Student Info

- Name: Boxuan Li
- GitHub username: li-boxuan
- Open Hub username: li-boxuan
- Email: liboxuan@connect.hku.hk
- Residence during the project: Suzhou, China
- Time Zone: UTC+08:00
- GSoC blog RSS feed URL: http://li-boxuan.github.io/feed.xml

---

Code Sample

I have finished the newcomer guide and get admitted into developer team. My relevant contributions can be found here. I joined in coala community quite late, thus I don't have much contribution yet, but I would keep contributing after the application deadline.
Sadly, I don't have any other open-source contribution, but you may take a look at Tutoria, an online platform for tutoring constructed by my schoolmates and me.

Project Info

Abstract

The goal of project coala Language Server is to offer a coala language server that conforms to Language Server Protocol and implement a plugin, e.g. VS Code plugin.

Motivation

coala is a great language-independent analysis tool which provides a command line interface. It is elegant and easy to use, but characteristics of the command line interface limit usage scenarios. A common scenario is that users may want real-time code analysis. By integrating coala into different editors / IDEs, users would be able to do code analysis when they are writing code. However, this is troublesome and requires a lot of work for each editor / IDE.

An alternative approach is to use Language Server Protocol (LSP). LSP is used between a tool (the client) and a language smartness provider (the server) to integrate features like

auto-complete, go to definition, etc. LSP decouples server logic from client logic, which makes support for multiple editors / IDEs possible. We have a state-of-art implementation in coala-vs-code but it has some limitations and issues. We need to design and implement a general mechanism for multiple editors/IDEs support, which is the aim of this project.

Implementation

This project would be built based on previous work on `coala-vs-code`. This repository would be forked into a new repository, presumably called `coala-language-server`. Work would be done in the new repository. Note that the old repository also offers a vs-client, which may be moved into a submodule of `coala-language-server` or kept in the `coala-vs-code` repo.

As this project is an improvement to an existing module, I would proceed the project by solving current issues and implementing new features. The old language server has following issues and limitations:

1. It only supports python. One of the milestones of coding phase 3 would be supporting multiple languages. This is one of the most important features of this project.
    a. For VS Code client, this line would be changed according to different documentation filters (the complete list is here).
    b. Our server, theoretically, should be able to handle multiple languages. One possible approach is: searching .coafile config file and then executing corresponding bears for the current language.

    Note that this needs to be tested thoroughly, as the current vs-code plugin only supports python.

2. It only supports VS Code (issue #3). Theoretically, the language server is client independent, but it is not tested with other clients. A plugin other than vs-client may or may not be implemented in this project. Also, a small issue is that the current language server adds ad-hoc support for VS Code (see this line). This would be improved in this project and possibly well tested with different clients. A vim language client has been integrated and tested (see here).

3. It does not have good performance (issue #4).

    The main reason is that a coala process is created every time the language server wants to do code analysis. Time and CPU resources would be saved if the language server (written in python) could call coala main function every time instead of recreating processes again and again. Possible approaches would be investigated before coding phases begin. A feasible approach is implemented and a PR has been submitted. This

works well by reducing overhead of starting new process again and again. Instead, it calls main function of coalib.coala directly to do code analysis.

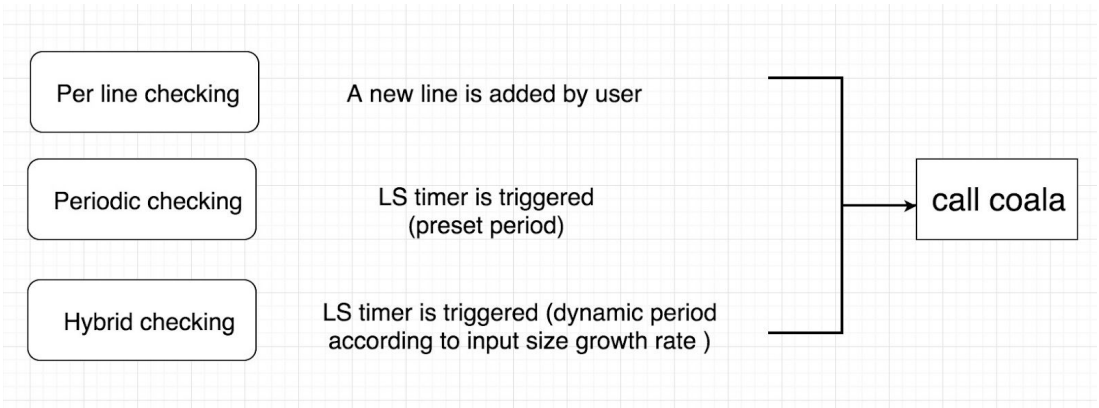
Also, current Language Server calls coala with flag `--find-config`. That means coala would search for the config file before starting code analysis every time. This can be optimized by caching config file path in Language Server runtime.

4. It does not support `textDocument/didChange` request. Currently, the language server handles `textDocument/didSave` request, but not `textDocument/didChange`.

   One reason is that our server should produce a different output when handling `textDocument/didSave` and `textDocument/didChange` requests. This would be handled in this project.

   The main reason, as mentioned above, is that it does not have good performance. Calling coala consumes too much time, and meanwhile, `textDocument/didChange` requests come very often. Current language server uses a compromising way to balance usage of time and instant feedback: it only does code analysis when receiving `textDocument/didSave` requests. If the performance issue could be solved, then this issue becomes much easier - we only need to care about the difference between return value of `didChange` and `didSave` requests. An alternative way (if performance is still a bottleneck) is to run coala occasionally when receiving `textDocument/didChange` requests. The key concern here is that checking needs to be acceptable performant. It may be noticeably slow, even ‘slightly annoyingly slow’, but not ‘unusably slow’. Some feasible techniques are as follows:

   - Per line checking. LS would only run coala if a newline is added in code by the user. That is, we offer so-called real-time analysis for every line of code.
   - Periodic checking. LS would run coala periodically except there are no requests within the period.
   - Hybrid checking. This would combine per line checking and periodic checking together. For instance, periodic checking would be used but instead of adopting a fixed period, the period would change according to rate of line increase. The hybrid checking approach could be complicated and difficult, and needs some design.



Three checking techniques

The checking technique would be done in Coding Phase 2. Further optimizations would be done in Coding Phase 3. Some feasible optimization techniques are as follows:

- A threshold for incremental word count must be reached before triggering coala. The scenario would be a user adds a blank line or adds a simple parenthesis for a new line. In this case, language server would not trigger coala. Note that the threshold might be dynamic. If it takes much time to run coala, the threshold would be increased, vice versa.
- Add an intentional delay before triggering coala. If many requests come within the bounded delay, only the last one would be handled. A scenario would be a user deletes code line by line in a short time. This intentional delay might also be dynamic.
- Again, those optimization techniques could be combined.

5. Lack of documentation. A coala vs-code plugin was released a year ago but relevant documents are still absent. This includes asciinema showing the functionality working, features and sample usage.

6. Lack of tests. Coala requires 100% test coverage. This project would be test driven developed, pursuing 100% test coverage.

7. Other issues. Current Language Server doesn't completely conform to Language Server Protocol. Some relevant issues are: coala-vs-code#37, coala-vs-code#43. This project would solve all issues to make sure it conforms to LSP.

Development Process

Community Bonding (April 23- May 14)

The aim in Community Bonding is to learn more about coala community, dive into Language Server Protocol, investigate possible ways to call coala faster, fork `coala-vs-code` repo and most importantly, finish design documentation including the architecture of the language server and mechanism to support different languages.

Proxy servers and stub servers may or may not be deliverables of this project. This remains further discussions.

Coding Phase 1 (May 14 - June 10)

There are 4 weeks in Coding Phase 1 and an evaluation after the phase.

The aim of Coding Phase 1 is to offer a basic language server for coala which supports python linting. The basic test suite and basic documentation including asciinema showing the functionality working in one editor client would also be provided.

Week 1 (May 14 - May 20)

Work would start on top of existing language server. Hopefully, a smarter way to run coala (rather than creating coala process every time) is found before coding phase. This week would be used to implement that method. Add necessary test cases. CI would be added to the project.

Week 2 (May 21 - May 27)

This week would be used to refactor the structure of existing language server. Since one of the final milestones is to open a pull request to add the server to http://langserver.org, the structure of server needs to be arranged in a better way. Note that support for different requests and languages would be taken into consideration, but not implemented yet. A proxy server might be implemented in this period, deferring to the final decision of server architecture.

Week 3 (May 28 - June 3)

Continue work on last week. Also, there is at least one known bug for current language server. This week would be used to test it thoroughly, reproduce errors and fix them. If there is time, this

week would be used to write an extension for our language server other than VS Code. This may be optional and defer to the final arrangement.

Week 4 (June 4 - June 10)

This week would be mainly used to complete basic test suite and write the documentation as well as asciinema. This is also a good chance to take a pause for code review. This week would be used partially as buffer to fix potential bugs in the previous three weeks.

Coding Phase 2 (June 11 - July 8)

There are 4 weeks in Coding Phase 2 and an evaluation after the phase.

The aim of Coding Phase 2 is to offer a basic language server for coala that supports multiple kinds of requests. The basic test suite and basic documentation including asciinema showing the functionality working in one editor client would also be provided.

Week 5 (June 11 - June 17)

This week would be used to implement `textDocument/didChange` handler. As mentioned in the implementation part, `didChange` requests would be handled only if there is one new line. Compared to a naive approach which handles every single `didChange` request, this can greatly reduce overhead. Note this is only one reasonable approach and would defer to the final design decision. The key concern is language server should offer sort of instant feedback, with acceptable latency. More optimizations could be done in coding phase 3.

Week 6 (June 18 - June 24)

This week would be used to continue work on last week and add necessary test cases. The behavior of `didSave` handler may be changed since real-time feedback could possibly be offered by the `didChange` handler.

Week 7 (June 25 - July 1)

There are some other requests including `textDocument/didOpen`, `textDocument/didClose`. For example, it may be a good idea to do code analysis when the user opens a new file. This week would be used to roughly finish handlers for different requests. Add test cases for different user scenarios.

Week 8 (July 2 - July 8)

This week would be used to take a pause and review previous code and tests thoroughly. Add corner test cases to make test suite complete. Complete relevant documents. This week can also be used as a buffer if some bugs or unfinished work exist.

Coding Phase 3 (July 9 - Aug 5)

There are 4 weeks in Coding Phase 3 and the final evaluation after the phase.

The aim of Coding Phase 3 is to offer a language server for coala that supports multiple languages. A full covering test suite and documentation including asciinema showing the functionality working in one editor client would also be provided.

Week 9 (July 9 - July 15)

Current VS Code plugin only supports python. As mentioned in the implementation section, it could be a minor change to this line if we want to support another language instead of python. A design decision would be made before implementation so as to improve extensibility. This week would be used to add different programming languages for the plugin.

Week 10 (July 16 - July 22)

In theory, current language server should support different programming languages. However, it has never been tested. Moreover, different coala-bears should be run according to the language user is using. This week would be used to deal with that. Add tests for different languages.

Week 11 (July 23 - July 29)

This week would be used to continue work on last week. Hopefully complete tests and documentation. The up-to-date plugin for VS Code (server and client) would be prepared. Also, per-line checking or any other checking method that is done in Coding Phase 2 would be optimized.

Week 12 (July 30 - Aug 5)

The final week would be used to review the whole project and do necessary cleanups before submitting pull requests to add the language server to http://langserver.org. 100% test coverage would be ensured by the end of the project.

Other Commitments

I have a summer internship starting from the June 19 - Aug 24, but I expect the workload for that would be quite light. I commit to contributing at least 30 hours per week during the 3 month coding period. I believe I would be able to contribute even more than that because I regard myself as a hardworking student who is good at multitasking. I don’t have any other short-term or long-term commitments.

I didn’t apply to any other organizations.

Extra Information

concealed