

Katalon Studio + Groovy

Katalon & Groovy & Selenium

Katalon: GUI (Eclipse)

→ Groovy

Selenium

→ open browser

→ DOM stránky najde elementy (xpath)

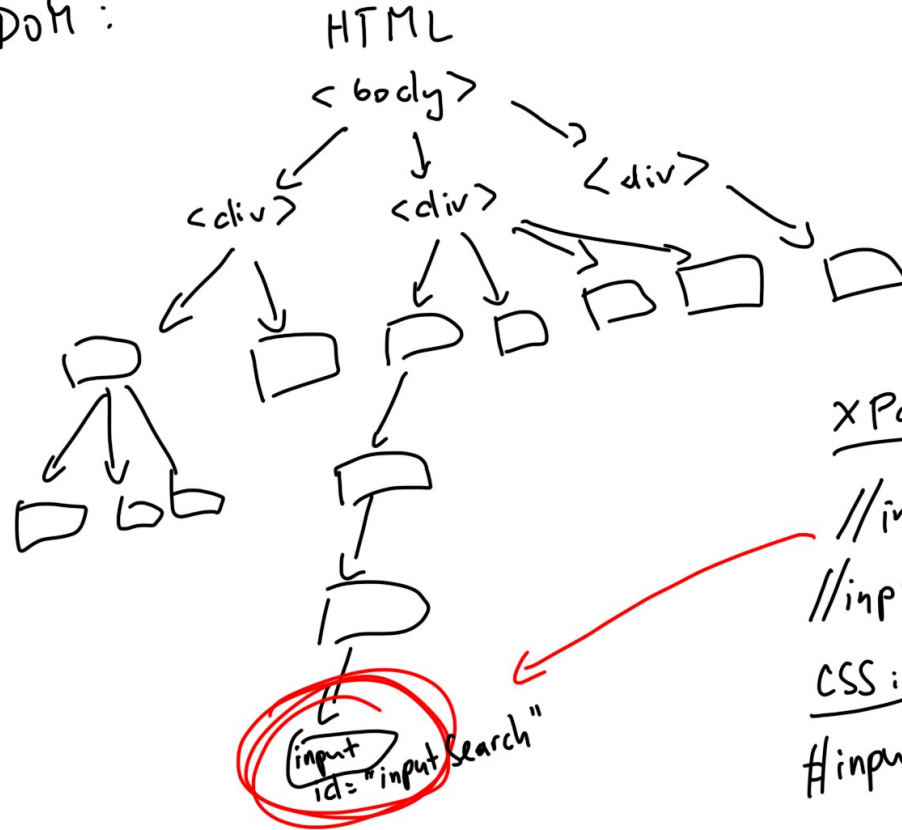
→ odesle formulář

kliknutí na tlačítko

vyplnění inputu

DOM (Document Object Model)

DOM :



XPath:

//input
//input[@id="inputSearch"]

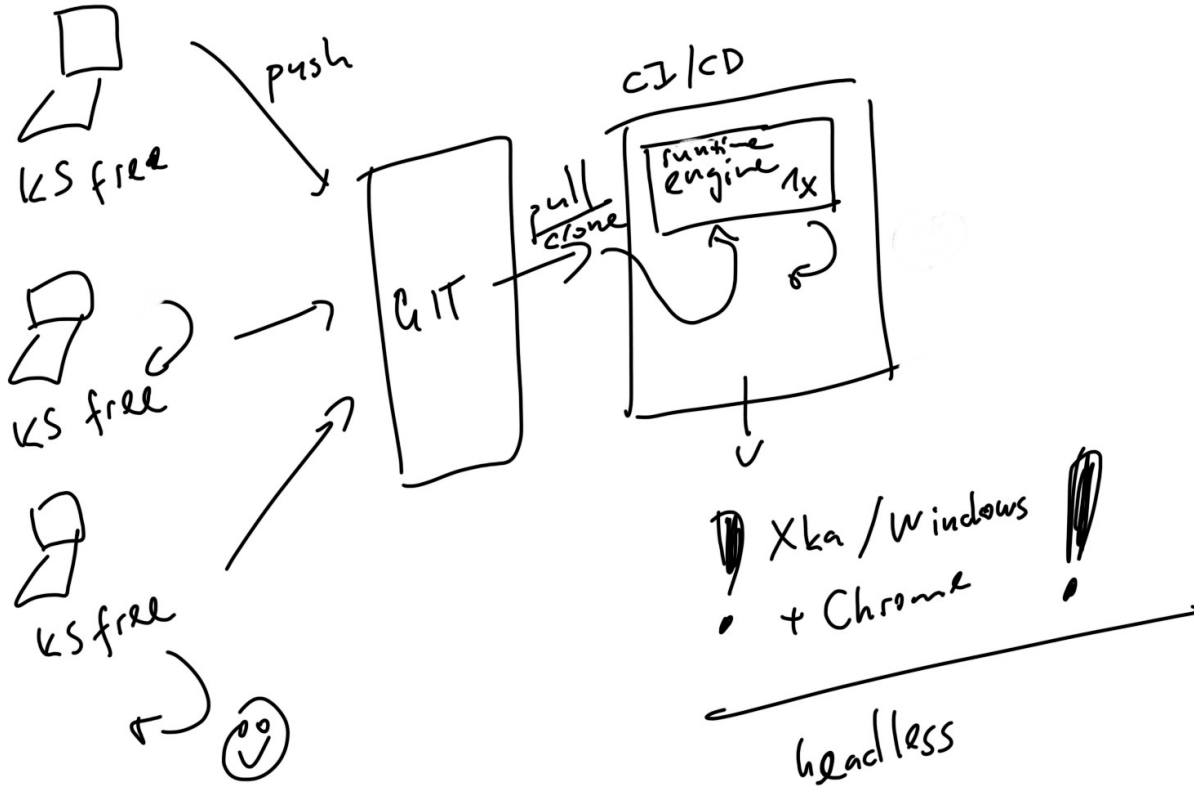
CSS:

#inputSearch

XPath

- Pomocí XPath výrazů lze najít nějaký element (například input) v DOM stromu:
 - https://docs.katalon.com/katalon-studio/docs/detect_elements_xpath.html#deal-with-dynamic-elements-using-xpath
 - <https://www.dev2qa.com/selenium-xpath-css-selector-example/>
 - <https://www.lambdatest.com/blog/complete-guide-for-using-xpath-in-selenium-with-examples/>
 - <https://devhints.io/xpath>

Katalon Studio & Runtime Engine



Groovy version

- Jak zjistit verzi Groovy:
 - V adresáři s Katalon Studio je adresář:
 - `plugins/org.codehaus.groovy_{VERSION}` a `VERSION` je používaná verze Groovy.
 - Katalon Studio 8 používá Groovy verze 2.4

Proměnné, komentáře

```
// jednoradkový komentář
```

```
/*
```

```
 * víceraďkový komentář
```

```
*/
```

```
def aaa = "aaa value"
```

```
println (aaa)    je to samé jako:    println aaa
```

(ale je preferovaná první varianta)

Poznámky:

- Název proměnné / metody / třídy atd. nesmí být jedno z klíčových slov: https://groovy-lang.org/syntax.html#_keywords
- Název nesmí začínat číslicí, také v názvu nesmí být speciální znaky jako pomlčka, plus, tečka, čárka apod.
- **POZOR!** Groovy (i Java) je case sensitive!!!

Text (String)

- Text (String):

```
def aa1 = "aaa value";  
def aa2 = "aaa value"  
def aa3 = 'aaa value'  
  
String aa4 = "aaa value"  
  
String aa5 = 'aaa value'
```

- Středník na konci výrazu je nepovinný
- Můžeme používat dvojité uvozovky, nebo apostrofy. Doporučoval bych dvojité uvozovky, protože tak se používají i v Javě.
- Stringy je možné spojovat operátorem "+":

```
println(aa1 + " " + aa2)
```

- Multi-line string:

```
def str = """multi  
line  
string  
"""
```

- Escapování speciálních znaků:
 - https://groovy-lang.org/syntax.html#_escaping_special_characters
- String interpolation:
 - https://groovy-lang.org/syntax.html#_string_interpolation

Number

- Číslo:

```
def bb1 = 10
```

```
def bb2 = 10.5
```

```
def bb3 = bb1 + bb2
```

```
int bb4 = 10
```

```
double bb5 = 10.5
```

- U velkých čísel pro vizuální oddělení tisíců se dá použít podtržítko: 1_000_000

- Čísla jsou dvou základních typů:

- Integrální (bez desetinné tečky)
- Decimální (s desetinnou tečkou)

List

- List se definuje tímto způsobem:

```
def numbers = [1, 2, 3]
```

- <https://docs.oracle.com/javase/8/docs/api/java/util/List.html>
- Je také možné používat klasické statické pole:
 - https://groovy-lang.org/syntax.html#_arrays

Proměnné: rozsah

Zejména v případě, že se používají typy, pak pozor na jejich rozsah!

```
println(2_000_000_000 + 2_000_000_000)
```

Výsledek: -294967296

Type	Size (in bits)	Range
byte	8	-128 to 127
short	16	-32,768 to 32,767
int	32	-2^{31} to $2^{31}-1$
long	64	-2^{63} to $2^{63}-1$
float	32	1.4e-045 to 3.4e+038
double	64	4.9e-324 to 1.8e+308
char	16	0 to 65,535
boolean	1	true or false

Primitivní datové typy

- int, double apod. nejsou třídy, ale tzv. primitivní datové typy. Díky tomu se při jejich vytváření nepoužívá operátor “new”.

https://groovy-lang.org/objectorientation.html#_primitive_types

Proměnné: operátory

- Groovy obsahuje klasické operátory: +, -, *, /, % (procento je zbytek po dělení)
- Dále také ++ (increment o 1), -- (decrement o 1)
- Nebo operátory přiřazení jako +=, -= apod.
 - https://groovy-lang.org/operators.html#_assignment_arithmetic_operators

Boolean

- Boolean reprezentuje pravdu / nepravdu: true / false:

```
def b1 = true
```

```
boolean b2 = false
```

- Funguje i toto, ale to bych nepoužíval (v Javě by tohle byla chyba):

```
def b1 = "true"
```

```
def b1 = 1
```

- Boolean výrazy se často používají v podmínkovém větvení (viz. dále), nebo v assertech:

```
assert b1
```

- Když b1 není “true”, pak se vyhodí chyba

Podmínkové větvení

- V Groovy je klasické if – else if – else:

```
def x = false, y = false
```

```
if ( !x ) {  
    x = true  
}
```

```
if ( x ) {  
    x = false  
} else {  
    y = true  
}
```

- Plná syntaxe:

```
if ( ... ) {  
    ...  
} else if (...) {  
    ...  
} else {  
    ...  
}
```

Operátory

- Nejpoužívanější relační operátory:

- Rovnost (==), nerovnost (!=), menší (<), větší (>), menší rovno (<=), větší rovno (>=)

```
assert 1 + 2 == 3
```

```
assert 3 != 4
```

- Logické operátory:

- “and”: &&, “or”: ||, “not”: !

```
assert !false
```

```
assert true && true
```

```
assert true || false
```

Poznámka: Výchozí precedence operátorů je jako v matematice (čili největší má NOT, pak AND a nakonec OR). Precedence se dá změnit pomocí závorek.

Třídy, atributy, metody

- Veškerý kód je v tzv. třídách (fyzicky je třída soubor na disku).
- Příklady tříd:
 - String, WebUI apod.
- Třídy typicky obsahují atributy a metody.

```
class Trida1 {  
    void test() {  
        println("byla zavolana metoda test()")  
    }  
}
```

```
def objekt = new Trida1()  
objekt.test()
```

Vytvoření instance třídy a zavolání instanční metody test()



Instance vs. Static

- Každý atribut a metoda je ve výchozím nastavení instanční. To znamená, že pro přístup je zapotřebí instance třídy (vytváří se pomocí operátoru new). Existují ale také statické atributy a statické metody, ke kterým se pak přistupuje napřímo přes název třídy bez nutnosti vytváření objektu:

```
class Trida1 {  
    static void test() {  
        println("byla zavolana metoda test()")  
    }  
}  
  
Trida1.test()
```

Importy

- Každá třída se nachází v tzv. balíčku (fyzicky to je adresář na disku). Například třída `String` se nachází na disku na cestě `java/lang/String.class`.
- V Groovy (a Javě) existuje pojem “plný název třídy”, přičemž pokud chceme nějakou třídu používat, pak ji musíme referencovat tímto plným názvem. Kdyby byl toto jediný způsob použití tříd, pak by byl kód nesnesitelně dlouhý a neustále by se opakovala informace z jakého balíčku nějaká třída pochází.
- Proto existuje princip importu. Importy musí být jako první věc ve skriptu, nemohou být nikde jinde. Pomocí importů se dá zkrátit přístup k názvům tříd nebo pomocí statických importů se dá zkrátit přístup ke statickým atributům / metodám.
- <https://groovy-lang.org/structure.html>

Cykly

- Klasický for loop:

```
String message = ''  
for (int i = 0; i < 5; i++) {  
    message += 'Hi '  
}  
assert message == 'Hi Hi Hi Hi Hi '
```

- While loop:

```
def x = 0  
def y = 5  
while ( y-- > 0 ) {  
    x++  
}  
assert x == 5
```

Výjimky

- Výjimky fungují klasickým způsobem jako v Javě (klasický try-catch-finally).
- Jakmile se v průběhu testu vyhodí chyba, tak se běh testu ukončí (test fail) a do konzole se vypíše stacktrace.

https://groovy-lang.org/semantics.html#_try_catch_finally

Test Case Variables

[illegible]

Test Case Variables

- Použití proměnných v kódu:

```
KeywordUtil.LogInfo('${employee} : ${department}')
```

- Obdobně funguje:

```
def aaa = 'aaa value'
```

```
KeywordUtil.LogInfo("${aaa}")
```

```
KeywordUtil.LogInfo(aaa)
```

waitForXXX

- DOM se tvoří postupně. Lehce může nastat situace, že element (například INPUT) ještě není načtený v DOMu a už do něj chceme zapisovat. V takovém případě potřebujeme waitForXXX.
- Příklad:

```
def link = findTestObject('TODO_OBJECT_ID')
```

```
WebUI.waitForElementVisible(link, 30)
```

```
WebUI.click(link)
```

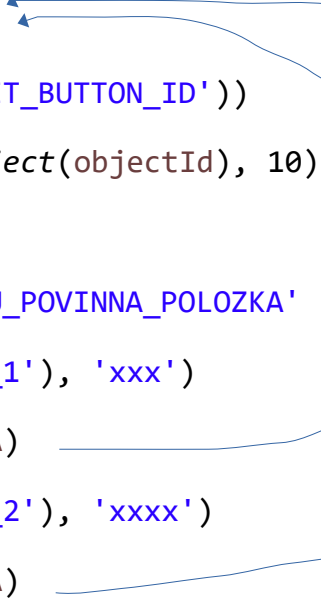
Zavolá operaci pro nalezení TestObjectu a čeká až 30 vteřin než bude element “visible”. Poté na něj klikne. Další zajímavá operace je například waitForElementPresent, která čeká až bude element v DOMu.

Poznámka: WebUI.delay(30) by natvrdo čekalo 30 vteřin.

Custom metody

- Pro omezení duplikování kódu je vhodné používat metody:

```
def sendAndCheckRequired(def objectId) {  
    WebUI.click(findTestObject('TODO_SUBMIT_BUTTON_ID'))  
    WebUI.verifyElementPresent(findTestObject(objectId), 10)  
}  
  
def TEXT_POVINNA_POLOZKA = 'TODO_ID_TEXTU_POVINNA_POLOZKA'  
WebUI.setText(findTestObject('TODO_INPUT_1'), 'xxx')  
sendAndCheckRequired(TEXT_POVINNA_POLOZKA)  
WebUI.setText(findTestObject('TODO_INPUT_2'), 'xxxx')  
sendAndCheckRequired(TEXT_POVINNA_POLOZKA)
```



V tomto příkladu se vyplní input ve formuláři a poté se klikne na submit tlačítko a zjistí se, jestli je na stránce text “Povinná položka”. Protože by se volání click & verifyElementPresent neustále opakovalo, tak je v metodě sendAndCheckRequired()

Jak z TestObject dynamicky získat property (například “id”, “class”, “name”)

```
def testObject = findTestObject('TODO_OBJECT_ID')  
  
def clazz = testObject.findPropertyValue("class")  
  
// println clazz  
  
assert clazz == "form-control"
```

Stažení (a následné smazání) souboru

```
WebUI.navigateToUrl('TODO_PAGE')

def link = findTestObject('TODO_LINK_TO_FILE')

WebUI.waitForElementClickable(link, 60)

WebUI.click(img)

while (true) {
    WebUI.delay(1)

    def downloadedFile = new File('TODO_PATH_TO_DOWNLOADED_FILE')

    if (downloadedFile.exists()) {
        downloadedFile.delete()

        break
    }
}
```

Jak zjistit že aktuální URL je rovno očekávanému URL?

```
def currentUrl = WebUI.getUrl()  
  
assert currentUrl == 'TODO_EXPECTED_URL'
```

Groovy dokumentace

- <https://groovy-lang.org/documentation.html>