Requirements for a new development language

Goal

We need to choose a new development language for future development for server side code on MusicBrainz. This document outlines what requirements we have for choosing a new language.

Anti-goal

We are not actually discussing any languages in this document. We are only discussing desired attributes of languages. **This document shall not name any languages**, save for Perl since that is the language we're moving away from.

Open questions

Do we need to choose more than one language?

- We already have three languages in production. Ideally we should not add any more languages. In any case, we need to settle on as few languages as possible to get our job done.
- If we choose one language one or two major codebases need to be re-written. And our core toolkit (Lucene) is strongly tied to its native language. If we choose one language we need to replace Lucene or move everything to its native language. Both of these options are a huge amount of work that is not strictly necessary.
- We should choose appropriate DSL's (e.g. templating language) where they make sense. Also supplemental languages and standards (shell, HTML, SQL) shall not be considered primary languages for this discussion.

Roadmap

We should avoid a NGS-style large make-over of our code. We should aim to incrementally replace or rewrite our code base to be less tightly coupled and more homogenous. This means needing to support perl initially, but with the goal in mind that eventually all perl code will go away.

Concerns

Writing a new DAL in the new language means that we need to interface an existing body of Perl code to it. To tackle that, we could:

1. Create the DAL as a library **and** create a private web service from this library which allows the existing Perl code to interface to it. This is more work, but less soul sucking than #3. (preferred approach)

2. Create the DAL as a web service, in which case DAL access from Perl becomes an HTTP call. My preferred option, seems simplest, and gives head-start towards ws/3
3. Create the DAL as a library, in which case we need to create an interface layer between the new language and Perl. In reality this will be **2** interface layers, since C will be the common language; this C interface layer will need to interface between perl and the new language. Writing these interface layers is time consuming, soul sucking work.

Current language issues

We currently use Perl on the server and for various reasons we're sick of it. Lets quantify the pros and cons about Perl:

Pro

1. Established language with many libraries that support it (CPAN).

Con

1. CPAN. CPAN has a lot of complications that we need to work around.
2. Community participation in Perl 5 is declining and Perl 6 isn't ready for prime-time.
3. While it is possible to build large web-sites with it, there are better languages for web development.
4. Perl syntax without a style guide leaves a lot to be desired.
5. Lacks powerful compilation/linting tools
6. Errors are hard to decipher
7. Reference counting makes memory management a chore (we've given up on this, and just expect to have to restart processes).

New language requirements

The new language we choose should satisfy the following requirements:

1. It should be actively developed and have a strong community participation.
2. Should have a rich standard library and a rich ecosystem of available modules and a reasonable expectation of finding the best tools for most tasks from within that ecosystem. Ideally these third party libraries should be managed in some standard fashion that allows for ease of installation and stability.
3. It should be sufficiently mainstream in order to not preclude new contributors to MusicBrainz. Ideally it should be in the top 10 languages on at least 3 of the net's most popular programming languages pages (github, ohloh, google code).
4. If should be suitable for large scale web application development. It should support building large web sites that can scale appropriately.
5. The language itself should be open enough to make the team happy.
6. A project system that allows lower *and* upper bounds of versions of third party dependencies to be specified and that doesn't purge old versions, making our life hell.
7. The ability to distribute/deploy without requiring all dependencies installed first.

8. Fast compilation/application startup. I want to be able to make changes and verify them quickly.
9. A decent test system with support for running individual tests without having to build this myself.
10. I want a build system, a profiler, a code coverage tool and good memory usage inspection. (removed “good” from this, since that is subjective)
11. It should provide enough benefit above Perl to be worth the significant overall investment we’ll necessarily spend rewriting things, beyond the benefits of careful refactoring of our existent code. (i.e.: we should choose a language that will provide benefit in more ways than simply allowing us an opportunity to be more strict with ourselves)
12. team fluency

Discussion

ocharles wants...

- A language that one member of the ‘core’ team feels they are proficient enough that they could produce production ready applications. +1
    - I feel like I should qualify this a bit more. So this means:
        - Understands the language syntax fluently.
        - Understands the standard library/recommended packages well. Having to find documentation is fine, as long as you know *what* you’re looking for.
        - Understands the languages warts, and knows what to avoid and what to be cautious of.
        - Understands how to build/package in a standard manner compatible with the rest of that language’s eco-system.
        - Preferably actually has work they can show that interacts with databases and renders web pages.
- Something that can be run without compilation for one-off things like shell scripts, as I think we should start making our shell scripts more polished and distributing proper executables. (Think `musicbrainz-replication import packet.tar.gz` or `musicbrainz run-job subscription-emails`) +1 +1, noting that I’m not sure I want real compilation at all
- Garbage collection
- Ideally, strongly typed.
    a. I don’t agree with this as a core requirement. We may care about the potential benefits of strong typing, but those benefits are what to list here, not this specific language feature.
- In terms of language features:
    - Guarantees that if I change a method or variable name without updating references, I will be told off *before* any code is executed. Unit tests are not the answer to this.
        i. This excludes most popular dynamically (duck)-typed interpreted

languages, although some do have “strict” modes that warn/error at parse time.

ii. I personally am OK with this, but then I have lost quite a bit of faith in dynamic languages :)

- Higher order functions are a must. +1 +1, but these are getting more and more common
- Defining functions outside classes is quite important for code reuse (so preferably not a everything-is-an-object language). +1
  - I think I disagree and don’t see how exactly having functions outside classes helps for code reuse. +1
  - I agree with the conclusion (“not everything-is-an-object”) but I’m less sure about the reasoning and I think some ways of everything being an object aren’t so terrible (Perl, for example, doesn’t bother me here)
- An expressive type/object system with support for sum types would be nice. This is sort of like enumerated types (Color is red OR green OR blue), but a little more complicated. Also a bit like a union in C, but less painful.
  - I’d appreciate some further reading on sum types; at present I don’t know enough to comment on this suggestion.
    - It’s hard to summarize succinctly, but sum types let you model a type based on various different ‘states’. For example, a tree is either a leaf, or it’s a node with a list of sub trees. Having these in the language ensures that you deal with all the cases when you work with that type. For example, the ‘Maybe’ type is a good example of a simple sum type, as is Either. This want might be a little too specific.
    - There is a Wikipedia article on “Tagged union”.

- The ability to write it in Emacs productively. Agree you shouldn’t be forced to use an ide, and certainly not a particular ide but i really query emacs being the most productive way to write code
- It must have a REPL that I can interact with in order to rapidly iterate on small new features, and inspect existing functions/methods when doing bug checking.
- It must have a debugger that I can add breakpoints to, single step, and inspect variables.
- Must not tie me to an IDE.

In no particular order, and some options more than the others (they are not equally weighted).

A few more ideas from ijabz

1. Ideally combine strengths of classic compiled robust languages, and quick to prototype scripting/dynamic languages.
2. Should work similarly on different platforms , especially considering likely move to AWS
3. Code should be self documenting, should be able to see what the code is meant do

without having to read the code itself.
    a. I agree, but I think this is much more related to being strict with ourselves than the language choice.

Contributors

ruaok writes in black, like a boss!
ianmcorvidae chose a darker, more conservative purple color this time :P
ocharles has chosen his favourite blue
warp has chosen the colour of the royal family >_<
murdos just picked green because it was available
kepstin is picking... dark reddish but changed it to pink because it was too similar to others.
chirlu has chosen cyan, which at least is sufficiently different from the others
ijabz has gone for a murky brown.