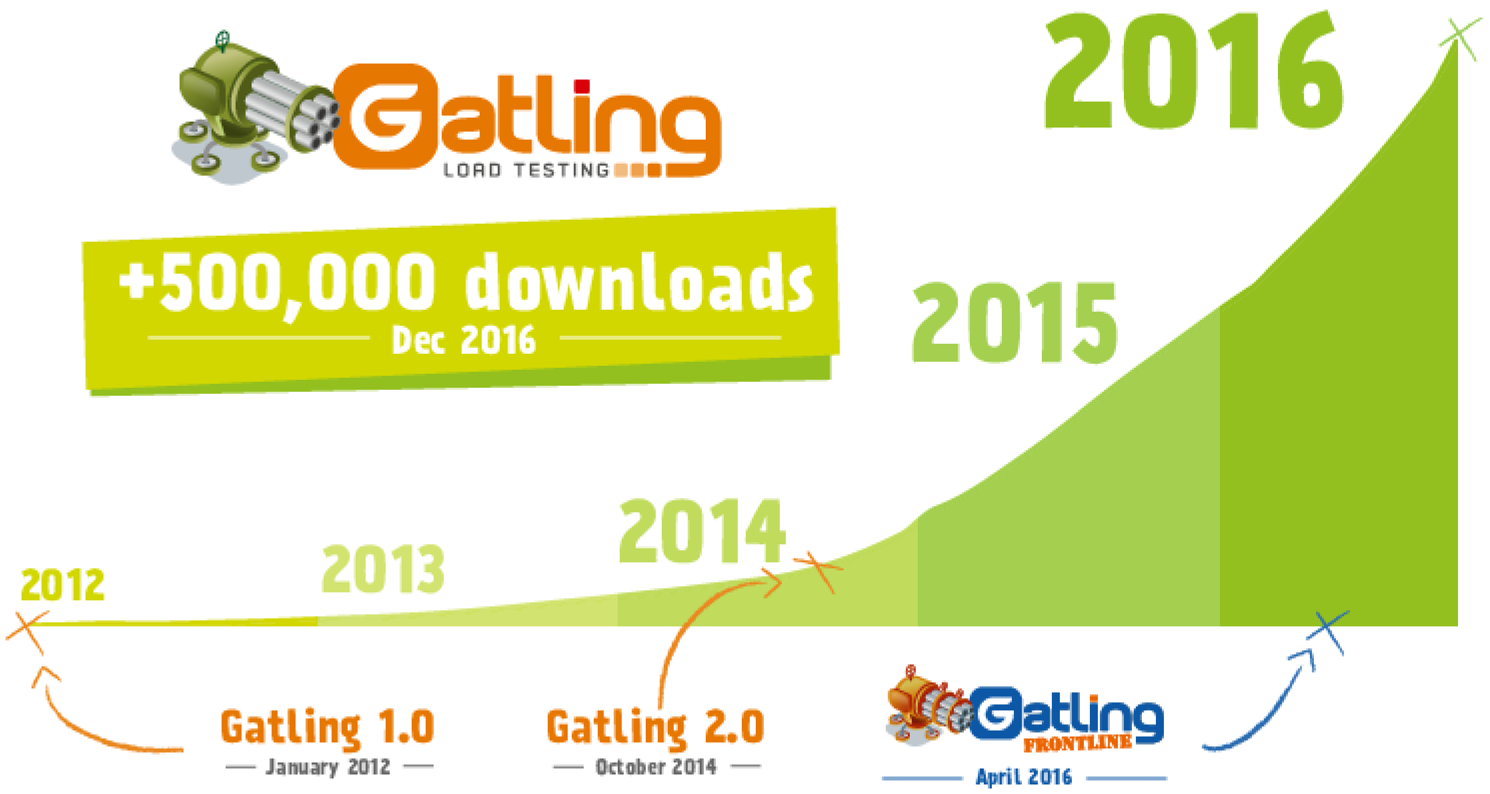
Gatling Training

gatling.io

@GatlingTool

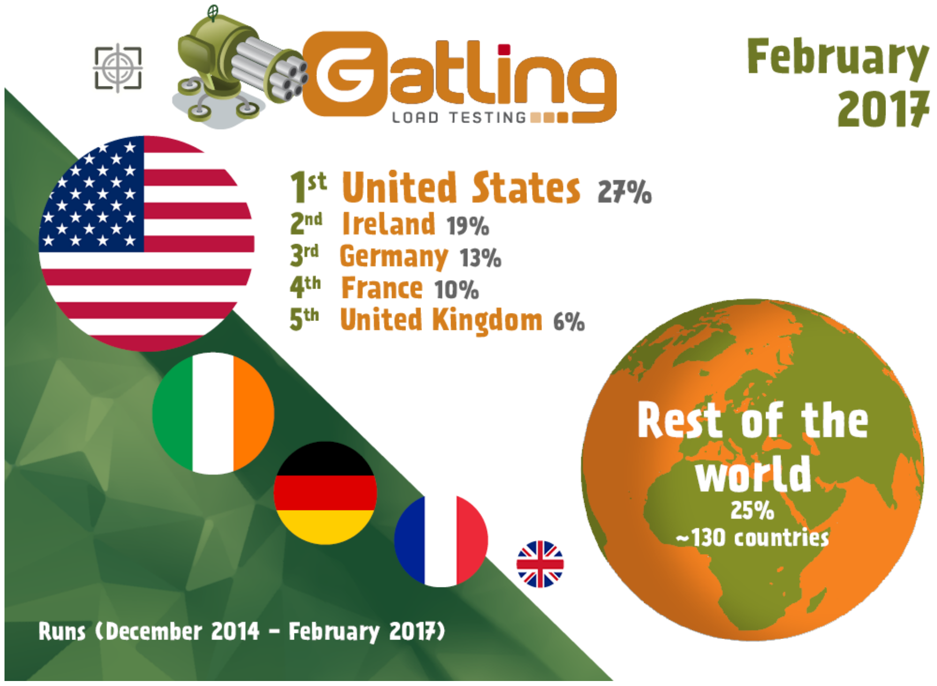
Who We Are

Popular Open Source Load Test Tool



Used All Over the World

- +10k companies
- +30k downloads/month
- +1M runs/month
- 26 conferences in 2016



Baked in France (Cachan)

CÉDRIC COUSSERAN

Developer

GUILLAUME CORRÉ

Developer

ALEXANDRE CHAOUAT

Developer

THOMAS GRENIER

Developer

STÉPHANE LANDELLE

CEO and Founder

PAUL-HENRI PILLET

COO

by GatlingCorp

- OSS: Support, Mentoring, Training, Feature Sponsoring
- Enterprise: FrontLine
- Professional Services: Load Test Campaign

Gatling, in Short

- Aiming at Agile Testing
- Resource Efficient
- Detailed Reporting

Aiming at Agile Testing

Code Centric

- Tests value is quick feedback (Continuous Integration)
- Tests must be maintained and shared
- Unit Tests and Integration Tests are code
- Deal with Load Testing the same way!

Aiming at Agile Testing

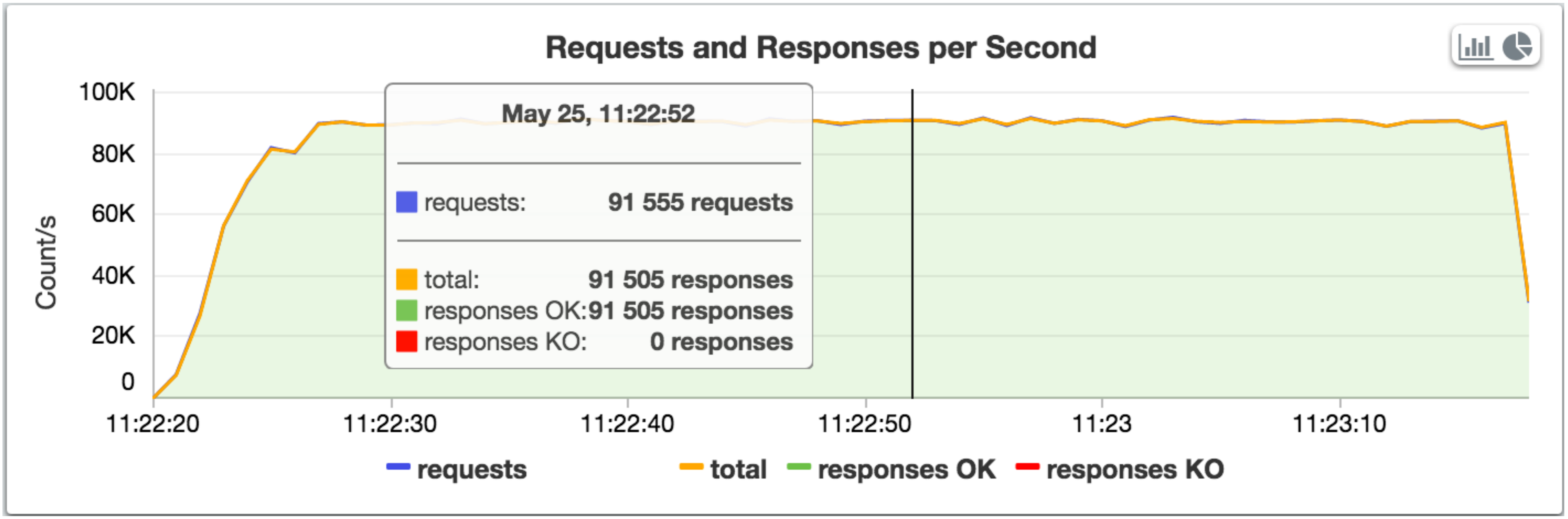
DSL + Scala

```
val scn = scenario("Google Search")
    .exec(http("Search Gatling")
        .get("https://www.google.fr/?#q=gatling+load+testing" ))
    .pause(1)
```

Resource Efficient

- Orchestration: Akka actors
- NIO: Netty

Resource Efficient



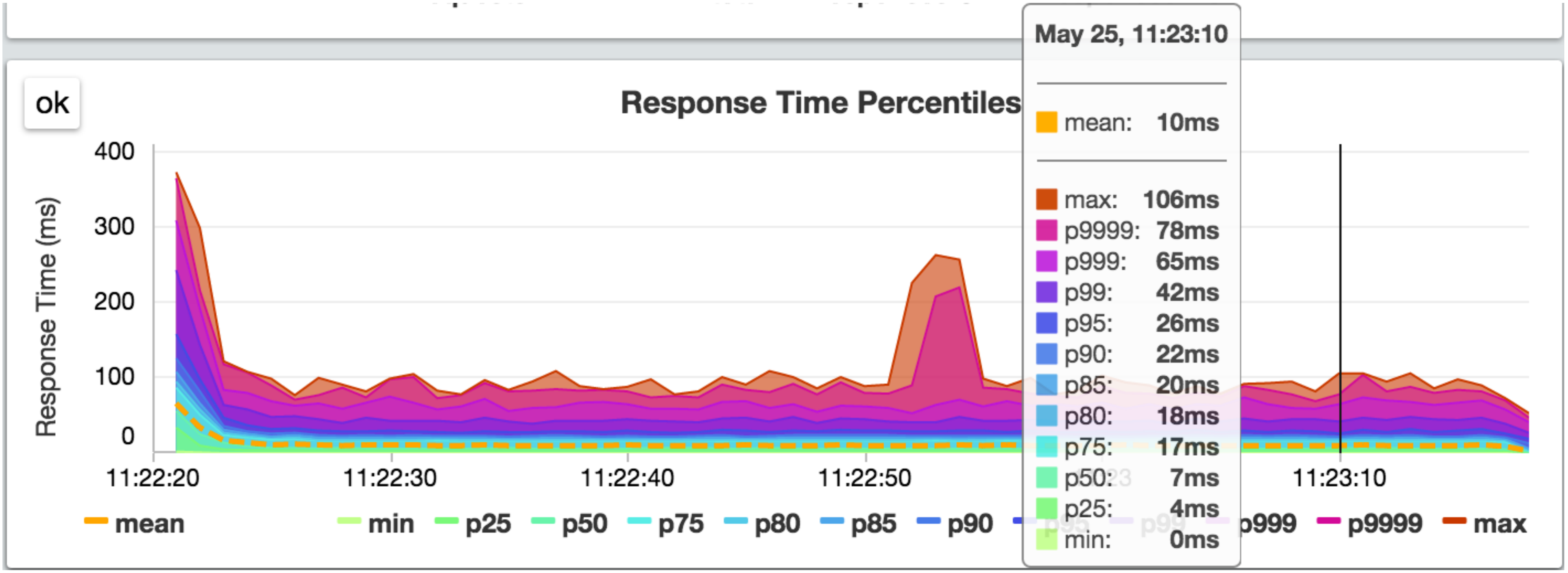
May 25, 11:22:52

🟦 requests: 91 555 requests

🟧 total: 91 505 responses
🟩 responses OK: 91 505 responses
🟥 responses KO: 0 responses

Detailed Reporting

Response Time Distribution: Percentiles



May 25, 11:23:10

mean:	10ms
max:	106ms
p9999:	78ms
p999:	65ms
p99:	42ms
p95:	26ms
p90:	22ms
p85:	20ms
p80:	18ms
p75:	17ms
p50:	7ms
p25:	4ms
min:	0ms

Glossary

- **Simulation** = Gatling Test, launched by the Gatling engine
- **Run** = Simulation execution
- **Scenario** = Workflow = Chain + name
- **Action** = Workflow Step
- **Chain** = Sequence of Actions
- **Users** = Messages
- **Injection Profile** = Arrival rate over time

Installing

Objectives

- Getting familiar with Gatling distributions
- Using Gatling in IDE

Distributions

- Bare zip bundle: download
- Maven artifacts on Maven central

Directories Structure

- **/bin**: launch scripts
- **/lib**: libraries (jars)
- **/conf**: configurations files (e.g. Gatling and logging)
- **/user-files/simulations**: Scala code to be compiled
- **/user-files/data**: data files (e.g. CSV files)
- **/user-files/bodies**: body files (e.g. upload, JSON templates)

Development Environment

- Get latest JDK8
- Get latest IntelliJ Community Edition (possibly EAP)
- Install IntelliJ Scala plugin
- Create a new IntelliJ project
- Git Clone or download in there the maven demo project
- Import it as a new module from existing source
- Verify that `computerdatabase.BasicSimulation` class gets compiled as a Scala source
- Run `Engine` class
- Open generated report file

First Steps with Gatling

Objectives

- Simulation class structure
- Recorder usage
- DSL usage
- Expression Language (EL)

1 Simulation Class and DSL

```scala
package basic

import io.gatling.core.Predef._ // (1)
import io.gatling.http.Predef._
import scala.concurrent.duration._

class Simple extends Simulation { // (2)

  val httpConf = http.baseURL("http://gatling.io") // (3)

  val scn = scenario("scenario")  // (4)
    .exec(http("home").get("/"))

  setUp(scn.inject(atOnceUsers(1))) // (5)
          .protocols(httpConf) // (6)
}
```

1 Simulation Class and DSL

- (1) import DSL
- (2) entrypoints must extend Simulation
- (3) shared HTTP config
- (4) declared scenario as workflow of execs
- (5) inject users
- (6) attach HTTP config

2 BrowseSimulation

2.1 Configure Recorder

- Start Recorder from launcher class
- Configure `gatling` as package and `BrowseSimulation` as class name
- Disable resource inferring
- Exclude static resources
- Click on "Save Preference"
- Click on "Start"
- Configure your browser to use it as a HTTP Proxy

2.2 Record

- Browse to http://computer-database.gatling.io
- Click on `Next` and go to list next page
- Click and `Stop and Save` in Recorder, then kill it
- Check generated Simulation class in IntelliJ
- Run it with Gatling
- Check out the report

2.3 Give proper request names

- Rename requests into "Page 0" and "Page 1"
- Run again and check the report

2.4 Add more requests

- Add requests for "Page 2" and "Page 3" with proper urls
- Run and check console output

2.5 Add a during loop

- Check the Gatling cheat sheet
- Have the virtual user loop over the full sequence `during` 30 seconds

```
.during(???) {
  ???
}
```

- Run and check console output
- Change loop duration to 1 minute (don't use 60 seconds)

2.5 Add a during loop

Solution

```
.during(1 minute) {
  exec(http("Page 0").get("/computers?p=0"))
  .pause(1)
  .exec(http("Page 1").get("/computers?p=1"))
  .pause(1)
  .exec(http("Page 2").get("/computers?p=2"))
  .pause(1)
  .exec(http("Page 3").get("/computers?p=3"))
  .pause(1)
  .exec(http("Page 4").get("/computers?p=4"))
  .pause(1)
}
```

2.6 Use a repeat loop

- Replace the inner sequence
- Instead `repeat` 5 times "Page 0" and a pause

```
.repeat(???) {
  ???
}
```

- Run and check console output

2.6 Use a repeat loop

Solution

```
.repeat(5) {
  exec(http("Page 0")
    .get("/computers?p=0"))
    .pause(1)
}
```

2.7 Make page index dynamic

- Make the `repeat` loop visit "Page 0" to "Page 4" instead
- Check the cheat sheet and force the counter name
- Use Gatling Expression Language to inject the counter value into the request names and urls

```
.repeat(???, ???) {
  exec(http("Page ${???}").get(???))
  .pause(1)
}
```

- Run and check reports

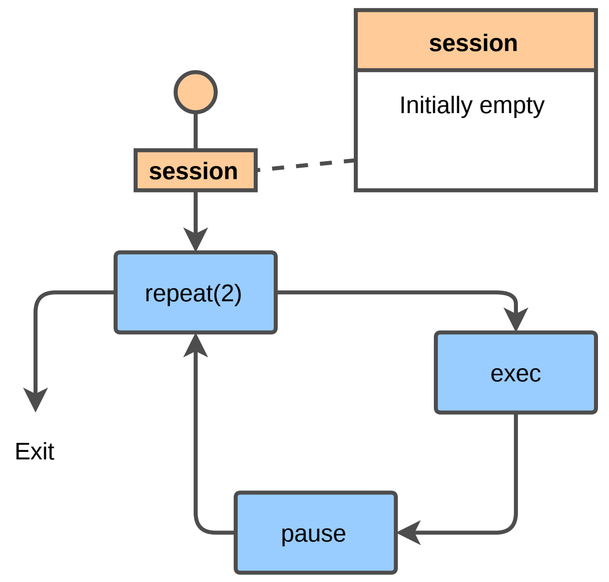
2.7 Make page index dynamic

Solution

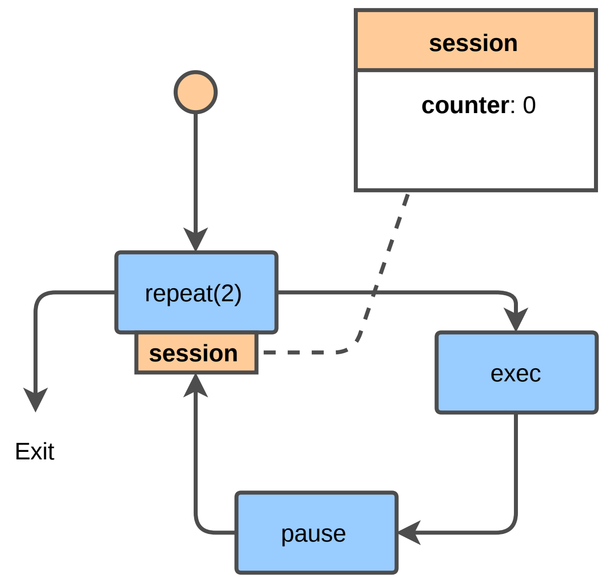
```
.repeat(5, "i") {
  exec(http("Page ${i}")
    .get("/computers?p=${i}"))
    .pause(1)
}
```

2.8 Repeat loop behavior

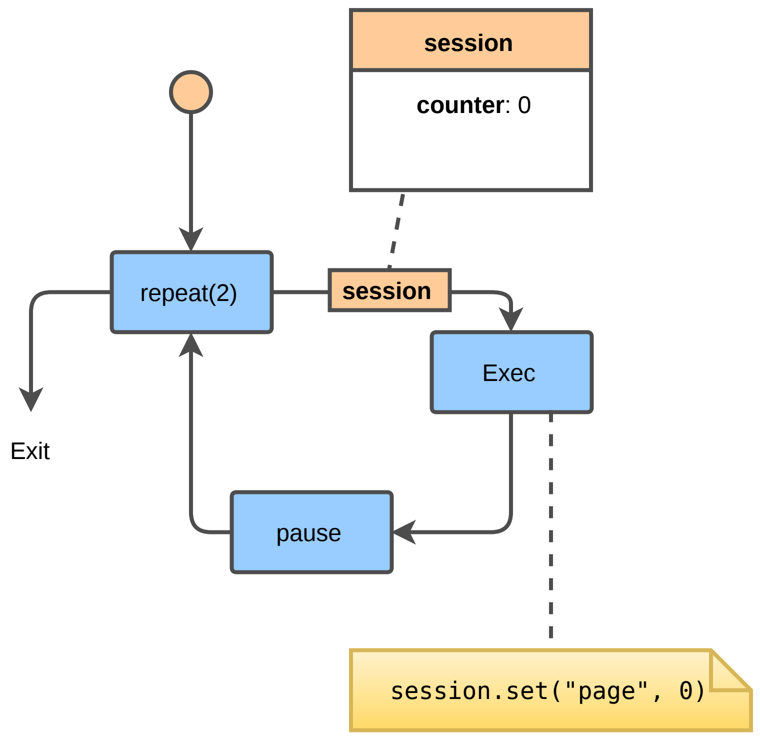
2.8 Repeat loop behavior



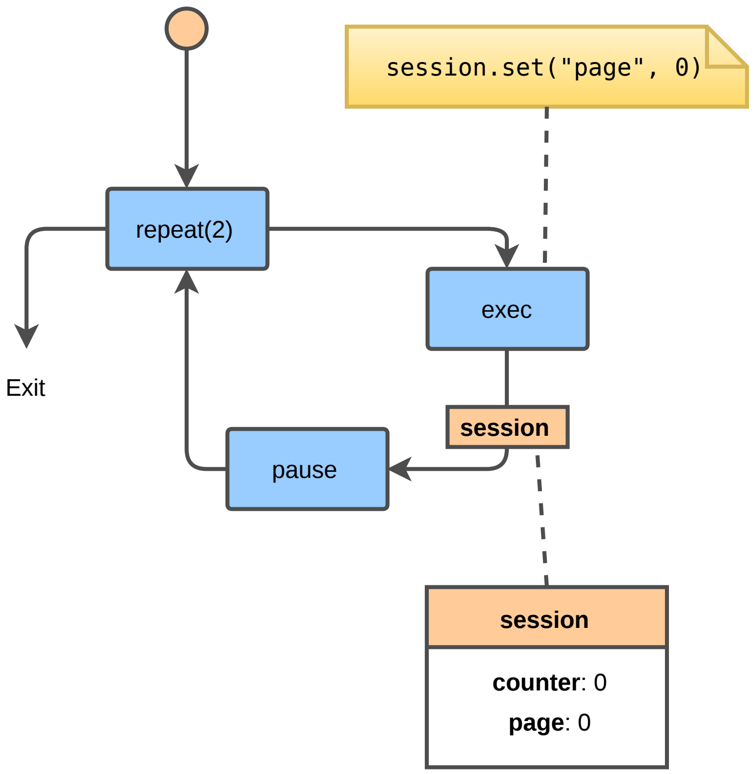
2.8 Repeat loop behavior



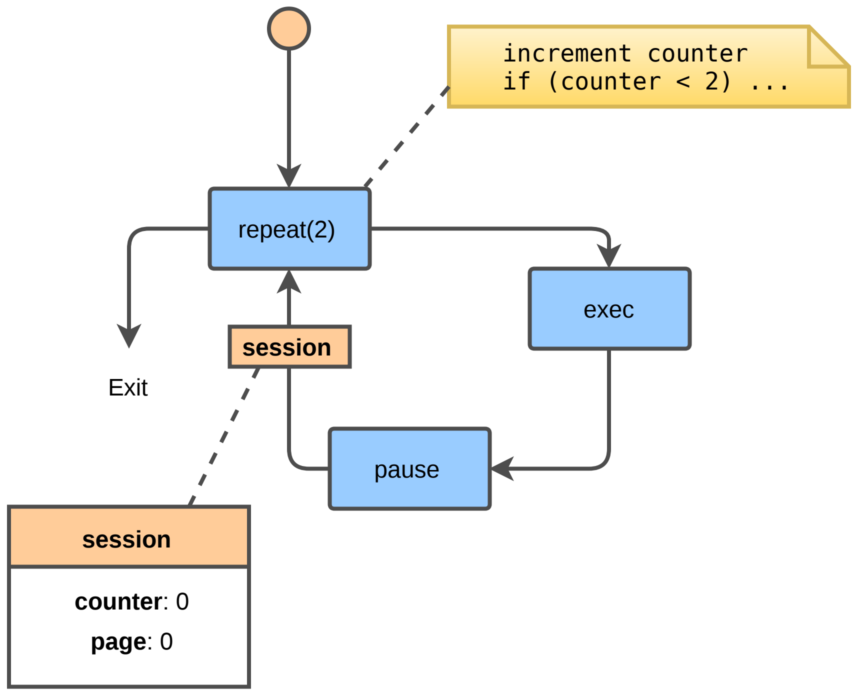
2.8 Repeat loop behavior



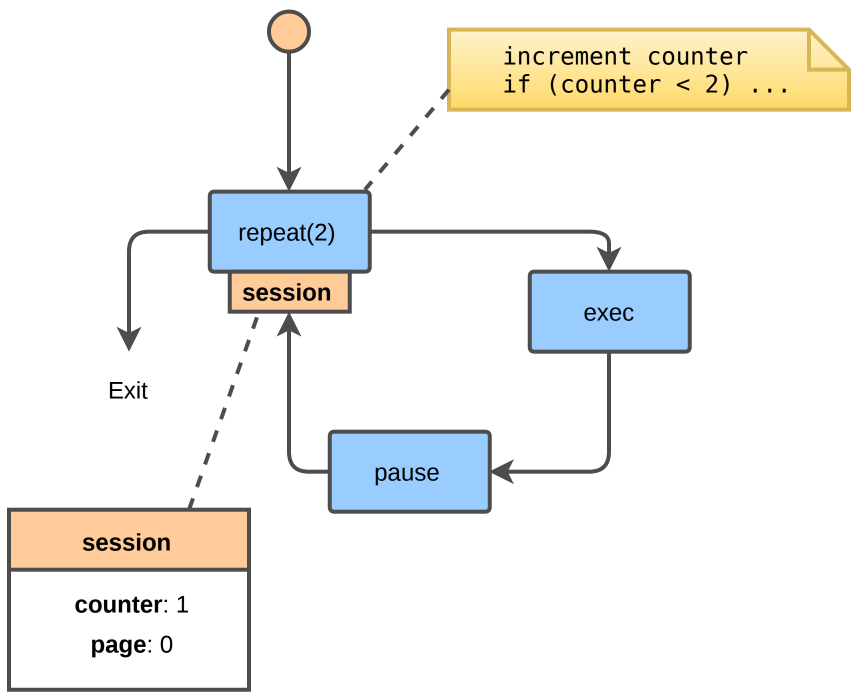
2.8 Repeat loop behavior



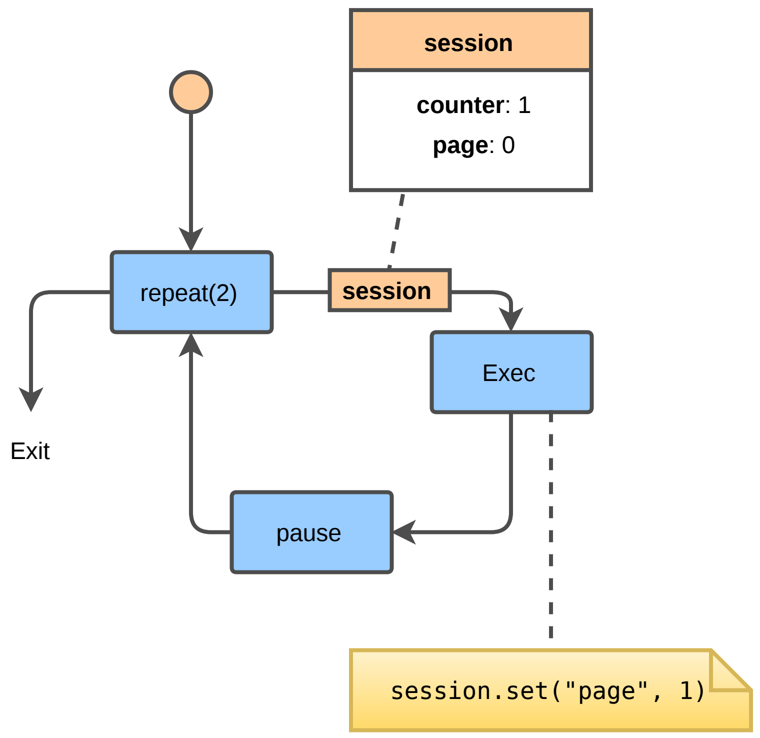
2.8 Repeat loop behavior



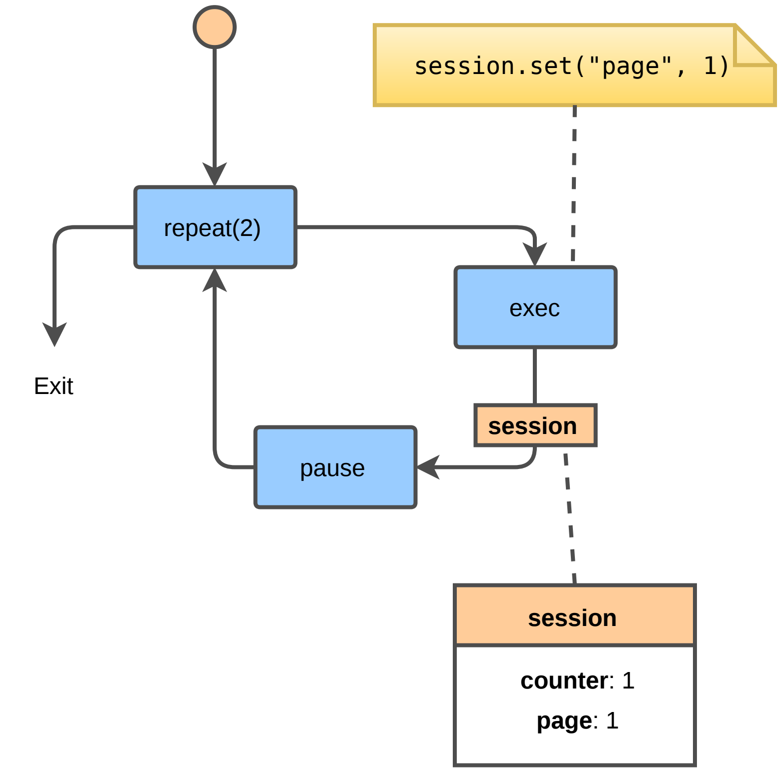
2.8 Repeat loop behavior



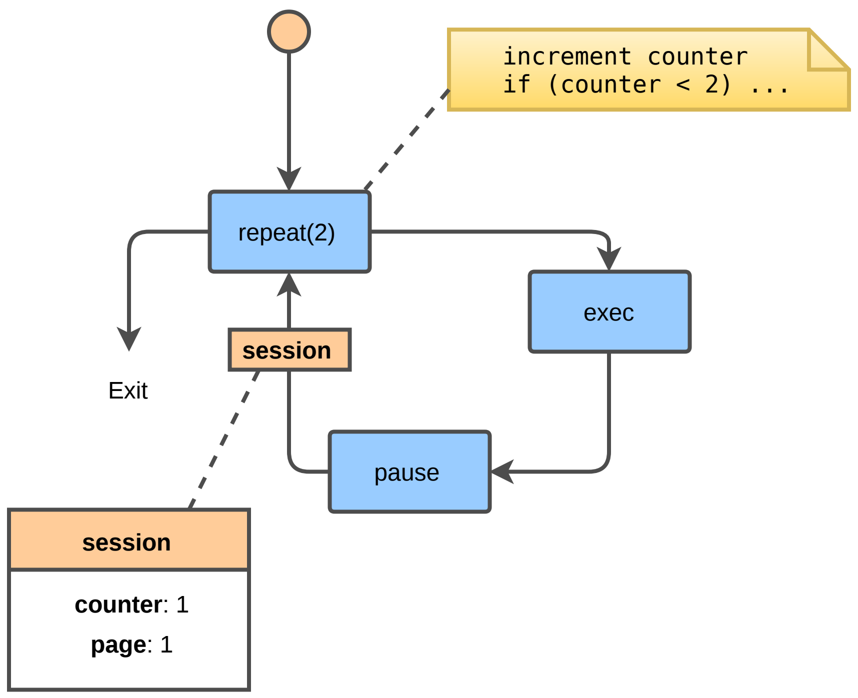
2.8 Repeat loop behavior



2.8 Repeat loop behavior

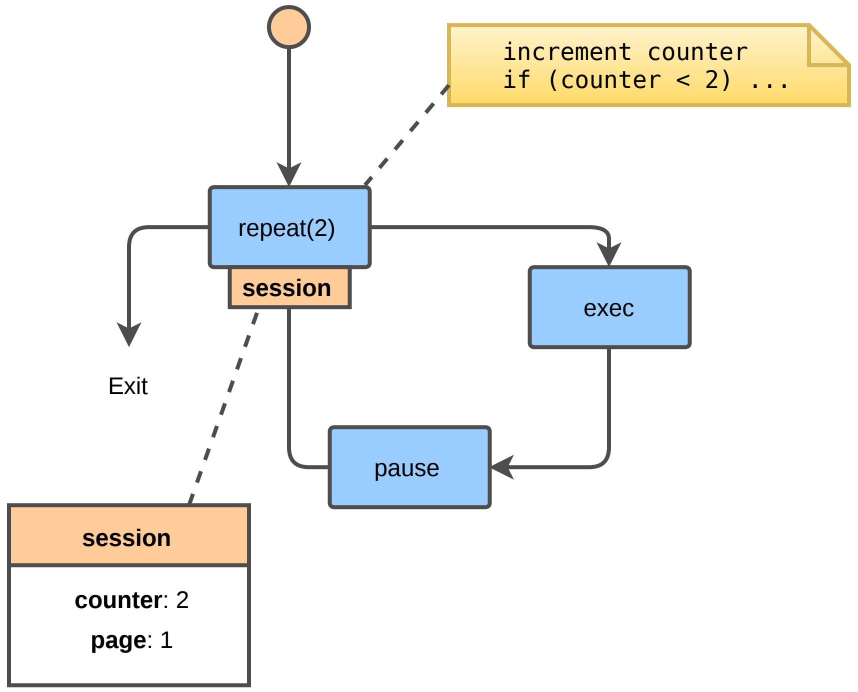


2.8 Repeat loop behavior

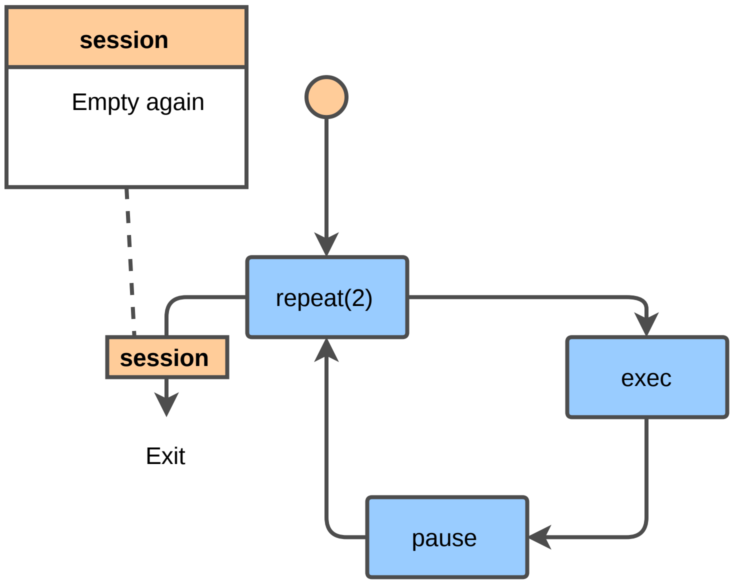


```
increment counter
if (counter < 2) ...
```

2.8 Repeat loop behavior



2.8 Repeat loop behavior



3 SearchSimulation

3.1 Record

- Start Recorder
- Configure `gatling` as package and `SearchSimulation` as class name
- Start
- Browse to http://computer-database.gatling.io/computers
- Search for "MacBook"
- Select "MacBook Pro" from the list
- Stop, Save and Run

3.2 Add a Check

- Checks are primarily used to asserting that responses are correct
- Gatling automatically performs a HTTP status check (2XX or 304)
- Rename requests into "Search" and "Select"
- Check that the "Search" response contains the "MacBook Pro" String

```
.exec(http("Select").get("http://...")
  .check(???))
```

- Run and check reports
- Bonus: Look for a non existing String instead

3.2 Add a Check

Solution

```
.exec(http("Search")
  .get("http://computer-database.gatling.io/computers?f=MacBook" )
  .check(substring( "MacBook Pro" )))
```

3.3 Use a CSS selector

- `substring` is very efficient but limited
- `regex` is not easy to maintain and ill suited for HTML
- Prefer `css` and CSS selectors instead
- Replace check with a CSS selector that looks for a "`<a>`" HTML tag whose text is "Macbook Pro"

```
.exec(http("Search").get("http://...")
  .check(css(???)))
```

- Run and check reports

3.3 Use a CSS selector

Solution

```
.exec(http("Search")
  .get("http://computer-database.gatling.io/computers?f=Macbook" )
  .check(css("a:contains('MacBook Pro')" )))
```

3.4 Use Checks to capture data

- Checks can save the capture data with `.saveAs("attributeName")`
- Instead of hard-coding "Select" url, follow link in "Search" response

```
.exec(http("Search").get("http://...")
  .check(css(???).saveAs("url")))
.pause(1)
.exec(http("Select").get(???))
```

3.4 Use Checks to capture data

Solution

```
.exec(http("Search").get("http://...")
  .check(css("a:contains('MacBook Pro')", "href").saveAs("url")))
.pause(1)
.exec(http("Select").get("${url}"))
```

3.5 Stop if the select fails

- It is useless to do the Select if the Search fails
- Check the cheat sheet and exit the scenario if the check fails

3.5 Stop if the select fails

Solution

```
.exec(http("Search").get("http://...")
  .check(css("a:contains('MacBook Pro')", "href").saveAs("url"))).exitHe
.pause(1)
.exec(http("Select").get("${url}"))
```

3.6 Make data dynamic

Having all the virtual users hit the same data isn't proper load testing.

System under test behavior would differ from live conditions.

You would test your caches, not your application.

3.6 Make data dynamic

Feeders are data sources shared amongst virtual users.

Every time a virtual user reaches a `feed` action, it gets populated with a record.

When a feeder instance is empty, Gatling shuts down.

3.6 Make data dynamic

- Create a file named "search.csv" (note header and records)

```
searchCriterion,searchComputerName
MacBook,MacBook Pro
Eee,ASUS Eee PC 1005PE
```

- Add a `feed` action that reads from this file

```
.feed(???)
```

- Replace hard coded values in "Search" request with Expression Language placeholders
- Run with 2 users, then 3
- Change feeder's strategy so it never gets empty

3.4 Make data dynamic

Solution

```
.feed(csv("search.csv").circular)
.exec(http("Search")
  .get("/computers?f=${searchCriterion}")
  .check(css("a:contains('${searchComputerName}')", "href")
      .saveAs("url")))
```

4 EditSimulation

4.1 Record

- Record a simulation named "EditSimulation"
- Click on "Add a new computer"
- Fill the form and click on "Create"
- Note that Gatling automatically followed redirect
- Note that form "Content-Type" header is automatically added

4.2 Use automatic retry

- Rename requests into "Form" and "Post"
- Wrap the chain with a `tryMax` block that retries at max 3 times
- Forcefully make "Post" crash with an incorrect `status` check
- Run

4.2 Use automatic retry

- Instead of always crashing, randomly check against 200 or 201
- Hint: `java.util.concurrent.ThreadLocalRandom`
- Hint: beware of things happening at load time vs runtime

4.2 Use automatic retry

Solution

```
tryMax(3) {
  exec(http("Form")
    .get("/computers/new"))
  .pause(1)
  .exec(http("Post")
    .post("/computers")
    .formParam("name", "Shiny")
    .formParam("introduced", "2015-01-01")
    .formParam("discontinued", "")
    .formParam("company", "20")
    .check(status.is(session => 200
          + ThreadLocalRandom.current.nextInt( 2))))
}
```

5 Recompose Chains

5.1 Extract Chains

Having code let you isolate reusable parts and recompose them.

- For each Simulation, move scenarios content into `objects` that are located in the same file as their Simulation sibling

```
object Browse {
  val browse = exec...
}

class BrowseSimulation extends Simulation {
  val scn = scenario("scenario") // now empty
}
```

5.2 Re-attach Chains

- Use `exec to re-attach the chains to the scenarios`

```
class BrowseSimulation extends Simulation {
  val scn = scenario("scenario")
    .exec(Browse.browse)
}
```

5.3 Re-compose

- Create a new Simulation named "GlobalSimulation"
- Create a first scenario named "Users" that performs "Browse" and "Search" sequentially
- Create a second scenario named "Admins" that performs "Browse", "Search" and "Edit "sequentially
- Inject 10 users and 2 admins

5.3 Re-compose

Solution

Note: Such reuse is only possible because DSL components are **immutable builders**!

```
class GlobalSimulation extends Simulation {

  val users = scenario("Users")
          .exec(Browse.browse, Search.search)
  val admins = scenario("Admins")
          .exec(Browse.browse, Search.search, Edit.edit)

  setUp(users.inject(atOnceUsers(10)),
        admins.inject(atOnceUsers(2))).protocols(httpProtocol)
}
```

5.4 Inject

You have full control over the injection profile

You can either reason in terms of users, or in terms of users/second

- Instead of launching users all at once, have them arrive over 10 seconds

5.4 Inject

Solution

Note: Gatling emphasizes open workload models, i.e. you control the arrival rate, not the number of concurrent users in the system (closed model)

```
setUp(users.inject(rampUsers(10) over(10)),
        admins.inject(rampUsers(2) over(10)))
```

5.5 Assertions

Assertions let you define acceptance criteria.

Failing assertions can trigger CI build failure.

- Add assertions verifying that the global count of failed requests is 0 and that the response time "first percentile" is below 20ms.
- Run and observe results in the console output.

5.5 Assertions

Solution

```
setUp(???)
      .assertions(
          global.failedRequests.count.is(0),
          global.responseTime.percentile1.lt(20)
      )
```

5.6 Groups

Group of requests to model process or requests in a same page.

Groups can be nested.

- Add a group in each Chain.
- Run and open the report.

5.6 Groups

Solution

```
group("groupName") {
      //Chain
  }
```

Scala Basics 1

1 Getting Started

Objectives

- Use worksheet
- Type declaration and inference
- `val` vs `var`

1.1 Assignation

- Create a worksheet in IntelliJ named "basics"
- Add `import scala.concurrent.duration._`
- Type 5 seconds, see worksheet evaluation
- Store into a val named "duration", see type is inferred from right operand
- Select "duration", click on light bulb and click on "Add type annotation to value definition", see type is now explicit

1.1 Assignation

Solution

```scala
import scala.concurrent.duration._

val duration: FiniteDuration = 5 seconds
```

1.2 Re-assignation

- In a new line, try reassigning 3 seconds into "duration", see that vals can't be reassigned
- Change from `val` to `var`, see that it compiles and vars can be reassigned (yet, code smell)

1.2 Re-assignation

Solution

```scala
import scala.concurrent.duration._

var duration: FiniteDuration = 5 seconds
duration = 3 seconds
```

2 Methods and Functions

Objectives

- `def` syntax
- function syntax
- result of block's last operation is automatically returned

2.1 First Method

- Create a method named "addOne" that takes an `Int` and return this value + 1

```
def addOne(i: Int): Int = {
  i + 1
}
```

- see def keyword, parameter list and types, return type, body wrapped with {}, returned result with `return` keyword
- Execute it with different values

2.2 Shorter Syntax

- Remove braces as single body operation
- Remove explicit return type (warning: explicit return types are good for compilation time and API stability)

```
def addOne(i: Int) = i + 1
```

2.3 Function Syntax

- Turn this method into a function

```
val addOne: Int => Int = (i: Int) => i + 1
```

- Note function type: "from" input type "to" output type
- Note that a function can be stored in a reference, hence passed as a parameter or returned by another function

2.4 Type Inference

- Return type inferred from right operand:

```
val addOne = (i: Int) => i + 1
```

- Input parameter type can be inferred from result type:

```
val addOne: Int => Int = i => i + 1
```

- Shorter syntax with _ input parameter placeholder:

```
val addOne: Int => Int = _ + 1
```

3 Strings

Objectives

- multiline Strings
- stripMargin
- String interpolation

3.1 Multiline

- Create a multiline String by wrapping it with triple double quotes

```
val text = """line 1
line 2
line 3
"""
```

Note that inner double quotes no longer have to be escaped with a backslash

3.2 stripMargin

- Use `stripMargin` to have properly aligned multilines

```
val text =
  """line 1
    |line 2
    |line 3""".stripMargin
```

Beware `stripMargin` is a method, hence happens at runtime!

3.3 String interpolation

- Declare some values and inject them into placeholders

```
val string1 = "Hello"
val string2 = "Bar"
val text = s"$string1 $string2"
```

- See what happens if string2 is not defined

Note that String interpolation is a macro, hence String template parsing happens at compile time

Parsing JSON

- JsonPath
- Typing

1 JsonPath

- XPath for JSON
- Spec: vague and no TCK
- Many implementations (PHP, Javascript, Java) and online evaluators:
  - Most are very buggy
  - Only use https://jsonpath.herokuapp.com/
  - But don't trust Gatling perf results there

2 JsonPath Exercise

- Get familiar with JsonPath syntax
- Check http://jsonplaceholder.typicode.com/ APIs
- Write a simulation:
  - Check that the name of the first comment whose author's email is Eliseo@gardner.biz is equal to "id labore ex et quam laborum"

Hint: Use triple quotes so you don’t have to escape inner quotes

""""foo "bar" baz""""

2 JsonPath Exercise

Solution

```
exec(http("Get comment")
  .get("http://jsonplaceholder.typicode.com/comments" )
  .check(
    jsonPath("""$[?(@.email == "Eliseo@gardner.biz")].name""")
    .is("id labore ex et quam laborum" )))
```

3 ofType

- By default, jsonPath extracts Strings
- If you want other types, you have to say so explicitly

```
jsonPath(???).ofType[T]
```

4 ofType Exercise

- Using previous exercise
- Instead of fetching name, fetch post id and validate it's equal to 1

4 ofType Exercise

Solution

```
.check(
  jsonPath("""$[?(@.email == "Eliseo@gardner.biz")].postId""")
  .ofType[Int]
  .is(1)))
```

Scala Basics 2

1 Controls

Objectives

- if
- for
- expression orientation

1.1 for

- Create a new worksheet named "controls"
- Create a for loop that iterates from 0 to 9 and prints the current index

```
for (i <- 0 to 9) {
  println(i)
}
```

- Instead of printing, yield current index, see that the for loop actually returned a `Vector`

```
for (i <- 0 to 9) yield i
```

1.2 if

- Create a method that takes a number and returns an "even" or "odd" String

```
def evenOrOdd(number: Int): String =
  if (number % 2 == 0)
    "even"
  else
    "odd"
```

- See how result of each branch is returned

2 Collections

Objectives

- List
- Tuple
- Map
- immutable vs mutable Collections
- for comprehension

2.1 List

- Create a List containing 1 and 2, see List is automatically imported

```
val list = List(1, 2)
```

- Add an element to this list, see that new element is prepended and original instance is not modified (List is an immutable linked list)

```
val list = List(1, 2)
val list2 = 3 :: List(1, 2)
println(list2)
```

2.2 Tuples

- Create a pair of 2 numbers

```
val pair = (1, 2)
```

- Access elements by rank

```
val element1 = pair._1
val element2 = pair._2
```

- Access elements by projection (pattern matching)

```
val (element1, element2) = pair
```

2.3 Map

- Create a Map from (key, value) pairs where keys are Strings and values are Ints

```
val map = Map(("a", 1), ("b", 2))
```

- Use "->" syntactic sugar to create pairs

```
val map = Map("a" -> 1, "b" -> 2)
```

2.4 Mutable Collections

- Only immutable collections are imported by default
- Create a mutable ArrayBuffer containing 1 and 2 and add 3, see it's similar to Java's ArrayList

```
val arrayBuffer = collection.mutable.ArrayBuffer(1, 2)
arrayBuffer += 3
println(arrayBuffer)
```

2.5 Looping

2.5.1 With side effect only

- Iterate over List instance and print all elements

```
for (i <- List(1, 2)) println(i)
```

- Have 2 embedded iterations and print products

```
for {
  i <- List(1, 2)
  j <- List(3, 4)
} println(i * j)
```

- Only loop over first List's even values

```
for {
  i <- List(1, 2) if i % 2 == 0
  j <- List(3, 4)
} println(i * j)
```

2.5 Looping

2.5.2 Yielding a result

- Return in a new List with the results

```scala
val results =
  for {
    i <- List(1, 2) if i % 2 == 0
    j <- List(3, 4)
  } yield i * j
println(results)
```

2.5 Looping

- without yield: side-effecting only
- yield: project results into a new container
- can embed multiple loops
- looped containers must be of same type
- result is of same same type as input

3 Option

Objectives

- Option[T] is a generic container
- Abstraction for data that could be missing
- Some, None implementations
- Looping

3.1 Option implementations

- Create an Option containing "foo", see actual type

```
val opt1 = Option("foo")
```

- Create an Option from null, see actual type

```
val i: String = null
val opt2 = Option(i)
```

3.2 Looping

- Option is just a collection with 0 or 1 element
- Use for loop

```
for {
  opt1 <- Some("foo")
} println(opt1)
```

3.3 Exercise

- Create a "concat" method that takes 2 Option[String] and concatenate their content when both exist
- Test the different possibilities

```
def concat(opt1: Option[String],
           opt2: Option[String]): Option[String] = ???

concat(Some("foo"), Some("bar")) // Some("foobar")
concat(Some("foo"), None)        // None
concat(None, Some("bar"))        // None
concat(None, None)               // None
```

3.3 Exercise

Solution

```
def concat(opt1: Option[String],
           opt2: Option[String]): Option[String] =
  for {
    value1 <- opt1
    value2 <- opt2
  } yield value1 + value2
```

Session

- Virtual User memory space
- ~ Map[String, Any]
- Immutable (case class)
- Safe fetching (missing and wrong type)

1 Introspecting

- Create a very basic Simulation with one empty scenario
- Add `props.noReports()` to Engine so it doesn't complain about no requests
- Add an `exec` Action that takes a `(Session => Session)` function
- Use it to print the session in the console
- Run the Simulation with 2 users

1 Introspecting

Solution

```
.exec { mySession =>
  println(mySession)
  mySession
}
```

Note: We usually use curly braces instead of parens for passing functions over multiple lines

2 Adding data

- Add an attribute named "foo" whose value is "bar"
- Run the Simulation

```
.exec { mySession =>
  mySession.set("foo", "bar")
  println(mySession)
  mySession
}
```

- Figure out what happens and fix

2 Adding data

Solution

Session is immutable!

```
.exec { mySession =>
  val newSession = mySession.set("foo", "bar")
  println(newSession)
  newSession
}
```

3 Fetching Data

3.1 Expression Language

- Hold Ctrl (Cmd for Mac) and mouse over the `get` method of a request
- See that method takes an `Expression[String]` not a String
- Click on the `get` method to go to the sources
- Figure out what `Expression[String]` is
- Figure out how we can pass a String instead

3.1 Expression Language

- Select the content of what's passed to `get`
- Click on Ctrl+Shift+Q (Ctrl+Q for Mac) and see applied implicit
- Click on light bulb and "Provide implicit conversion"

3.1 Expression Language

- Does this work?

```
.exec { mySession =>
  val newSession = mySession.set("baz", "${foo} qix")
  println(newSession)
  newSession
}
```

3.1 Expression Language

- Gatling DSL components actually expect function parameters
- Strings gets implicitly parsed and turned into functions
- Parsing usually happens only once, when String is passed to the DSL
- Conversion only happen because types don't match
- There's no magic!

3.2 Manually with `as[T]`

- Add a ("foo", "bar") attribute
- Print "foo"'s value

3.2 Manually with `as[T]`

Solution

```scala
.exec { mySession =>
  val newSession = mySession.set("foo", "bar")
  println(newSession("foo").as[String])
  newSession
}
```

3.2 Manually with `as[T]`

- What happens if the attribute is undefined?

```
.exec { mySession =>
  val newSession = mySession.set("foo", "bar")
  println(newSession("BAZ").as[String])
  newSession
}
```

3.2 Manually with `as[T]`

- What happens if we want an Int but the attribute is a String?

```scala
.exec { mySession =>
  val newSession = mySession.set("foo", "bar")
  println(newSession("foo").as[Int] + 1)
  newSession
}
```

3.3 Manually with `asOption[T]`

- Replace `as` with `asOption`
- Test again when data is missing or of a different type

```
.exec { mySession =>
  val newSession = mySession.set("foo", "bar")
  println(newSession("BAZ").asOption[String])
  println(newSession("foo").asOption[Int])
  newSession
}
```

3.4 Manually with `validate[T]`

- Replace as with `validate`
- Test again when data is missing or of a different type

```
.exec { mySession =>
  val newSession = mySession.set("foo", "bar")
  println(newSession("BAZ").validate[String])
  println(newSession("foo").validate[Int])
  newSession
}
```

3.5 Exercise

- Set 2 attributes with Int values in the Session
- Compute the sum of them and add it as a new attribute
- Use "as" then "validate"
- Hint: use for loops

3.4 Exercise

Solution with ”as”

```
.exec { mySession =>
  val newSession =
          mySession.setAll("foo" -> 2, "bar" -> 3)
  val foo = newSession("foo").as[Int]
  val bar = newSession("bar").as[Int]
  newSession.set("baz", foo + bar)
}
```

3.4 Exercise

Solution with "validate"

```scala
.exec { mySession =>
  val newSession =
          mySession.setAll("foo" -> 2, "bar" -> 3)
  for {
   foo <- newSession("foo").validate[Int]
   bar <- newSession("bar").validate[Int]
  } yield newSession.set("baz", foo + bar)
}
```

4 Custom Feeders

4.1 Understanding Feeder type

Feeder is a type alias:

```
type Record[T] = Map[String, T]

type Feeder[T] = Iterator[Record[T]]
```

In short: a key-value pairs generator

```
Iterator[Map[String, Any]]
```

4 Custom Feeders

4.2 Exercise

- Generate a random email feeder
- Use the pattern user<number>@gatling.io
- Print the generated assigned emails in the console

Hint: check Iterator factories

Hint: use ”map” method to convert values

4 Custom Feeders

4.3 Solution

```scala
val feeder = Iterator.from(1)
           .map(i => Map("email" -> s"user$i@gatling.io"))

feed(feeder)
.exec { mySession =>
  println(mySession("email").as[String])
  mySession
}
```

Crafting Request Bodies

- Using EL
- Programmatic Templating

1 Using EL

- Template as Scala String: StringBody
- Template as external file: ElFileBody

```
.exec(http(???)
  .put(???)
  .body(StringBody("""{ "foo": "bar" }"""))
  )
```

2 Exercise

- Patch comment with id 1: Prefix name with FOO
- Use a StringBody, then an ElFileBody

Hint: first, save current name

Hint: don’t forget Content-Type header

2 Exercise

Solution

```
.exec(http("get")
  .get("http://jsonplaceholder.typicode.com/comments/1" )
  .check(jsonPath("$.name").saveAs("commentName"))
)
.exec(http("patch")
  .patch("http://jsonplaceholder.typicode.com/comments/1" )
  .header("Content-Type", "application/json")
  .body(StringBody("""{"name": "FOO ${commentName}"}""" ))
)
```

3 Programmatic Templating

- What if data needs prior processing?
- Gatling EL is convenient yet limited

3 Programmatic Templating

Work-around: pre-generate process data

- Use a prior action
- Fetch session data
- Store new session data
- Use regular EL template

4 Exercise

- Patch comment 1 with the reversed name
- Use "as" then "validate"
- Hint: use IDE auto-completion to find out how to reverse a String

4 Exercise

```
.exec(http("get")
  .get("http://jsonplaceholder.typicode.com/comments/1" )
  .check(jsonPath("$.name").saveAs("commentName"))
)
.exec { session =>
  ???
}
.exec(http("patch")
  .patch("http://jsonplaceholder.typicode.com/comments/1" )
  .header("Content-Type", "application/json")
  .body(StringBody("""{"name": "${reversedCommentName}"}"""))
)
```

4 Exercise

Solution with ”as”

```
.exec { mySession =>
  val commentName = mySession("commentName").as[String]
  mySession.set("reversedCommentName", commentName.reverse)
}
```

4 Exercise

Solution with ”validate”

```
.exec { mySession =>
  for {
      commentName <- mySession("commentName").validate[String]
  } yield mySession.set("reversedCommentName", commentName.reverse)
}
```

5 Programmatic Templating

- Problems with workaround:
  - Extra step
  - Session pollution
- Directly pass a function to StringBody

6 Exercise

- Achieve previous results without an intermediate step
- Hint: create a val
- Hint: use Scala String interpolation for concatenating

6 Exercise

Solution with ”as”

```scala
val patchTemplate: Expression[String] = ???
```

```scala
.exec(http("get")
  .get("http://jsonplaceholder.typicode.com/comments/1" )
  .check(jsonPath("$.name").saveAs("commentName"))
)
.exec(http("patch")
  .patch("http://jsonplaceholder.typicode.com/comments/1" )
  .header("Content-Type", "application/json")
  .body(StringBody(patchTemplate))
)
```

6 Exercise

Solution with ”as”

```
val patchTemplate: Expression[String] = session => {
  val commentName = session("commentName").as[String]
  s"""{"name": "${commentName.reverse}"}"""
}
```

6 Exercise

Solution with ”validate”

```
val patchTemplate: Expression[String] = session =>
  for {
    commentName <- session("commentName").validate[String]
  } yield s"""{"name": "${commentName.reverse}"}"""
```

Automating

- Using Jenkins with the Gatling Plugin

Purpose

- Automate runs
- Fail on defined threshold
- Get notified

Principles

- Fetch from a Git repository
- Build with Maven
- Run with Gatling
- Handle with Gatling Jenkins' Plugin

How

- We'll use an already configured instance
- Content Security Policy disabled

Preparation

- Source repository
- Fork the repository
- Keep it public
- Add some assertions
- For some inspiration, look at the assertions branch

Logging into the demo server

- http://35.156.73.41
- Login: gatling-training
- Password: training

Configuring a new job

- From the home page: New Item
- Freestyle project
- Use your name as a job name

Source Code Management

- Check Git
- Choose the right repository URL
- Don't use any credentials

Build Triggers

- This is where the automation takes place
- Untick everything for now

Build

- Choose "Invoke top-level Maven targets"
- Set Goals to:

```
gatling:execute -Dgatling.useOldJenkinsJUnitSupport=true -Dgatling.simul
```

Build Settings

- Notifications

Post-build Actions

- Add "Publish JUnit test result report"
- Set "Test report XMLS" to "target/gatling/assertions-*.xml"
- Add "Track a Gatling load simulation"
- Make sure "Enable simulation tracking" is ticked

Running the job

- First save the job
- Click the "Build Now" link on the left

Going further

- When to run the tests?
- Which server should runs the tests?

Scala Basics 3

1 `map` method

- Create a new List instance with all values translated by 1

```
val list = List(1, 2).map(i => i + 1)
```

- Create a List of List[Int] where the latter contains i and i + 1

```
val list = List(1, 2).map(i => List(i, i + 1))
```

- `map` takes a function for converting values into other values and return a new container of the original type

2 `flatMap` method

- Create a List[Int] instead

```
val list = List(1, 2).flatMap(i => List(i, i + 1))
```

- `map` takes a function for converting values into containers and return a new container of the original type

3 `filter` method

- Only keep even values from List(1, 2)

```
val list = List(1, 2).filter(_ % 2 == 0)
```

4 `foreach` method

- Print each collection elements (side-effect)

```
List(1, 2).foreach(println)
```

5 for comprehension implementation

- for loops are actually a syntactic sugar!
- Implement the code below with `map`, `flatMap` and such

```
for {
  i <- List(1, 2)
  if i % 2 == 0
  j <- List(3, 4)
} println(i * j)
```

5 for comprehension implementation

Solution

```
val list1 = List(1, 2)
val list2 = List(3, 4)

list1
  .filter(_ % 2 == 0)
  .flatMap { i =>
    list2.map(j => i * j)
  }.foreach(println)
```