

Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Ярков Иван Сергеевич

Поддержка языка описания модулей в GoLand

Бакалаврская работа

Научный руководитель:
к.ф.-м.н., доц. С. В. Григорьев

Консультант:
ст. преп. кафедры системного программирования СПбГУ Я. А. Кириленко

Рецензент:
Инженер-программист М. Р. Хабибуллин

Санкт-Петербург
2020

Оглавление

Введение	3
1. Постановка задачи	5
2. Обзор	6
2.1. Семантическое версионирование	6
2.2. Система управления зависимостями Go	7
2.3. Платформа IntelliJ	8
2.4. Сравнение существующих решений	8
3. Реализация	9
3.1. Генерация лексического и синтаксического анализаторов	9
3.2. Подсветка конструкций и ошибок синтаксиса языка . . .	9
3.3. Навигация к директории зависимости	11
3.4. Автоматическое дополнение	12
3.5. Скачивание отсутствующей зависимости	13
3.6. Обнаружения неиспользуемых зависимостей	14
3.7. Обнаружение локальных путей при коммите	15
4. Апробация	16
Заключение	17
4.1. Текущие результаты	17
Список литературы	18

Введение

Практически все современное программное обеспечение переиспользует ранее написанный код. Программы, написанные на языке Go, не являются исключением. До версии Go 1.11 переиспользуемой компонентой считались пакеты, которые хранились в указанной программистом единой директории. Чтобы скачать пакет из интернета, нужно было использовать стандартный инструмент командой строки от разработчиков компилятора Go. Этот подход показал себя неудачным, потому что появились такие проблемы, как невозможность хранить зависимости вне единой директории, возможность скачать только самую новую версию пакета, невозможность хранить несколько зависимостей разных версий, отсутствие отдельного файла с зависимостями, что ведет к размещению их всех вместе с проектом.

С появлением версии Go 1.11 появилась предварительная система управления зависимостями в экспериментальном режиме. С того момента зависимость является не отдельным пакетом, а модулем, который является совокупностью пакетов с файлом описывающим его зависимости. Данная система целиком опирается на идею семантического версионирования [9], зависимости описывается в единственном на весь модуль `go.mod` файле. Следующая версия Go 1.14 примет использование данной системы управления зависимостями [2] как стандарт, а использование отдельных пакетов будет считаться устаревшим.

Компания JetBrains разрабатывает интегрированную среду разработки GoLand [6] для языка Go. На данный момент среда разработки не поддерживает `go.mod` файлы на уровне языка, что являлось причиной начать данную работу. Учитывая, что данная система управления зависимостями станет стандартом в ближайшей версии Go 1.14 и то, что GoLand используется больше, чем сотня тысяч пользователей, задача предоставления пользователям такой поддержки является крайне актуальной. Поддержка включает в себя добавление подсветки конструкций и ошибок синтаксиса языка, навигации к директории зависимости, автоматического дополнения кода, статических анализаторов кода для

обнаружения неиспользуемых зависимостей и локальных путей зависимостей при коммите, а также возможности скачать отсутствующие зависимости.

1. Постановка задачи

В рамках настоящей работы целью является реализация поддержки языка описания модулей в GoLand и интеграции полученной функциональности в продукт. Были поставлены следующие задачи:

1. Реализовать подсветку конструкций и ошибок синтаксиса языка.
2. Реализовать навигацию к директории зависимости.
3. Реализовать автоматическое дополнение по ключевым словам и именам зависимостей.
4. Реализовать статические анализаторы кода для обнаружения неиспользуемых зависимостей и локальных путей зависимостей при коммите.
5. Реализовать возможность скачать отсутствующую зависимость.

2. Обзор

2.1. Семантическое версионирование

Идея семантического версионирования [9] появилась в результате такой проблемы как "Ад зависимостей" [1]. Она является ее решением и представляет собой набор правил и требований. У проекта есть определенный программный интерфейс, изменение которого ведет к определенному изменению версии. Полную версию проекта составляют три разделенных числа.

Первое число представляет собой мажорную версию, увеличение которой говорит о том, что появились обратно несовместимые изменения в программном интерфейсе.

Второе число является минорной версией, увеличение которой говорит, что произошло обратно совместимое изменение программного интерфейса, относительно текущей мажорной версии.

Третье число называется патч версией, при увеличении которой, подразумевается, что изменение программного интерфейса не произошло вообще, изменилась только реализация.

2.2. Система управления зависимостями Go

Появившаяся в Go 1.11 система управления зависимостями [2] оперирует понятием модуль, который является набором пакетов, хранящихся в файловом дереве, у которого в корне лежит `go.mod` файл с описанием модуля и его зависимостей. У файла всегда единственное имя – `go.mod`. При сборке проекта, вызывается инструмент командной строки от разработчиков Go, который проверяет какие зависимости были использованы в проекте и в случае необходимости скачивает их в специальную директорию для зависимостей. Система использует принцип семантического версионирования, потому у каждой зависимости в файле должна быть указана ее версия. В файле могут быть использованы пять директив.

`Module` директива устанавливает имя модуля. На весь файл такая директива должна быть единственной.

`Go` директива указывает какой версией компилятора должен быть собран модуль. Директива должна быть единственной для всего файла.

`Require` директива указывает зависимости, которые использует модуль. Зависимость представлена в виде имени и версии.

`Replace` директива позволяет подменить указанные зависимости другими, которые могут находиться локально на машине пользователя.

`Exclude` директива исключает указанные версии зависимостей.

2.3. Платформа IntelliJ

Компания JetBrains разрабатывает платформу с открытым исходным кодом для создания инструментов разработчика – IntelliJ Platform [4]. На ее базе создаются различные интегрированные среды разработки и плагины к ним. Платформа предоставляет программный интерфейс для реализации различной функциональности связанной с поддержкой языков, например подсветку синтаксиса, навигацию, автоматическое дополнение и многое другое. Платформа IntelliJ предоставляет Program Structure Interface [8]. Он является унифицированным интерфейсом к построению и работе с абстрактным синтаксическим деревом исходного кода.

Также, компания JetBrains выпустила плагин Grammar-Kit [3], позволяющий генерировать лексические и синтаксические анализаторы на языке Java поддерживающие PSI. Также он добавляет поддержку файлов с лексическими и синтаксическими правилами. Лексические правила представляют собой регулярные выражения в формате определенном инструментом JFlex [5], а синтаксические правила записываются в расширенной форме Бэкуса-Наура.

2.4. Сравнение существующих решений

На данный момент, ни одна из существующих интегрированных сред разработки не имеет поддержку языка описания модулей, в том числе и конкурирующая с GoLand среда разработки – Visual Studio Code [10] с плагином для поддержки Go.

3. Реализация

3.1. Генерация лексического и синтаксического анализаторов

При реализации лексического и синтаксического анализаторов был использован плагин Grammar-Kit [3]. Генерация лексического и синтаксического анализаторов осложнялась тем, что разработчики языка Go не выкладывали спецификацию языка описания модулей. Также, было поставлено требование к синтаксическому анализатору, что он должен уметь восстанавливаться после некорректной синтаксической конструкции как можно быстрее и обрабатывать следующую конструкцию. К счастью, Grammar-Kit предоставляет возможность определить специальные правила, для восстановления анализатора.

Проведенный анализ языка позволил выписать его лексические и грамматические правила. Также были разработаны и реализованы правила для восстановления анализатора при некорректной синтаксической конструкции.

3.2. Подсветка конструкций и ошибок синтаксиса языка

Подсветка ошибок реализована на уровнях синтаксического анализатора и аннотатора. Для отображения ошибок уровня синтаксического анализатора, достаточно определить грамматику языка и они автоматически отображаются в редакторе среды разработки благодаря интеграции платформы и Grammar-Kit плагина. Для другого уровня отображения ошибок, уровня аннотатора, необходимо явно определить логику и шаблоны синтаксического дерева, на которых он работает. К таким ошибкам относятся, например, множественные директивы `module` и `go`, когда должно быть не больше одной для каждой из них. Вынесение части проверок корректности файла с зависимостями в аннотатор позволило существенно облегчить лексические и грамматические пра-

вила и написать более простую грамматику.

Ключевые слова и комментарии языка подсвечиваются на этапе лексического анализа программы, так как их возможно определить на этапе разбиения текста на токены. Имена зависимостей и их версии подсвечиваются на уровне аннотатора, так как осуществление подсветки зависит от того, в какой синтаксической конструкции они находятся (рис. 1).

```
1  module awesomeProject7
2
3  go 1.14|
4
5  require (
6      gopkg.in/yaml.v2 v2.2.8
7      k8s.io/api v0.18.2
8  )
```

Рис. 1: Подсветка синтаксиса

3.3. Навигация к директории зависимости

Платформа IntelliJ предоставляет механизм определения ссылок на элементах абстрактного синтаксического дерева. Используя данный механизм была реализована навигация к директории зависимости. Найденная директория подсвечивается в окне структуры проекта (рис. 2). Реализация данной функциональности также учитывает устаревшие проекты, которые не располагаются внутри кэширующей директории.

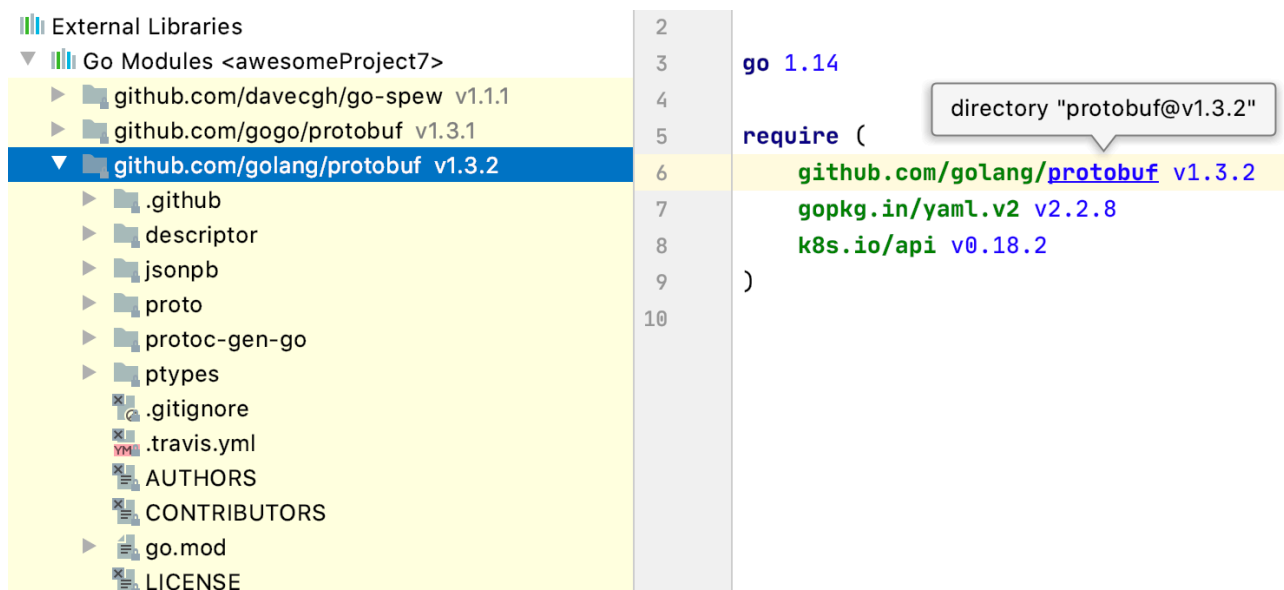


Рис. 2: Навигация к директории зависимости

3.4. Автоматическое дополнение

Автоматическое дополнение по ключевым словам было реализовано с помощью стандартного средства платформы IntelliJ, системы Live Templates [7] и базовых возможностей платформы. Было необходимо задать шаблон синтаксического дерева на котором данная система будет предлагать автоматическое дополнение. Для ключевых слов, это должна быть директива верхнего уровня (рис. 3).

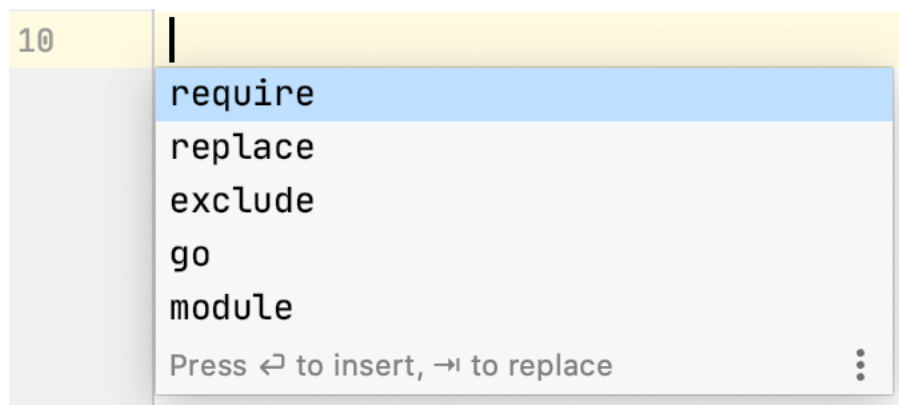


Рис. 3: Автодополнение ключевых слов

Автоматическое дополнение имен зависимостей ищет имена всех загруженных зависимостей в кэширующей директории. Дополнение учитывает устаревшие проекты, которые не используют систему управления зависимостями и располагаются вне кэширующей директории, а также проекты использующие Go вендоринг. Данное дополнение доступно только внутри require, replace или exclude директив и только на токене представляющем имя зависимости (рис. 4).



Рис. 4: Автодополнение имен зависимостей

3.5. Скачивание отсутствующей зависимости

Для реализации скачивания отсутствующих модулей была проведена интеграция с инструментом управления зависимостями Go. Отсутствующие зависимости подсвечиваются как ошибки на уровне аннотатора и среда разработки предлагает пользовательский интерфейс для их скачивания (рис. 5).

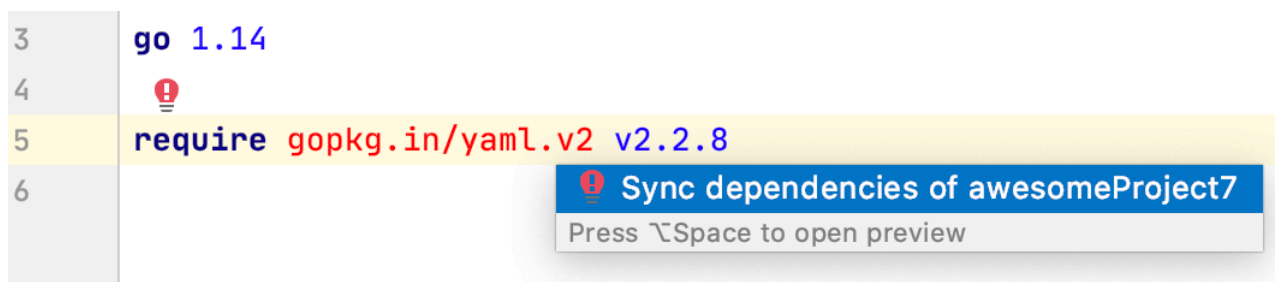


Рис. 5: Интерфейс для скачивания зависимости

3.6. Обнаружения неиспользуемых зависимостей

Для отображения неиспользуемых зависимостей был реализован статический анализатор кода, которая ищет в проекте наличие импортирования пакетов из зависимости (рис. 6). Для реализации анализатора был построен специальный индекс, который отображает имя зависимости в список файлов в которых она используется. Такой подход позволяет эффективно проверять используется ли она в проекте. При изменении файла, индекс инкрементально перестраивается, что обеспечивает хорошую производительность решения. Также, анализатор корректно обрабатывает особые случаи с транзитивными зависимостями из устаревших проектов.

```
5   require (
6       github.com/golang/protobuf v1.3.2
7       gopkg.in/
8       k8s.io/ap
9   )
```

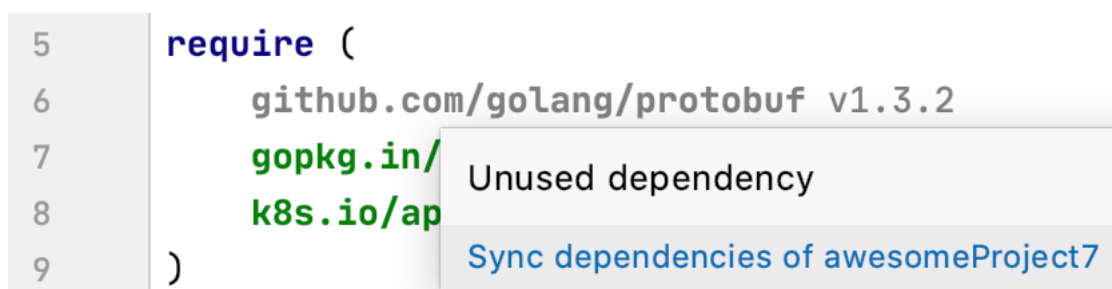


Рис. 6: Предупреждение о неиспользуемой зависимости

3.7. Обнаружение локальных путей при коммите

Для предупреждения о локальных путях был также реализован статический анализатор кода, который запускается вместе с рядом других при коммите изменений с помощью интерфейса среды разработки (рис. 7). Коммит локальных путей является запахом кода потому, что он может сломать сборку проекта для других пользователей. Анализатор ищет `replace` директиву, в правой части которой прописан локальный путь.

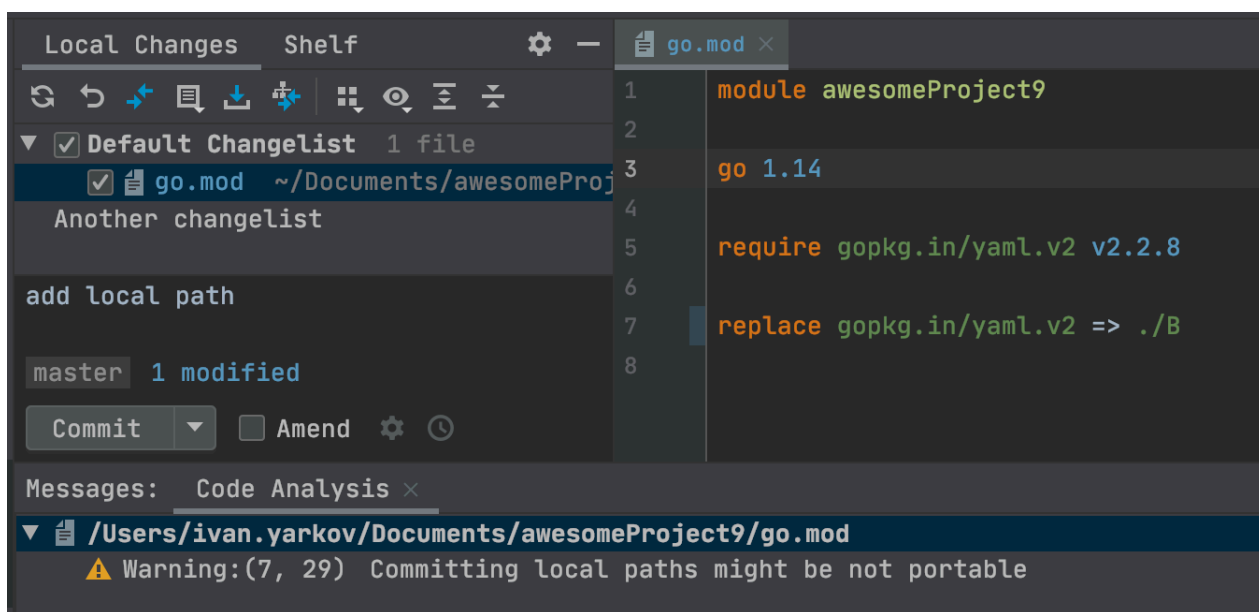


Рис. 7: Предупреждение о локальном пути

4. Апробация

Реализованная функциональность была покрыта обширным набором модульных тестов. Она проходит существующие модульные, интеграционные тесты, а также тесты на производительность проекта GoLand. Функциональность была проверена отделом контроля качества, исходный код был просмотрен командой GoLand и интегрирован в продукт.

Заключение

4.1. Текущие результаты

В рамках настоящей работы были выполнены все поставленные задачи:

1. Реализована подсветка конструкций и ошибок синтаксиса языка.
2. Реализована навигация к директории зависимости.
3. Реализовано автоматическое дополнение по ключевым словам и именам зависимостей.
4. Реализованы статические анализаторы кода для обнаружения неиспользуемых зависимостей и локальных путей зависимостей при коммите.
5. Реализована возможность скачать отсутствующую зависимость.

Работа была интегрирована в продукт GoLand и получила положительные отзывы от пользователей среды разработки.

Список литературы

- [1] Dependency hell. (дата обращения: 05.12.2019). — URL: https://ru.wikipedia.org/wiki/Dependency_hell.
- [2] Go modules. (дата обращения: 03.12.2019). — URL: <https://github.com/golang/go/wiki/Modules>.
- [3] Grammar-Kit. (дата обращения: 08.12.2019). — URL: <https://github.com/JetBrains/Grammar-Kit>.
- [4] IntelliJ Platform. (дата обращения: 04.12.2019). — URL: <https://www.jetbrains.com/ru-ru/opensource/idea/>.
- [5] JFlex. (дата обращения: 08.12.2019). — URL: <https://www.jflex.de/>.
- [6] JetBrains. GoLand. (дата обращения: 01.12.2019). — URL: <https://www.jetbrains.com/go>.
- [7] Live templates. (дата обращения: 10.12.2019). — URL: https://www.jetbrains.org/intellij/sdk/docs/tutorials/live_templates.html?search=Live.
- [8] Program Structure Interface. (дата обращения: 09.12.2019). — URL: https://www.jetbrains.org/intellij/sdk/docs/basics/architectural_overview/psi.html/.
- [9] Semantic versioning. (дата обращения: 01.12.2019). — URL: <https://semver.org>.
- [10] Visual Studio Code. (дата обращения: 06.12.2019). — URL: <https://code.visualstudio.com/>.