trimer小知识(2) 配置文件中的一些yaml语法

boboIqiqi edited this page on 7 May 2016 · 4 revisions

yaml小知识

Yaml简介

首先，yaml的全称 Yaml Ain't Markup Language。即Yaml不是标记语言,这是针对XML而言的，即Yaml中不需要使用标签。 注: Ain't是am not的缩写,后来又变成isn't等否定的缩写

YAML is a human friendly data serialization standard for all programming languages.

官方给的定义是，Yaml是一种可读的数据序列化标准。

简单的讲，就是yaml就是一种描述各种结构化数据的纯文本语言。与它同样地位的，可以相互类比的是xml和json。 关于这三者的比较，可以看这里 序列化格式：XML、JSON、YAML

数据表示方法

Yaml是如何描述数据的结构呢？主要有以下三条:

- Structure通过空格来展示,即通过缩进来表示包含关系，也可以通过[]和{}后面有介绍
- Sequence里的项用"-"来代表
- Map里的键值对用":"分隔.

```
qwerty:
  ascii_mode: 0
  author: "osfans <waxaca@163.com>"
  height: 55
  keys: [{click: q, long_click: "!", swipe_up: 1}, {click: w, long_click: "@", swipe_up: 2}, {click: e, long_click: "#
```

上面的文件定义了一个名叫qwerty的映射，这个映射包含几个个字段:

- ascii_mode
- author
- height
- keys： 这是一个复合类型，其中包含了多个按键，每个按键，又包含多个属性值

每个字段的取值，在冒号后面指定。 其中keys是一个复合字段，它本身又是一个列表，它包含多个键值。另外，每个键值，又是一个字典(又叫映射)，包含多个属性值对。

列表和字典的区别在于，列表是有序的，而字典是无序的。上面的keys要表示键盘每个键的排列，所以必须是有序的排列。 而每个key的属性值，只是标明各个属性，并不需要有序。

上面为了简洁，上面的keys以单行模式写出，如果展开会是

```
qwerty:
  keys:
    - {click: q, long_click: "!", swipe_up: 1}
    - {click: w, long_click: "@", swipe_up: 2}
...
```

进一步展开会变成:

```
qwerty:
  keys:
    -
      click: q
      long_click: "!"
      swipe_up: 1
    -
```

▸ Pages 12

Trimer小知识

- Rime前端汇总
- Rime经典资料汇总
- Yaml文件开头注释是什么意思?
- 配置文件中的一些yaml语法

配置同文输入法

- 简易索引
- 前端配置 (trime.yaml詳解)

案例参考

- 五笔双键配置案例详解(待完善)
  - (一) 准备篇
  - (二) 添加一个输入方案
  - (三) 用模糊音实现双键转换
  - (四) 实现手机上的双键键盘

Clone this wiki locally

https://github.com/osfans/trime

Clone in Desktop

```
  click: w
  long_click: "@"
  swipe_up: 2
```

从上面，我们可以看出列表和映射的两种表示方法：

列表的单行与展开模式：

```
#单行模式:
name: [value1, value2]
#展开模式:
name:
  - value1
  - value2
```

映射的单行与展开模式：

```
#单行模式:
name: {p1: n1, p2: n2}
#展开模式:
name:
  p1: n1
  p2: n2
```

数据引用方式

打patch时，需要先取得准备修改的属性，然后才能对其进行修改。

参见晓群老师，给我的示例，

```
 "preset_keyboards/qwerty/keys/@31": {label: "英", click: Keyboard_default}
```

总结如下：

列表数据的引用:

通过@下标，引用 。

映射数据的引用:

"/"一层一层的引用

关于@的更多语法，见 定制指南

```
patch:
  "一級設定項/二級設定項/三級設定項": 新的設定值
  "另一個設定項": 新的設定值
  "再一個設定項": 新的設定值
  "含列表的設定項/@0": 列表第一個元素新的設定值
  "含列表的設定項/@last": 列表最後一個元素新的設定值
  "含列表的設定項/@before 0": 在列表第一個元素之前插入新的設定值（不建議在補靶中使用）
  "含列表的設定項/@after last": 在列表最後一個元素之後插入新的設定值（不建議在補靶中使用）
  "含列表的設定項/@next": 在列表最後一個元素之後插入新的設定值（不建議在補靶中使用）
```

打补丁注意事项

打补丁，相当于对某个数据域重新进行赋值。该数据域如果是一个映射，那么它原始值中未被赋值的属性就会被删除。 所以，要精确引用到需要进行修改的数据域很重要。 如果引用的数据范围过大，而又没有给其中的所有属性赋值，布署时，程序由于读不到某些关键的属性，就会崩哦。

比如，现在需要定制输入方案，切换到英文模式时，调用标准的qwerty键盘。但是需要对qwerty键盘打补丁，对其中的某些项作修改。

```
#对键高度，和某个按键的事件，以及中/英模式，做补丁修改

#方法1，OK，可以正常工作，精确指定了修改的属性
  "preset_keyboards/qwerty/height": 60
  "preset_keyboards/qwerty/keys/@31":  {label: "英", click: Keyboard_default}
```

```
  "preset_keyboards/qwerty/ascii_mode": 1

#方法2，用全覆盖的方式定制
  "preset_keyboards/qwerty":   #注意这里，相当于对整个qwerty重新赋值
    height: 60  #每行的高度
    keys:    #这里必须重新把所有要使用的按键都声明一遍
      - {click: space}
    ascii_mode: 1

#注：用这种方式定制时，相当于对整个qwerty赋值，不能只赋值其中的一部分，而是要全部赋值。否则，未出现的
```

其他

另外，网上还有数据引用和合并的介绍，但是我还没有验证，在我们程序中是否可行。

可以使用&符号定义一个引用标签，使用符号*引用这个标签的数据，使用符号<<进行hash值合并操作，例如：

```
# sequencer protocols for Laser eye surgery
---
- step:  &id001                # defines anchor label &id001
    instrument:      Lasik 2000
    pulseEnergy:     5.4
    pulseDuration:   12
    repetition:      1000
    spotSize:        1mm
- step:
    <<: *id001               # merges key:value pairs defined in step1 anchor
    spotSize:        2mm        # overrides "spotSize" key's value
- step:
    <<: *id001               # merges key:value pairs defined in step1 anchor
    pulseEnergy:     500.0        # overrides key
    alert: >                 # adds additional key
      warn patient of
      audible pop
```

1. &id001定义了一个id001的引用标签（引用文档中第一个step元素的所有属性）；
2. 第二个step元素引用id001后，重写spotSize属性；
3. 第三个step元素引用id001后，重写pulseEnergy属性，并添加alert属性