VERSIONS LANGUAGES

约定式提交

一种用于给提交信息增加人机可读含义的规范

阅读规范 GitHub

约定式提交 1.0.0-beta.4

概述

约定式提交规范是一种基于提交消息的轻量级约定。 它提供了一组用于创建清晰的提交历史的简单规则； 这使得编写基于规范的自动化工具变得更容易。 这个约定与 **SemVer** 相吻合， 在提交信息中描述新特性、bug 修复和破坏性变更。

提交说明的结构如下所示：

```
<类型>[可选的作用域]: <描述>

[可选的正文]

[可选的脚注]
```

提交说明包含了下面的结构化元素，以向类库使用者表明其意图：

1. **fix:** *类型* 为 `fix` 的提交表示在代码库中修复了一个 bug（这和语义化版本中的 `PATCH` 相对应）。
2. **feat:** *类型* 为 `feat` 的提交表示在代码库中新增了一个功能（这和语义化版本中的 `MINOR` 相对应）。
3. **BREAKING CHANGE:** 在可选的正文或脚注的起始位置带有 `BREAKING CHANGE:` 的提交，表示引入了破坏性 API 变更（这和语义化版本中的 `MAJOR` 相对应）。 破坏性变更可以是任意 *类型* 提交的一部分。
4. **其它情况:** 除 `fix:` 和 `feat:` 之外的提交 *类型* 也是被允许的，例如 **@commitlint/config-conventional**（基于 **Angular 约定**）中推荐的 `chore:`、`docs:`、`style:`、`refactor:`、`perf:`、`test:` 及其他标签。 我们也推荐使用 `improvement`，用于对当前实现进行改进而没有添加新功能或修复错误的提交。 请注意，这些标签在约定式提交规范中并不是强制性的。并且在语义化版本中没有隐式的影响（除非他们包含 BREAKING CHANGE）。
   可以为提交类型添加一个围在圆括号内的作用域，以为其提供额外的上下文信息。例如 `feat(parser): adds ability to parse arrays.`。

示例

包含了描述以及正文内有破坏性变更的提交说明

```
feat: allow provided config object to extend other configs

BREAKING CHANGE: `extends` key in config file is now used for extending other co
```

包含了可选的 ！ 字符以提醒注意破坏性变更的提交说明

```
chore!: drop Node 6 from testing matrix

BREAKING CHANGE: dropping Node 6 which hits end of life in April
```

不包含正文的提交说明

```
docs: correct spelling of CHANGELOG
```

包含作用域的提交说明

```
feat(lang): add polish language
```

为 fix 编写的提交说明，包含（可选的） issue 编号

```
fix: correct minor typos in code

see the issue for details on the typos fixed

closes issue #12
```

约定式提交规范

本文中的关键词“必须（MUST）”、“禁止（MUST NOT）”、“必要（REQUIRED）”、“应当（SHALL）”、“不应当（SHALL NOT）”、“应该（SHOULD）”、“不应该（SHOULD NOT）”、“推荐（RECOMMENDED）”、“可以（MAY）”和“可选（OPTIONAL）”，解释参考 **RFC 2119** 中所述。

1. 每个提交都**必须**使用类型字段前缀，它由一个名词组成，诸如 `feat` 或 `fix` ，其后接一个**可选的**作用域字段，以及一个**必要的**冒号（英文半角）和空格。
2. 当一个提交为应用或类库实现了新特性时，**必须**使用 `feat` 类型。
3. 当一个提交为应用修复了 bug 时，**必须**使用 `fix` 类型。

4. 作用域字段**可以**跟随在类型字段后面。作用域**必须**是一个描述某部分代码的名词，并用圆括号包围，例如： `fix(parser):`
5. 描述字段**必须**紧接在类型/作用域前缀的空格之后。描述指的是对代码变更的简短总结，例

5. 描述字段**必须**紧接在类型/作用域前缀的空格之后。描述指的是对代码变更的简短总结，例如：[illegible]
6. 在简短描述之后，**可以**编写更长的提交正文，为代码变更提供额外的上下文信息。正文**必须**起始于描述字段结束的一个空行后。
7. 在正文结束的一个空行之后，**可以**编写一行或多行脚注。脚注**必须**包含关于提交的元信息，例如：关联的合并请求、Reviewer、破坏性变更，每条元信息一行。
8. 破坏性变更**必须**标示在正文区域最开始处，或脚注区域中某一行的开始。一个破坏性变更**必须**包含大写的文本 `BREAKING CHANGE`，后面紧跟冒号和空格。
9. 在 `BREAKING CHANGE:` 之后**必须**提供描述，以描述对 API 的变更。例如：[illegible]
10. 在提交说明中，**可以**使用 `feat` 和 `fix` 之外的类型。
11. 工具的实现**必须不**区分大小写地解析构成约定式提交的信息单元，只有 `BREAKING CHANGE` **必须**是大写的。
12. **可以**在类型/作用域前缀之后，`:` 之前，附加 `!` 字符，以进一步提醒注意破坏性变更。当有 `!` 前缀时，正文或脚注内**必须**包含 `BREAKING CHANGE: description`

为什么使用约定式提交

- 自动化生成 CHANGELOG。
- 基于提交的类型，自动决定语义化的版本变更。
- 向同事、公众与其他利益关系者传达变化的性质。
- 触发构建和部署流程。
- 让人们探索一个更加结构化的提交历史，以便降低对你的项目做出贡献的难度。

FAQ

在初始开发阶段我该如何处理提交说明?

我们建议你按照假设你已发布了产品那样来处理。因为通常总 *有人* 使用你的软件，即便那是你软件开发的同事们。他们会希望知道诸如修复了什么、哪里不兼容等信息。

提交标题中的类型是大写还是小写?

大小写都可以，但最好是一致的。

如果提交符合多种类型我该如何操作?

回退并尽可能创建多次提交。约定式提交的好处之一是能够促使我们做出更有组织的提交和PR。

这不会阻碍快速开发和迭代吗?

它阻碍的是以杂乱无章的方式快速前进。它助你能在横跨多个项目以及和多个贡献者协作时长期

地快速演进。

约定式提交会让开发者受限于提交的类型吗（因为他们会想着已提供的类型）？

约定式提交鼓励我们更多地使用某些类型的提交，比如 `fixes` 。除此之外，约定式提交的灵活性也允许你的团队使用自己的类型，并随着时间的推移更改这些类型。

这和 SemVer 有什么关联呢?

`fix` 类型提交应当对应到 `PATCH` 版本。 `feat` 类型提交应该对应到 `MINOR` 版本。带有 `BREAKING CHANGE` 的提交不管类型如何，都应该对应到 `MAJOR` 版本。

我对约定式提交做了形如 `@jameswomack/conventional-commit-spec` 的扩展，该如何版本化管理这些扩展呢?

我们推荐使用 SemVer 来发布你对于这个规范的扩展（并鼓励你创建这些扩展！）

如果我不小心使用了错误的提交类型，该怎么办呢?

当你使用了在规范中但错误的类型时，例如将 `feat` 写成了 `fix`

在合并或发布这个错误之前，我们建议使用 `git rebase -i` 来编辑提交历史。而在发布之后，根据你使用的工具和流程不同，会有不同的清理方案。

当使用了*不*在规范中的类型时，例如将 `feat` 写成了 `feet`

在最坏的场景下，即便提交没有满足约定式提交的规范，也不会是世界末日。这只意味着这个提交会被基于规范的工具错过而已。

所有的贡献者都需要使用约定式提交规范吗?

并不！如果你使用基于 squash 的 Git 工作流，主管维护者可以在合并时清理提交信息——这不会对普通提交者产生额外的负担。 有种常见的工作流是让 git 系统自动从 pull request 中 squash 出提交，并向主管维护者提供一份表单，用以在合并时输入合适的 git 提交信息。

关于

约定式提交规范受到了 **Angular 提交准则**的启发，并在很大程度上以其为依据。

该规范的首个草案来自下面这些项目中若干贡献者们的协作：

- **conventional-changelog**：一套从 git 历史中解析出约定式提交说明的工具。
- **parse-commit-message**：兼容规范的解析工具，可以将给定提交信息的字符串解析成对象，结果形如 `{ header: { type, scope, subject }, body, footer }` 。
- **bumped**：一个用于发布软件的工具，可以在为你的软件发布新版本前后轻松地执行操作

- **bumped**：一个用于发布软件的工具，可以在为你的软件发布新版本前后轻松地执行操作。
- **unleash**：一个用于自动化软件发行和发布生命周期的工具。
- **lerna**：一个用于管理宏仓库（monorepo）的工具，源自 Babel 项目。

用于约定式提交的工具

- **php-commitizen**：一个用于创建遵循约定式提交规范提交信息的工具。 可配置，并且可以作为 composer 依赖项用于 PHP 项目，或可在非 PHP 项目中全局使用。
- **conform**：一个可用以在 git 仓库上施加配置的工具，包括约定式提交。
- **standard-version** 基于 GitHub 的新 squash 按钮与推荐的约定式提交工作流，自动管理版本和 CHANGELOG。
- **commitsar**： 一个检查提交信息是否符合约定式提交规范的 Go 语言工具。可在 CI 的 Docker 镜像中使用。

使用约定式提交的项目

- **yargs**：广受欢迎的命令行参数解析器。
- **istanbuljs**：一套为 JavaScript 测试生成测试覆盖率的开源工具和类库。
- **uPortal-home** 和 **uPortal-application-framework**：用于增强 **Apereo uPortal** 的可选用户界面。
- **massive.js**：一个用于 Node 和 PostgreSQL 的数据访问类库。
- **electron**：用 JavaScript、HTML 和 CSS 构建跨平台应用。
- **scroll-utility**：一个居中元素和平滑动画的滚屏工具包实例。
- **Blaze UI**：无框架开源 UI 套件。
- **Monica**：一个开源的人际关系管理系统。
- **mhy**：🧩 一个零配置、开箱即用的、多用途工具箱与开发环境。
- **sharec**：一个用于模板和配置文件版本化的极简工具。


Conventional Commits

想让你的项目出现在上面吗？ **提交 pull request** 吧。