<https://eggjs.org/zh-cn/contributing.html>

**代码贡献规范**

有任何疑问，欢迎提交 [issue](https://github.com/eggjs/egg/issues)， 或者直接修改提交 [PR](https://github.com/eggjs/egg/pulls)!

**提交 issue**

* 请确定 issue 的类型。
* 请避免提交重复的 issue，在提交之前搜索现有的 issue。
* 在标签(分类参考**标签分类**), 标题 或者内容中体现明确的意图。

随后 egg 负责人会确认 issue 意图，更新合适的标签，关联 milestone，指派开发者。

标签可分为两类，type 和 scope

* type: issue 的类型，如 feature, bug, documentation, performance, support ...
* scope: 修改文件的范围，如 core: xx，plugin: xx，deps: xx

**常用标签说明**

* support: issue 提出的问题需要开发者协作排查，咨询，调试等等日常技术支持。
* bug: 一旦发现可能是 bug 的问题，请打上 bug，然后等待确认，一旦确认是 bug，此 issue 会被再打上 confirmed。
  + 此时 issue 会被非常高的优先级进行处理。
  + 如果此 bug 是正在影响线上应用正常运行，会再打上 critical，代表是最高优先级，需要马上立刻处理！
  + bug 会在最低需要修复的版本进行修复，如是在 0.9.x 要修复的，而当前最新版本是 1.1.x， 那么此 issue 还会被打上 0.9，0.10，1.0，1.1，代表需要修复到这些版本。
* core: xx: 代表 issue 跟 core 内核相关，如 core: antx 代表跟 antx 配置相关。
* plugin: xx: 代表 issue 跟插件相关，如 deps: session 代表跟 session 插件相关。
* deps: xx: 代表 issue 跟 dependencies 模块相关，如 deps: egg-cors 代表跟 egg-cors 模块相关。
* chore: documentation: 代表发现了文档相关问题，需要修复文档说明。
* cbd: 代表跟服务器部署相关

**编写文档**

所有功能点必须提交配套文档，文档须满足以下要求

* 必须说清楚问题的几个方面：what（是什么），why（为什么），how（怎么做），可根据问题的特性有所侧重。
* how 部分必须包含详尽完整的操作步骤，必要时附上 **足够简单，可运行** 的范例代码， 所有范例代码放在 [eggjs/examples](https://github.com/eggjs/examples" \t "_blank) 库中。
* 提供必要的链接，如申请流程，术语解释和参考文档等。
* 同步修改中英文文档，或者在 PR 里面说明。

**提交代码**

**提交 Pull Request**

如果你有仓库的开发者权限，而且希望贡献代码，那么你可以创建分支修改代码提交 PR，egg 开发团队会 review 代码合并到主干。

|  |
| --- |
| *# 先创建开发分支开发，分支名应该有含义，避免使用 update、tmp 之类的* $ git checkout -b branch-name  *# 开发完成后跑下测试是否通过，必要时需要新增或修改测试用例* $ npm test  *# 测试通过后，提交代码，message 见下面的规范*  $ git add . *# git add -u 删除文件* $ git commit -m "fix(role): role.use must xxx" $ git push origin branch-name |

提交后就可以在 [egg](https://github.com/eggjs/egg/pulls) 创建 Pull Request 了。

由于谁也无法保证过了多久之后还记得多少，为了后期回溯历史的方便，请在提交 MR 时确保提供了以下信息。

1. 需求点（一般关联 issue 或者注释都算）
2. 升级原因（不同于 issue，可以简要描述下为什么要处理）
3. 框架测试点（可以关联到测试文件，不用详细描述，关键点即可）
4. 关注点（针对用户而言，可以没有，一般是不兼容更新等，需要额外提示）

**代码风格**

你的代码风格必须通过 eslint，你可以运行 $ npm run lint 本地测试。

**Commit 提交规范**

根据 [angular 规范](https://github.com/angular/angular.js/blob/master/CONTRIBUTING.md#commit-message-format)提交 commit， 这样 history 看起来更加清晰，还可以自动生成 changelog。

|  |
| --- |
| <type>(<scope>): <subject> <BLANK LINE> <body> <BLANK LINE> <footer> |

（1）type

提交 commit 的类型，包括以下几种

* feat: 新功能
* fix: 修复问题
* docs: 修改文档
* style: 修改代码格式，不影响代码逻辑
* refactor: 重构代码，理论上不影响现有功能
* perf: 提升性能
* test: 增加修改测试用例
* chore: 修改工具相关（包括但不限于文档、代码生成等）
* deps: 升级依赖

（2）scope

修改文件的范围（包括但不限于 doc, middleware, core, config, plugin）

（3）subject

用一句话清楚的描述这次提交做了什么

（4）body

补充 subject，适当增加原因、目的等相关因素，也可不写。

（5）footer

* **当有非兼容修改(Breaking Change)时必须在这里描述清楚**
* 关联相关 issue，如 Closes #1, Closes #2, #3
* 如果功能点有新增或修改的，还需要关联文档 doc 和 egg-init 的 PR，如 eggjs/egg-bin#123

示例

|  |
| --- |
| fix($compile): [BREAKING\_CHANGE] couple of unit tests for IE9  Older IEs serialize html uppercased, but IE9 does not... Would be better to expect case insensitive, unfortunately jasmine does not allow to user regexps for throw expectations.  Document change on eggjs/egg#123  Closes #392  BREAKING CHANGE:   Breaks foo.bar api, foo.baz should be used instead |

查看具体[文档](https://docs.google.com/document/d/1QrDFcIiPjSLDn3EL15IJygNPiHORgU1_OOAqWjiDU5Y/edit" \t "_blank)

**发布管理**

egg 基于 [semver](http://semver.org/lang/zh-CN/" \t "_blank) 语义化版本号进行发布。

**分支策略**

master 分支为当前稳定发布的版本，next 分支为下一个开发中的大版本。

* 只维护两个版本，除非有安全问题，否则修复只会 patch 到 master 和 next 分支，其他更新推动上层框架升级到稳定大版本的最新版本。
* 所有 API 的废弃都需要在当前的稳定版本上 deprecate 提示，并保证在当前的稳定版本上一直兼容到新版本的发布。
* master 分支不设置 publish tag，上层框架基于 semver 依赖稳定版本。
* next 分支设置 tag 为 next，上层框架可以通过 egg@next 引用开发中的版本进行测试。
* egg 持续维护的版本以 Milestone 为准，只要是开着的版本都会进行修复。

**发布策略**

每个大版本都有一个发布经理管理（PM），他/她要做的事情

**准备工作：**

* 建立 milestone，确认需求关联 milestone，指派和更新 issues，如 [1.x milestone](https://github.com/eggjs/egg/milestone/1)。
* 从 master 分支新建 next 分支，并设置 tag 为 next。

**发布前：**

* 确认当前 Milestone 所有的 issue 都已关闭或可延期，完成性能测试。
* 发起一个新的 [Release Proposal MR](https://github.com/nodejs/node/pull/4181)，按照 [node CHANGELOG](https://github.com/nodejs/node/blob/master/CHANGELOG.md) 进行 History 的编写，修正文档中与版本相关的内容，commits 可以自动生成。

|  |
| --- |
| $ npm run commits |

* 指定下一个大版本的 PM。

**发布时：**

* 将老的稳定版本（master）备份到以当前大版本为名字的分支上（例如 1.x），并设置 tag 为 release-{v}.x（ v 为当前版本，例如 release-1.x）。
* 将 next 分支推送到 master，成为新的稳定版本分支，并去除 next tag，修改 README 中与分支相关的内容。
* 发布新的稳定版本到 [npm](http://npmjs.com/" \t "_blank)，并通知上层框架进行更新。
* npm publish 之前，请先阅读[『我是如何发布一个 npm 包的』](https://fengmk2.com/blog/2016/how-i-publish-a-npm-package" \t "_blank)。

上述描述中所有的设置 tag 都是指在 package.json 中设置 npm 的 tag。

|  |
| --- |
| "publishConfig": {  "tag": "next" } |