Dart REPL PoC - possible directions for a real thing (brain-dump)

PoC architecture

Dart_repl copies the template files into a temp directory and spawns a sandbox isolate
repl connects to its own VM service. It uses dart:mirrors to find the execution environment, so the code does not have to import the environment directly. Instead for adhoc imports, a simple custom_repl_runner.dart is generated that imports everything and passes control to the actual repl code.

IPython integration

IPython offers a way to add support for adding support for languages other than Python by implementing the kernel interface and registering it with it. Doing that would make Dart work in IPython and Jypiter out of the box, which would be very cool.

IPython uses ZeroMQ, which does not have a Dart language binding (yet).
Ways to do that:

- write a ZeroMQ binding in Dart
- write a proxy from ZeroMQ to JsonRPC in Python or similar (or another RPC mechanism that has bindings in Dart and the intermediate language)

A proxy in a different language adds another layer of complexity, but testing should be fairly straight forward and self-contained. It also allows us to have a simpler API for the Dart implementation.

Dynamic loading of modules (instead of command-line only)

There is a `ReloadSources` VM Service RPC that allows to selectively reload libraries. We could use this, together with --reload_every=0 to reload only when needed.

PackageResolver based on local pub cache instead of existing .packages

Right now, repl uses either the local .packages or $packageDir/.packages. We could create one on the fly. It would be cool if we could use pub's constraint-based package resolution to specify versioned packages on the command-line and get everything needed for them on the fly.

This would allow us to provide a playground that let's you explore packages easily in an interactive environment.

Mostly done: Slim execution environment Isolates

Currently the Isolate that contains that is used for evaluation also connects to the VM service and imports the analyzer. This requires quite a few package dependencies. This could cause package conflicts with other packages (and also makes the Isolates heavier than necessary). It would be nice to have slim isolates that only provide the execution scope and a minimal amount of helpers and delegate to a heavier Isolate that talks to the VM service. This slim Isolate would not use any packages and only use relative imports to make it easy to bind it to custom Isolate code.

Use data: URI imports instead of file system code generation

Don't use TempDir, use memory. This is crazy but would be oh so cool for dynamic programming. Does it work? (The only problem is that it won't work with hot reloading…?)

Open questions: functions and classes

This requires hot reloading to be fast/work well. Instead of using a dynamic Scope class with noSuchMethod magic, we could have each declarative cell be its own library and we import previous cells in the later ones hiding older declarations.

Non-declarative code (expressions and statements) are always executed in the latest library. This would do away with automatic variable creations ala `a = 1`.
One would have to execute `var a = 1` and later `a = a + 1`, which would result in different behavior.

Ultimately, it would be nice if one could dump the current symbols into merged code.

Open question: Workflow in general

Hot reloading in Dart might also deprioritize above idea: we could simple import one module that you work on in your IDE and you use the REPL to verify it behaves as expected. Could we use this to make writing tests easier? We could allow one to dump the history into a file and reformulate inputs and outputs as tests. Ie. the given input code for a cell it is expected to match the cell's output. This would require support for `repr`-like serialization that makes it easy to recreate cell outputs in code. Alternatively, we could have tests based on notebooks directly (which we get for free with colab).