



# Redmineの脆弱性診断を 「Ruby × Selenium」を使って自動化

ファーエントテクノロジー株式会社 坂本 想

# 自己紹介



坂本 想

Sakamoto So

## ■ 所属

- ・ファーエンドテクノロジー株式会社  
入社3年目

## ■ 業務

- ・既存プロダクトの改善
- ・テストの自動化

# アジェンダ

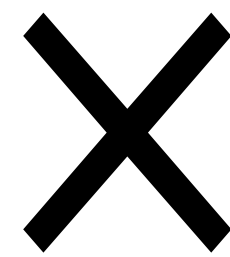
- 1 脆弱性診断の必要性
- 2 Redmine脆弱性診断の流れ
- 3 Ruby × Selenium メソッドの紹介
- 4 脆弱性診断を自動化してみても
- 5 本日のまとめ

# 1. 脆弱性診断の必要性

---

# 脆弱性診断の定期実行

Redmineの脆弱性診断を定期的にしていきます！



# 脆弱性とは？

- サービスの**問題点**

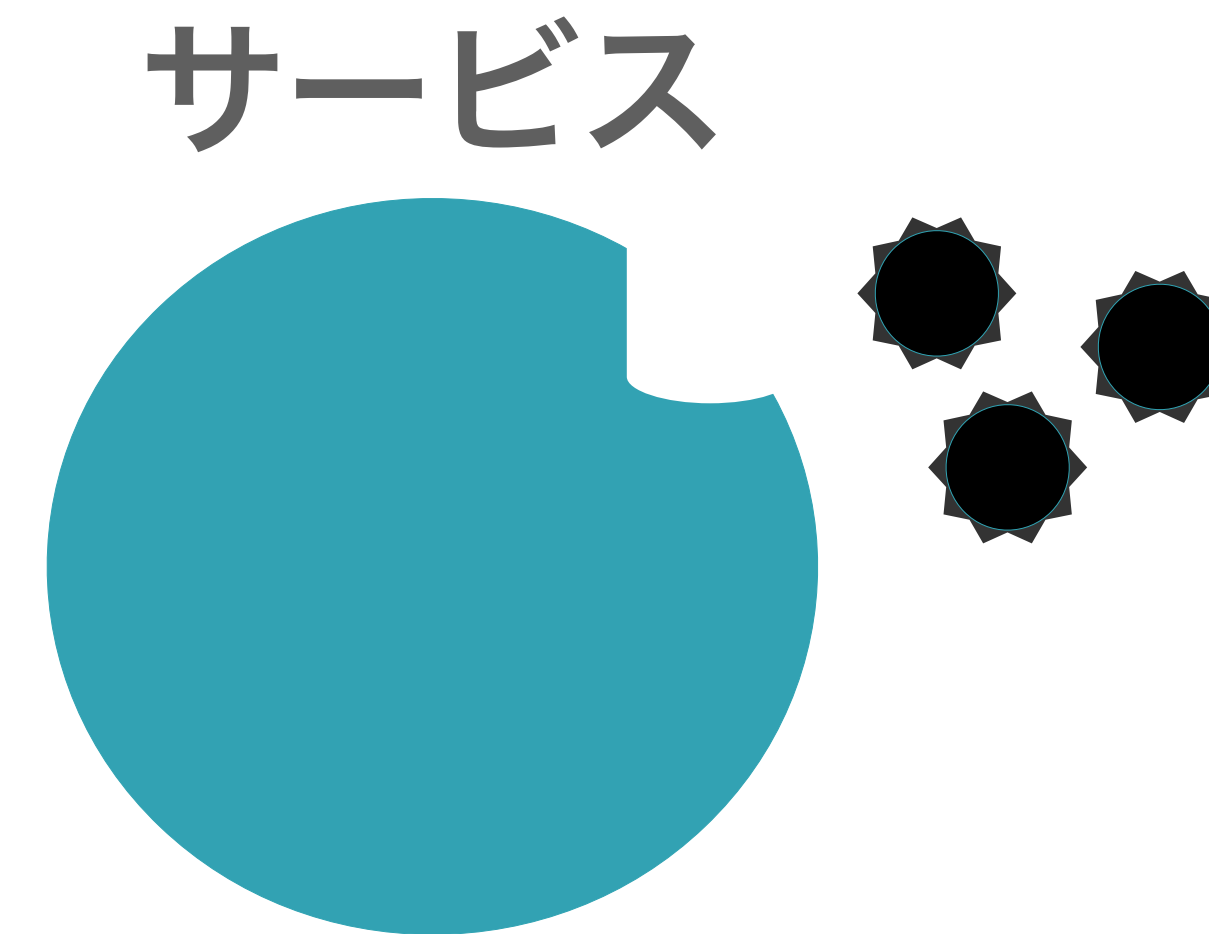


# 脆弱性とは？

- サービスの**問題点**

- 脆弱性があると、**ウイルス感染の危険性**がある

- ・個人情報漏洩
- ・サイバー攻撃



# 脆弱性診断とは

- サービスの**問題点**がないか**チェック**する

サービス





# 脆弱性診断とは

- サービスの**問題点**がないか**チェック**する
- より**品質の高いサービス**を提供できる

サービス



## 2. Redmine脆弱性診断の流れ

---

# 脆弱性診断の流れ

大きく分けて**2つ**のステップ

Step 1

- クロールデータ の作成
  - ・リンクをクリックするなどのRedmineの操作を記録したデータ

# 脆弱性診断の流れ

## 大きく分けて2つのステップ

### Step1

#### ■ クロールデータ の作成

- ・リンクをクリックするなどのRedmineの操作を記録したデータ

### Step2

#### ■ Scan

- ・クロールデータ を選択して、脆弱性診断を開始

# 脆弱性診断の流れ

本日メイン



## Step1

### ■ クロールデータ の作成

- ・リンクをクリックするなどのRedmineの操作を記録したデータ

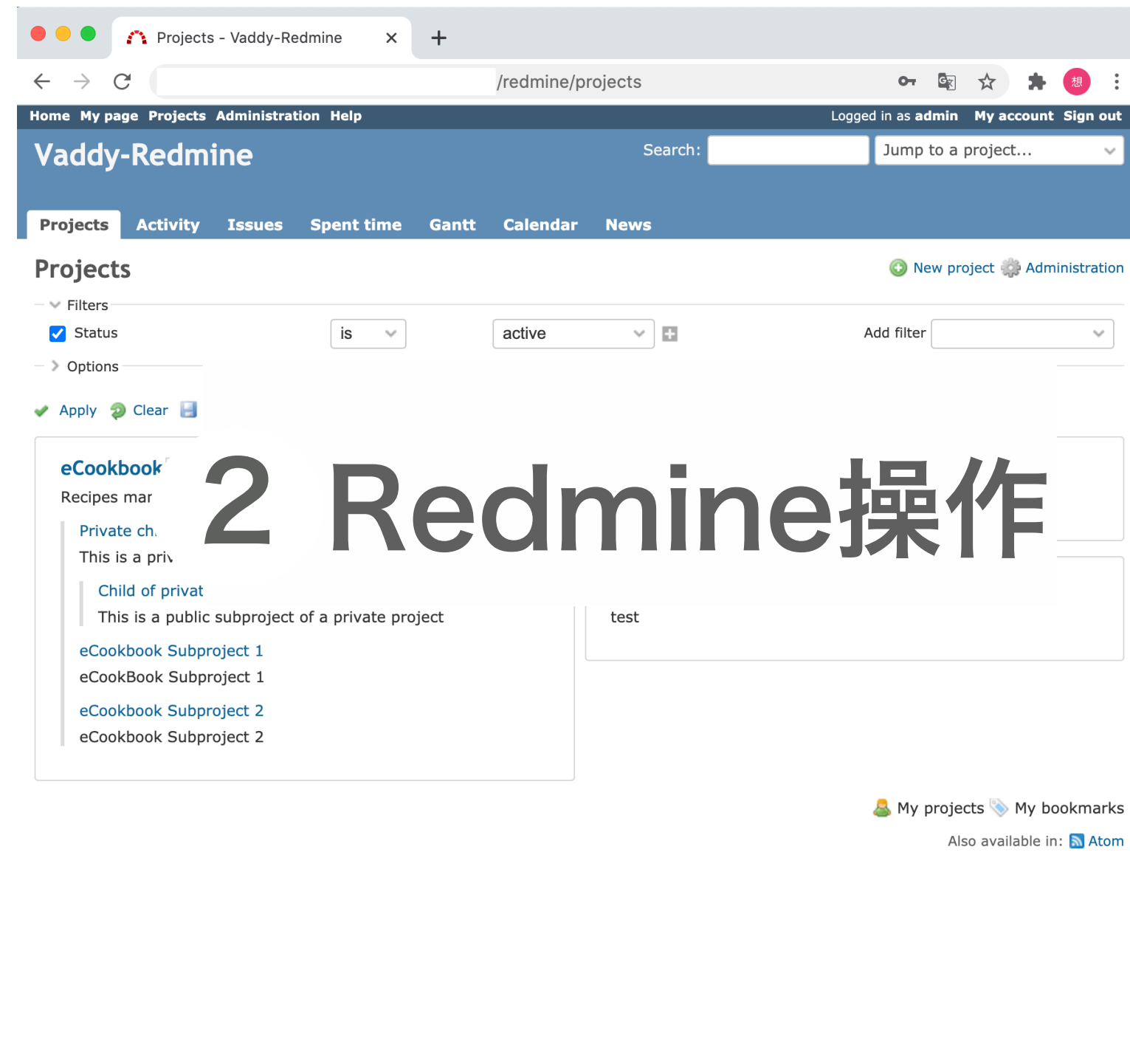
## Step2

### ■ Scan

- ・クロールデータ を選択して、脆弱性診断を開始

# クローラデータの作成

## Step 1



2 Redmine操作

1

Proxy

3

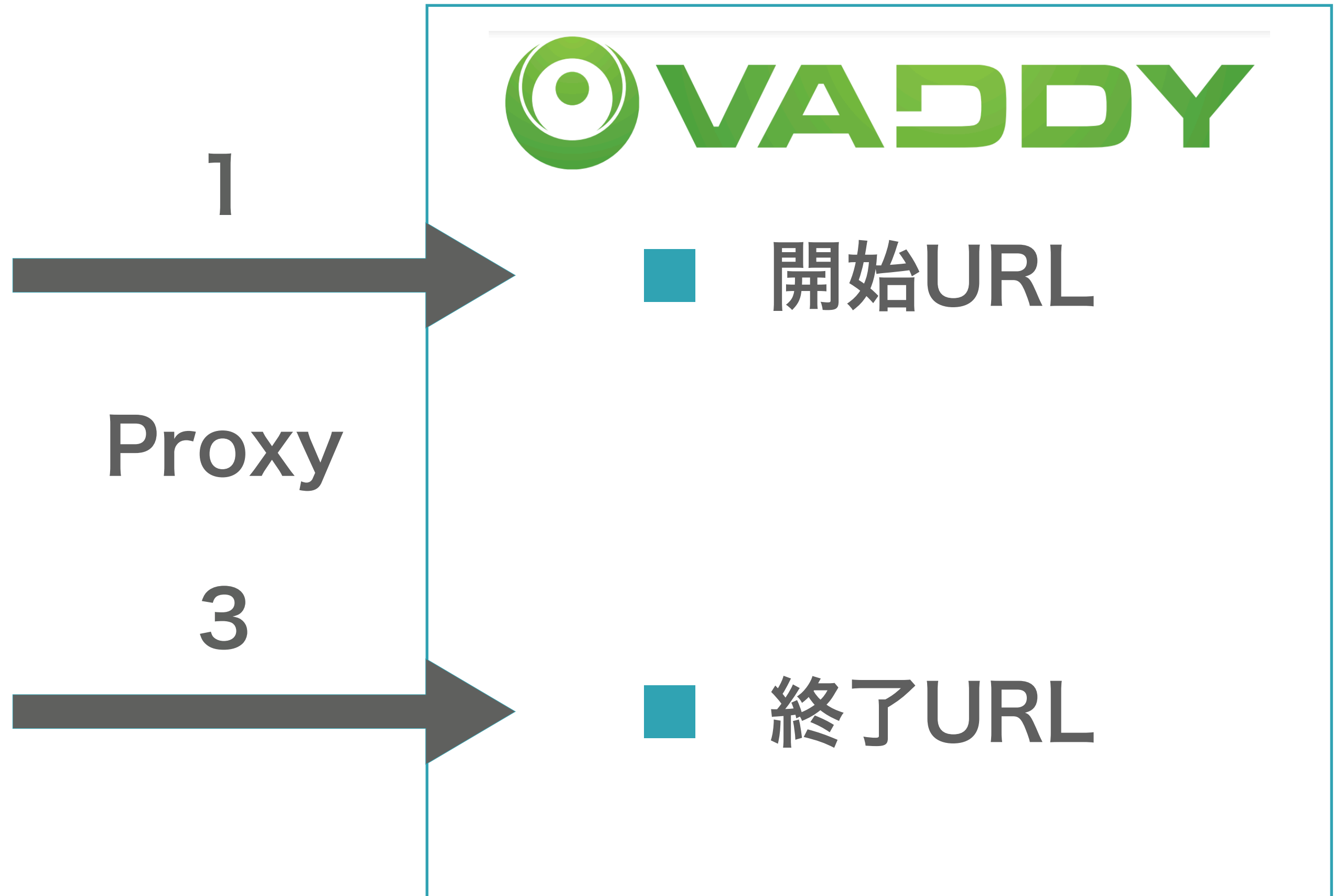
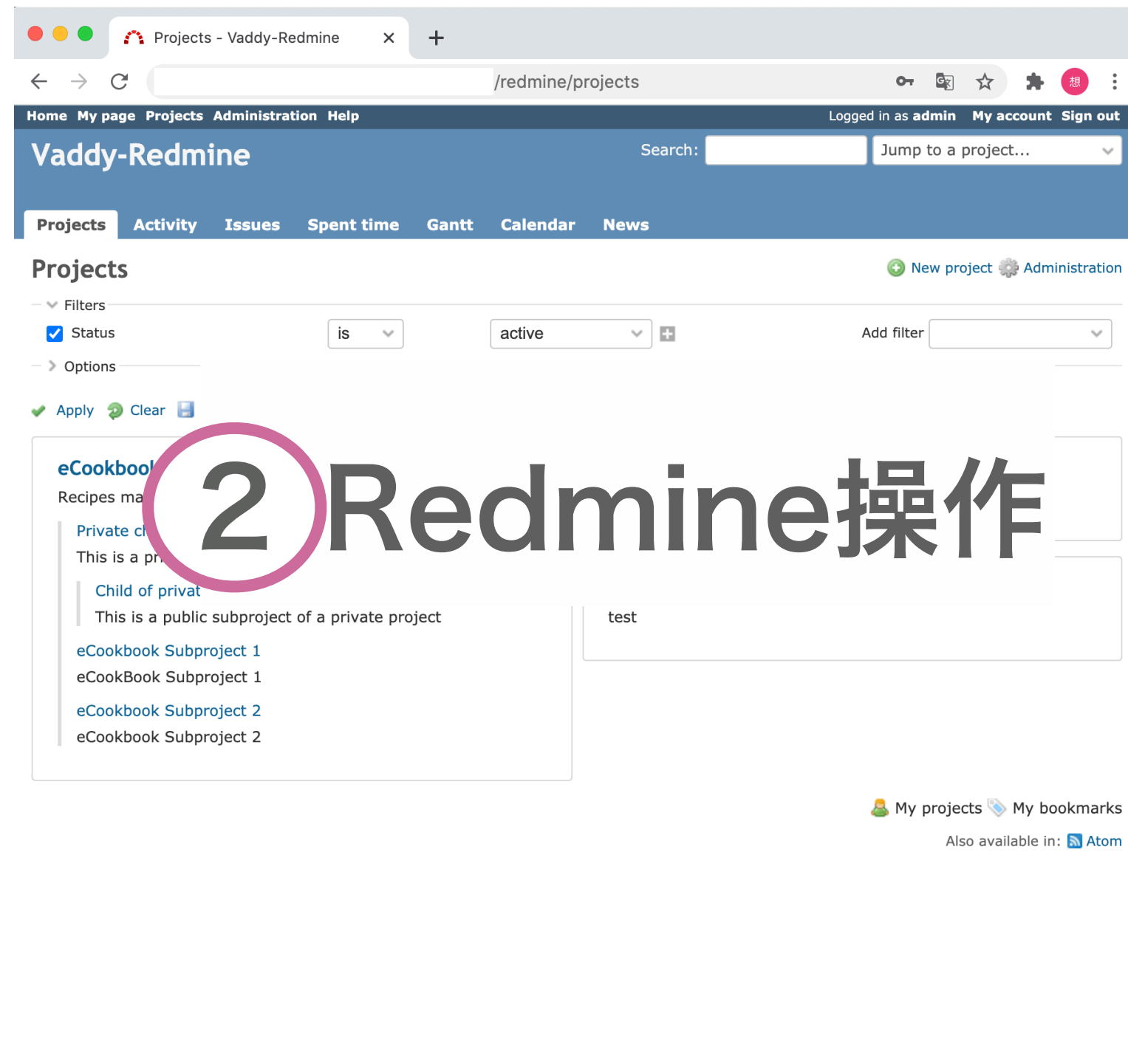


■ 開始URL

■ 終了URL

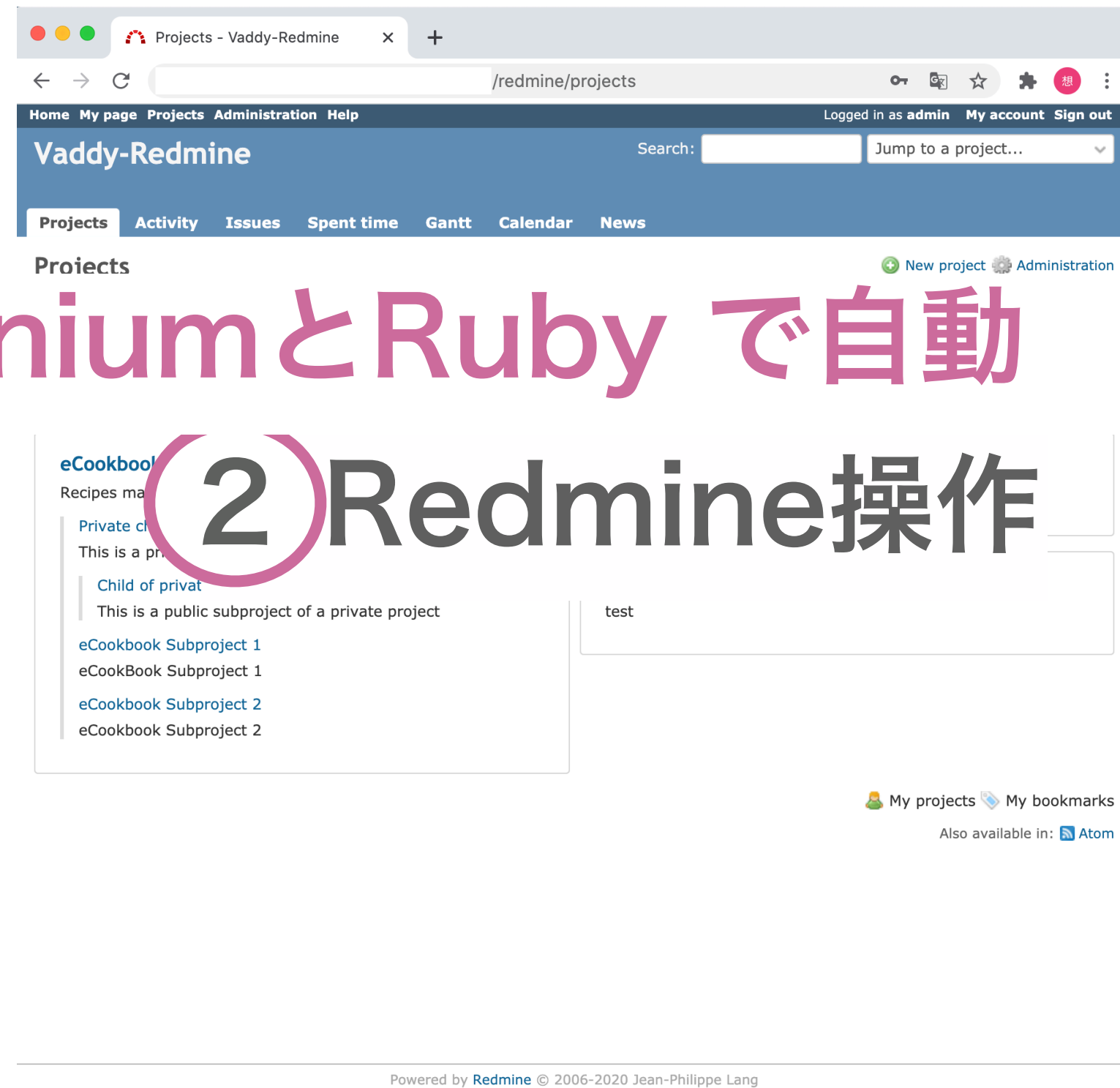
# クローラデータの作成

## Step 1



# クローラデータの作成

## Step 1



SeleniumとRubyで自動

Redmine操作

Proxy

3



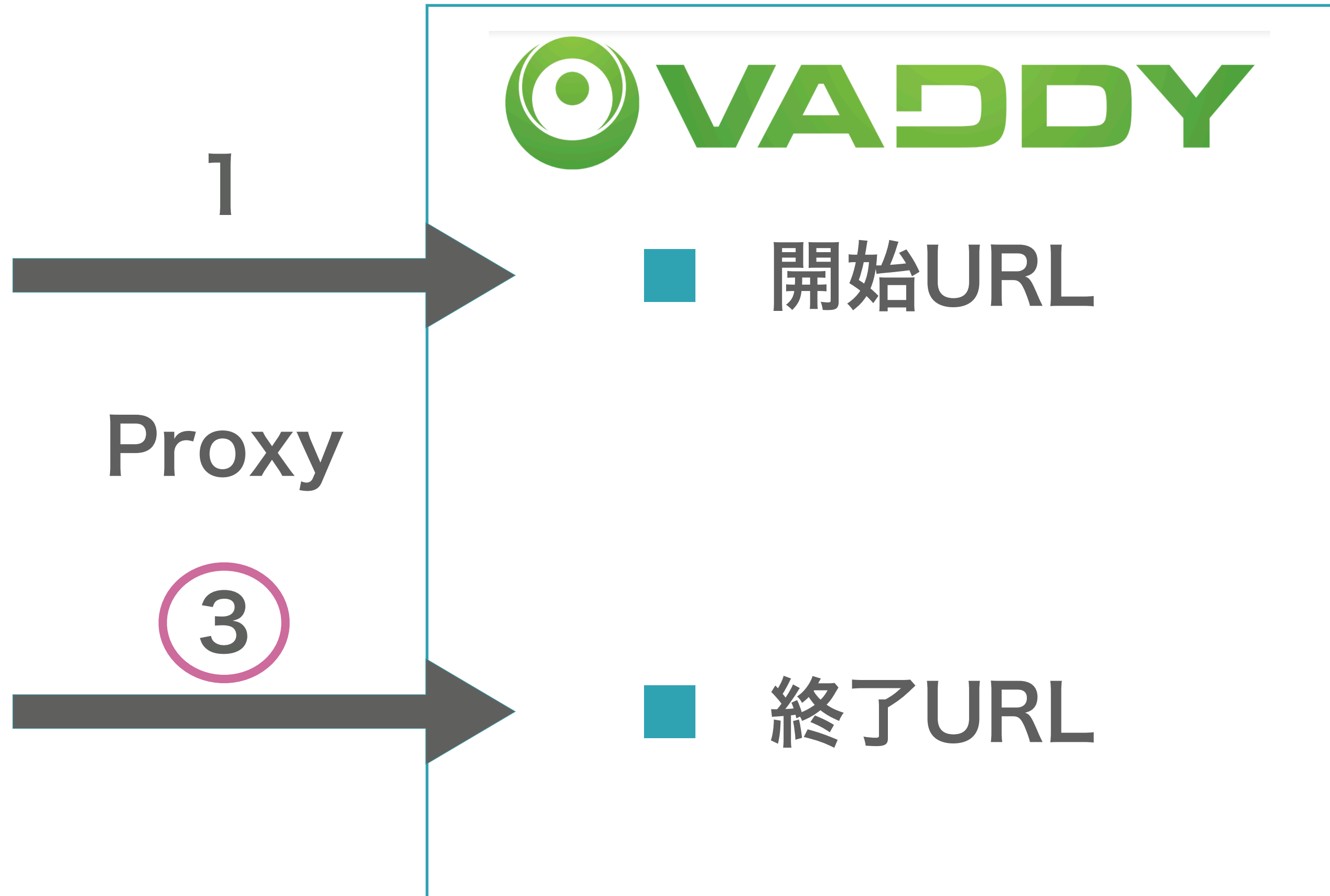
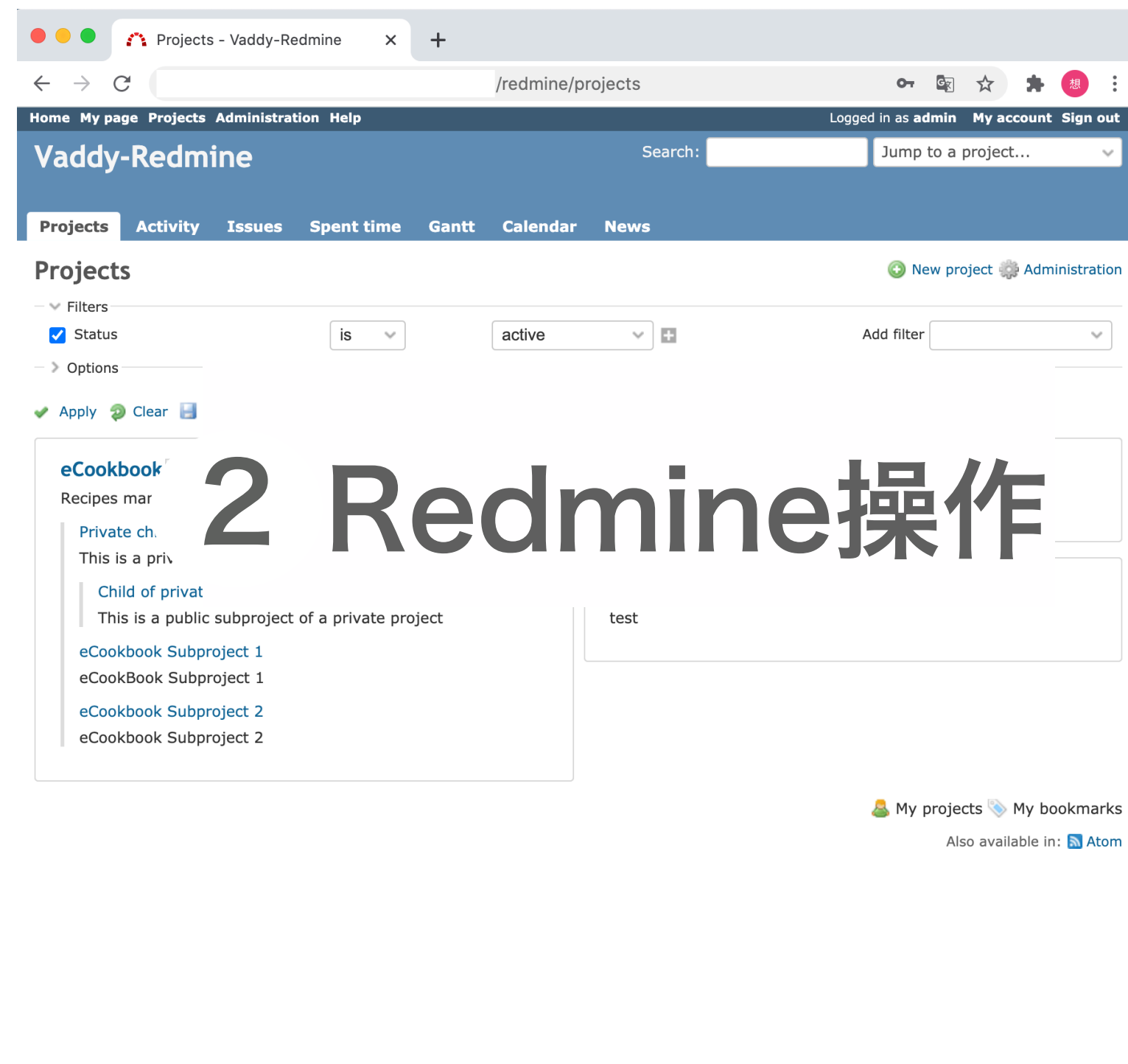
■ 開始URL

■ 終了URL



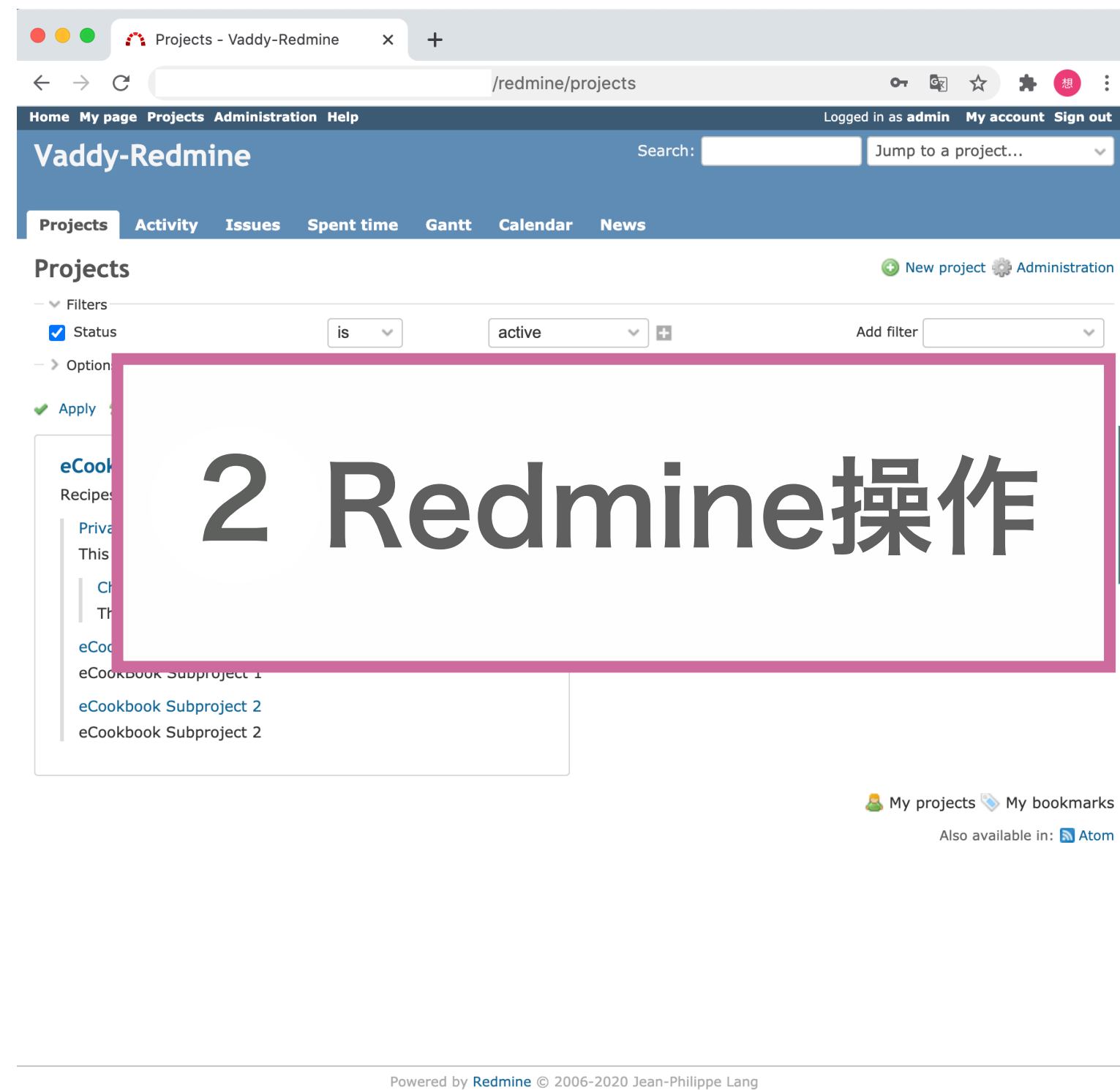
# クローラデータの作成

## Step 1



# クローldata の作成

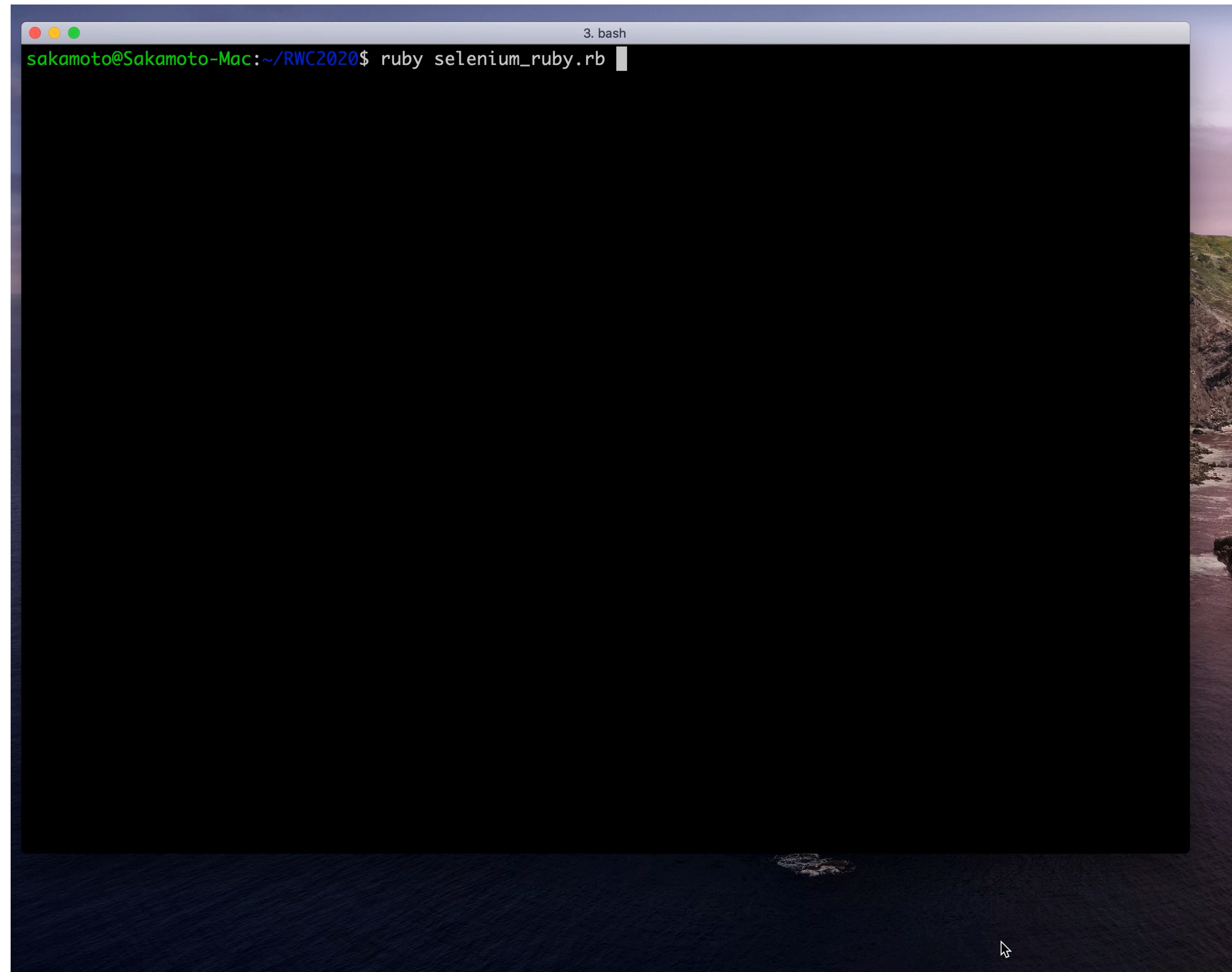
## Step 1



## 3. Ruby × Selenium メソッドの紹介

---

# Seleniumが動く動画



# コードの中身

```
require 'selenium-webdriver'

driver = Selenium::WebDriver.for :chrome
driver.manage.timeouts.implicit_wait = 10

# redmineにログイン
driver.navigate.to('http://vaddy-redmine.jp/redmine/login')
driver.find_element(:id, 'username').send_keys 'admin'
driver.find_element(:id, 'password').send_keys 'admin'
driver.find_element(:id, 'login-submit').click

# 新しいチケット作成画面へ移動
driver.navigate.to('http://vaddy-redmine.jp/redmine/issues/new')

# 新しいチケット(ticket1)を作成
driver.find_element(:id, 'issue_subject').send_keys 'ticket1'
driver.find_element(:id, 'issue_description').send_keys 'ticket1'
# ファイルを添付
driver.find_element(:xpath, '//*[@id="attachments_form"]/span/span[2]/input').send_keys '~/dummy.png'
driver.find_element(:xpath, '//*[@id="attachments_1"]/input[2]').send_keys 'dummy'
driver.find_element(:xpath, '//*[@id="issue-form"]/input[3]').click

driver.quit
```

# 紹介する操作

- ブラウザ起動
- ページ遷移
- 要素取得（テキスト入力、クリック）
- 待機処理

# 紹介する操作

- ブラウザ起動
- ページ遷移
- 要素取得（テキスト入力、クリック）
- 待機処理

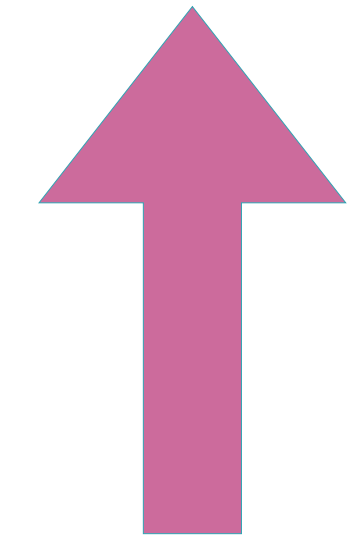
# ブラウザ起動

```
driver = Selenium::WebDriver.for :chrome
```



## ブラウザ起動

```
driver = Selenium::WebDriver.for :chrome
```



- 起動する **ドライバ** を指定

# ブラウザ起動

firefox

```
driver = Selenium::WebDriver.for :chrome
```

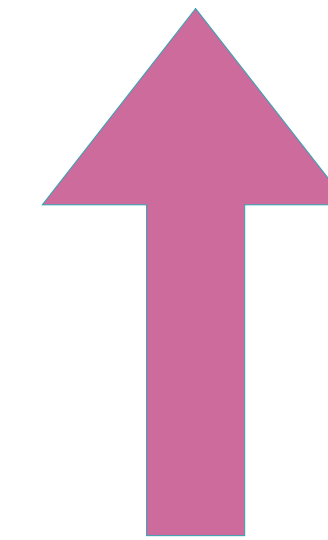
- 起動する **ドライバ** を指定
- **firefox** ドライバも指定できる

# 紹介する操作

- ブラウザ起動
- ページ遷移
- 要素取得（テキスト入力、クリック）
- 待機処理

## ページ遷移

```
driver.navigate.to( 'http://vaddy-redmine.jp/redmine/login' )
```



- アクセスしたいURLを指定

# ページ遷移

## ログイン画面

Chrome は自動テスト ソフトウェアによって制御されています。

ホーム プロジェクト ヘルプ ログイン 登録する

Redmine 検索: プロジェクトへ移動...

ログインID  
パスワード [パスワードの再設定](#)  
ログイン

## チケット作成画面

Chrome は自動テスト ソフトウェアによって制御されています。

Home My page Projects Administration Help Logged in as admin My account Sign out

eCookbook Search: eCookbook

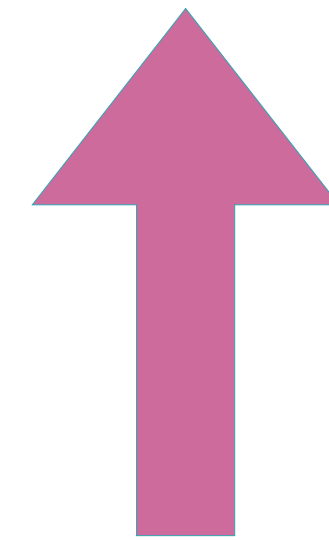
+ Overview Activity Roadmap **Issues** Spent time Gantt Calendar News Documents Wiki Forums Files Repository Settings

New issue

Project \* eCookbook  Private  
Tracker \* Bug  
Subject \*  
Description Edit Preview B I U S C  
Status \* New Parent task  
Priority \* Normal Start date 2020/11/04  
Assignee Assign to me Due date 年 / 月 / 日  
Category Estimated time Hours  
Target version % Done 0 %  
Searchable field Default string Custom date 年 / 月 / 日  
Database Project 1 cf 年 / 月 / 日  
Float field  
Files ファイル選択 選択されていません (Maximum size: 5 MB)  
Watchers  Dave Lopper  John Smith  
Search for watchers to add

## ページ遷移

```
driver.navigate.to( 'http://vaddy-redmine.jp/redmine/login' )
```



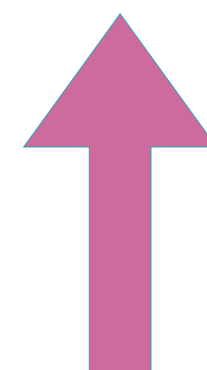
- アクセスしたいURLを指定
- ログインページなどにアクセスできる

# 紹介する操作

- ブラウザ起動
- ページ遷移
- 要素取得 (テキスト入力、クリック)
- 待機処理

## 要素取得

```
driver.findElement(:id, 'username')
```

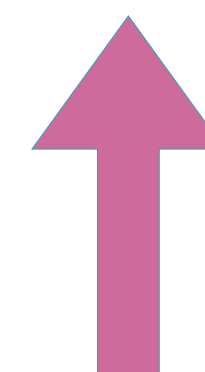


属性



## 要素取得

```
driver.findElement(:id, 'username')
```



属性值

# 要素取得

```
driver.find_element(:id, 'username')
```

ログインID

パスワード パスワードの再設定

ログイン

← 取得した要素

## 要素取得

```
driver.findElement(:id, 'username')
```

- idがusernameの要素を取得

## 要素取得

```
driver.find_element(:id, 'username')
```

- idがusernameの要素を取得
- xpath、name属性の要素も取得できる

## 要素取得

```
driver.find_element(:id, 'username')
```

+

- 要素に文字を入力するメソッド `send_keys`

## 要素取得

```
driver.find_element(:id, 'username')
```

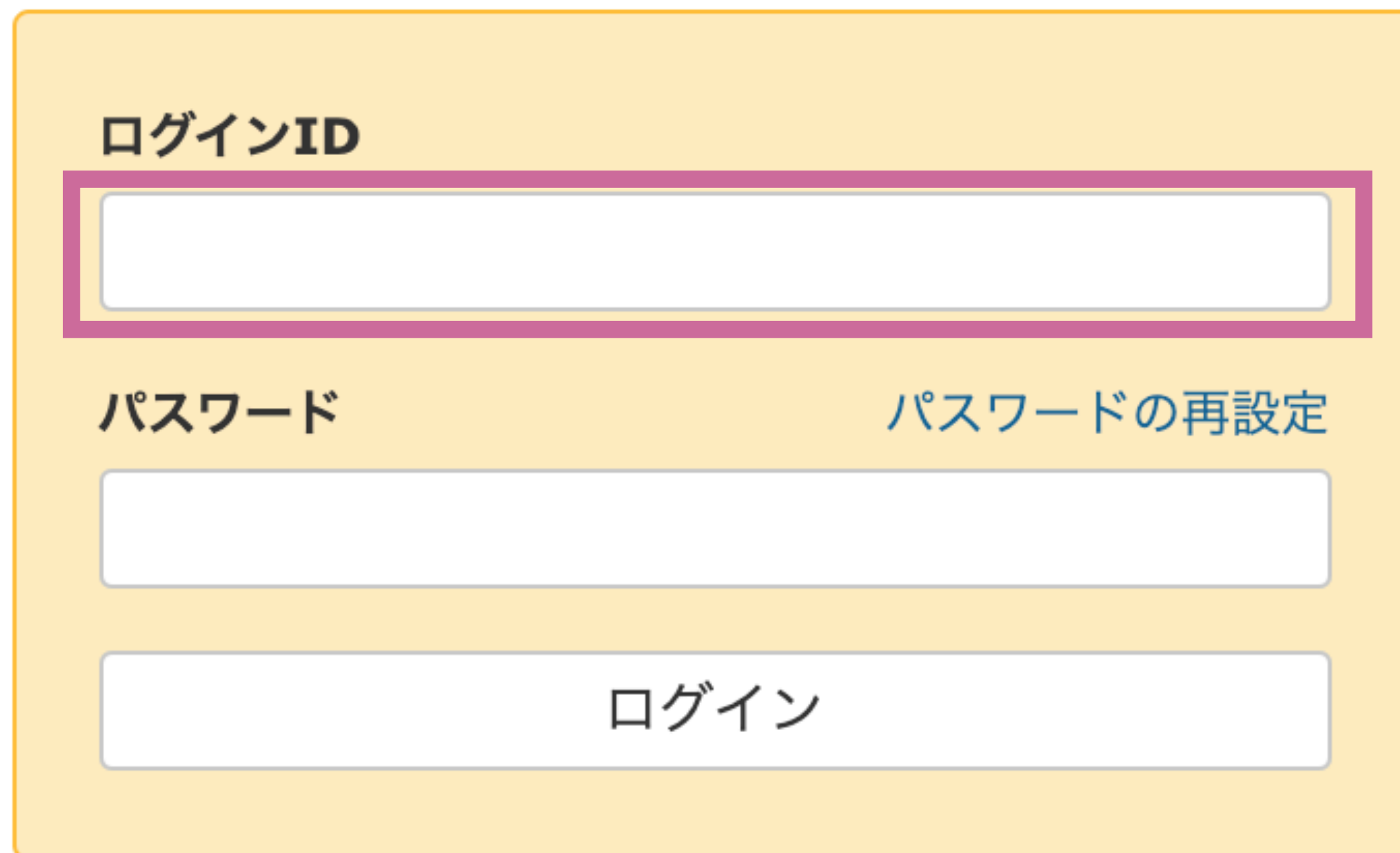
+

- 要素に**文字を入力**するメソッド **send\_keys**
- 要素を**クリック**するメソッド **click**

など...

## 文字入力: send\_keys

```
driver.find_element(:id, 'username').send_keys 'admin'
```



A screenshot of a login form with a yellow background. The form contains the following elements:

- ログインID**: A text input field, highlighted with a thick purple border.
- パスワード**: A text input field.
- パスワードの再設定**: A text input field.
- ログイン**: A button.

**send\_keys 'admin'**  
を指定

## 文字入力: send\_keys

```
driver.find_element(:id, 'username').send_keys 'admin'
```

ログインID

admin| "admin"を入力

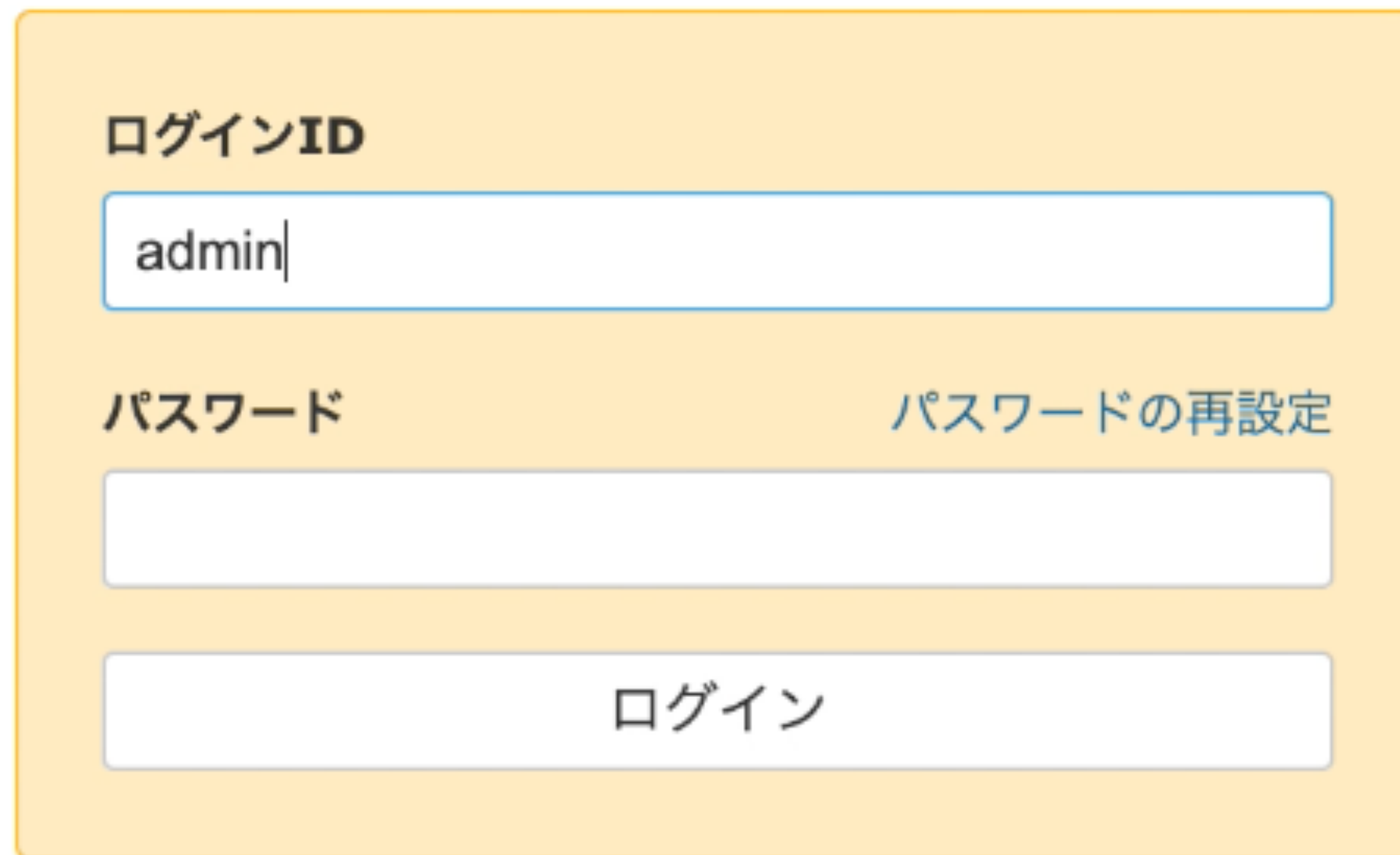
パスワード [パスワードの再設定](#)

ログイン



# 文字入力: send\_keys

```
driver.find_element(:id, 'username').send_keys 'admin'
```



ログインID

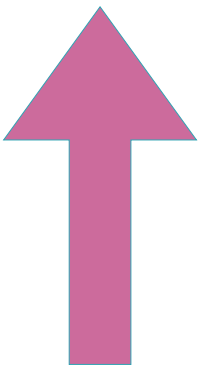
パスワード パスワードの再設定

ログイン

- send\_keysメソッドは  
使用頻度が高い

## クリック: click

```
driver.findElement(:id, 'login-submit').click
```



# クリック: click

```
driver.findElement(:id, 'login-submit').click
```

ログインID

admin

パスワード

パスワードの再設定

.....

クリックする

ログイン

# クリック: click

```
driver.findElement(:id, 'login-submit').click
```

ログインID

admin

パスワード

パスワードの再設定

.....

クリックする

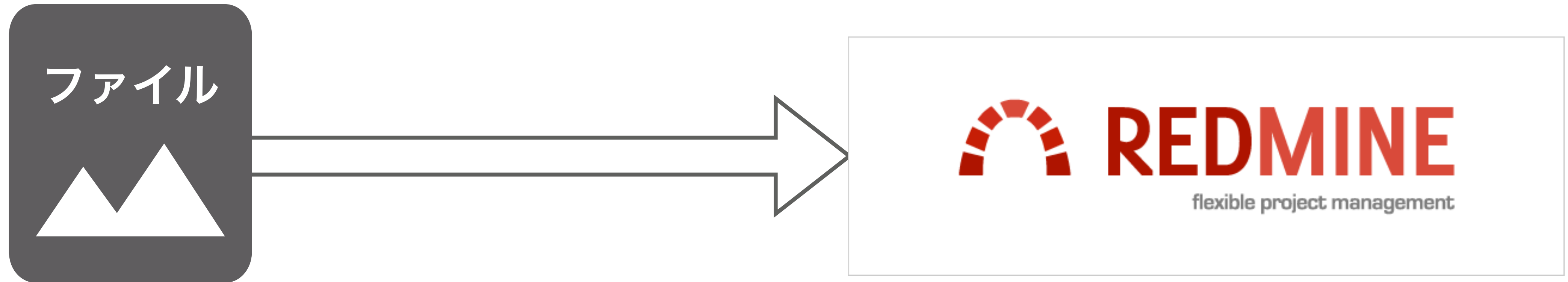
ログイン

■ ボタンの**クリック**  
をするときに使う

# 紹介する操作

- ブラウザ起動
- ページ遷移
- 要素取得（テキスト入力、クリック）
- 待機処理

# 待機処理



- 添付処理は**時間**がかかる
- 処理が終わる前に次の処理を実行すると**エラー**となる
- 処理が終わるまで**待機**する

# 待機処理

3種類試した中で、1番良いのは**implicit\_wait**

① Sleep

② Wait.until

③ **implicit\_wait**

## ① 待機処理: Sleepメソッド

```
sleep 5
```

5秒の間、一時停止状態にする



## ① 待機処理: Sleepメソッド

```
sleep 5
```

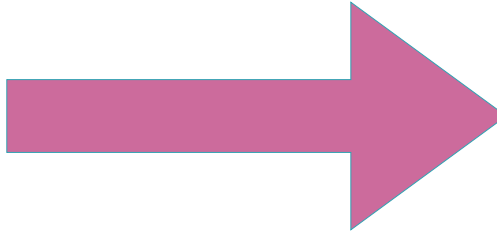
5秒の間、一時停止状態にする



明確な秒数がわからない

必要以上にスリープしてしまう

## ② 待機処理: Wait.untilメソッド



```
# waitに50秒のタイマーを持たせる
wait = Selenium::WebDriver::Wait.new(:timeout => 50)

# 指定した要素が表示されるまで待つ
wait.until{driver.find_element(:id, 'test').displayed?}
```

## ② 待機処理: Wait.untilメソッド

```
# waitに50秒のタイマーを持たせる  
wait = Selenium::WebDriver::Wait.new(:timeout => 50)  
  
# 指定した要素が表示されるまで待つ  
wait.until{driver.find_element(:id, 'test').displayed?}
```

## ② 待機処理: Wait.untilメソッド

```
# waitに50秒のタイマーを持たせる  
wait = Selenium::WebDriver::Wait.new(:timeout => 50)  
  
# 指定した要素が表示されるまで待つ  
wait.until{driver.find_element(:id, 'test').displayed?}
```



毎回呼び出しが必要

# 待機処理

3種類試した中で、1番良いのは**implicit\_wait**

① Sleep

② Wait.until

③ **implicit\_wait**



### ③ 待機処理: implicit\_waitメソッド

```
driver.manage.timeouts.implicit_wait = 10
```

指定した要素が見つかるまでの**待ち時間**を**設定**する

### ③ 待機処理: implicit\_waitメソッド

```
driver.manage.timeouts.implicit_wait = 10
```

指定した要素が見つかるまでの**待ち時間**を**設定**する

- **1度の設定**で良い
- 待ち時間を**一律**に指定することができる

## 4. 脆弱性診断を自動化してみても

---



# 脆弱性診断の流れ

自動化



Step1

## ■ クロールデータ の作成

- ・リンクをクリックするなどのRedmineの操作を記録したデータ

Step2

## ■ Scan

- ・クロールデータ を選択して、脆弱性診断を開始

# 脆弱性診断の流れ

## Step1

### ■ クロールデータ の作成

- ・リンクをクリックするなどのRedmineの操作を記録したデータ



## Step2

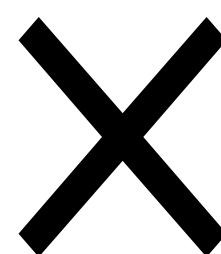
### ■ Scan

- ・クロールデータ を選択して、脆弱性診断を開始

# クローラデータ の作成

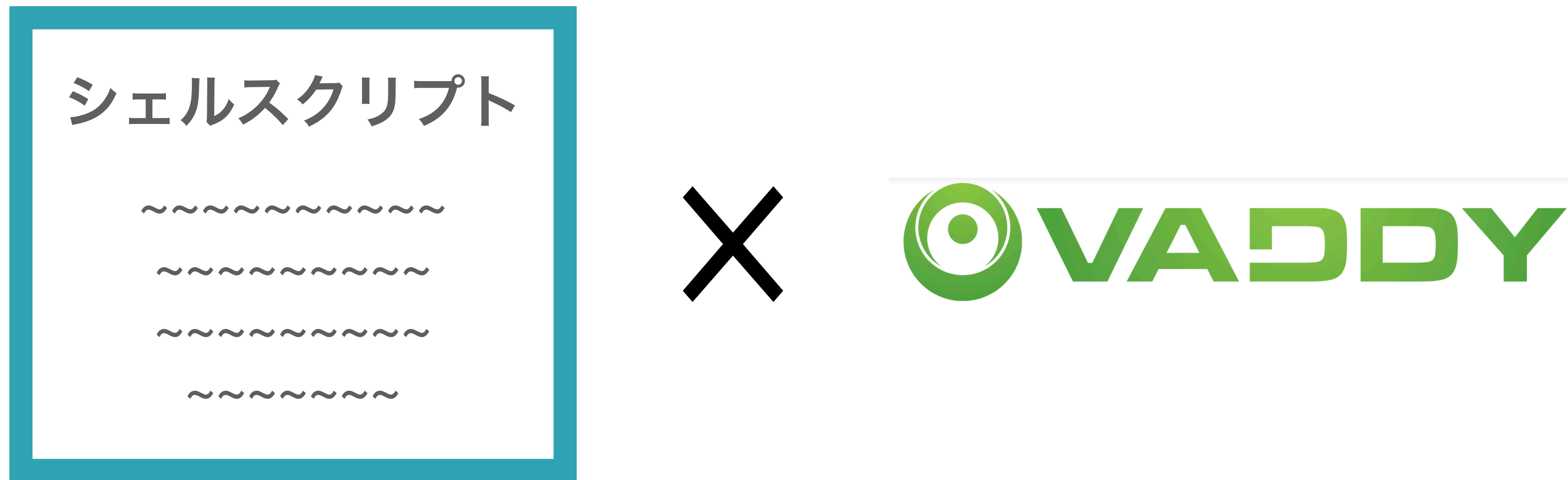
Step2 シェルスクリプトとVAddyの機能を使って**自動化**

シェルスクリプト  
~~~~~  
~~~~~  
~~~~~  
~~~~~



# クロールデータ の作成

## Step2 シェルスクリプトとVAddyの機能を使って**自動化**



## 今後の取り組み

- シェルスクリプト → **Ruby**で書く

# 脆弱性診断を自動化してみても

今までは…

- すべて**手作業**
- 脆弱性診断をするのに**時間**がかかっていた
- **丸一日**かかっていた

# 脆弱性診断の流れ

自動化



## Step1

### ■ クロールデータ の作成

- ・リンクをクリックするなどのRedmineの操作を記録したデータ

## Step2

### ■ Scan

- ・クロールデータ を選択して、脆弱性診断を開始

# 脆弱性診断を自動化してみても

## 自動化して

- 脆弱性診断をする**時間**が**少なくなった**
- 実行する**頻度**を**増やせた**
- 人為的作業ミスの**対策**にもなった

## 5. 本日のまとめ

---



## 今日のまとめ

- **自動化**することで**効率**よく**確実**に実行することができる
- **send\_keys**メソッドを活用する
- Seleniumを使う時は**待機処理**を使う
- **implicit\_wait**メソッドが一番**使いやすい**