链滴

距离全栈 你只差一个 kotlinx

作者：crick77

原文链接：https://ld246.com/article/1600502495289

来源网站：链滴

示例项目地址: https://github.com/wangyuheng/kotlin-dsl-html

全栈不能保证一定能够解决复杂的问题, 但却能帮你打开解决复杂问题的大门.

近些年，前端技术变得愈发复杂。这一趋势除了导致全球变暖，也让全栈开发的难度越来越大。

但是，阻碍一个后端开发去写页面的根本原因到底是什么呢?

我认为是开发环境的搭建，如果环境变量准备好，可以用自己平时使用的IDE直接写代码，刷新就能效果，这事貌似可行。

那么如何解决这个问题呢?

我觉得终极解决方案是用一门编程语言同时写前后端代码。

Kotlin配合DSL就可以。

DSL

DSL(Domain Specific Language)，领域特定语言。专注于一个方面而特殊设计的语言。

比如 SQL是数据库领域的DSL。

特点

1. 只描述和解决特定领域

优势

1. 语义更明确，直观

2. 可以屏蔽数据结构和技术细节，由领域专家编写

缺点

1. 额外的理解、学习成本
2. 抽象设计难度高，需要平衡表现力和实现成本。

比如描述2天前的时间，你可以定义为

2 days ago

也可以是

new Date().before(2, DAY)

这又引出了DSL的2种分类

- 内部(Internal)DSL，借助宿主语言(如:Scala、Kotlin)实现。和提取函数方法不同，提供了一套更接自然语言的语法表现形式
- 外部(External)DSL，语言无关，需要自定义语法并实现解析器。比如 XMl、YAML

kotlin DSL

Kotlin 借助 Lambda + Extensions扩展 来实现内部DSL

```
fun main(args: Array<String>) {
    expression {
        source = "a"
        target = "b"
        operator = Operator.ADD
        onBefore { println("before $source") }
    }
}

enum class Operator {
    ADD,
    SUBTRACT,
    MULTIPLY,
    DIVIDE
}

class Expression {
    var source: String? = null
    var target: String? = null
    var operator: Operator? = null

    internal var before: () -> Unit = { }

    fun onBefore(onBefore: () -> Unit) {
        before = onBefore
    }

    fun execute(): String {
```

```
        this.before()
        val result = "$source $operator $target"
        println(result)
        return result
    }
}

fun expression(init: Expression.() -> Unit) {
    val wrap = Expression()
    wrap.init()

    wrap.execute()
}
```

expression函数的入参是一个 Expression.() -> Unit类型的 lambda。未简化的代码为

```
val lambdaExpression:Expression.() -> Unit = {
    source = "a"
    target = "b"
    operator = Operator.ADD
    onBefore {
        println("before $source")
    }
}
expression(lambdaExpression)
```

简化后借助 Lambda argument should be moved out of parentheses， 变为

```
expression {
    source = "a"
    target = "b"
    operator = Operator.ADD
    onBefore {
        println("before $source")
    }
}
```

是不是有DSL内味了

kotlinx

原文链接：距离全栈 你只差一个 kotlinx

kotlinx.html 是一个通过DSL构造HTML的kotlin扩展库。

本质上是通过kotlin定义好了一套htmlTag类。

DSL定义

```
html {
    head {
        script {
            src = "https://code.jquery.com/jquery-3.5.1.slim.min.js"
        }
    }
    body {
        div {
            +"Hello $name"
        }
    }
}
```

生成的HTML

```
<html>
  <head>
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  </head>
  <body>
    <div>Hello DSL</div>
  </body>
</html>
```

在此基础上扩展并封装UI组件，达到简化开发成本的目的。比如

- 封装bootstrap V4的header资源引用

```
fun HEAD.b4() {
    link {
        href = "https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
        rel = "stylesheet"
    }
    script {
        src = "https://code.jquery.com/jquery-3.5.1.slim.min.js"
    }
    script {
        src = "https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"
    }
    script {
        src = "https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"
    }
}
```

- 封装基于bootstrap4的dropdown

```
fun DIV.b4dropdown(btn: String, block: DIV.() -> Unit) {
    div("dropdown") {
        a("#") {
```

```
            classes = setOf("btn", "btn-secondary", "dropdown-toggle")
            role = "button"
            attributes["aria-expanded"] = "false"
            attributes["data-toggle"] = "dropdown"
            +btn
        }
        div("dropdown-menu") {
            attributes["aria-labelledby"] = "dropdownMenuLink"
            block()
        }
    }
}

fun DIV.b4dropdownItem(content: String, href: String? = "#", target: String? = null) {
    a(href, target, "dropdown-item") { +content }
}
```

● 封装基于bootstrap4的table

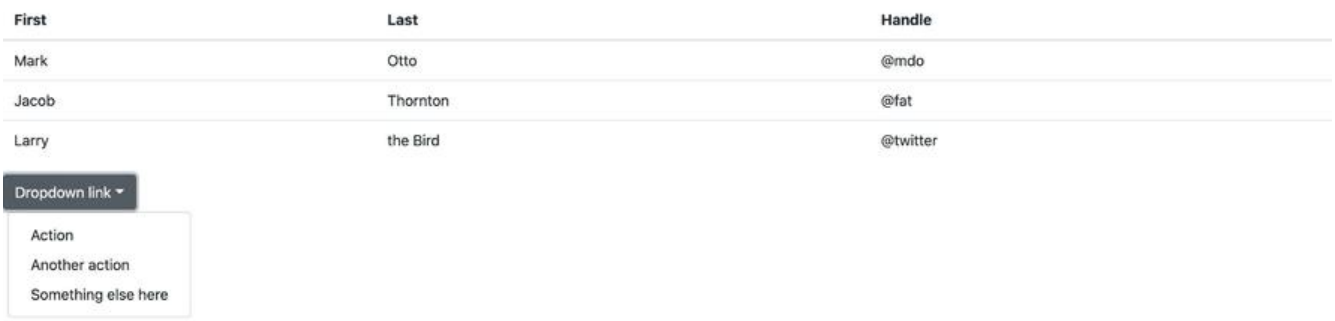
```
fun HtmlBlockTag.b4table(headers: List<String>, rows: List<List<() -> Unit>>, showIndex: Boo
ean = false) {
    table("table") {
        thead {
            tr {
                if (showIndex) th(ThScope.col) { +"#" }
                headers.forEach { th(ThScope.col) { +it } }
            }
        }
        tbody {
            for ((index, row) in rows.withIndex()) {
                tr {
                    if (showIndex) td { +"${index + 1}" }
                    row.forEach { td { it() } }
                }
            }
        }
    }
}
```

则View层代码为

```
val dropdownList = arrayListOf("Action", "Another action", "Something else here")
val tableHeaders = arrayListOf("First", "Last", "Handle")
val tableRows = arrayListOf(
        arrayListOf("Mark", "Otto", "@mdo"),
        arrayListOf("Jacob", "Thornton", "@fat"),
        arrayListOf("Larry", "the Bird", "@twitter")
)
createHTML()
        .html {
            head {
                b4()
            }
            body {
```

```
            b4table(tableHeaders, tableRowsUnit, false)
            div {
                b4dropdown("Dropdown link") {
                    dropdownList.forEach { b4dropdownItem(it) }
                }
            }
        }
    }
```

浏览器效果



通常table的最后一列为action列，所以table rows类型为 List<List<() -> Unit>>，可以传入html素

```
createHTML()
    .html {
        val tableRowsUnit = tableRows.map { r ->
            r.map {
                { +it }
            }
        }
        head {
            b4()
        }
        body {
            b4table(tableHeaders, tableRowsUnit, false)
            div {
                b4dropdown("Dropdown link") {
                    list.forEach { b4dropdownItem(it) }
                }
            }
            div {
                b4table(
                        tableHeaders.toList().plus("action"),
                        tableRowsUnit.map { r ->
                            r.plus {
                                ul {
                                    lia("#") { +"Edit" }
                                    lia("#") { +"Delete" }
                                }
                            }
                        },
```

```
                true
            )
        }
    }
}
```

其他

Javalin

示例项目使用了 Javalin

Javalin is more of a library than a framework. Some key points:

- You don't need to extend anything
- There are no @Annotations
- There is no reflection
- There is no other magic; just code.

想必你也听得出是在影射哪个框架

结论

1. kotlinx.html可以作为jsp、FreeMarker这类模板方案的替代，优势是无需在两种语法之间切换。
2. 如果没想清楚如何解决问题，那DSL不是一个好的选择