**
*
* Method: indentWithSpaces(int number)
*
*/
@Test(expected = IllegalArgumentException.class)
public void testIndentWithSpaces() throws Exception {
    PrettyPrint.indentWithSpaces(-2);
}

@Test
public void testIndentWithSpaces1() throws Exception {
    PrettyPrint.indentWithSpaces(4);
}

    /**
*
* Method: indentWithTabs()
*
*/
@Test
public void testIndentWithTabs() throws Exception {
    PrettyPrint.indentWithTabs();
}

/**
*
* Method: createWriter(Writer writer)
*
*/
@Test
public void testCreateWriter() throws Exception {
    prettyPrint.createWriter(writer);
}

/**
*
* Method: writeArrayOpen()
*
*/
@Test
public void testWriteArrayOpen() throws Exception {
    String string = "[\n--";
    prettyPrintWriter.writeArrayOpen();
    for ( int i = 0; i < writingBuffer.fill; i++) {
            System.out.print(writingBuffer.buffer[i]);
            Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}
```

WritingBufferTest.java

```
/**
*
* Method: writeArrayClose()
*
*/
@Test
public void testWriteArrayClose() throws Exception {
    String string = "\n]";
    prettyPrintWriter.writeArrayClose();
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

/**
*
* Method: writeArraySeparator()
*
*/
@Test
public void testWriteArraySeparator() throws Exception {
    String string = ",\n";
    prettyPrintWriter.writeArraySeparator();
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

/**
*
* Method: writeObjectOpen()
*
*/
@Test
public void testWriteObjectOpen() throws Exception {
    String string = "{\n--";
    prettyPrintWriter.writeObjectOpen();
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

/**
*
* Method: writeObjectClose()
*
*/
@Test
public void testWriteObjectClose() throws Exception {
    String string = "\n}";
    prettyPrintWriter.writeObjectClose();
    for ( int i = 0; i < writingBuffer.fill; i++) {
```

WritingBufferTest.java

```
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

/**
*
* Method: writeMemberSeparator()
*
*/
@Test
public void testWriteMemberSeparator() throws Exception {
    String string = ": ";
    prettyPrintWriter.writeMemberSeparator();
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

/**
*
* Method: writeObjectSeparator()
*
*/
@Test
public void testWriteObjectSeparator() throws Exception {
    String string = ",\n";
    prettyPrintWriter.writeObjectSeparator();
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}


/**
*
* Method: writeNewLine()
*
*/
@Test
public void testWriteNewLine() throws Exception {
    String string = "[\n--\n--\n]\n";
    prettyPrintWriter.writeArrayOpen();
    Assert.assertTrue(prettyPrintWriter.writeNewLine());
    prettyPrintWriter.writeArrayClose();
    Assert.assertTrue(prettyPrintWriter.writeNewLine());
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);

}
```

WritingBufferTest.java

```java
@Test
public void testWriteNewLine01() throws Exception {
    prettyPrint1 = new PrettyPrint(null);
    prettyPrintWriter1 = (PrettyPrint.PrettyPrintWriter)
prettyPrint1.createWriter(writer);
    String string = "[]";
    prettyPrintWriter1.writeArrayOpen();
    Assert.assertFalse(prettyPrintWriter1.writeNewLine());
    prettyPrintWriter1.writeArrayClose();
    Assert.assertFalse(prettyPrintWriter1.writeNewLine());
    for ( int i = 0; i < writingBuffer.fill; i++) {
        System.out.print(writingBuffer.buffer[i]);
        Assert.assertEquals(string.charAt(i), writingBuffer.buffer[i]);
    }
    Assert.assertEquals(string.length(), writingBuffer.fill);
}

@Test(timeout = 4000)
public void test02()  throws Throwable  {
    PrettyPrint prettyPrint0 = PrettyPrint.singleLine();
    StringWriter stringWriter0 = new StringWriter();
    JsonWriter jsonWriter0 = prettyPrint0.createWriter(stringWriter0);
    jsonWriter0.writeObjectSeparator();
    Assert.assertEquals(", ", stringWriter0.toString());
}

@Test(timeout = 4000)
public void test03()  throws Throwable  {
    char[] charArray0 = new char[5];
    PrettyPrint prettyPrint0 = new PrettyPrint(charArray0);
    StringWriter stringWriter0 = new StringWriter();
    JsonWriter jsonWriter0 = prettyPrint0.createWriter(stringWriter0);
    jsonWriter0.writeObjectSeparator();
    Assert.assertEquals(",\n", stringWriter0.toString());
}

@Test(timeout = 4000)
public void test04()  throws Throwable  {
    PrettyPrint prettyPrint0 = new PrettyPrint((char[]) null);
    StringWriter stringWriter0 = new StringWriter();
    JsonWriter jsonWriter0 = prettyPrint0.createWriter(stringWriter0);
    jsonWriter0.writeArraySeparator();
    Assert.assertEquals(", ", stringWriter0.toString());
}

@Test(timeout = 4000)
public void test05()  throws Throwable  {
    PrettyPrint prettyPrint0 = PrettyPrint.indentWithTabs();
    StringWriter stringWriter0 = new StringWriter(3092);
    JsonWriter jsonWriter0 = prettyPrint0.createWriter(stringWriter0);
    jsonWriter0.writeArraySeparator();
    Assert.assertEquals(",\n", stringWriter0.toString());
}
}
```