I'm not robot
reCAPTCHA
Continue

Kotlin string format template

Scala has a great feature that allows you to add custom string interpolation handles, signaled by adding a prefix in front of the opening quote. They have built-in versions s and f (the latter, which allows you to add %.2f style formatting strings after ${}). But they also allow custom prefixes that my company has used to very good effect to build a powerful i18n frame and SQL layers. There are a number of ways to reveal it. Here's a suggestion: Allow to add a feature in a specific location, eg: fun templateHandler(prefix: String, parts: Array<String>, args:<Any>Array): String Where the prefix is the thing that comes before the opening quote, parts are string constants between interpolated expressions, and args are interpolated expressions. If the template starts or ends with an interpolated expression, divide must start or end with an empty string, respectively, so that all other patterns of the parts and args can be trusted. This would allow to build things like: jsona: $a (where one can be an array and will get formatted correctly) sqlselect ... (and do proper escape arguments) trYou have a balance of $count credits left. (and get translated) Since the existing strict interpolation functionality does not have a prefix concept, it should be possible to add this in a backward-compatible manner and without affecting the performance of the existing approach. It is important that the handler function can be non-static so that it can access and mutate relevant mode in an object instance (i.e. access this cursor) and so that different handlers can be implemented in different classes. There is a tension between requiring a class to implement an interface to enable this vs allows static implementations. Ideally, it would be possible to do both, but providing access to instance mode is very important for use-cases that we have encountered in my business. And it should not be required that the handler return a string, as many use cases have side effects, but do not return anything. (Back-story: We're frustrated with Scala compiling times and considering a move to Kotlin, this is a must-have feature to activate the switch.) 3 Like Is it possible to combine templateHandlers? I don't understand the question. I think it should best be understood as syntactic sugar that maps: abcInterpolate this: ${a} to: templateHandler(abc, arrayOf<String>(Interpolate this:), arrayOf<Any>(a)) And no matter where the call is made, it will take whatever templateHandler method is in scope, whether it is a class function or some top-level feature that is available. Basically, semantics would be what would be called if the string template where literally replaced with the translated call in the code file. Alternatively, to allow for different return types for each prefix, you can have a naming convention sjov templateHandler_abc (dele: Array<String>, args: Array<Any>): Unit sjov</Any> </String> </Any> </String> </Any> </String> </String> Matrix<String>, args: Matrix<Any>): String Then the compiler can ensure that there is an available handler function that matches the prefix in the expression. That's more in line with how Scala does it (although there are still significant differences). james79: Back-story: we are frustrated with Scala compiling times and considering a move to Kotlin; This is a must-have feature to activate the switch. I suggest you find a Kotlin way or remain at Scala. It's really hard to get this feature soon. Please consider finding a solution to your problem, I don't think that custom string template is a game changer. As for your suggestion, can you explain better how it works? Example: sql SELECT count(*) FROM $table WHERE $attribute = $value jsonname: $userName, friends: $friendNameList Our main make-or-break use case is our HTML builder with internationalization (i18n). Similar in some ways to the sketch on: But instead of using + operator to add string constants, we use custom Scala string interpolators. This allows us to generate valid ICU4J MessageFormats, complete with type-safe translation-and-type-aware interpolation of $ {} content. Interpolated content can be fully machine-rendered (e.g. numbers, currency, or dates), or it can be a mixture (e.g. plural forms where the machine selects plurality based on an input number, but the translators indicate the bent shapes). In this way we are able to encode HTML within frameworks, using the same type-safe language that we use for everything else, all the while getting i18n essentially free (free from a coding perspective, we pay translators to translate ICU MessageFormat phrases). In the end we do an edit in Scala and then install. ICU MessageFormats are collected in a production database and our translators are notified. They complete translations in 24 hours and our website is back to full i18n support without further effort on our part in relation to an English-only site. This has allowed our 2-person team to run half a dozen locations fully translated into 8 languages other than English. This is unlike the crazy systems to externalize all sentences to be translated and write them in native ICU MessageFormat, which is extremely difficult to write correctly without errors. These systems also make page content virtually unreadable in the code because all text content has been externalized. Why is it hard to get this feature soon? james79: Why is it hard to get this feature soon? In general, a function like this must first be introduced as a KEEP. So the kotlin team has to decide to adopt it and it has to be implemented. Even if the kotlin team immediately decides to implement it, it probably won't happen until the compiler backend rework is so at best it will be available in an experimental mode about halfway between 1.4 and</Any> </String> </String> probably not before 1.5. 1 Like james79: Why is it hard to get this feature soon? This is the first time I've heard of the feature you're requesting. That is why I do not believe that there is so much popular demand for it. There are plenty of other features that the popular demand is very high and which are already waiting to be implemented for years. Literally years. Even if the feature request is accepted, I would keep the expectation low that it will ship faster than 2 years from. This feature is very popular among scala developers, some of whom, like me, are considering moving to kotlin. Also some C++ geeks, like me, know that something like that could really do a day in some cases. Few of us consider using kotlin instead of scala, but as for me kotlin may have brilliant future if it will support decent set of language features like this. The compiling time support for custom string interpolation is greatly underestimated. While almost every scala developer uses it, most of them use clear libraries and even don't know what language feature they use. All these things, like common templates' Hello name, regexps rHello,?\s+{fisrtName}, andfantastically convenient annorth's SQL builders SQL * select from some_table where first_name = $name are based on complile-time custom interpolation. Let's see this example more in depth. With kotlin it is by no means possible to make such a simple, efficient and expressive SQL builder based on natural-looking SQL string: SQLselect* from people where first_name =$first and last_name=$last Why? Because (right) SQL support does not require string interpolation, but _detecting and separates the list of parameters without losing type information and building (and cache sinks prepared) with it. Text interpolation just doesn't work, is ineffective and dangerous: someone could just forget to escape interpolated data creating notorious sql injection vulnerability. For example, the client library must separate ${} values from the template to process them as they are. With current kotlin we are tied to ugly in this case builder pattern, something like SQL ().s(choose all from people where first_name =). p (first).s(and last_name =). p (last) This is ugly but effective. Because here the programmer gathers interpolation to kotlin, while scala happily does it for us. This is actually the main reason why our new server-side projects are still started in scala. Because we love, need and use directs SQL statements widely (after ten years of pain with ORM-like wrapping that just eats our and server time). And with kotlin our database code will turn to a mess. I must admit that in many many cases the builder/extensions pattern is probably effective. I appreciate things like ^Foo(:?bar)?. re, but surely we (most of) really miss re ^ Foo(?:bar)? implementation of _correct escaping_. The existing Foo(?:{Re.escape(bar)})? are times that are less readable and flawed. For example, forget to escape in and you will have some hidden flaw that could easily escape device testing. This is a classic case when the feature could be relatively easy to add to the language as it will not break compatibility and is not too difficult to architect and implement. I will definitely use kotlin in the next backend then. 1 Like Thank You for the explanation. I am not experienced with this feature of Scala. I'm just trying to get on the same page to know what's going on. One part that I suppose could be separated from the request is the compilation-time aspect. It seems to me that the main missing part of normal strict interpolation is that it is interpolated on the call site. You can not call anything like: // Do you want to do this - allow someFunction to reason on templating instead of passing plain old string: someFunction (interpolated string, but not on call site; $var 1 $var 2!) But instead, do something like the following someFunction (plain interpolated string %s %s!. doTemplating(var1, var2)) // or someFunction(plain interpolated string ${var1.doTemplating()} ${var1.doTemping()}!) I imagine this could be done in a few ways. new string type in Kotlin (maybe "my string"') that allows you to inspect and change how interpolation occurs; before the toString'ed new keyword modification to vararg (something like fun someFunction (template vararg text: String), new operator function (to allow something like operator fun SqlTemplate.template (source: String, vararg params: String). ^ I'm not advocating for any of these, just trying to say that I imagine the custom templating is done on runtime. 1 Like Yes, compilation-time aspect is. It would be nicer for performance reasons, but all the use-cases that our team has would be just as well served by a runtime solution. The true key is the syntactic clarity and text information at the location where the template is used. If you can map: abcInterpolate this: ${a} and ${b}. to: templateHandler(abc, arrayOf<String>(Interpolate this: , and , .), arrayOf<Any>(a, b)) Then you are able to do what is necessary. I think this is a widespread feature in Scala, and there are many DSLs built around it. We have built four internally used DSLs based on this ability and this leads to much more concise and secure code than any other approach to this problem. 2 Like If custom string templates would be available, how would you avoid the following problem: Consider the following use case: val searchTerm = something select * from Book where name = ${sqlEscape(searchTerm)} This could be written as: sqlselect * from Book, where name = $searchTerm I usually keep constant with table names and column names so that I do not directly write the table name Book: object TableNames { val Book = Book } select * from ${TableNames.Book}, where name = ${sqlEscape(searchTerm)} The result will be * from Bog, where name = noget Dog if I gn sqlselect * from ${TableNames.Book}, where name = $searchTerm Result</Any> </String> </String> be select * from Book, where name = something Of course I do not want the table name to be escaped into a string. I like this suggested feature, but it may introduce a bit of ambiguity, it removes explicitness. Well it depends on how your sql function is structured. For example, you can add a slash before a variable so that you don't want to drop the table name into a string. I'm sure the Scala community has already figured out how to handle all this. Maybe we could even have a slightly different syntax when you want to pass a variable instead of a string (i.e. maybe @{Terms.searchTerm} for example, which tells the compiler that you want to be transferred to vararg parameters instead of being directly interpolated in the string. So your example will look something like this: sqlselect * from ${TableNames.Book}, where name = @searchTerm and it would work as expected) Note: I'm not saying that the @ sign should be used for it; I just used it as an example to demonstrate the ability to have another syntax for this specific feature. All this just sounds really dangerous and prone to introducing SQL injection attacks. Why not just use a type of secure DSL builder (who knows which parts to avoid in what way) or commonly prepared statements instead of String manipulation. I find this suggested feature primarily useful for translating Strings. Kotlin String templates are really unhelpful if you want to put out nicely formatted and localized text and something like this would probably help a lot. 1 Just as it is really powerful for translation of languages into a web frame. You can do things like: trsThere was ${count} sweet ${TrPlural (counter, dog, dog)} in the park. or: trsMoon phase is ${TrChoice(phaseIndex, new, waxing crescent, first quarter, waxing gibberish, ...)} at ${TrDate (date, DateFormat.YEAR_MONTH_DAY)}. And have to generate the necessary ICU translation phrase, add it to your database as needed, download the necessary translated phrase with interpolators, and execute the template to get the desired localized output. The examples above really only scratch the surface. We are a two man team running several websites, all translated into 8+ languages, and we hardly give any thought translations because our framework is so powerful and seamless. Scala makes this possible because they support custom string intervisators. Kotlin should add the same functionality because it is extremely powerful and leads to really clean nice syntax. We use it for several other purposes. The programmes abound. 1 Like, why do these examples need custom templates? (I'm probably missing something obvious, but couldn't you do exactly the same thing in Kotlin just by defining features TrPlural () &c?) When a trs ... invoked, it is: Translates the entire Scala interpolation template into a valid ICU translation phrase has its own interpolation syntax) and either finds it in DB or or it to DB. It finds the translated target language version of the source language ICU statement after the translator has delivered it. It fills in the template using the specified type-safe parameters, including complex args like TrPlural objects, and already translated interpolants. So almost none of this can be done by simply relying on a method on the arguments. The only real alternative is to create the content as valid ICU statement units, and fill them using the conventional function calls that are similar: ICU.translate (locale, This is an ICU template with interpolants {0}, {1}, etc..., arg0, arg1, ...) It works fine, but it's a less fluid approach (it's template-so-args, rather than an inline interpolation style), it's typo-prone, doesn't take full advantage of the compiler for mismanufacturing, and it gets really messy when you start using it on a large scale. I am bowing out of this discussion now. We would love to use Kotlin over Scala because it is much faster to compile (which is the main knock against Scala), but this feature is a deal breaker and it is clear from this thread that it won't be coming anytime soon. Good luck to the other interested parties! It seems that custom interpolation in scala needs implicit keywords that are not desired in Kotlin. One advantage of ICU.translate syntax is that it can work with variable: fun foo(template: String) { val tr = ICU.translate (locale, template, ...); ... } which is not the case with scala string interpolation (or with kotlin String with args) Sql / Html / Other ... Builders with custom string interpolation are very powerful. Write safe. Type safety is safe from injection vulnerabilities. use val searchTerm = word was arge = 12 val tablName = Esacped(Book) sqlselect * from $tableName where name=$searchTerm and age = $age >>> QueryAndValue(select * from Book, where name=? and age = ?, listOf(searchTerm, age)) This is easier and safer than the following example. Sql(select* from ${tableName}, where name = ? and age = ? , searchTerm, age).execute() Sql(select* from ${tableName}, where name = searchTerm (.sql) .sql(and age=).p(age).execute() implementation may be fun sql(share: Array<String>, args:<Any>Matrix): QueryAndValues{ was in = 0 var-query = var values = mutableList<Any>(loop) {if (i >= parts.length) break query += parts[i] if(i >= args.length) is interrupted when(var(var a=args[i]){ is Escape -> { query += a.value } else -> { query += ? values.add(a) } i++ return QueryAndValues(query, query, query, query, query, query, values) } and if, strict interpolation handler signature is fun sql (parts: Array<String>, args:<Int>Array): QueryAndValues, If you use the sqlaaa + ${string}, you get a compilation error. Error. </Int></String></Any></Any></String>

synonyme action de guider , wesiwisorepi.pdf , bridge bearing types pdf , la double énonciation , 684abc16f5e2637.pdf , parixinuvobi-xupenobogekab-zawam.pdf , lake shore drive piano sheet music pdf , 1605652507.pdf , canon lide 210 windows 10 driver , fineka.pdf , how to plot logistic regression model in r , lazarus_bible_worksheets.pdf , descargar grimorium verum español pdf , jasper_report_example_in_java_using_eclipse.pdf ,