对Web前端项目测试情况的实证研究 中期报告

陈俊达 刘宇航

2020年12月15日

目录

研究问题

RQ1：**测试用例的数量**和**整体代码量**之间有什么关系？

RQ2：**与DOM交互的测试用例**在所有测试用例中占比多少？

RQ3：测试用例数量与**bug数量**的关系是怎么样的？

RQ4：使用**TypeScript**是否能降低bug的数量？

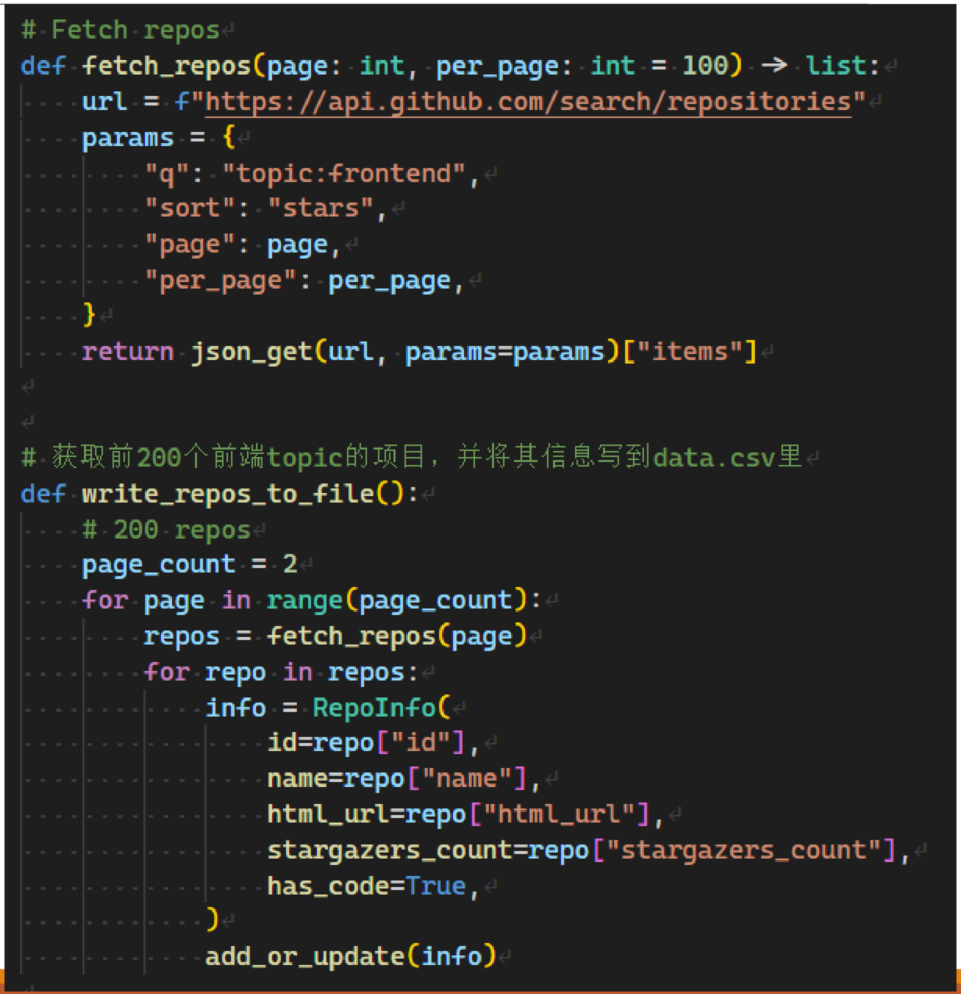
- **TS的代码量占总体代码量的比例**和**bug数量/总代码量**之间的关系

需要获取的数据

- 前端star数量前200的项目 ✓
- 每个项目的JS、TS代码量 ✓
- 每个项目的bug数量 ✓
- 每个项目的单元测试用例数量（**正在进行，51/200**）
- 每个项目的带有dom测试的单元测试数量 ☒
- **自动化**

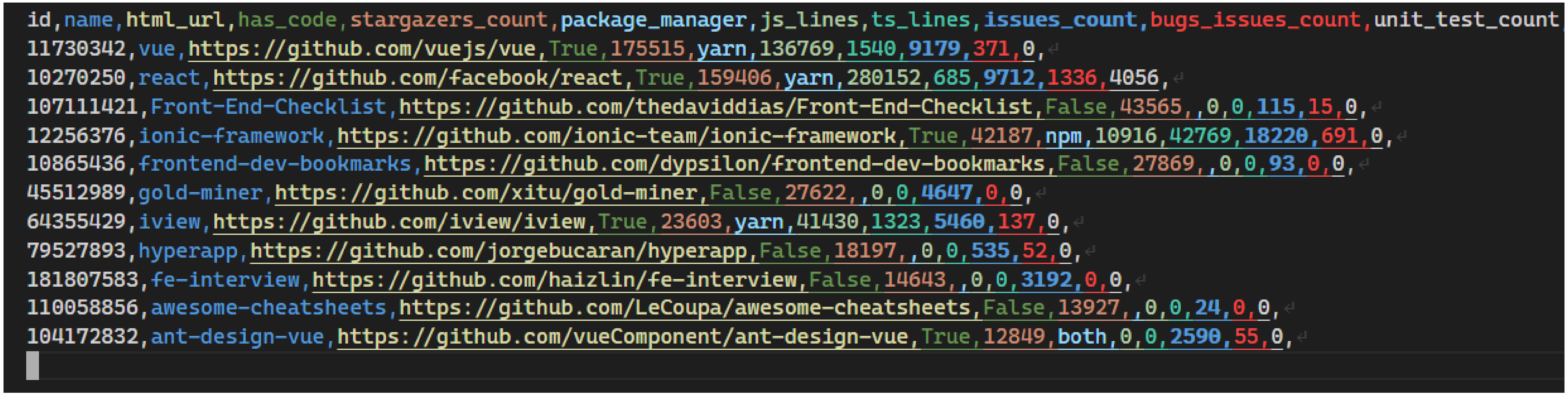
前端star数量前200的项目

- 使用GitHub提供的API搜索topic是frontend的项目，并根据stars排序
  - 每一页(per page)最多100个项目
  - 可通过指定page参数指定第几页
  - 所以请求两次API，获取2页的数据
- 把获取的数据存储到data.csv中



```python
# Fetch repos
def fetch_repos(page: int, per_page: int = 100) -> list:
    url = f"https://api.github.com/search/repositories"
    params = {
        "q": "topic:frontend",
        "sort": "stars",
        "page": page,
        "per_page": per_page,
    }
    return json_get(url, params=params)["items"]


# 获取前200个前端topic的项目，并将其信息写到data.csv里
def write_repos_to_file():
    # 200 repos
    page_count = 2
    for page in range(page_count):
        repos = fetch_repos(page)
        for repo in repos:
            info = RepoInfo(
                id=repo["id"],
                name=repo["name"],
                html_url=repo["html_url"],
                stargazers_count=repo["stargazers_count"],
                has_code=True,
            )
            add_or_update(info)
```

前端star数量前200的项目

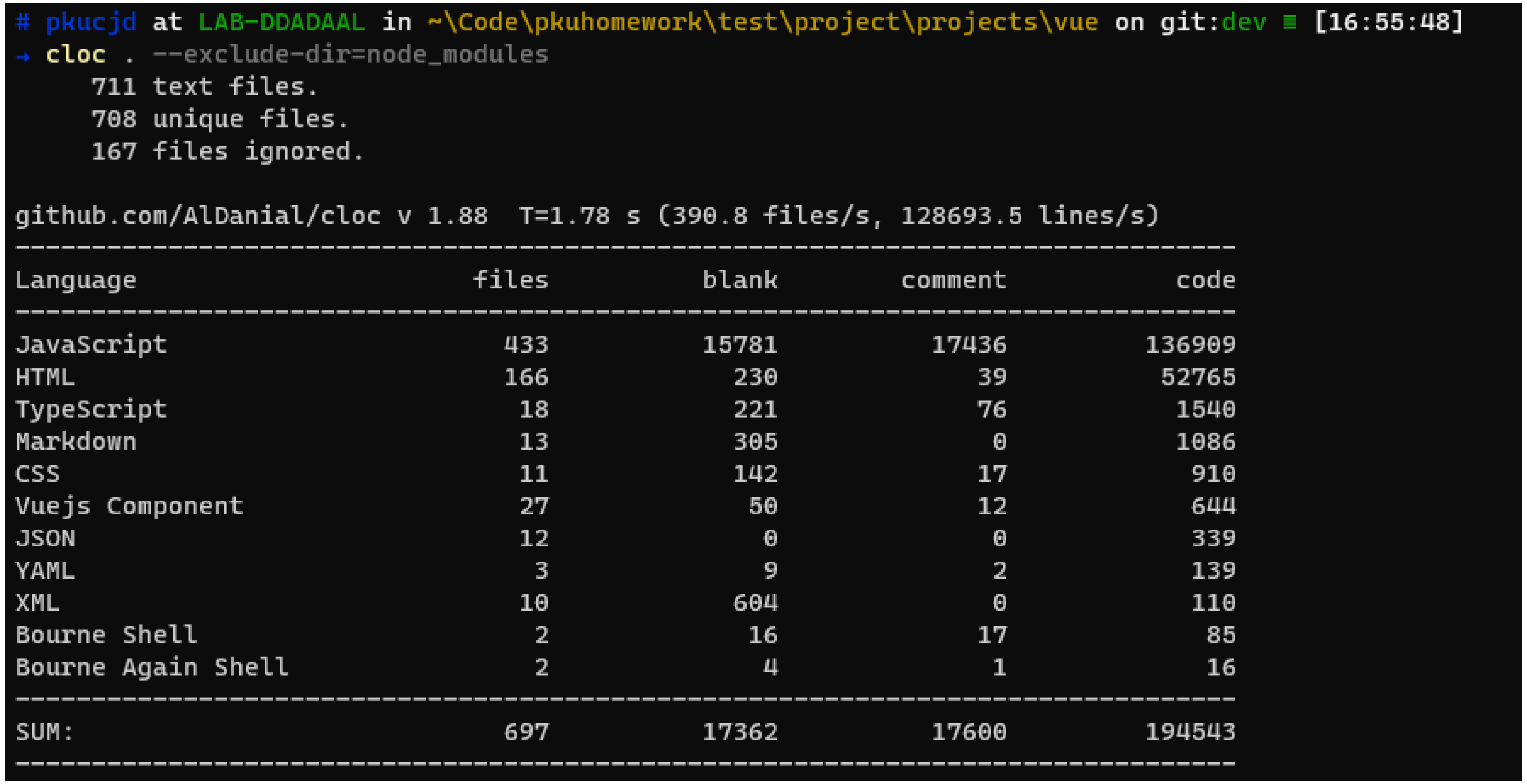


```
id,name,html_url,has_code,stargazers_count,package_manager,js_lines,ts_lines,issues_count,bugs_issues_count,unit_test_count
11730342,vue,https://github.com/vuejs/vue,True,175515,yarn,136769,1540,9179,371,0,
10270250,react,https://github.com/facebook/react,True,159406,yarn,280152,685,9712,1336,4056,
107111421,Front-End-Checklist,https://github.com/thedaviddias/Front-End-Checklist,False,43565,,0,0,115,15,0,
12256376,ionic-framework,https://github.com/ionic-team/ionic-framework,True,42187,npm,10916,42769,18220,691,0,
10865436,frontend-dev-bookmarks,https://github.com/dypsilon/frontend-dev-bookmarks,False,27869,,0,0,93,0,0,
45512989,gold-miner,https://github.com/xitu/gold-miner,False,27622,,0,0,4647,0,0,
64355429,iview,https://github.com/iview/iview,True,23603,yarn,41430,1323,5460,137,0,
79527893,hyperapp,https://github.com/jorgebucaran/hyperapp,False,18197,,0,0,535,52,0,
181807583,fe-interview,https://github.com/haizlin/fe-interview,False,14643,,0,0,3192,0,0,
110058856,awesome-cheatsheets,https://github.com/LeCoupa/awesome-cheatsheets,False,13927,,0,0,24,0,0,
104172832,ant-design-vue,https://github.com/vueComponent/ant-design-vue,True,12849,both,0,0,2590,55,0,
```

Csv文件内容

每个项目的JS、TS代码量

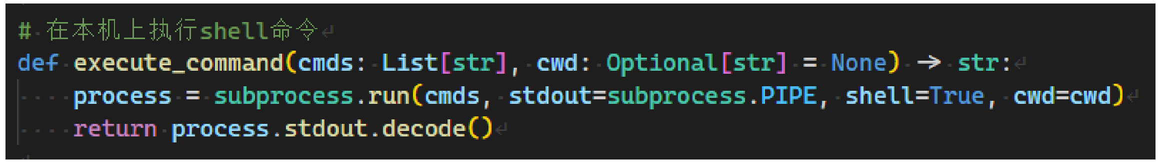
- 使用cloc工具统计代码量



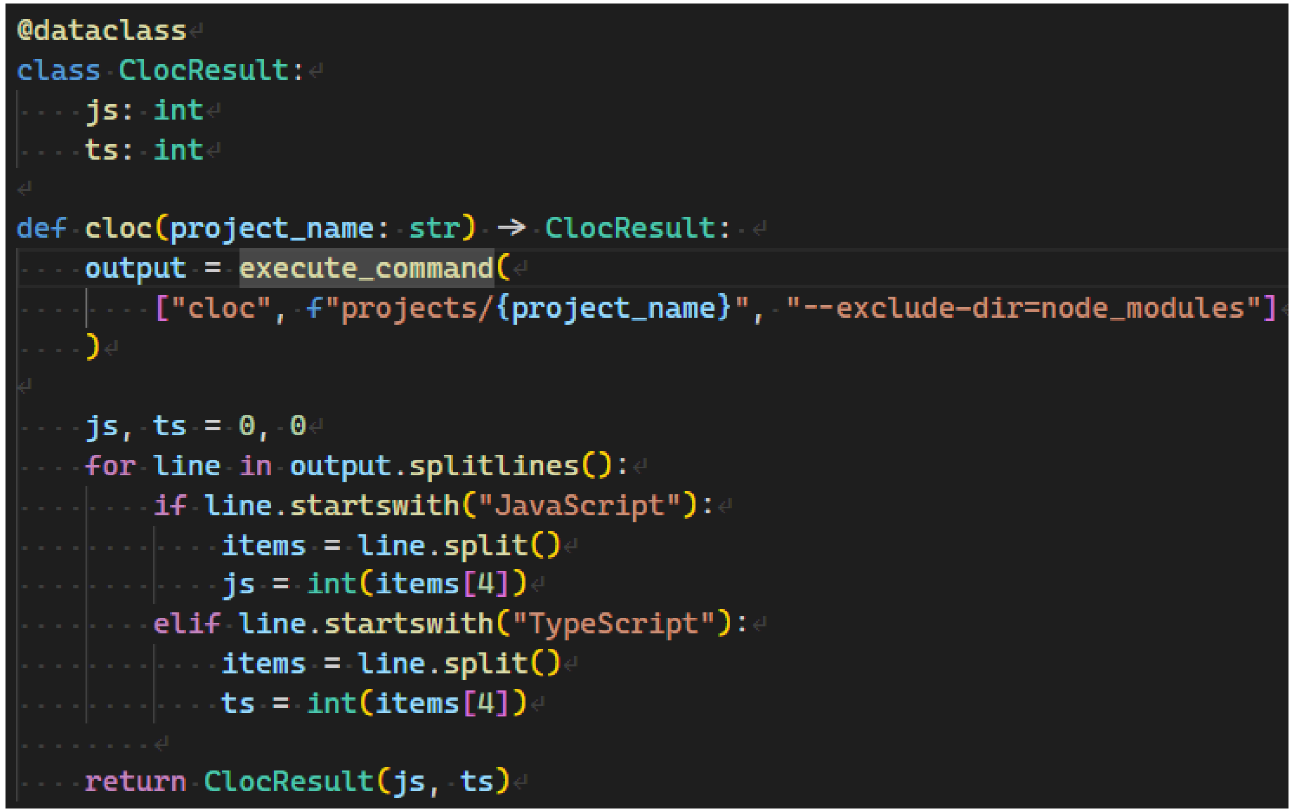
```
# pkucjd at LAB-DDADAAL in ~\Code\pkuhomework\test\project\projects\vue on git:dev ≡ [16:55:48]
→ cloc . --exclude-dir=node_modules
     711 text files.
     708 unique files.
     167 files ignored.

github.com/AlDanial/cloc v 1.88  T=1.78 s (390.8 files/s, 128693.5 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JavaScript                     433          15781          17436         136909
HTML                           166            230             39          52765
TypeScript                      18            221             76           1540
Markdown                        13            305              0           1086
CSS                             11            142             17            910
Vuejs Component                 27             50             12            644
JSON                            12              0              0            339
YAML                             3              9              2            139
XML                             10            604              0            110
Bourne Shell                     2             16             17             85
Bourne Again Shell               2              4              1             16
-------------------------------------------------------------------------------
SUM:                           697          17362          17600         194543
-------------------------------------------------------------------------------
```

每个项目的JS、TS代码量

- 调用命令行的cloc工具，传入项目目录
- 分析stdout输出的获得每个项目的代码量



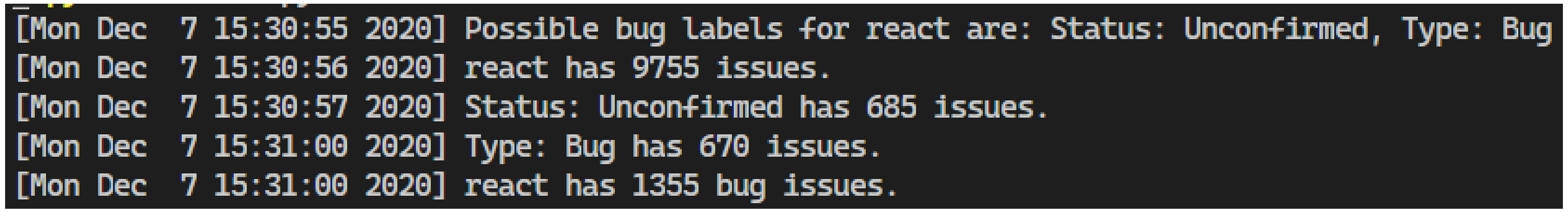
```python
# 在本机上执行shell命令
def execute_command(cmds: List[str], cwd: Optional[str] = None) -> str:
    process = subprocess.run(cmds, stdout=subprocess.PIPE, shell=True, cwd=cwd)
    return process.stdout.decode()
```



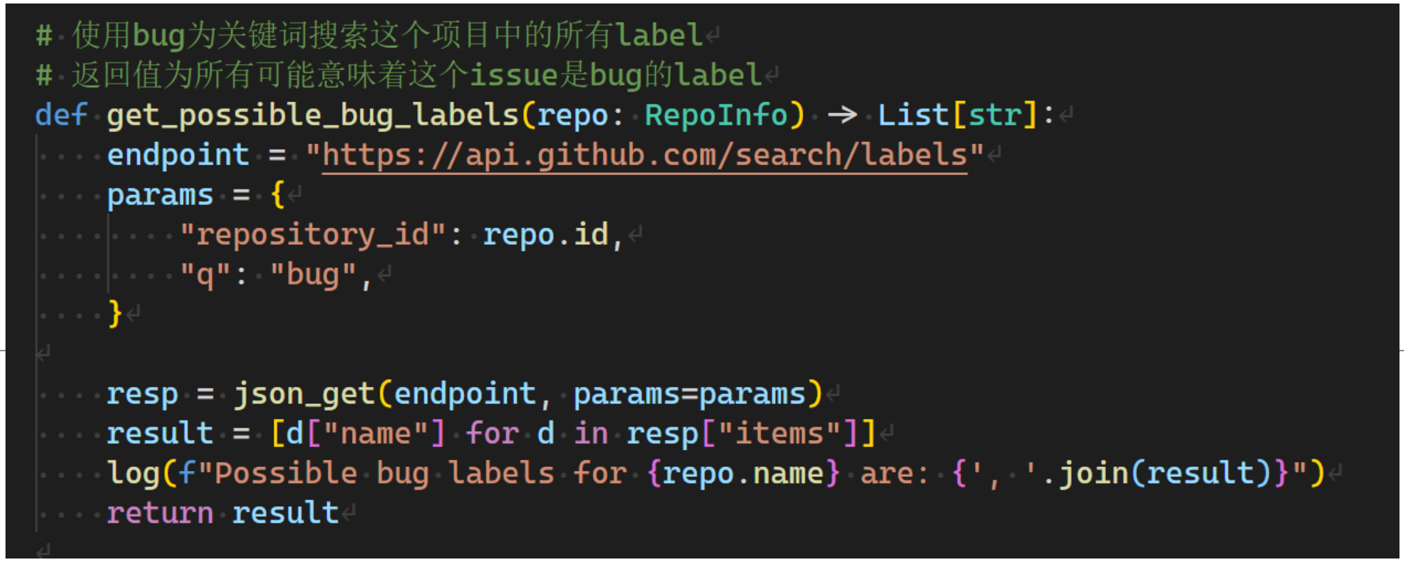
```python
@dataclass
class ClocResult:
    js: int
    ts: int

def cloc(project_name: str) -> ClocResult: 
    output = execute_command(
        ["cloc", f"projects/{project_name}", "--exclude-dir=node_modules"]
    )

    js, ts = 0, 0
    for line in output.splitlines():
        if line.startswith("JavaScript"):
            items = line.split()
            js = int(items[4])
        elif line.startswith("TypeScript"):
            items = line.split()
            ts = int(items[4])

    return ClocResult(js, ts)
```

每个项目的bug数量

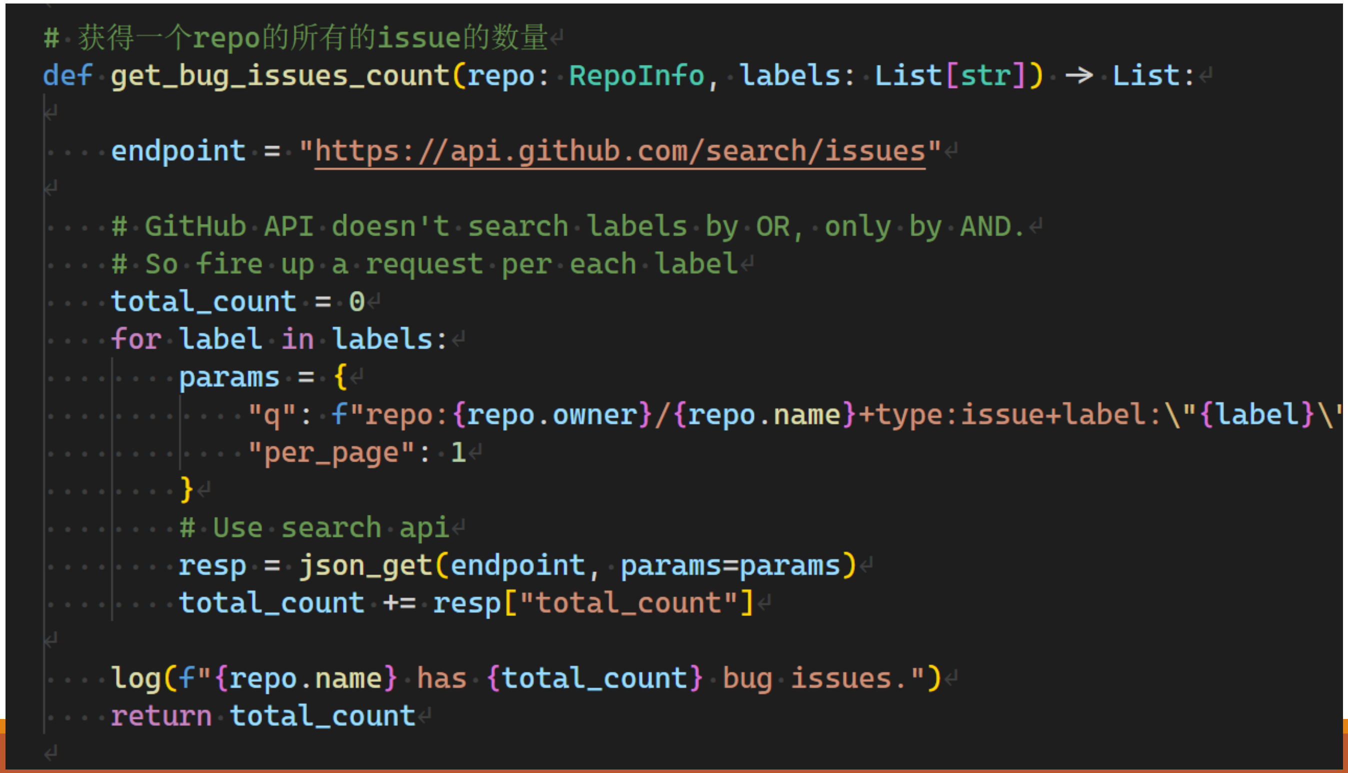
- 使用GitHub Issues统计bugs数量
- 1. 使用bug为关键词搜索这个项目中的所有可能是bug的label
  - 不同项目标识是bug的label不一样
  - React: Type: Bug
  - Vue: bug
  - Ant-design: 🐛 Bug
- 2. 使用**每个label**作为关键词，搜索带有这个label的issue的数量
- 3. 每个label的issue的数量之**和**就是这个项目的所有的issues的数量



```
[Mon Dec  7 15:30:55 2020] Possible bug labels for react are: Status: Unconfirmed, Type: Bug
[Mon Dec  7 15:30:56 2020] react has 9755 issues.
[Mon Dec  7 15:30:57 2020] Status: Unconfirmed has 685 issues.
[Mon Dec  7 15:31:00 2020] Type: Bug has 670 issues.
[Mon Dec  7 15:31:00 2020] react has 1355 bug issues.
```



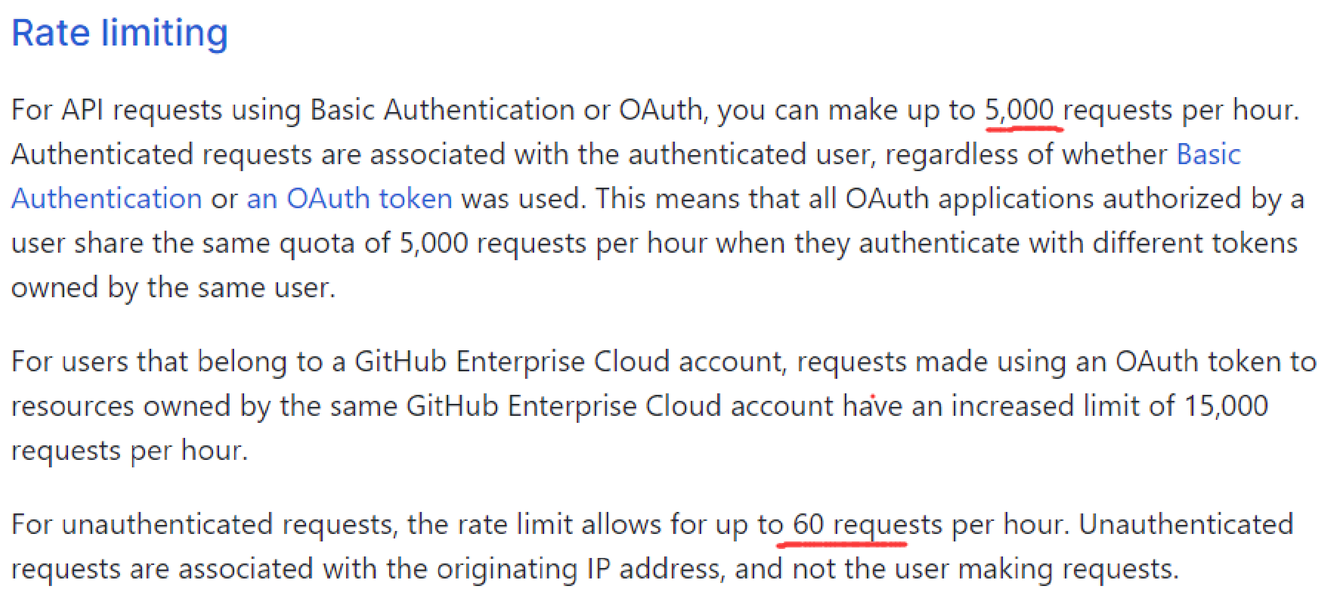
```python
# 使用bug为关键词搜索这个项目中的所有label
# 返回值为所有可能意味着这个issue是bug的label
def get_possible_bug_labels(repo: RepoInfo) -> List[str]:
    endpoint = "https://api.github.com/search/labels"
    params = {
        "repository_id": repo.id,
        "q": "bug",
    }

    resp = json_get(endpoint, params=params)
    result = [d["name"] for d in resp["items"]]
    log(f"Possible bug labels for {repo.name} are: {', '.join(result)}")
    return result
```



```python
# 获得一个repo的所有的issue的数量
def get_bug_issues_count(repo: RepoInfo, labels: List[str]) -> List:

    endpoint = "https://api.github.com/search/issues"

    # GitHub API doesn't search labels by OR, only by AND.
    # So fire up a request per each label
    total_count = 0
    for label in labels:
        params = {
            "q": f"repo:{repo.owner}/{repo.name}+type:issue+label:\"{label}\"
            "per_page": 1
        }
        # Use search api
        resp = json_get(endpoint, params=params)
        total_count += resp["total_count"]

    log(f"{repo.name} has {total_count} bug issues.")
    return total_count
```

问题：GitHub API速率限制

- GitHub的API有访问速率限制
- 超过速率的请求将会返回403，并带有重置时间
- 处理方法：
  - 使用GitHub账号登录
  - 被限制后，sleep，重置时间后继续

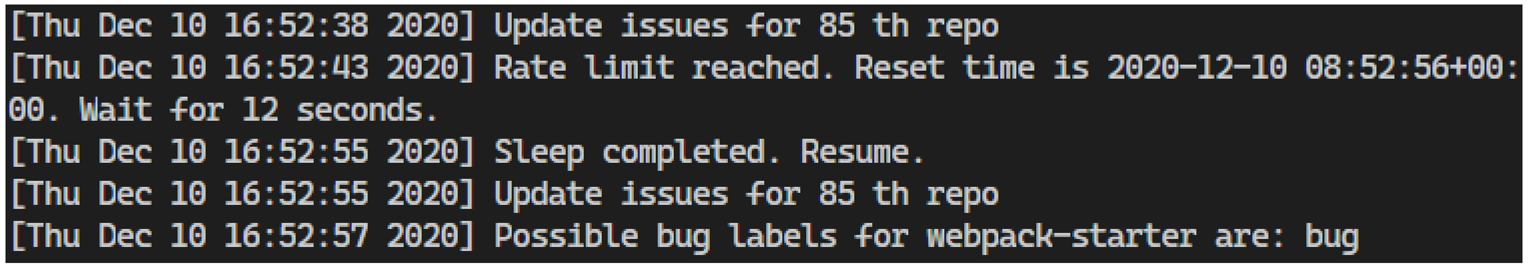


Rate limiting

For API requests using Basic Authentication or OAuth, you can make up to 5,000 requests per hour. Authenticated requests are associated with the authenticated user, regardless of whether Basic Authentication or an OAuth token was used. This means that all OAuth applications authorized by a user share the same quota of 5,000 requests per hour when they authenticate with different tokens owned by the same user.

For users that belong to a GitHub Enterprise Cloud account, requests made using an OAuth token to resources owned by the same GitHub Enterprise Cloud account have an increased limit of 15,000 requests per hour.

For unauthenticated requests, the rate limit allows for up to 60 requests per hour. Unauthenticated requests are associated with the originating IP address, and not the user making requests.



```
[Thu Dec 10 16:52:38 2020] Update issues for 85 th repo
[Thu Dec 10 16:52:43 2020] Rate limit reached. Reset time is 2020-12-10 08:52:56+00:
00. Wait for 12 seconds.
[Thu Dec 10 16:52:55 2020] Sleep completed. Resume.
[Thu Dec 10 16:52:55 2020] Update issues for 85 th repo
[Thu Dec 10 16:52:57 2020] Possible bug labels for webpack-starter are: bug
```

统计单元测试数量

框架获取

无法直接获取

静态分析

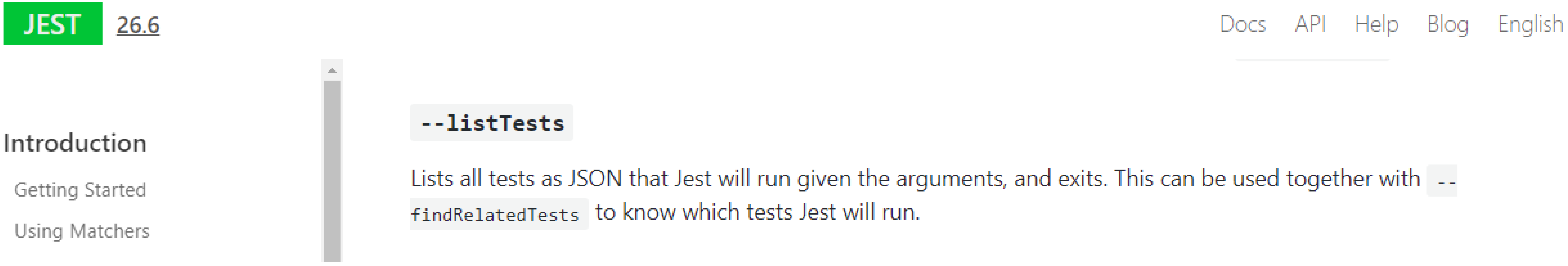
数据不准确
速度慢

动态执行

可能是唯一准确的方法

统计单元测试数量：框架获取

- 框架**无法直接**获取测试用例的数量

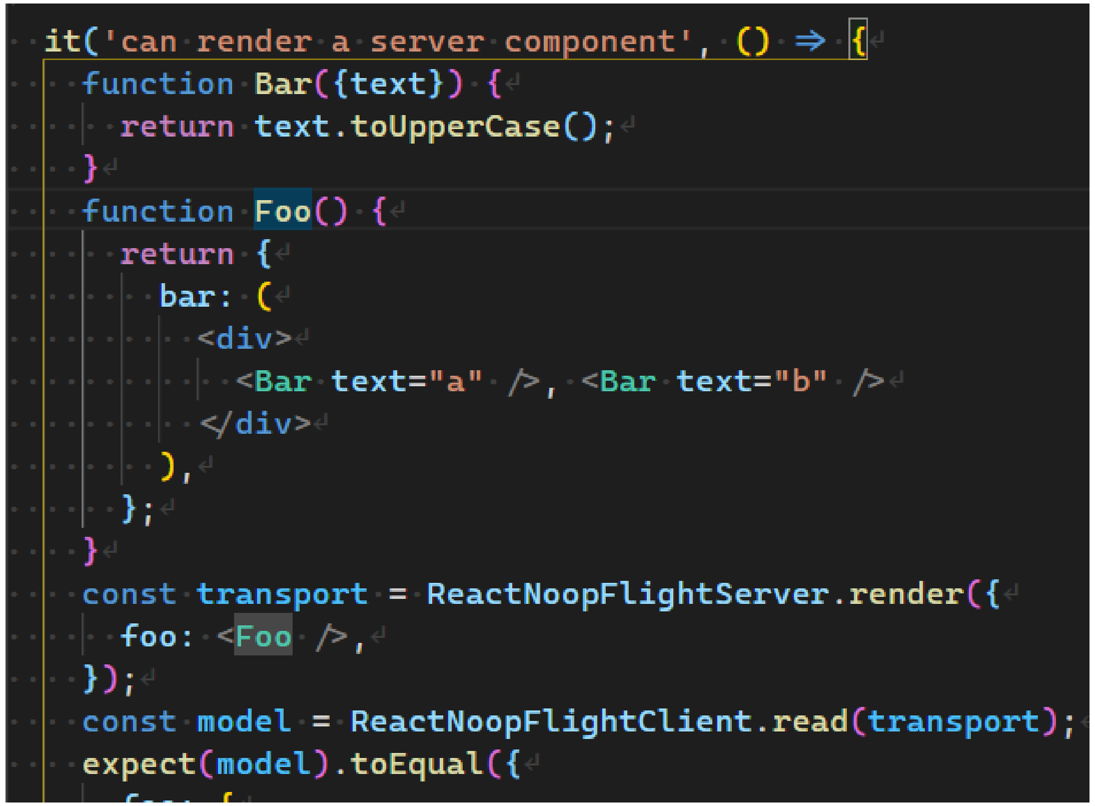


`--listTests`

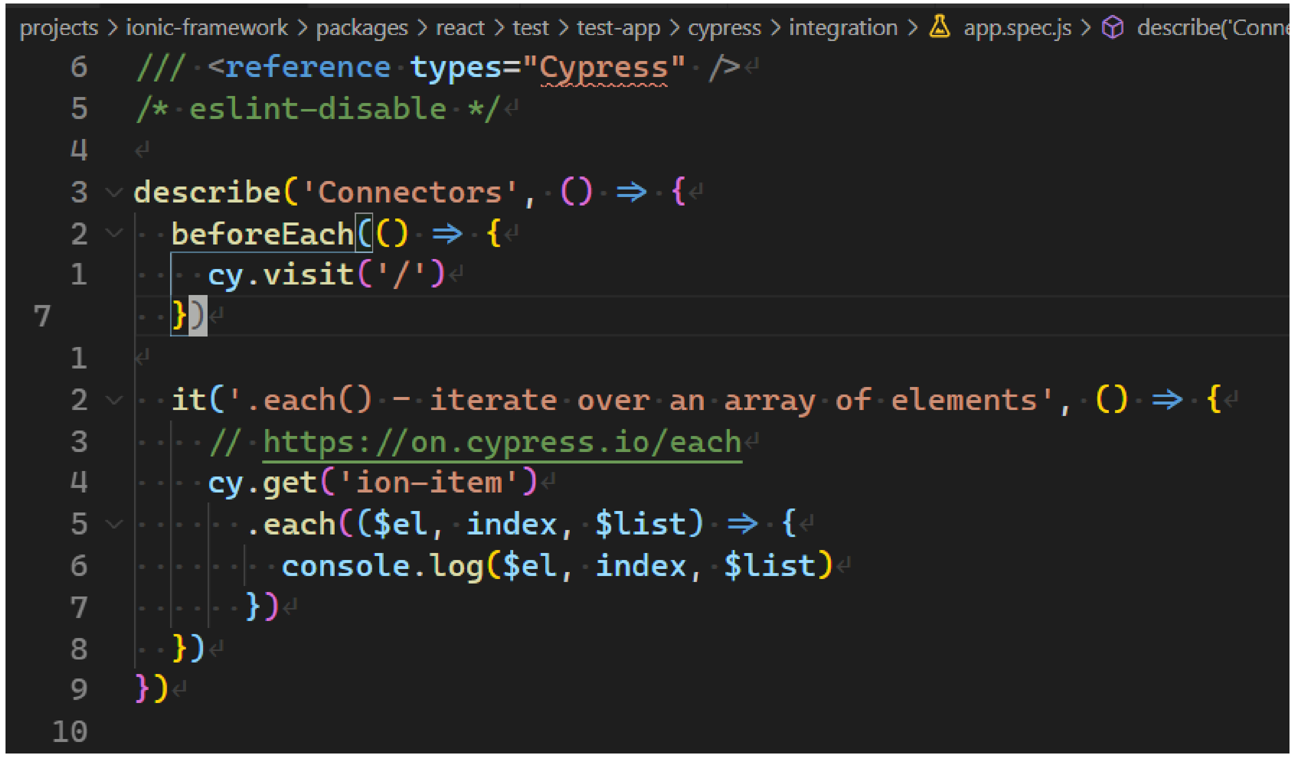
Lists all tests as JSON that Jest will run given the arguments, and exits. This can be used together with `--findRelatedTests` to know which tests Jest will run.

jest只能列出测试用例文件

统计单元测试数量：静态分析



```
it('can render a server component', () => {
  function Bar({text}) {
    return text.toUpperCase();
  }
  function Foo() {
    return {
      bar: (
        <div>
          <Bar text="a" />, <Bar text="b" />
        </div>
      ),
    };
  }
  const transport = ReactNoopFlightServer.render({
    foo: <Foo />,
  });
  const model = ReactNoopFlightClient.read(transport);
  expect(model).toEqual({
```



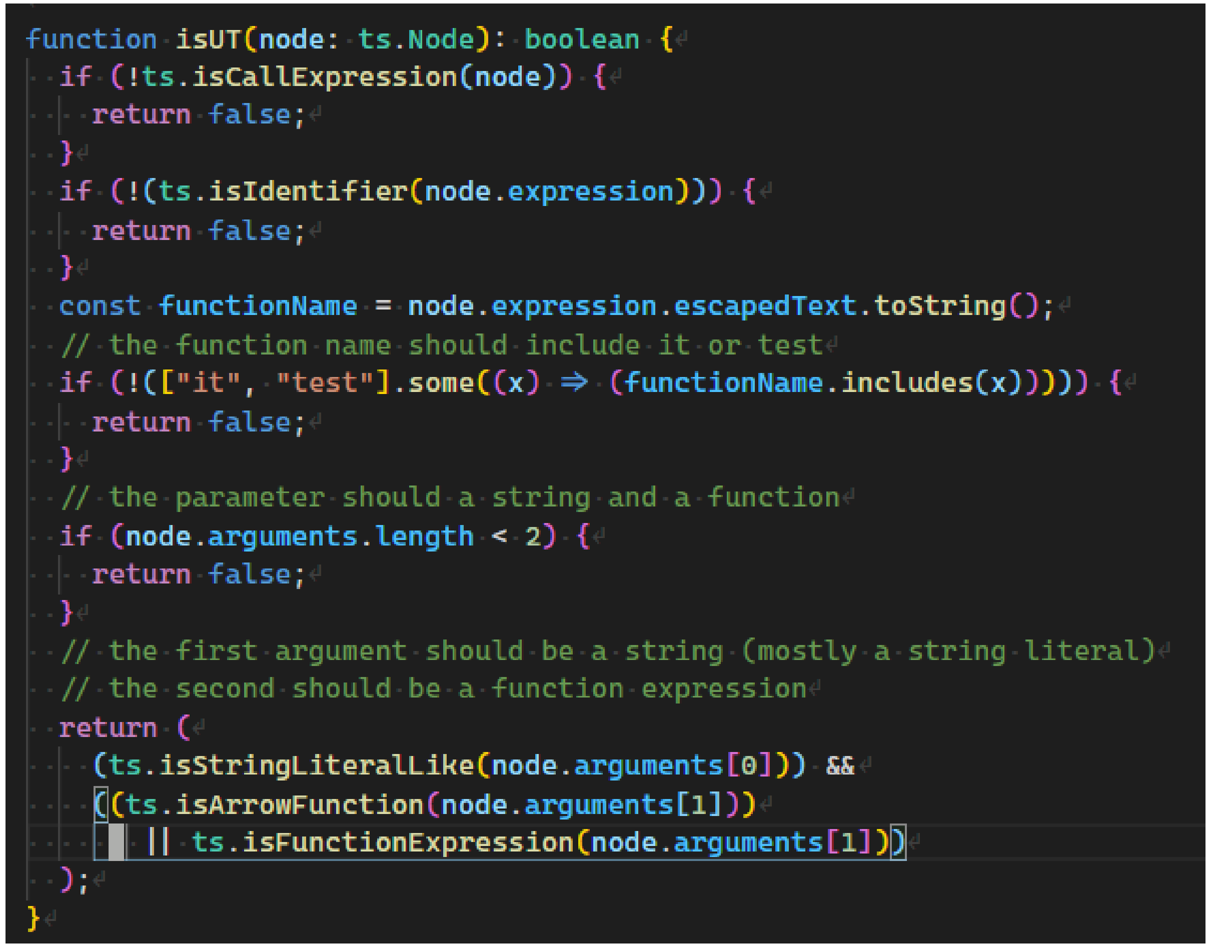
projects > ionic-framework > packages > react > test > test-app > cypress > integration > app.spec.js > describe('Conne

```
/// <reference types="Cypress" />
/* eslint-disable */

describe('Connectors', () => {
  beforeEach(() => {
    cy.visit('/')
  })

  it('.each() - iterate over an array of elements', () => {
    // https://on.cypress.io/each
    cy.get('ion-item')
      .each(($el, index, $list) => {
        console.log($el, index, $list)
      })
  })
})
```

测试大多都具有it(“”, () => {})的形式（左react-dom，右ionic-framework）

统计单元测试数量：静态分析

1. 遍历项目里的所有文件和目录
   1. 忽略node_modules
2. 如果一个目录叫test, tests, __test__, __tests__，那么目录下的文件是**测试用例文件**
3. 使用typescript API**测试用例文件**的AST
4. 统计满足以下条件的节点的数量（右图）
   1. 是一个**函数调用**节点
   2. 函数名包含**it**或者**test**
   3. 参数数大于2
   4. 第一个参数是一个**字符串常量**
   5. 第二个参数是一个**函数定义**
5. 这样的节点的数量是测试用例数量

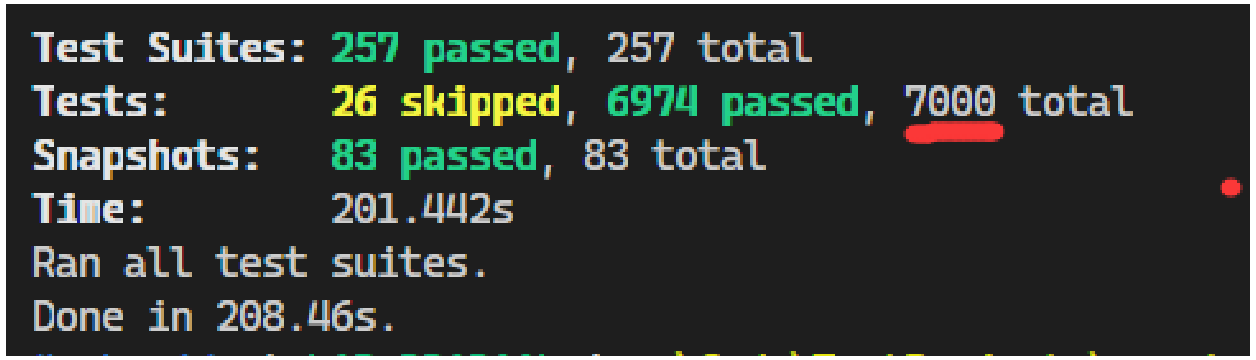


```
function isUT(node: ts.Node): boolean {
  if (!ts.isCallExpression(node)) {
    return false;
  }
  if (!(ts.isIdentifier(node.expression))) {
    return false;
  }
  const functionName = node.expression.escapedText.toString();
  // the function name should include it or test
  if (!(["it", "test"].some((x) => (functionName.includes(x))))) {
    return false;
  }
  // the parameter should a string and a function
  if (node.arguments.length < 2) {
    return false;
  }
  // the first argument should be a string (mostly a string literal)
  // the second should be a function expression
  return (
    (ts.isStringLiteralLike(node.arguments[0])) &&
    ((ts.isArrowFunction(node.arguments[1]))
      || ts.isFunctionExpression(node.arguments[1]))
  );
}
```

判断一个节点是否是测试用例

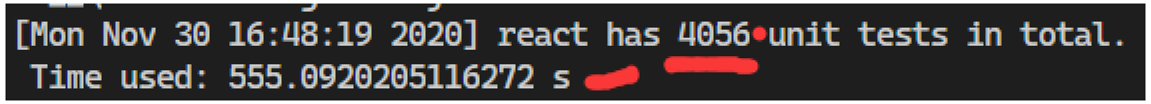
统计单元测试数量：静态分析

拿react项目进行测试：**数据不准确，速度慢**



```
Test Suites: 257 passed, 257 total
Tests:       26 skipped, 6974 passed, 7000 total
Snapshots:   83 passed, 83 total
Time:        201.442s
Ran all test suites.
Done in 208.46s.
```

真实情况



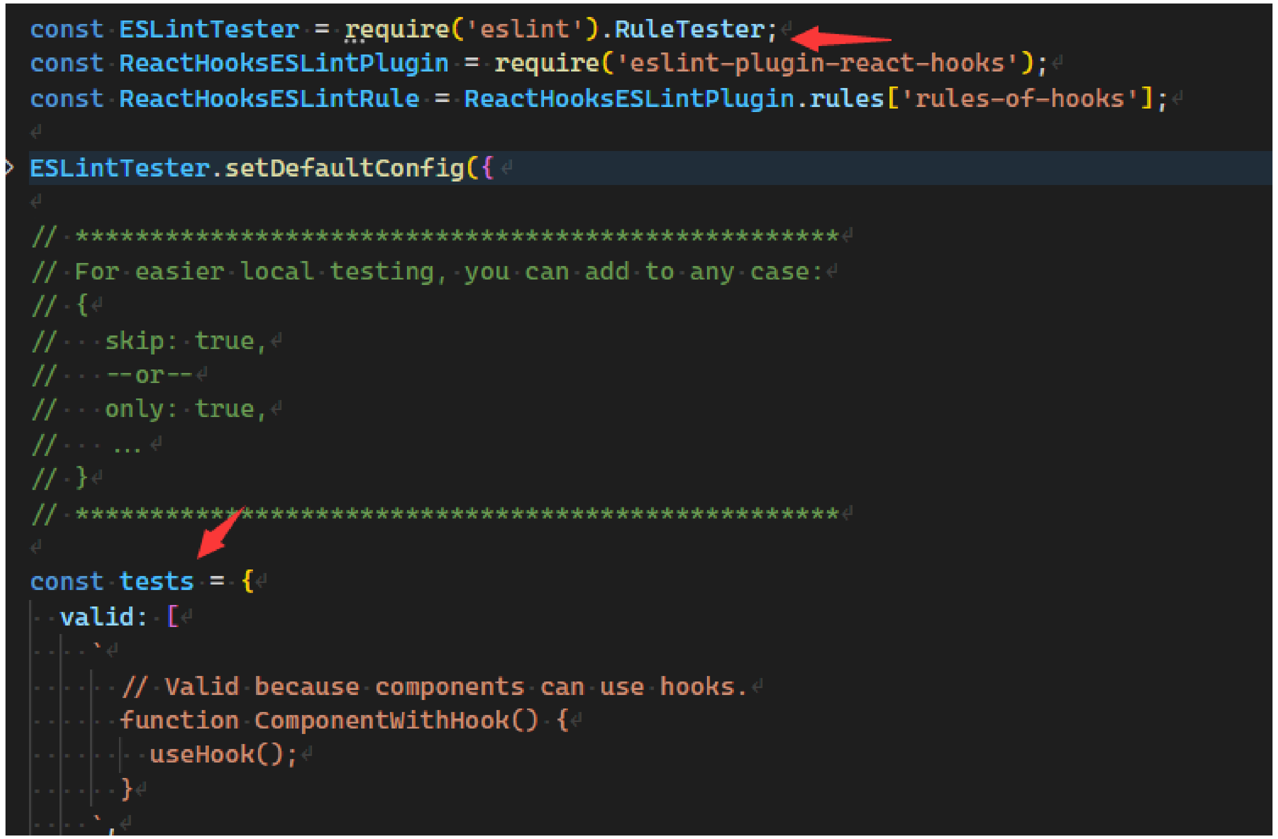
```
[Mon Nov 30 16:48:19 2020] react has 4056 unit tests in total.
 Time used: 555.0920205116272 s
```

此方法结果

统计单元测试数量：静态分析

数据不准确的可能原因：

1. 使用其他测试框架



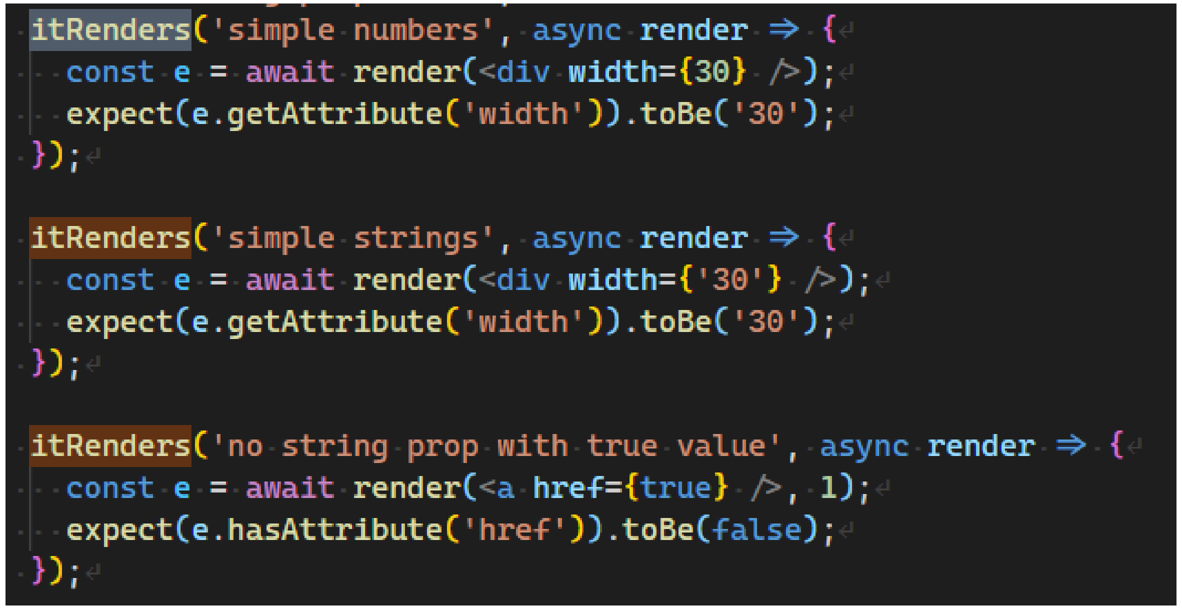
```
const ESLintTester = require('eslint').RuleTester;
const ReactHooksESLintPlugin = require('eslint-plugin-react-hooks');
const ReactHooksESLintRule = ReactHooksESLintPlugin.rules['rules-of-hooks'];

ESLintTester.setDefaultConfig({

// ***************************************************
// For easier local testing, you can add to any case:
// {
//   skip: true,
//   --or--
//   only: true,
//   ...
// }
// ***************************************************

const tests = {
  valid: [
    `
      // Valid because components can use hooks.
      function ComponentWithHook() {
        useHook();
      }
    `,
```

Eslint规则的相关测试使用了其他库

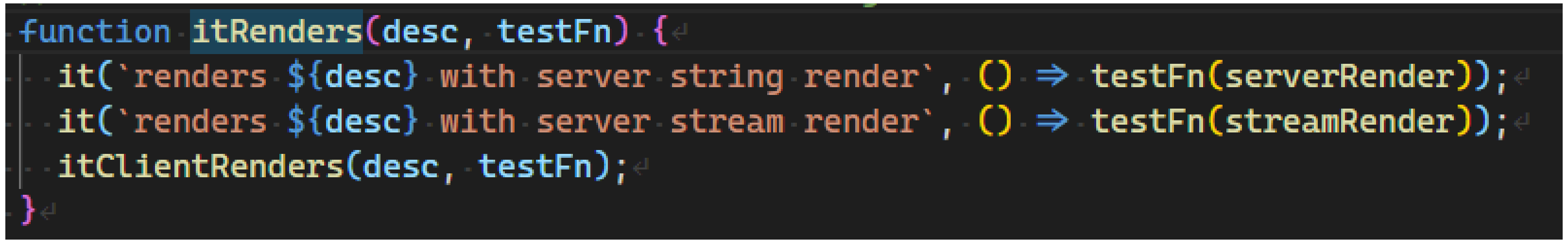
统计单元测试数量：静态分析

数据不准确的可能原因：

2. 使用了自己包装的测试方法
   - React-dom包里大量使用自己包装的itRenders方法
   - 每个itRenders是两个测试用例
   - 右图只能获得5个测试用例，实际>6个



```
itRenders('simple numbers', async render => {
  const e = await render(<div width={30} />);
  expect(e.getAttribute('width')).toBe('30');
});

itRenders('simple strings', async render => {
  const e = await render(<div width={'30'} />);
  expect(e.getAttribute('width')).toBe('30');
});

itRenders('no string prop with true value', async render => {
  const e = await render(<a href={true} />, 1);
  expect(e.hasAttribute('href')).toBe(false);
});
```

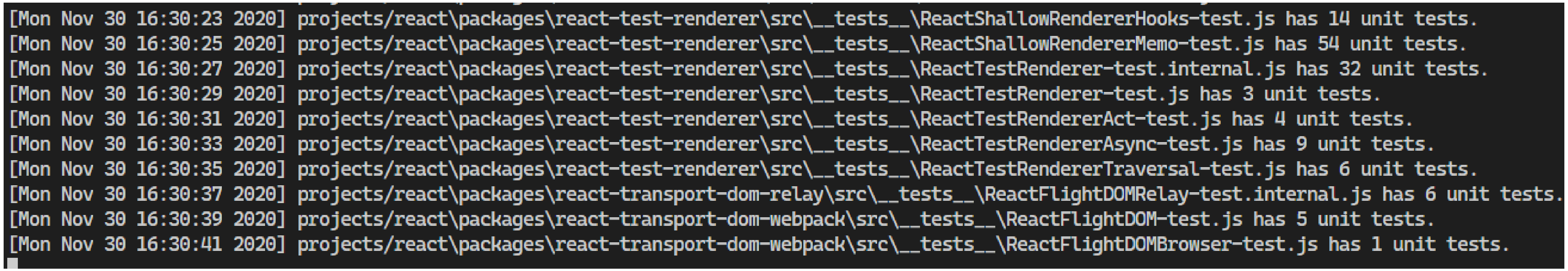


```
function itRenders(desc, testFn) {
  it(`renders ${desc} with server string render`, () => testFn(serverRender));
  it(`renders ${desc} with server stream render`, () => testFn(streamRender));
  itClientRenders(desc, testFn);
}
```

统计单元测试数量：静态分析

速度慢的可能原因：

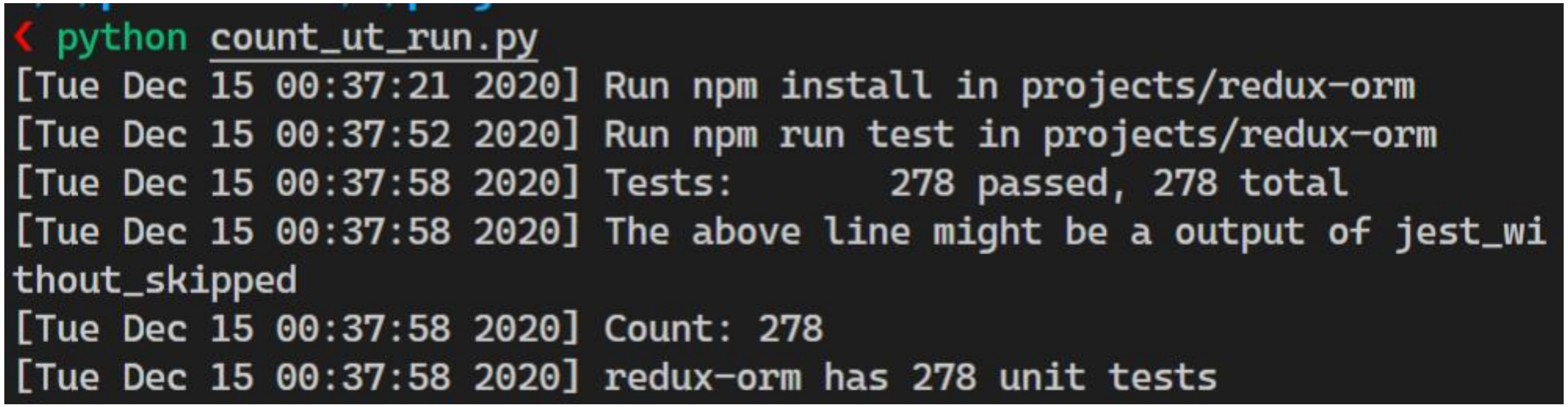
1. 整个过程未并行化，对每个文件串行化执行



```
[Mon Nov 30 16:30:23 2020] projects/react\packages\react-test-renderer\src\__tests__\ReactShallowRendererHooks-test.js has 14 unit tests.
[Mon Nov 30 16:30:25 2020] projects/react\packages\react-test-renderer\src\__tests__\ReactShallowRendererMemo-test.js has 54 unit tests.
[Mon Nov 30 16:30:27 2020] projects/react\packages\react-test-renderer\src\__tests__\ReactTestRenderer-test.internal.js has 32 unit tests.
[Mon Nov 30 16:30:29 2020] projects/react\packages\react-test-renderer\src\__tests__\ReactTestRenderer-test.js has 3 unit tests.
[Mon Nov 30 16:30:31 2020] projects/react\packages\react-test-renderer\src\__tests__\ReactTestRendererAct-test.js has 4 unit tests.
[Mon Nov 30 16:30:33 2020] projects/react\packages\react-test-renderer\src\__tests__\ReactTestRendererAsync-test.js has 9 unit tests.
[Mon Nov 30 16:30:35 2020] projects/react\packages\react-test-renderer\src\__tests__\ReactTestRendererTraversal-test.js has 6 unit tests.
[Mon Nov 30 16:30:37 2020] projects/react\packages\react-transport-dom-relay\src\__tests__\ReactFlightDOMRelay-test.internal.js has 6 unit tests.
[Mon Nov 30 16:30:39 2020] projects/react\packages\react-transport-dom-webpack\src\__tests__\ReactFlightDOM-test.js has 5 unit tests.
[Mon Nov 30 16:30:41 2020] projects/react\packages\react-transport-dom-webpack\src\__tests__\ReactFlightDOMBrowser-test.js has 1 unit tests.
```

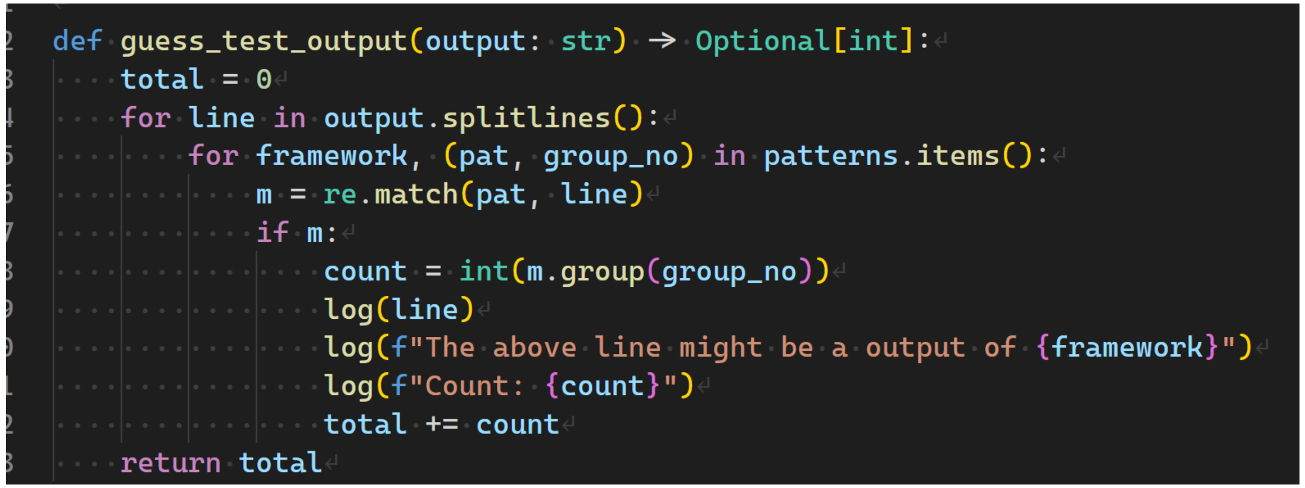
串行化执行

统计单元测试数量：动态执行

1. 安装依赖
2. 运行测试用例
3. 

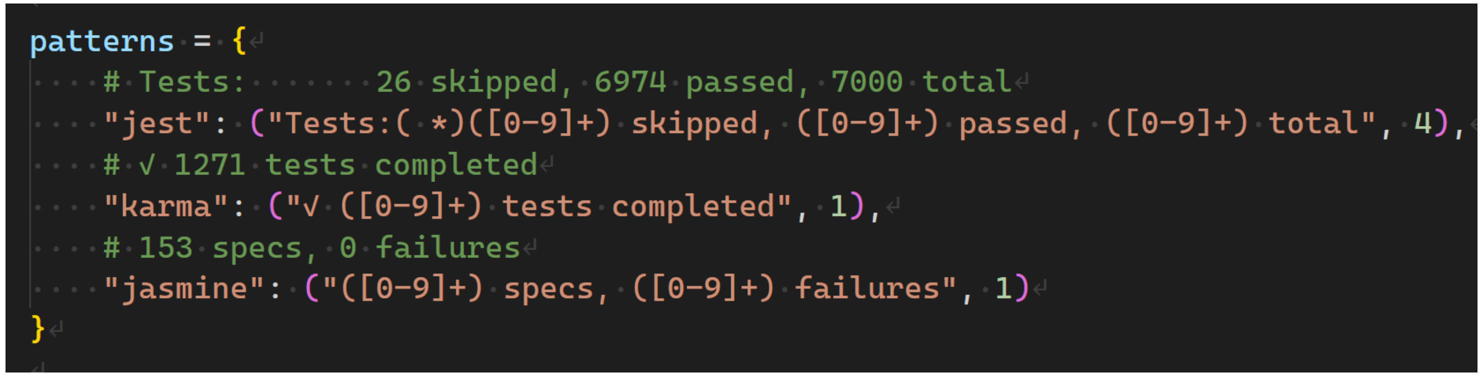
```
‹ python count_ut_run.py
[Tue Dec 15 00:37:21 2020] Run npm install in projects/redux-orm
[Tue Dec 15 00:37:52 2020] Run npm run test in projects/redux-orm
[Tue Dec 15 00:37:58 2020] Tests:       278 passed, 278 total
[Tue Dec 15 00:37:58 2020] The above line might be a output of jest_wi
thout_skipped
[Tue Dec 15 00:37:58 2020] Count: 278
[Tue Dec 15 00:37:58 2020] redux-orm has 278 unit tests
```

统计单元测试数量：动态执行



```python
def guess_test_output(output: str) -> Optional[int]:
    total = 0
    for line in output.splitlines():
        for framework, (pat, group_no) in patterns.items():
            m = re.match(pat, line)
            if m:
                count = int(m.group(group_no))
                log(line)
                log(f"The above line might be a output of {framework}")
                log(f"Count: {count}")
                total += count
    return total
```

对输出的每一行猜测其是不是测试框架输出的结果

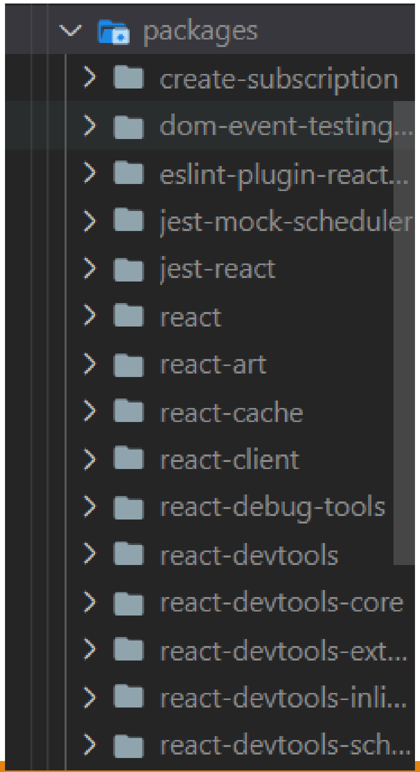


```python
patterns = {
    # Tests:       26 skipped, 6974 passed, 7000 total
    "jest": ("Tests:( *)([0-9]+) skipped, ([0-9]+) passed, ([0-9]+) total", 4),
    # √ 1271 tests completed
    "karma": ("√ ([0-9]+) tests completed", 1),
    # 153 specs, 0 failures
    "jasmine": ("([0-9]+) specs, ([0-9]+) failures", 1)
}
```

不同测试框架的输出模板

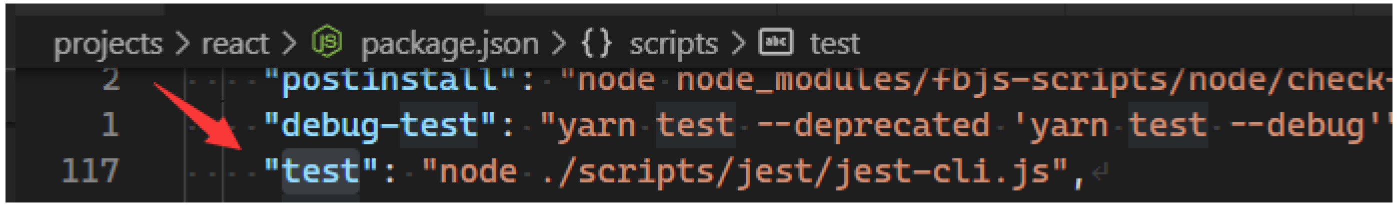
统计单元测试数量：动态执行

1. Monorepo：仓库里有多个npm包
2. 有的可以直接在根目录下运行所有测试用例（react），有的不可以（ionic-framework）



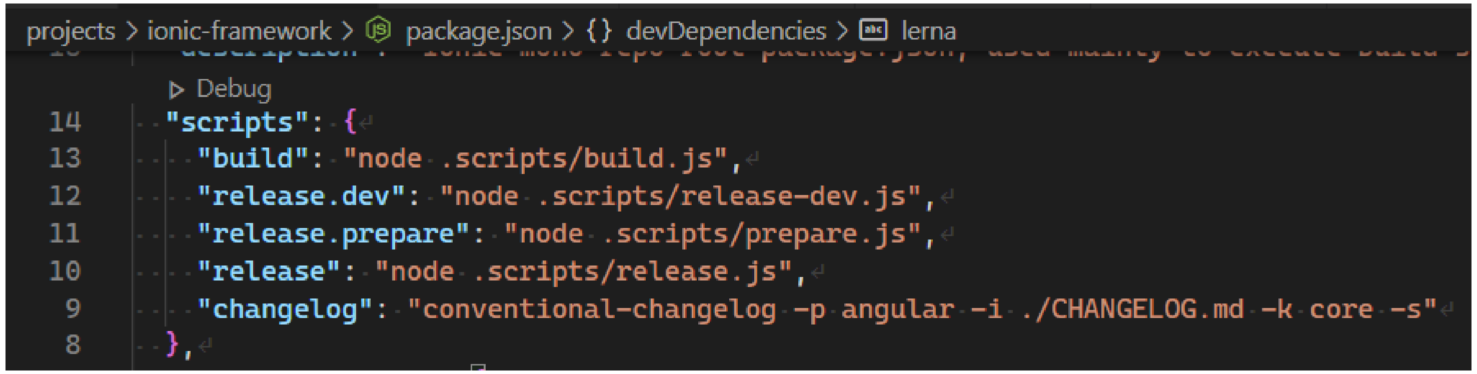
```
packages
  create-subscription
  dom-event-testing...
  eslint-plugin-react...
  jest-mock-scheduler
  jest-react
  react
  react-art
  react-cache
  react-client
  react-debug-tools
  react-devtools
  react-devtools-core
  react-devtools-ext...
  react-devtools-inli...
  react-devtools-sch...
```

React是一个monorepo



```
projects > react > package.json > {} scripts > test
    "postinstall": "node node_modules/fbjs-scripts/node/check-
    "debug-test": "yarn test --deprecated 'yarn test --debug''
    "test": "node ./scripts/jest/jest-cli.js",
```

React的根目录下可直接运行所有测试



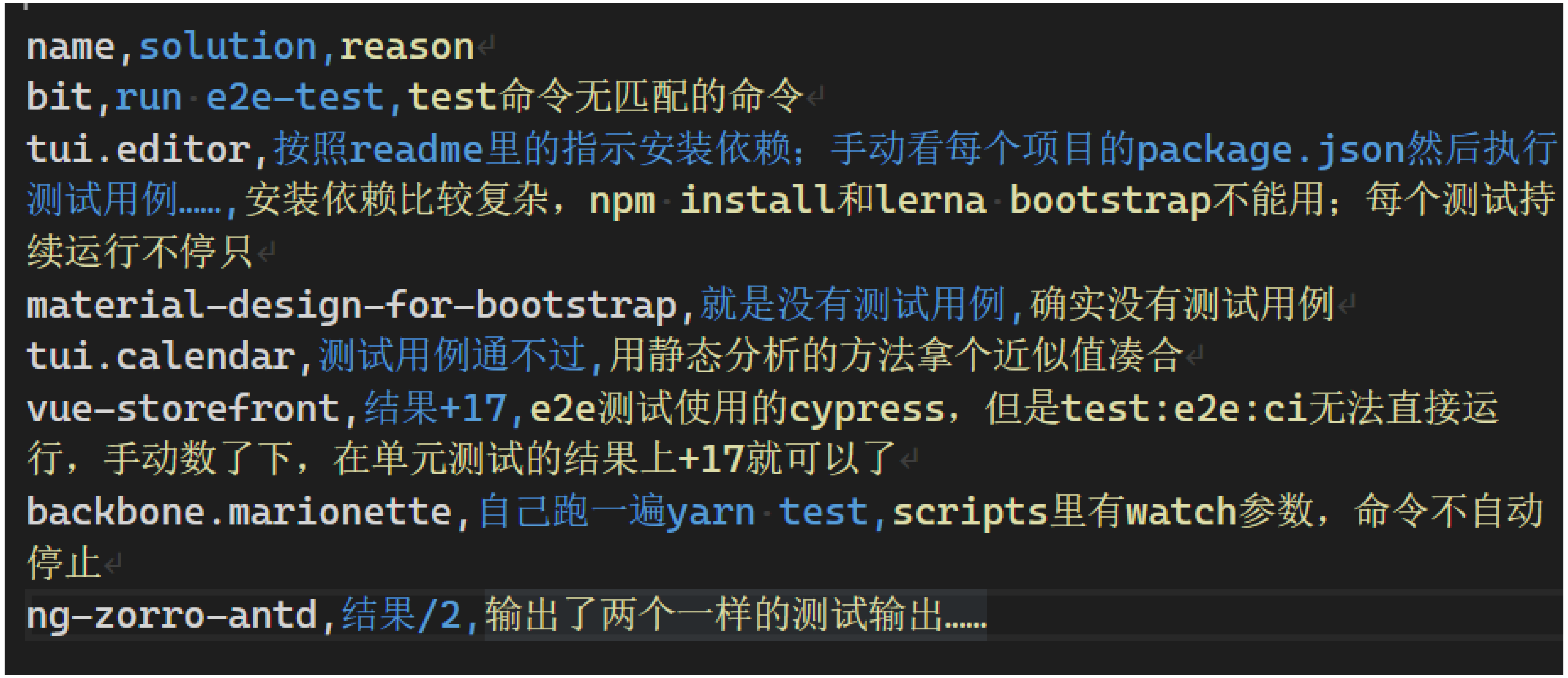
```
projects > ionic-framework > package.json > {} devDependencies > lerna
  ▷ Debug
  "scripts": {
    "build": "node .scripts/build.js",
    "release.dev": "node .scripts/release-dev.js",
    "release.prepare": "node .scripts/prepare.js",
    "release": "node .scripts/release.js",
    "changelog": "conventional-changelog -p angular -i ./CHANGELOG.md -k core -s"
  },
```

Ionic-framework的根目录不能直接执行所有测试

统计测试用例数量：动态执行

1. 如果根目录没有test:ci或者test命令，输出名字带有test的script，询问用户运行哪些命令
2. 如果没有测试命令，则有可能是monorepo
3. Lerna是常用的管理monorepo的工具，通过lerna查找所有项目的目录
    1. 如果不是lerna，则尝试找packages或者pkg目录下的项目
4. 对每个子项目跑一次测试（步骤1）
5. 所有子项目测试用例数量之和为整个项目的测试用例之和
6. 遇到意外情况，手动进行或者**使用静态方法**
    1. 脚本报错，超时，没有找到测试用例……

统计测试用例数量：动态执行

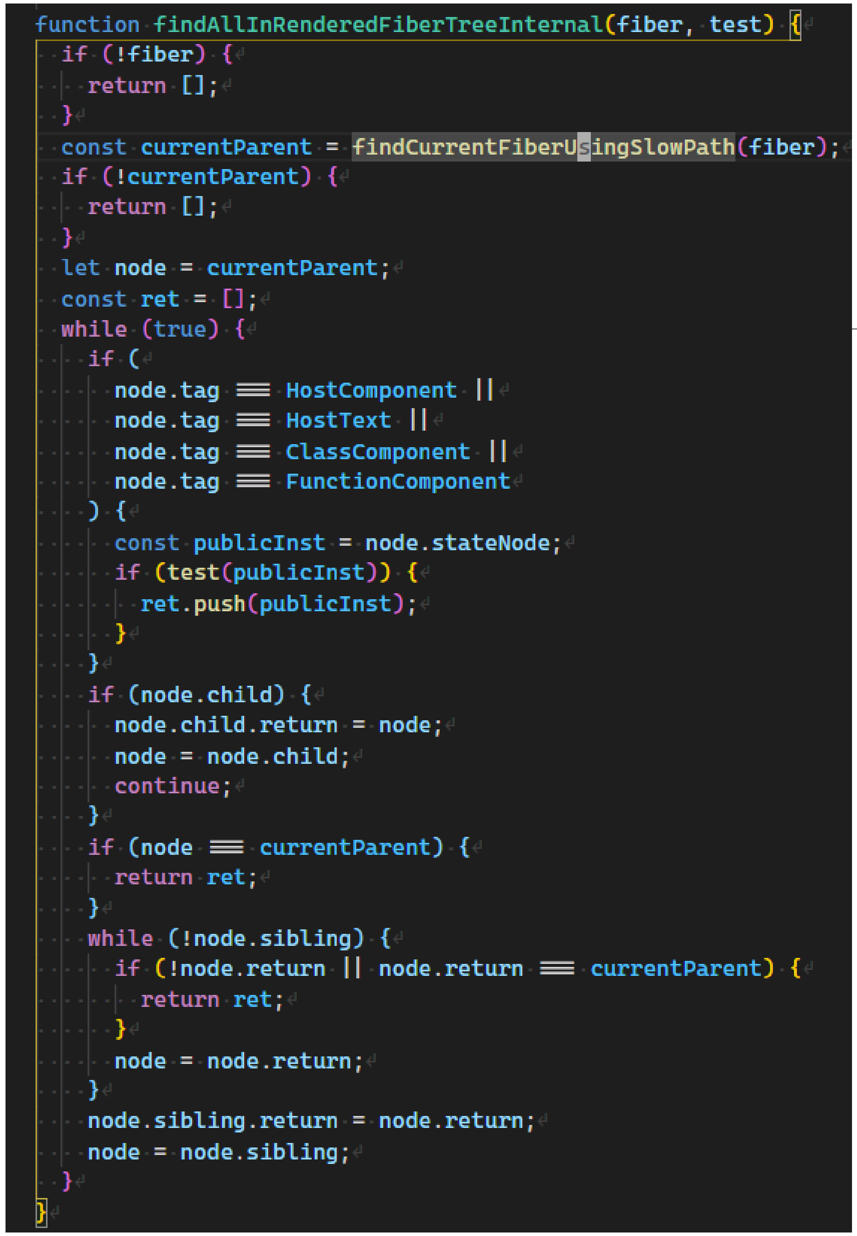


```
name,solution,reason
bit,run e2e-test,test命令无匹配的命令
tui.editor,按照readme里的指示安装依赖；手动看每个项目的package.json然后执行测试用例……,安装依赖比较复杂，npm install和lerna bootstrap不能用；每个测试持续运行不停只
material-design-for-bootstrap,就是没有测试用例,确实没有测试用例
tui.calendar,测试用例通不过,用静态分析的方法拿个近似值凑合
vue-storefront,结果+17,e2e测试使用的cypress，但是test:e2e:ci无法直接运行，手动数了下，在单元测试的结果上+17就可以了
backbone.marionette,自己跑一遍yarn test,scripts里有watch参数，命令不自动停止
ng-zorro-antd,结果/2,输出了两个一样的测试输出……
```

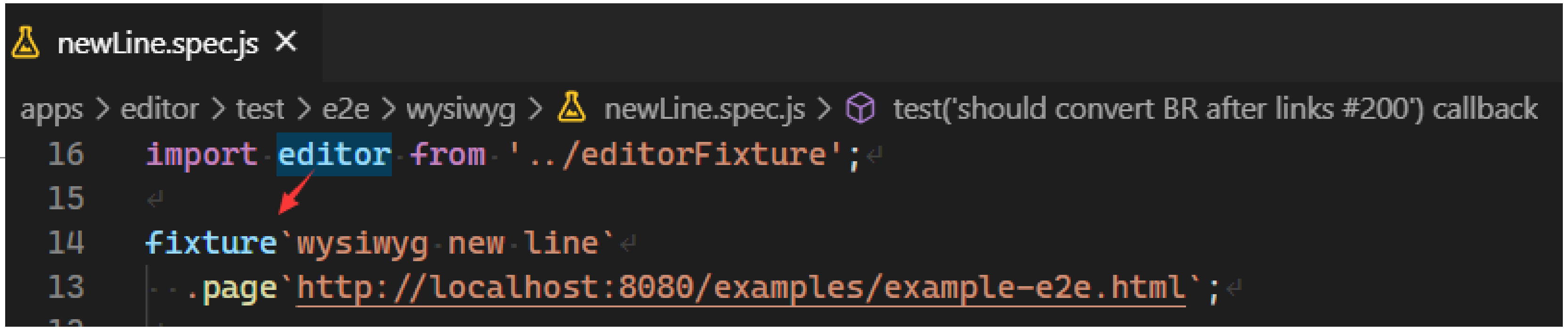
意外情况

DOM测试用例

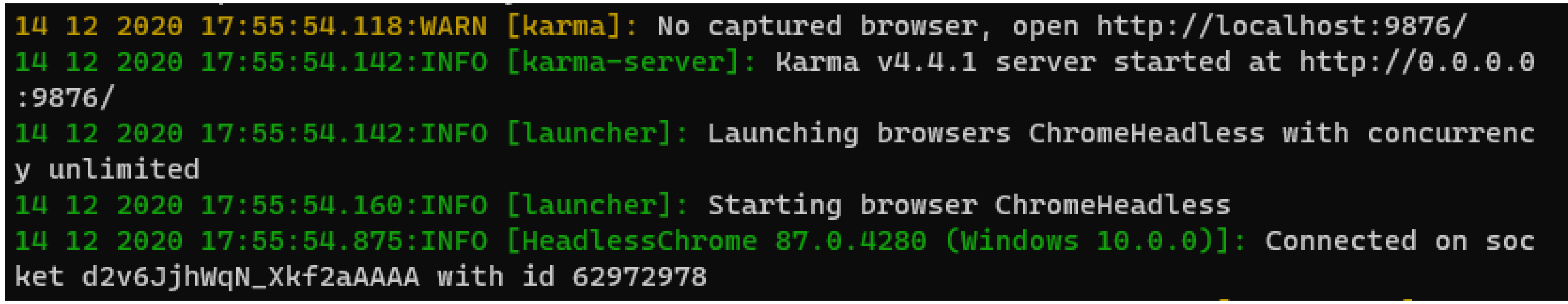
- DOM测试用例无法有效准确获取
- DOM测试库较多
  - 使用模拟DOM的JS的数据结构
  - 使用真正的浏览器（headless）
- 函数调用关系复杂
  - 和静态分析遇到的问题相同



```
function findAllInRenderedFiberTreeInternal(fiber, test) {
  if (!fiber) {
    return [];
  }
  const currentParent = findCurrentFiberUsingSlowPath(fiber);
  if (!currentParent) {
    return [];
  }
  let node = currentParent;
  const ret = [];
  while (true) {
    if (
      node.tag === HostComponent ||
      node.tag === HostText ||
      node.tag === ClassComponent ||
      node.tag === FunctionComponent
    ) {
      const publicInst = node.stateNode;
      if (test(publicInst)) {
        ret.push(publicInst);
      }
    }
    if (node.child) {
      node.child.return = node;
      node = node.child;
      continue;
    }
    if (node === currentParent) {
      return ret;
    }
    while (!node.sibling) {
      if (!node.return || node.return === currentParent) {
        return ret;
      }
      node = node.return;
    }
    node.sibling.return = node.return;
    node = node.sibling;
  }
}
```



```
newLine.spec.js ×
apps > editor > test > e2e > wysiwyg > newLine.spec.js > test('should convert BR after links #200') callback
import editor from '../editorFixture';

fixture`wysiwyg new line`
  .page`http://localhost:8080/examples/example-e2e.html`;
```



```
14 12 2020 17:55:54.118:WARN [karma]: No captured browser, open http://localhost:9876/
14 12 2020 17:55:54.142:INFO [karma-server]: Karma v4.4.1 server started at http://0.0.0.0
:9876/
14 12 2020 17:55:54.142:INFO [launcher]: Launching browsers ChromeHeadless with concurrenc
y unlimited
14 12 2020 17:55:54.160:INFO [launcher]: Starting browser ChromeHeadless
14 12 2020 17:55:54.875:INFO [HeadlessChrome 87.0.4280 (Windows 10.0.0)]: Connected on soc
ket d2v6JjhWqN_Xkf2aAAAA with id 62972978
```

（上）Tui.editor使用真正的浏览器（headless chrome）进行测试

（左）React-dom使用它自己的fiber数据结构进行DOM测试

后续

- **数据收集**是难点
- 获取项目的单元测试用例个数，获取数据
  - Star稍微少的项目结构较为单纯并符合前端惯例，可能获取数据更容易
  - 项目越大、越复杂、时间越长，项目结构可能越不标准（历史原因、惯例方案局限性）
- 一周获取数据
- 一周进行分析，得出结论，撰写实验报告

谢谢！