《Java性能优化与面试21讲》

李国

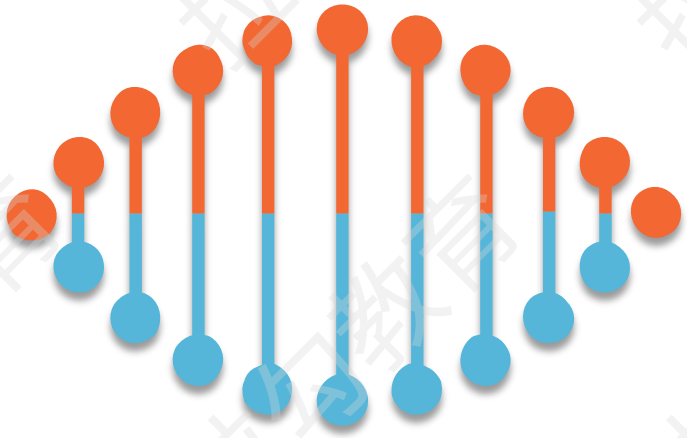
— 拉勾教育出品 —

16 | 案例分析：常见 Java 代码优化法则

语言本身对性能是有影响的

Java 有一套优化法则，这些细微的性能差异

经过多次调用和迭代，会产生越来越大的影响



代码优化法则

使用局部变量可避免在堆上分配

堆资源是多线程共享的

是垃圾回收器工作的主要区域，过多的对象会造成 GC 压力

可以通过局部变量的方式，将变量在栈上分配

代码优化法则

减少变量的作用范围

```
public void test1(String str) {
    final int a = 100;
    if (!StringUtils.isEmpty(str)) {
        int b = a * a;
    }
}
```

代码优化法则

访问静态变量直接使用类名

```
public class StaticCall{
  public static final int A = 1;

  void test(){
    System.out.println(this.A);
    System.out.println(StaticCall.A);
  }
}
```

代码优化法则

访问静态变量直接使用类名

```
void test();
  descriptor: ()V
  flags:
  Code:
    stack=2, locals=1, args_size=1
       0: getstatic     #2          // Field java/lang/System.out:Ljava/io/PrintStream;
       3: aload_0
       4: pop
       5: iconst_1
       6: invokevirtual #3          // Method java/io/PrintStream.println:(I)V
       9: getstatic     #2          // Field java/lang/System.out:Ljava/io/PrintStream;
      12: iconst_1
      13: invokevirtual #3          // Method java/io/PrintStream.println:(I)V
      16: return
    LineNumberTable:
      line 5: 0
      line 6: 9
      line 7: 16
```

代码优化法则

字符串拼接使用 StringBuilder

```
public String test() {
    String str = "-1";
    for (int i = 0; i< 10; i++) {
        str += i;
    }
    return str;
}
```

代码优化法则

拉勾教育
互联网人实战大学

字符串拼接使用 StringBuilder

```
 5: iload_2
 6: bipush        10
 8: if_icmpge     36
11: new           #3                  // class java/lang/StringBuilder
14: dup
15: invokespecial #4                  // Method java/lang/StringBuilder."<init>":()V
18: aload_1
19: invokevirtual #5                  // Method
java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
22: iload_2
23: invokevirtual #6                  // Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
26: invokevirtual #7                  // Method java/lang/StringBuilder.toString:()Ljava/lang/String;
29: astore_1
30: iinc          2, 1
33: goto          5
```

代码优化法则

拉勾教育
互联网人实战大学

重写对象的 HashCode，不要简单地返回固定值

代码 review 时，发现有开发重写 HashCode 和 Equals 方法时

会把 HashCode 的值返回固定的 0，这样做是不恰当的

固定地返回 0，相当于把 Hash 寻址功能废除

代码优化法则

HashMap 等集合初始化的时候，指定初始值大小

参见“10 | 案例分析：大对象复用的目标和注意点”

代码优化法则

遍历 Map 的时候，使用 EntrySet 方法

EntrySet 方法

可以直接返回 set 对象，直接拿来用即可

KeySet 方法

获得的是key 的集合

需要再进行一次 get 操作，多了一个操作步骤

代码优化法则

遍历 Map 的时候，使用 EntrySet 方法

EntrySet 方法

可以直接返回 set 对象，直接拿来用即可

KeySet 方法

获得的是key 的集合

需要再进行一次 get 操作，多了一个操作步骤

代码优化法则

不要在多线程下使用同一个 Random

Random 类的 seed 会在并发访问的情况下发生竞争，造成性能降低

建议在多线程环境下使用 ThreadLocalRandom 类

Linux 通过加入 JVM 配置 -Djava.security.egd=file:/dev/./urandom

使用 urandom 随机生成器，在进行随机数获取时，速度会更快

代码优化法则

自增推荐使用 LongAddr

自增运算可以通过 synchronized 和 volatile 的组合

也可以使用原子类（比如 AtomicLong）

代码优化法则

不要使用异常控制程序流程

```
Exception table:
   from    to  target type
      7    17    20   any
     20    23    20   any
```

代码优化法则

不要在循环中使用 try catch

不要把异常处理放在循环里，而应该把它放在最外层

但实际测试情况表明这两种方式性能相差并不大

推荐根据业务的需求进行编码

代码优化法则

不要捕捉 RuntimeException

代码优化法则

不要捕捉 RuntimeException

```
//BAD
public String test1(List<String> list, int index) {
    try {
        return list.get(index);
    } catch (IndexOutOfBoundsException ex) {
        return null;
    }
}
?
//GOOD
public String test2(List<String> list, int index) {
    if (index >= list.size() || index < 0) {
        return null;
    }
    return list.get(index);
}
```

代码优化法则

合理使用 PreparedStatement

PreparedStatement 使用预编译对 SQL 的执行进行提速

还能提高程序的安全性，能够有效防止 SQL 注入

代码优化法则

日志打印的注意事项

```
logger.debug("xjjdog:"+ topic + " is awesome" );
```

代码优化法则

拉勾教育
互联网人实战大学

日志打印的注意事项

```
if(logger.isDebugEnabled()){
    logger.debug("xjjdog:"+topic + " is awesome" );
}
```

代码优化法则

拉勾教育
互联网人实战大学

日志打印的注意事项

```
logger.debug("xjjdog:{} is awesome" ,topic);
```

代码优化法则

减少事务的作用范围

```
@Transactional
public void test(String id){
  String value = rpc.getValue(id); //高耗时
  testDao.update(sql,value);
}
```

代码优化法则

拉勾教育
互联网人实战大学

减少事务的作用范围

```
public void test(String id){
  String value = rpc.getValue(id); //高耗时
  testDao(value);
}

@Transactional
public void testDao(String value){
  testDao.update(value);
}
```

代码优化法则

使用位移操作替代乘除法

```
int a = 2;
int b = (a++) << (++a) + (++a);
System.out.println(b);
```

- << 左移相当于乘以 2
- >> 右移相当于除以 2
- >>> 无符号右移相当于除以 2，但它会忽略符号位，空位都以 0 补齐

代码优化法则

拉勾教育
互联网人实战大学

不要打印大集合或者使用大集合的 toString 方法

将集合作为字符串输出到日志文件中，是**非常不好的习惯**

代码优化法则

不要打印大集合或者使用大集合的 toString 方法

```
public String toString() {
    Iterator<E> it = iterator();
    if (! it.hasNext())
        return "[]";

    StringBuilder sb = new StringBuilder();
    sb.append('[');
    for (;;) {
        E e = it.next();
        sb.append(e == this ? "(this Collection)" : e);
        if (! it.hasNext())
            return sb.append(']').toString();
        sb.append(',').append(' ');
    }
}
```

程序中少用反射

反射的功能很强大

但它是通过解析字节码实现的，性能就不是很理想

代码优化法则

程序中少用反射

```
import java.lang.invoke.MethodHandle;
import java.lang.invoke.MethodHandles;
import java.lang.invoke.MethodType;

public class MethodHandleDemo {
  static class Bike {
    String sound() {
      return "ding ding";
    }
  }

  static class Animal {
    String sound() {
      return "wow wow";
    }
  }
```

代码优化法则

程序中少用反射

```
static class Man extends Animal {
    @Override
    String sound() {
        return "hou hou";
    }
}


String sound(Object o) throws Throwable {
    MethodHandles.Lookup lookup = MethodHandles.lookup();
    MethodType methodType = MethodType.methodType(String.class);
    MethodHandle methodHandle = lookup.findVirtual(o.getClass(), "sound",
methodType);

    String obj = (String) methodHandle.invoke(o);
    return obj;
}
```

代码优化法则

程序中少用反射

```
        MethodType methodType = MethodType.methodType(String.class);
        MethodHandle methodHandle = lookup.findVirtual(o.getClass(), "sound",
methodType);

        String obj = (String) methodHandle.invoke(o);
        return obj;
    }

    public static void main(String[] args) throws Throwable {
        String str = new MethodHandleDemo().sound(new Bike());
        System.out.println(str);
        str = new MethodHandleDemo().sound(new Animal());
        System.out.println(str);
        str = new MethodHandleDemo().sound(new Man());
        System.out.println(str);
    }
}
```

代码优化法则

正则表达式可以预先编译，加快速度

```
Pattern pattern = Pattern.compile({pattern});
Matcher pattern = pattern.matcher({content});
```

案例分析

案例 1：正则表达式和状态机

```
select * from USERS
where id>:smallId
##{
 and FIRST_NAME like concat('%',:firstName,'%') }
```

案例分析

案例 1：正则表达式和状态机

```
select * from USERS
where id>:smallId
##{
 and FIRST_NAME like concat('%',:firstName,'%') }
```

```
#\{(.*?:([a-zA-Z0-9_]+).*?)\}
```

案例分析

案例 1：正则表达式和状态机

```
pairStart = '#{';
pairEnd = '}';
namedQueryStringFull = ( ':'alnum+)
    ->buffer
      %namedQueryStringFull
      ;
pairBlock =
    (pairStart
      any*
      namedQueryStringFull
      any*
      pairEnd)
    ->pairBlockBegin %pairBlockEnd
    ;
main := any* pairBlock any*;
```

案例分析

案例 1：正则表达式和状态机

```
ragel -G2 -J -o P.java P.rl
```

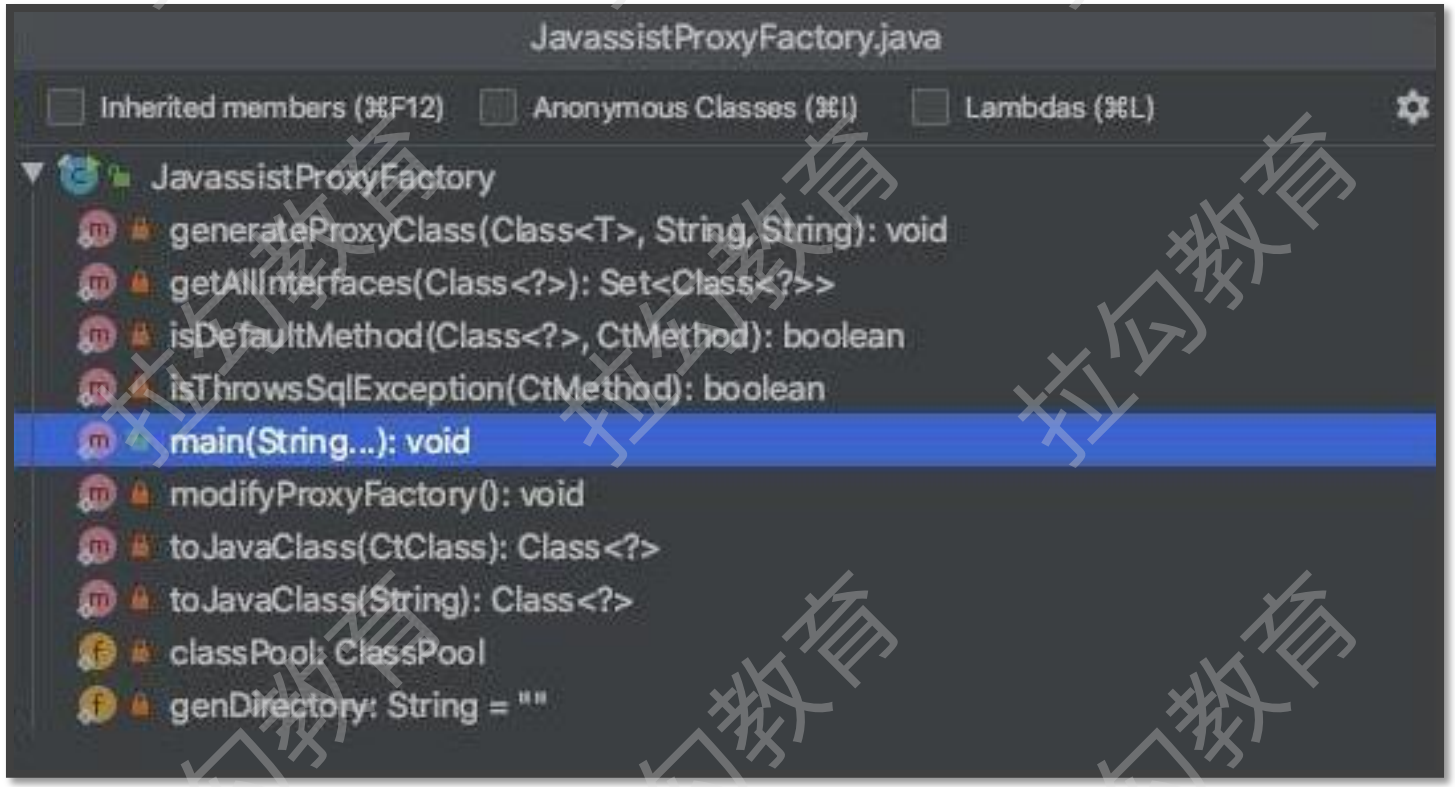
案例分析

案例 1：正则表达式和状态机

```
Benchmark                          Mode  Cnt    Score    Error  Units
RegexVsRagelBenchmark.ragel       thrpt   10  691.224 ± 446.217  ops/ms
RegexVsRagelBenchmark.regex       thrpt   10  201.322 ±  47.056  ops/ms
```

案例分析

案例 2：HikariCP 的字节码修改



案例分析

案例 2：HikariCP 的字节码修改

```
Generating com.zaxxer.hikari.pool.HikariProxyConnection
Generating com.zaxxer.hikari.pool.HikariProxyStatement
Generating com.zaxxer.hikari.pool.HikariProxyResultSet
Generating com.zaxxer.hikari.pool.HikariProxyDatabaseMetaData
Generating com.zaxxer.hikari.pool.HikariProxyPreparedStatement
Generating com.zaxxer.hikari.pool.HikariProxyCallableStatement
Generating method bodies for com.zaxxer.hikari.proxy.ProxyFactory
```

案例分析

案例 2：HikariCP 的字节码修改

```
public class HikariProxyResultSet extends ProxyResultSet implements Wrapp
    public boolean isWrapperFor(Class var1) throws SQLException {
        try {
            return super.delegate.isWrapperFor(var1);
        } catch (SQLException var3) {
            throw this.checkException(var3);
        }
    }

    public void close() throws Exception {
        ((ResultSet)super.delegate).close();
    }
```

案例分析

案例 2：HikariCP 的字节码修改

```
public class HikariProxyResultSet extends ProxyResultSet implements Wrapp
    public boolean isWrapperFor(Class var1) throws SQLException {
        try {
            return super.delegate.isWrapperFor(var1);
        } catch (SQLException var3) {
            throw this.checkException(var3);
        }
    }

    public void close() throws Exception {
        ((ResultSet)super.delegate).close();
    }
```

1. 在代码中只需要实现需要修改的 JDBC 接口方法

   其他的交给代理类自动生成的代码，极大地减少了编码数量

2. 出现问题时，可以通过 checkException 函数对错误进行统一处理

案例分析

invokevirtual

```
public final java.sql.PreparedStatement prepareStatement(java.lang.String, java.lang.String[]) throws
java.sql.SQLException;
  flags: ACC_PRIVATE, ACC_FINAL
  Code:
   stack=5, locals=3, args_size=3
      0: getstatic     #59          // Field PROXY_FACTORY:Lcom/zaxxer/hikari/proxy/ProxyFactory;
      3: aload_0
      4: aload_0
      5: getfield      #3           // Field delegate:Ljava/sql/Connection;
      8: aload_1
      9: aload_2
     10: invokeinterface #74,  3          // InterfaceMethod
java/sql/Connection.prepareStatement:(Ljava/lang/String;[Ljava/lang/String;)Ljava/sql/PreparedState
ment;
     15: invokevirtual #69          // Method
com/zaxxer/hikari/proxy/ProxyFactory.getProxyPreparedStatement:(Lcom/zaxxer/hikari/proxy/Connect
ionProxy;Ljava/sql/PreparedStatement;)Ljava/sql/PreparedStatement;
     18: return
```

案例分析

invokestatic

```
private final java.sql.PreparedStatement prepareStatement(java.lang.String, java.lang.String[]) throws
java.sql.SQLException;
    flags: ACC_PRIVATE, ACC_FINAL
    Code:
      stack=4, locals=3, args_size=3
         0: aload_0
         1: aload_0
         2: getfield      #3                  // Field delegate:Ljava/sql/Connection;
         5: aload_1
         6: aload_2
         7: invokeinterface #72,  3           // InterfaceMethod
java/sql/Connection.prepareStatement:(Ljava/lang/String;[Ljava/lang/String;)Ljava/sql/PreparedStatem
ent;
        12: invokestatic  #67                 // Method
com/zaxxer/hikari/proxy/ProxyFactory.getProxyPreparedStatement:(Lcom/zaxxer/hikari/proxy/Connecti
onProxy;Ljava/sql/PreparedStatement;)Ljava/sql/PreparedStatement;
        15: areturn
```

案例分析

拉勾教育
互联网人实战大学

Invokevirtual 属于需方法调用

```
public final java.sql.PreparedStatement prepareStatement(java.lang.String, java.lang.String[]) throws
java.sql.SQLException;
  flags: ACC_PRIVATE, ACC_FINAL
  Code:
   stack=5, locals=3, args_size=3
      0: getstatic     #59          // Field PROXY_FACTORY:Lcom/zaxxer/hikari/proxy/ProxyFactory;
      3: aload_0
      4: aload_0
      5: getfield      #3           // Field delegate:Ljava/sql/Connection;
      8: aload_1
      9: aload_2
     10: invokeinterface #74,  3          // InterfaceMethod
java/sql/Connection.prepareStatement:(Ljava/lang/String;[Ljava/lang/String;)Ljava/sql/PreparedState
ment;
     15: invokevirtual #69          // Method
com/zaxxer/hikari/proxy/ProxyFactory.getProxyPreparedStatement:(Lcom/zaxxer/hikari/proxy/Connect
ionProxy;Ljava/sql/PreparedStatement;)Ljava/sql/PreparedStatement;
     18: return
```

小结

学习 Java 规范，你还可以细读**《阿里巴巴 Java 开发规范》**

语言层面的性能优化，都是在各个资源之间的权衡（比如开发时间、代码复杂度、扩展性等）

在编码中选择合适的工具，根据实际的工作场景进行灵活变动

Next：第17讲《案例分析：从 BIO 到 NIO，再到 AIO》

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容

L / A / G / O / U