

1.单元测试

1.1概述【理解】

JUnit是一个 Java 编程语言的单元测试工具。JUnit 是一个非常重要的测试工具

1.2特点【理解】

- JUnit是一个开放源代码的测试工具。
- 提供注解来识别测试方法。
- JUnit测试可以让你编写代码更快，并能提高质量。
- JUnit优雅简洁。没那么复杂，花费时间较少。
- JUnit在一个条中显示进度。如果运行良好则是绿色；如果运行失败，则变成红色。

1.3使用步骤【应用】

- 使用步骤

1. 将junit的jar包导入到工程中 junit-4.9.jar
2. 编写测试方法该测试方法必须是公共的无参数无返回值的非静态方法
3. 在测试方法上使用@Test注解标注该方法是一个测试方法
4. 选中测试方法右键通过junit运行该方法

- 代码示例

```
XXXXXXXXXX

10
1
public class JunitDemol {
2
    @Test
3
    public void add() {
4
        System.out.println(2 / 0);
5
        int a = 10;
6
        int b = 20;
7
        int sum = a + b;
8
        System.out.println(sum);
}
```

```
9
    }
10
}
```

1.4相关注解【应用】

- 注解说明

注解	含义
@Test	表示测试该方法
@Before	在测试的方法前运行
@After	在测试的方法后运行

- 代码示例

```
XXXXXXXXXX
17
1
public class JunitDemo2 {
2
    @Before
3
    public void before() {
4
        // 在执行测试代码之前执行，一般用于初始化操作
5
        System.out.println("before");
6
    }
7
    @Test
8
```

```
public void test() {  
9  
    // 要执行的测试代码  
10  
    System.out.println("test");  
11  
}  
12  
@After  
13  
public void after() {  
14  
    // 在执行测试代码之后执行，一般用于释放资源  
15  
    System.out.println("after");  
16  
}  
17  
}
```

2.日志

2.1概述【理解】

- 概述

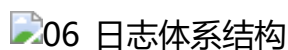
程序中的日志可以用来记录程序在运行的时候点点滴滴。并可以进行永久存储。

- 日志与输出语句的区别

	输出语句	日志技术
取消日志	需要修改代码，灵活性比较差	不需要修改代码，灵活性比较好
输出位置	只能是控制台	可以将日志信息写入到文件或者数据库中
多线程	和业务代码处于一个线程中	多线程方式记录日志，不影响业务代码的性能

2.2 日志体系结构和Log4J【理解】

- 体系结构



- Log4J

Log4j是Apache的一个开源项目。

通过使用Log4j，我们可以控制日志信息输送的目的地是控制台、文件等位置。

我们也可以控制每一条日志的输出格式。

通过定义每一条日志信息的级别，我们能够更加细致地控制日志的生成过程。

最令人感兴趣的就是，这些可以通过一个配置文件来灵活地进行配置，而不需要修改应用的代码。

- Apache基金会

Apache软件基金会（也就是Apache Software Foundation，简称为ASF），为支持开源软件项目而办的一个非盈利性组织。

2.3 入门案例【应用】

- 使用步骤

1. 导入log4j的相关jar包
2. 编写log4j配置文件
3. 在代码中获取日志的对象
4. 按照级别设置记录日志信息

- 代码示例

```

XXXXXXXXXX

42
1

// log4j的配置文件, 名字为log4j.properties, 放在src根目录下

2

```

```
log4j.rootLogger=debug,my,fileAppender
3

4
### direct log messages to my ###
5
log4j.appender.my=org.apache.log4j.ConsoleAppender
6
log4j.appender.my.ImmediateFlush = true
7
log4j.appender.my.Target=System.out
8
log4j.appender.my.layout=org.apache.log4j.PatternLayout
9
log4j.appender.my.layout.ConversionPattern=%d %t %5p %c{1}:%L - %m%n
10

11
# fileAppender'
12
log4j.appender.fileAppender=org.apache.log4j.FileAppender
13
log4j.appender.fileAppender.ImmediateFlush = true
14
log4j.appender.fileAppender.Append=true
15
log4j.appender.fileAppender.File=D:/log4j-log.log
16
log4j.appender.fileAppender.layout=org.apache.log4j.PatternLayout
17
log4j.appender.fileAppender.layout.ConversionPattern=%d %5p %c{1}:%L - %m%n
18

19
// 测试类
20
public class Log4JTest01 {
```

```
21
22
23    //使用log4j的api来获取日志的对象
24
25    //弊端：如果以后我们更换日志的实现类，那么下面的代码就需要跟着改
26
27    //不推荐使用
28
29    //private static final Logger LOGGER = Logger.getLogger(Log4JTest01.class);
30
31
32    //使用slf4j里面的api来获取日志的对象
33
34    //好处：如果以后我们更换日志的实现类，那么下面的代码不需要跟着修改
35
36    //推荐使用
37
38    private static    final Logger LOGGER = LoggerFactory.getLogger(Log4JTest01.class);
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
        LOGGER.warn("warn级别的日志");
40
        LOGGER.error("error级别的日志");
41
    }
42
}
```

2.4配置文件详解【理解】

- 三个核心

- Loggers(记录器) 日志的级别

Loggers组件在此系统中常见的五个级别：DEBUG、INFO、WARN、ERROR 和 FATAL。

DEBUG < INFO < WARN < ERROR < FATAL。

Log4j有一个规则：只输出级别不低于设定级别的日志信息。

- Appenders(输出源) 日志要输出的地方

把日志输出到不同的地方，如控制台（Console）、文件（Files）等。

- org.apache.log4j.ConsoleAppender（控制台）
 - org.apache.log4j.FileAppender（文件）

- Layouts(布局) 日志输出的格式

可以根据自己的喜好规定日志输出的格式

常用的布局管理器：

org.apache.log4j.PatternLayout（可以灵活地指定布局模式）

org.apache.log4j.SimpleLayout（包含日志信息的级别和信息字符串）

org.apache.log4j.TTCCLayout（包含日志产生的时间、线程、类别等信息）

- 配置根Logger

- 格式

log4j.rootLogger=日志级别, appenderName1, appenderName2, ...

- 日志级别

OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL或者自定义的级别。

- appenderName1

就是指定日志信息要输出到哪里。可以同时指定多个输出目的地，用逗号隔开。

例如：log4j.rootLogger = INFO, ca, fa

- ConsoleAppender常用的选项

- ImmediateFlush=true

表示所有消息都会被立即输出，设为false则不输出，默认值是true。

- Target=System.err

默认值是System.out。

- FileAppender常用的选项

- ImmediateFlush=true

表示所有消息都会被立即输出。设为false则不输出，默认值是true

- Append=false

true表示将消息添加到指定文件中，原来的消息不覆盖。

false则将消息覆盖指定的文件内容，默认值是true。

- File=D:/logs/logging.log4j

指定消息输出到logging.log4j文件中

- PatternLayout常用的选项

- ConversionPattern=%m%n

设定以怎样的格式显示消息

07_PatternLayout常用的选项

2.5在项目中的应用【应用】

- 步骤

1. 导入相关的依赖
2. 将资料中的properties配置文件复制到src目录下
3. 在代码中获取日志的对象
4. 按照级别设置记录日志信息

- 代码实现

```
xxxxxxxxx

18
1
@WebServlet(urlPatterns = "/servlet/loginservlet")

2

public class LoginServlet implements HttpServlet{

3

4

    //获取日志的对象

5

    private static final Logger LOGGER = LoggerFactory.getLogger(LoginServlet.class);
```

6

7

```
@Override
```

8

```
public void service(HttpServletRequest httpRequest, HttpServletResponse httpResponse) {
```

9

```
    //处理
```

10

```
        System.out.println("LoginServlet处理了登录请求");
```

11

12

```
        LOGGER.info("现在已经处理了登录请求，准备给浏览器响应");
```

13

14

```
    //响应
```

15

```
        httpResponse.setContentType("text/html;charset=UTF-8");
```

16

```
        httpResponse.write("登录成功");
```

17

```
    }
```

18

```
}
```

