第一部分

1. 什么是单元测试框架？

框架是指：构成一类特定软件可复用设计的一组相互协作的类。框架规定了应用的体系结构。它定义了整体结构，类和对象的分割，各部分的主要责任，类和对象怎么协作，以及控制流程。框架预定义了这些设计参数，以便于应用设计者或实现者能集中精力于应用本身的特定细节。[1]

单元测试框架：针对软件的最小单位（函数，方法）进行正确性的检查测试，构成此测试软件可复用设计的一组相互协作的类,即为单元测试框架。

2. 开发人员如何利用框架？

开发人员可直接复用测试框的类，在此基础上进行修改，可以跟据框架结构方便地构建测试用例，设计测试规则，执行测试并生成结果，通过断言来判断预期结果和实际结果的差异，统计测试进度、耗时、用例通过率，生成测试报告。

3. 框架可提供什么好处？

框架让测试变得更简单灵活，容易上手，框架已考虑测试的方方面面，测试者只需灵活考虑应该使用哪些模块，框架支持参数化，支持简单的单元测试和复杂的功能测试，还可用来做自动化测试、接口自动化测试，支持方便快速地生成测试报告等。

---

[1] “框架_百度百科.” *百度百科*, https://baike.baidu.com/item/%E6%A1%86%E6%9E%B6/1212667?fr=aladdin.

4. Jasmine 测试框架与 Pytest 测试框架的比较

	jasmine	unittest
开发语言	javascript	python
适用场景	单元测试，行为驱动开发（BDD）测试 适用于网页、Node.js 项目或任何可以运行 JavaScript 的地方	不仅可以适用于单元测试，还可以适用 WEB 自动化测试用例的开发与执行
优点	除了 JavaScript ，还可以运行在 Python 和 Ruby 中 被许多 CIS 使用和支持 内置用于断言的语法	python 自带，不用额外安装，学习便利，网上文档和例子多，可以较方便的收集用例处理断言并输出测试报告 可以参数化 二次开发方便
缺点	多数情况下，它需要一个测试运行器（如 Karma）。 难以异步测试。 比较适合一个统一的（客户端 - 服务器）单元测试解决方案。[2]	需要遵守框架的规则，学习成本较高，不支持失败重跑功能 用例格式复杂，兼容性无，插件少

第二部分

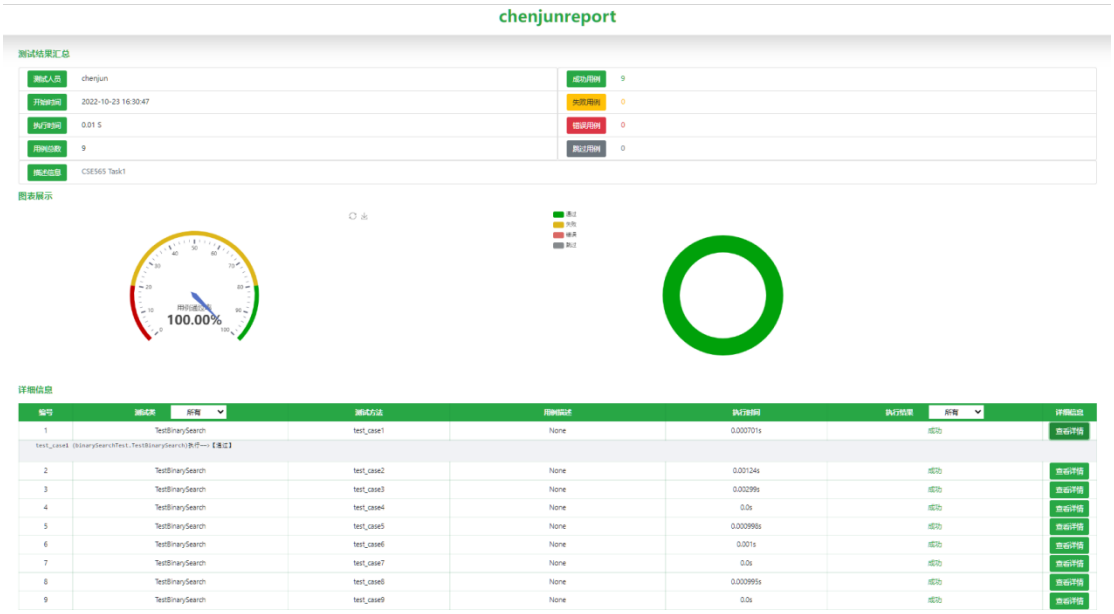
5. 测试代码和测试用例

[2] “自动化测试框架对比.” 自动化测试框架对比 - 走看看, http://t.zoukankan.com/yaoteng-p-10979005.html.

运用了等价类和边界值测试方法，等价类：分了全数字和全字母两类，边界值取了数组左右边界和中间的数，例如：[104 为左边界取值 103,104　右边界 980]取值 980,981　中间数 518 是跟据数组长度计算的，取值 572，518，616 最后测了空值的特殊情况，详细测试用例和代码见下面的图片：

```
import unittest
from binarySearch import binary_search
from unittestreport import TestRunner


class TestBinarySearch(unittest.TestCase):
    # equivalence partitioning testing --all data is numbers
    # boundary value testing  --none set
    def test_case9(self):
        arr = []
        self.assertEqual(binary_search(arr, 2), -1)

    # boundary value testing  --lef boundary
    def test_case1(self):
        arr = [104, 185, 219, 253, 313, 351, 412, 434, 518, 572, 626, 662, 674, 679, 736, 802, 825, 877, 923, 980]
        self.assertEqual(binary_search(arr, 103), -1)

    # boundary value testing  --lef boundary
    def test_case2(self):
        arr = [104, 185, 219, 253, 313, 351, 412, 434, 518, 572, 626, 662, 674, 679, 736, 802, 825, 877, 923, 980]
        self.assertEqual(binary_search(arr, 104), 0)

    # boundary value testing  --middle boundary
    def test_case3(self):
        arr = [104, 185, 219, 253, 313, 351, 412, 434, 518, 572, 626, 662, 674, 679, 736, 802, 825, 877, 923, 980]
        self.assertEqual(binary_search(arr, 572), 9)

    # boundary value testing  --middle boundary
    def test_case4(self):
        arr = [104, 185, 219, 253, 313, 351, 412, 434, 518, 572, 626, 662, 674, 679, 736, 802, 825, 877, 923, 980]
        self.assertEqual(binary_search(arr, 518), 8)

    # boundary value testing  --middle boundary
    def test_case5(self):
        arr = [104, 185, 219, 253, 313, 351, 412, 434, 518, 572, 626, 662, 674, 679, 736, 802, 825, 877, 923, 980]
        self.assertEqual(binary_search(arr, 626), 10)

    # boundary value testing  --right boundary
    def test_case6(self):
        arr = [104, 185, 219, 253, 313, 351, 412, 434, 518, 572, 626, 662, 674, 679, 736, 802, 825, 877, 923, 980]
        self.assertEqual(binary_search(arr, 980), 19)

    # boundary value testing  --right boundary
    def test_case7(self):
        arr = [104, 185, 219, 253, 313, 351, 412, 434, 518, 572, 626, 662, 674, 679, 736, 802, 825, 877, 923, 980]
        self.assertEqual(binary_search(arr, 981), -1)

    # equivalence partitioning testing --all data is letters
    def test_case8(self):
        arr = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't']
        self.assertEqual(binary_search(arr, 'c'), 2)

if __name__ == '__main__':
    case = unittest.defaultTestLoader.discover(r'D:\test',pattern='binarySearchTest.py')
    runner = TestRunner(case, tester="chenjun", filename="chenjunreport", report_dir=r"D:\test", title="chenjunreport",
                        templates=1,desc="CSE565 Task1")
    runner.run()
```

6. 测试报告


chenjunreport
测试结果汇总
测试人员
chenjun
开始时间
2022-10-23 16:30:47
执行时间
0.01 S
用例总数
9
描述信息
CSE565 Task1
成功用例
9
失败用例
0
错误用例
0
跳过用例
0
图表展示
100.00%
详细信息
编号
测试类
所有
测试方法
用例描述
执行时间
执行结果
所有
详细信息
1
TestBinarySearch
test_case1
None
0.000701s
成功
查看详情
test_case1 (binarySearchTest.TestBinarySearch)执行——【通过】
2
TestBinarySearch
test_case2
None
0.00124s
成功
查看详情
3
TestBinarySearch
test_case3
None
0.00299s
成功
查看详情
4
TestBinarySearch
test_case4
None
0.0s
成功
查看详情
5
TestBinarySearch
test_case5
None
0.000998s
成功
查看详情
6
TestBinarySearch
test_case6
None
0.001s
成功
查看详情
7
TestBinarySearch
test_case7
None
0.0s
成功
查看详情
8
TestBinarySearch
test_case8
None
0.000995s
成功
查看详情
9
TestBinarySearch
test_case9
None
0.0s
成功
查看详情