Item 184
git_comments:

git_commits:

1. **summary:** Add more tests for GROOVY-9637
   message: Add more tests for GROOVY-9637
   label: test

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Improve the performance of GString
   description: {{GString}} will run {{toString()}} whenever its literal string is needed, e.g. methods like {{equals}}, {{hashCode}} are called, but unfortunately its literal string will be re-constructed for each time and the process costs quite a little of time. So I propose to check whether {{GString}} values are all of immutable type, e.g. primitive types and their boxed type, {{String}}, etc. If yes, use the cached string literal, otherwise re-construct the literal string. Here is the test script, Groovy 3.0.5 costs about {{8700ms}}, and PR1329 costs about {{290ms}} on my machine. About {{96.7%}} time is reduced. {code:groovy} def gstr = makeGString() long b = System.currentTimeMillis() for (int i = 0; i < 10000000; i++) { gstr.toString() } long e = System.currentTimeMillis() println "${e - b}ms" //@groovy.transform.CompileStatic def makeGString() { def now = java.time.LocalDateTime.now() "integer: ${1}, double: ${1.2d}, string: ${'x'}, class: ${Map.class}, boolean: ${true}, now: ${now}" } {code}

jira_issues_comments:

1. **body:** Since GStrings aren't themselves immutable, we probably have more work to do. E.g. while bad style and strongly discouraged, we don't disallow code like below: {code} def x = 42G def y = "Answer is $x" println y // => Answer is 42 y.strings[0] = '6 x 7 = ' println y // => 6 x 7 = 42 y.values[0] = 'the question' println y // => 6 x 7 = the question {code} We could place in a change to make them immutable. That might have been a good candidate change for Groovy 3 but perhaps too late now unless discussed and agreed on mailing lists. Seems a better fit for Groovy 4? Also, the PR doesn't taken into account if a metaclass is in play. Due to GROOVY-2599, this would take some work to be an issue (i.e. extremely rare). E.g.: {code} def x = 42G def y = "Answer is $x" def z = "Answer is ${x.toString()}" println y println z BigInteger.metaClass.toString { '' + delegate * 2 } println 10G println 10G.toString() println y println z {code} Due to GROOVY-2599, Groovy's InvokerHelper.format also ignores if a metaClass is in play for some hard-coded cases. But if we ever fixed GROOVY-2599, the behavior seen by the closure variant would also be seen by the non-closure variant. Which means the GString patch would turn that off. We could argue that isn't a big issue. Or turn caching on only in CompileStatic code? Perhaps worth discussing in mailing lists regardless. Others might have better suggestions. The idea is certainly nice.
   label: code-design
2. If we return a copy of the array of strings/values, users can not change them as follows: {code:java} public Object[] getValues() { return values.clone(); } {code} {code:groovy} def x = 42G def y = "Answer is $x" y.strings[0] = '6 x 7 = ' y.values[0] = 'the question' {code} As to {{toString}} behavior changed by MOP, it is hard to make the constructed string literal constant unless STC is enabled.
3. Just swapped to the more usual label we have been using - easier when we create release notes.
4. The proposed PR is merged. Thanks!
5. Removed the breaking label since while caching is enabled by default for applicable values, it automatically turns off if the internal data structures are accessed at all unless an explicit freeze is called.