разработка производительного пользовательского DSL

для анализа временных рядов

алексей семин

октябрь 2017

DEVEXPERTS

предметно-ориентированные языки

- HTML
- SQL
- shell-scripts
- ...

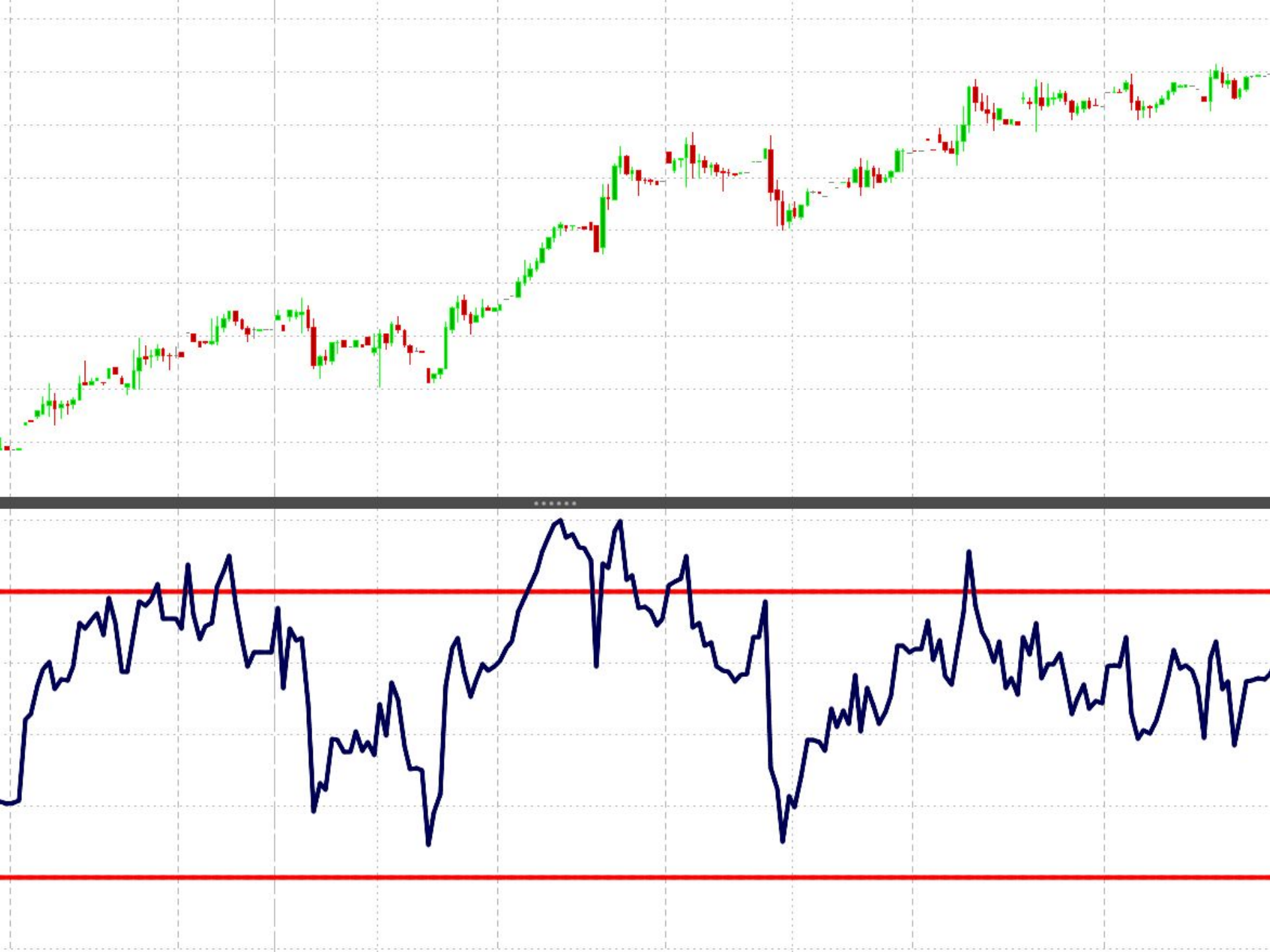
```
<h1>Hello!</h1>
```

```
SELECT year FROM books
```

```
ls -l | grep "foo"
```

причины использования DSL подхода

- громоздкий API или UI
- сериализация
- использование конечными пользователями



критерии для языка dxScript

- простота
- выразительность
- безопасность
- производительность

простота

- декларативный синтаксис
- декларативное задание синтаксиса
  - формальная грамматика
  - парсер-генератор ANTLR

dxScript

```
def fib = fib[1] + fib[2] default 1
```

выразительность

- значения параметров по умолчанию
- многозначность результатов
- вывод типов

dxScript

```
in x = price
in n = 10
out simple = sma(x, n)
out exp = ema(x, n)
```

foreign function interface (FFI)

```java
public static double sma(
    @In(value="x", size="n") RealSeries x,
    @PureIn("n") double n
) {
    double v = 0;
    for (int i = 0; i < n; i++) {
        v += x.getReal(i);
    }
    return v / n;
}
```

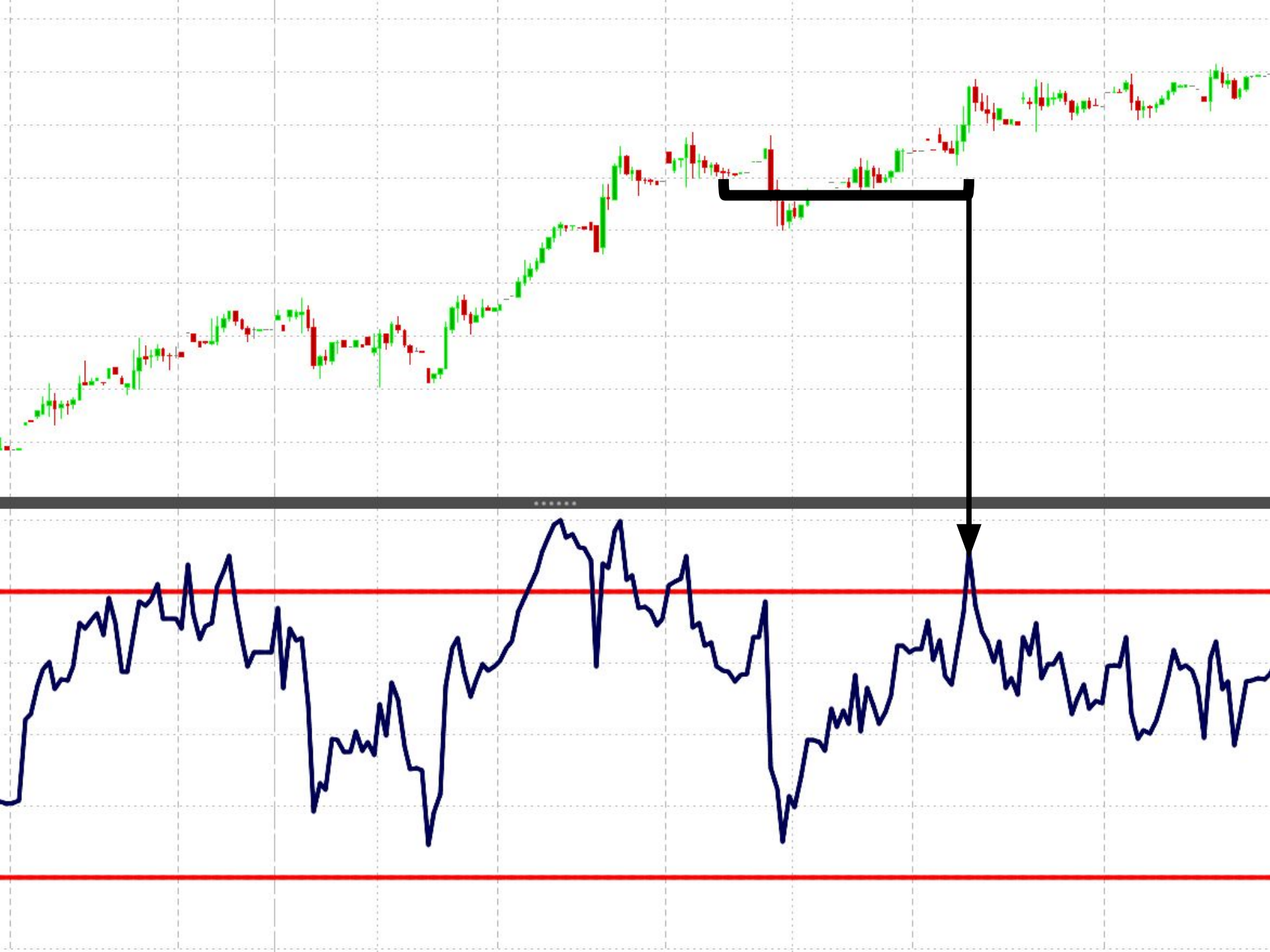
Java

безопасность

- отсутствие Тьюринг-полноты
  - запрещена рекурсия
  - отсутствие явных циклов
- конечная память
  - конечный размер окна данных

dxScript

```
in x = price
in n = 1
out change = x - x[n]
```



производительность

- компиляция в JVM байткод с помощью ASM
- снижение нагрузки на garbage collector (GC)
- оптимизации кода
  - constant folding
  - dead code elimination
  - common subexpression elimination

Java

```java
class World {
    public World() {
        super();
        return;
    }
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        return;
    }
}
```

Kotlin

```kotlin
asmClassWriter("com.hello.World") {
    asmConstructor {
        asmCtorInvoke()
        asmReturn()
    }

    asmStaticMethod("main", "args".ofType<Array<String>>()) {
        asmInvoke(
          receiver = asmStaticGetter<System, PrintStream>("out"),
          func = overloaded<PrintStream.(String) → Unit>(PrintStream::println),
          args = asmPusher("Hello, World!")
        )
        asmReturn()
    }
}
```

Java

ASM

```
ClassWriter cw = new ClassWriter(0);
MethodVisitor mv;
cw.visit(52, ACC_SUPER, "com/hello/World", null, "java/lang/Object", null);
{
    mv = cw.visitMethod(ACC_PUBLIC, "<init>", "()V", null, null);
    mv.visitCode();
    Label l0 = new Label();
    mv.visitLabel(l0);
    mv.visitLineNumber(16, l0);
    mv.visitVarInsn(ALOAD, 0);
    mv.visitMethodInsn(INVOKESPECIAL, "java/lang/Object", "<init>", "()V", false);
    Label l1 = new Label();
    mv.visitLabel(l1);
    mv.visitLineNumber(17, l1);
    mv.visitInsn(RETURN);
    Label l2 = new Label();
    mv.visitLabel(l2);
    mv.visitLocalVariable("this", "Lcom/hello/World;", null, l0, l2, 0);
    mv.visitMaxs(1, 1);
    mv.visitEnd();
}
{
    mv = cw.visitMethod(ACC_PUBLIC + ACC_STATIC, "main", "([Ljava/lang/String;)V", null, null);
    mv.visitCode();
    Label l0 = new Label();
    mv.visitLabel(l0);
    mv.visitLineNumber(20, l0);
    mv.visitFieldInsn(GETSTATIC, "java/lang/System", "out", "Ljava/io/PrintStream;");
    mv.visitLdcInsn("Hello, World!");
    mv.visitMethodInsn(INVOKEVIRTUAL, "java/io/PrintStream", "println", "(Ljava/lang/String;)V", false);
    Label l1 = new Label();
    mv.visitLabel(l1);
    mv.visitLineNumber(21, l1);
    mv.visitInsn(RETURN);
    Label l2 = new Label();
    mv.visitLabel(l2);
    mv.visitLocalVariable("args", "[Ljava/lang/String;", null, l0, l2, 0);
    mv.visitMaxs(2, 1);
    mv.visitEnd();
}
```

~~критерии~~ достоинства языка dxScript

- простота
- выразительность
- безопасность
- производительность

алексей семин @ DEVEXPERTS
alllexsm@gmail.com asemin@devexperts.com

октябрь 2017