Практика разработки веб-серверов на Rust

Панков Михаил

17 мая 2016

Обо мне

Кто я

- Системный программист
- Работал над компилятором, операционными системами
- В данный момент отвечаю за инфраструктуру обеспечения качества в проекте Kaspersky OS
- В свободное время занимаюсь веб-разработкой
- Один из основных переводчиков «The Rust Programming Language» на русский язык
- Основатель русскоязычного сообщества Rust

  `http://rustycrate.ru`

Задача

Задача

- Нужен «всегда зелёный master»
- Много разработчиков, большой поток Merge Request’ов
- Простой проверки Merge Request’а на момент его создания недостаточно
- Нужно упорядочивать изменения и проверять «кандидата на новый master» именно в том виде, в котором он будет влит

Стек

- GitLab Community Edition 8.x
- Jenkins 1.x

Решение

Нужен сторож!

Commit Gatekeeper под нашу инфраструктуру

Познакомьтесь с Шуриком

- Смотрит в GitLab, получает по Webhook API оповещения о Merge Request'ах (MR) и комментариях
- Получает команды от автора и рецензента через комментарии к MR
- Сливает MR с актуальным master, делает прогон Jenkins на нём, если успешно — обновляет master
- Уведомляет автора, если очередной MR невозможно слить автоматически

Как это выглядит

Mikhail Pankov @pankov · about 2 hours ago

@shurik r+

Небезызвестный сторож @shurik · about 2 hours ago

⏳ проверяю коммит 433239cb

Небезызвестный сторож @shurik · about 2 hours ago

☀️ тесты прошли, сливаю

Небезызвестный сторож @shurik · about 2 hours ago · about 2 hours ago

👌 успешно

Небезызвестный сторож @shurik · about 2 hours ago

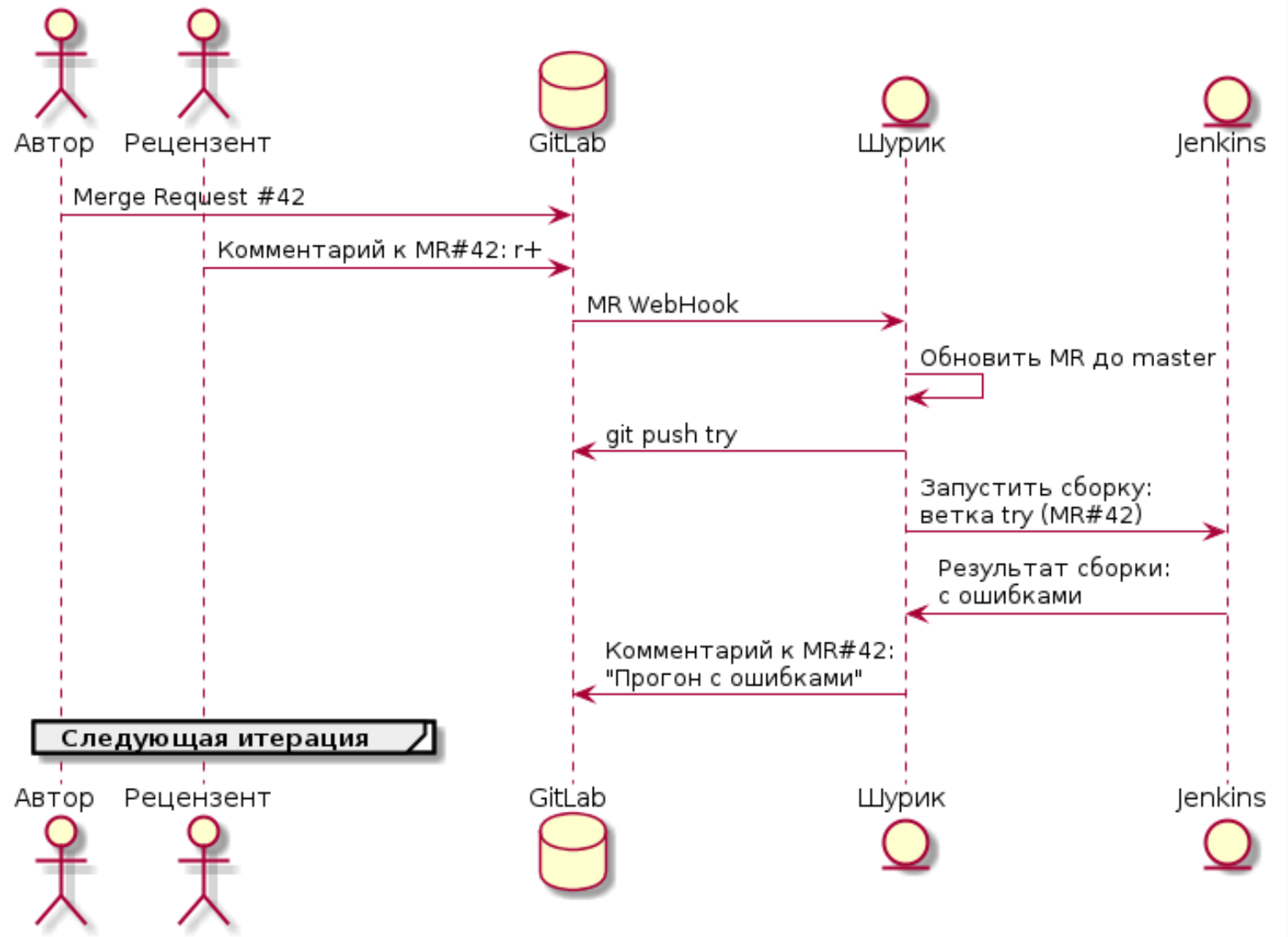
mentioned in commit 283f67da

Небезызвестный сторож @shurik · about 2 hours ago

Status changed to merged

Как это работает

Рецензент одобрил, но прогон не успешен



Плохой код...

не пройдёт!

Что ещё есть

Что ещё есть

- Сохранение состояния, восстановление при сбое через перезапуск приложения
- Изоляция проектов: сбой в обработке одного проекта не влияет на другие
- Логи
- Конфигурация в TOML-файле

Почему Rust?

Почему Rust?

- Хорошая поддержка многопоточности стандартной библиотекой: мьютексы, каналы, `RwLock`, `CondVar`...
- Надёжные многопоточность и обработка ошибок, при этом нулевые накладные расходы
- Хорошая изоляция сбоев: по умолчанию, паника останавливается на границе потока

Почему Rust? (продолжение)

- Редко ломается
- Когда ломается — это происходит в изолированных местах
- Вся наша большая команда разработчиков не будет останавливаться и ждать, пока починят инфраструктуру

Рабочее окружение

Используем стабильный Rust

- Любой `stable` проходит 12-недельный цикл использования людьми, которые сидят на `beta`
- Все изменения, которые попадают в `beta` после её отрезания - исправления багов
- Т.е. `stable` содержит меньше неизвестных багов
- `nightly` опасен тем, что нестабильные возможности могут изменить или удалить

multirust

- Берём `multirust` и ставим все нужные версии компилятора через него
- Новая версия `multirust` называется `rustup.rs` и работает на Windows
- `https://www.rustup.rs/`

IDE

IDE

- Их есть у нас
- Построены на базе всех мыслимых и немыслимых редакторов и IDE
- Почти все используют для навигациии и автодополнения `racer`
- `http://is.gd/rust_ide`
  `https://www.rust-lang.org/ides.html`
  `https://areweideyet.com/`

racer

- Он не всегда хорошо работает

IDE RFC

- Принят
  `https://github.com/rust-lang/rfcs/pull/1317`
- Был заблокирован рефакторингом компилятора — MIR
- Демон-«оракул» будет отвечать на вопросы IDE о программе
  `https://github.com/rust-lang/rust/issues/31548`
- У нас скоро будут первоклассные IDE!

Экосистема веб-приложений

На Rust уже можно писать веб-приложения?

- Можно
- `http://www.arewewebyet.org/`

Состояние экосистемы

- Веб-сервер
  - hyper
- Веб-фреймворки
  - iron
  - nickel
  - conduit
  - sappers
- Драйверы к БД
  - MySQL
  - PostgreSQL
  - Redis
- ORM
  - rustorm
  - diesel

Как найти библиотеки?

crates.io

Центральное хранилище с поиском

Как найти хорошие библиотеки?

Хорошие библиотеки

- Работают
- Имеют удобное API
- Хорошо документированы

Косвенные признаки

- Имеют много пользователей
  - Есть на crates.io
- Обновляются чаще, чем раз в тысячелетие
  - Есть на crates.io
- Есть документация и примеры
  - Ссылка на документацию на crates.io
- Люди хорошо отзываются
  - За этим мы тут и собрались

Нужной библиотеки нет, что делать?

Foreign Function Interface

У Rust первоклассный FFI, и он хорошо встраивается во многие языки

Легко позвать существующую библиотеку на другом языке

`http://jakegoulding.com/rust-ffi-omnibus/`

Веб-фреймворк

Iron

Почему?

- Самый матёрый
- Простой
- Модульный

GitLab API

GitLab API/1

Написана своя минималистичная обёртка

GitLab API/2: Типы отражают состояния

```
pub struct Api {
    root: Url,
}

pub struct Session {
    root: Url,
    private_token: String,
}
```

GitLab API/3: Конструирование API проверяет URL

```
impl Api {
    pub fn new<T: IntoUrl>(maybe_url: T)
           -> Result<Self, ::url::ParseError> {
        let url = try!(maybe_url.into_url());
        Ok(
            Api {
                root: url,
            })
    }
```

GitLab API/4: Обработка ошибок

```
impl Api {
    pub fn login(
            &self, username: &str, password: &str)
            -> Result<Session, LoginError> {
        let client = Client::new();
        let mut res =
            try!(
              client.post(
                &*format!("{}/session", self.root))
              .body(
                &*format!(
                  "login={}&password={}",
                  username, password))
              .send());
```

GitLab API/5: Как эта функция вызывается

```
let gitlab_session =
  gitlab_api.login(gitlab_user, gitlab_password).unwrap();
```

GitLab API/6: Вспомогательный макрос

```
quick_error! {
    #[derive(Debug)]
    pub enum LoginError {
        Http(err: ::hyper::error::Error) { from() }
        Read(err: ::std::io::Error) { from() }
        Json(err: JsonError) { from() }
        JsonObject(err: JsonObjectError) { from() }
        JsonObjectString(err: JsonObjectStringError) {
            from()
        }
    }
}
```

Общая архитектура

Разделяемое состояние

```
let gitlab_login_config =
  get_gitlab_login_config(&*config).unwrap();
let gitlab_session =
  login_to_gitlab(gitlab_login_config).unwrap();
let gitlab_session = Arc::new(gitlab_session);
```

Разделяемое состояние (продолжение)

```
let gitlab_session = Arc::new(gitlab_session);
for (psid, project_set) in project_sets {
    init_project_set(
        gitlab_session.clone(), psid, project_set,
        state_save_dir, &mut router, &mut builders,
        config.clone());
}
Iron::new(router).http(
    (&*gitlab_address, gitlab_port))
    .expect("Couldn't start the web server");
```

Многопоточность

```
{
    let mr_storage = mr_storage.clone();
    let queue = queue.clone();
    let project_set = project_set.clone();
    let state_save_dir = state_save_dir.clone();

    let builder = thread::spawn(move || {
        handle_build_request(...);
    });
    builders.push(builder);
}
```

Многопоточность (продолжение)

```
let builder = thread::spawn(move || {
    handle_build_request(...);
});
router.post(format!("/api/v1/{}/mr", psid),
            move |req: &mut Request|
            handle_mr(...));
router.post(format!("/api/v1/{}/comment", psid),
            move |req: &mut Request|
            handle_comment(...));
```

Сериализация в JSON

```
#[derive(RustcDecodable, RustcEncodable)]
#[derive(Debug, Clone)]
struct MergeRequest {
    id: MrUid,
    human_number: u64,
    checkout_sha: String,
    status: Status,
    approval_status: ApprovalStatus,
    merge_status: MergeStatus,
}
```

Рефакторинг

Рефакторинг

- Очень легко делать*
- Просто чинишь все места, где возникают ошибки компиляции
    - *С поправкой на сами сообщения об ошибках компиляции

Отладка

Отладка

- `println!`
- `rust-gdb`
- backtrace
- следующая версия `gdb` будет поддерживать Rust нативно

Профилирование

Профилирование

- Производительность
  - valgrind

    `https://llogiq.github.io/2015/07/15/profiling.html`
  - oprofile
  - perf
  - Instruments

    `http://carol-nichols.com/2015/12/09/rust-profiling-on-osx-cpu-time/`
- Покрытие
  - kcov

    `https://is.gd/rust_kcov`
  - coveralls.io

    `https://is.gd/rust_coveralls`

Развёртывание компилятора и приложений

Развёртывание компилятора и приложений

- Ставим компилятор **`rustup`**'ом с флагом **`--save`**
- Внимание: зависимости на системные библиотеки!

Особенно классное

Ни одного segmentation fault’a

- Я не вру
- Никаких звёздочек

Все падения только по assert или unwrap

- Внимание: `.unwrap()` — это `assert!()` в овечьей шкуре

Ошибки и проблемы

Deadlock

- Deadlock очень легко сделать из-за lock guards
- Возникает детерминированно
- Отлаживается элементарно

Медленная компиляция

- Сборка всего проекта, с зависимостями, в релизе
  2 минуты
- Сборка только изменённых файлов проекта, без зависимостей, в дебаге
  8 секунд
- Инкрементальная компиляция скоро будет
- Ждали окончания рефакторинга компилятора (MIR)
  `https://github.com/rust-lang/rust/issues/2369`

Нет рекомендаций по архитектуре

Всевозможные шаблоны только зарождаются

`https://github.com/nrc/patterns`

Иногда приходится обновлять библиотеки

- Не проблема, когда это библиотеки на Rust
- Если это обёртки к системным библиотекам (`.so`, `.dll`, `.dylib`), начинается веселье
- Приходится сводить все зависимости к совместимым версиям обёртки

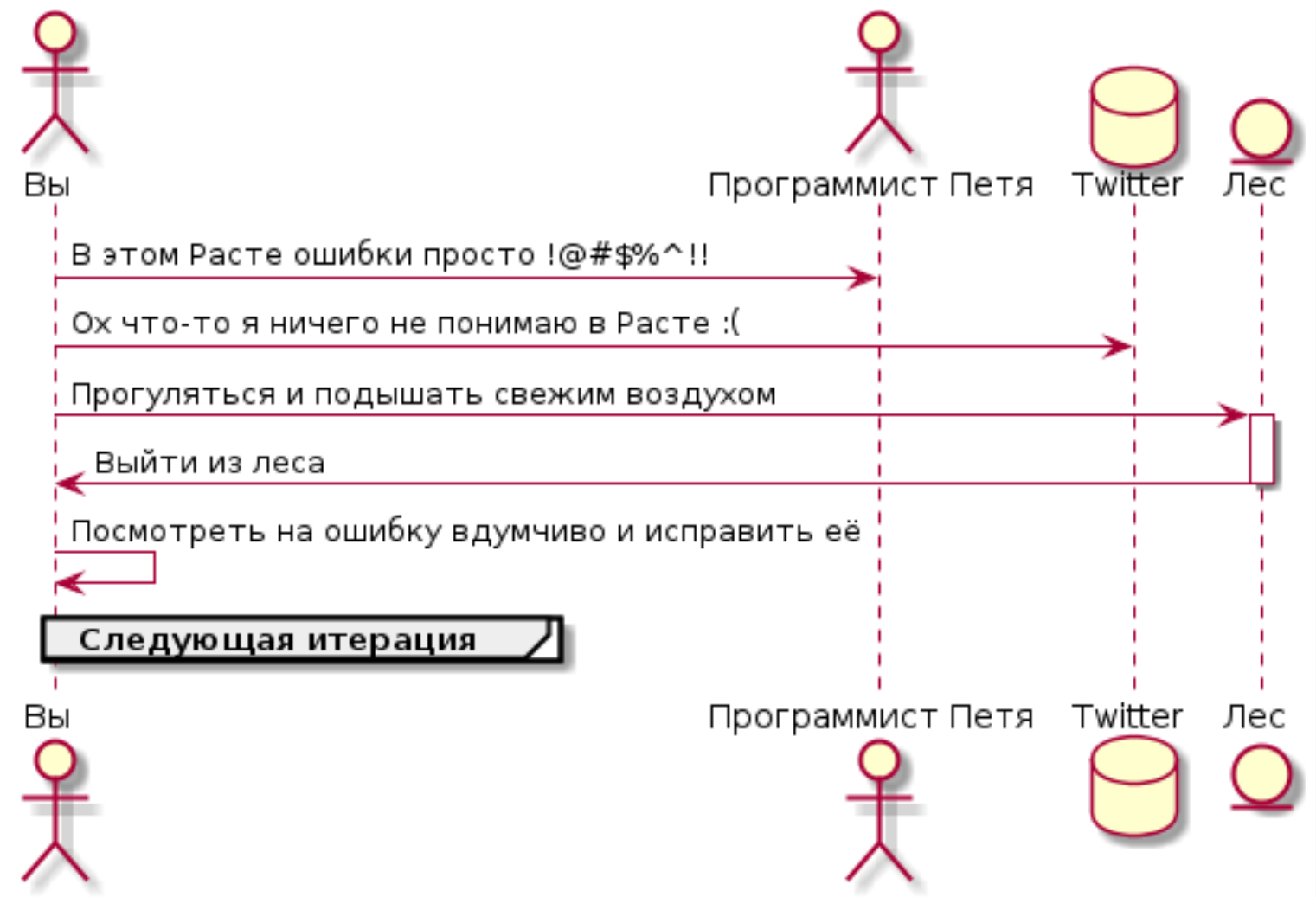
Сообщения об ошибках

Сообщения об ошибках

```
   Compiling shurik v0.1.0 (file:...)
src/main.rs:1253:19: 1253:32 error: the type
  `[closure@src/main.rs:1253:33: 1257:6
  mr_storage:alloc::arc::Arc<
    std::sync::mutex::Mutex<
      std::collections::hash::map::HashMap<
        MrUid, MergeRequest>>>,
  queue:alloc::arc::Arc<(
    std::sync::mutex::Mutex<
      collections::linked_list::LinkedList<
        WorkerTask>>,
      std::sync::condvar::Condvar)>,
  config:&alloc::arc::Arc<toml::Value>,
```

```
  project_set:alloc::arc::Arc<ProjectSet>,
  state_save_dir:
    alloc::arc::Arc<collections::string::String>,
  gitlab_session:alloc::arc::Arc<gitlab::Session>]`
does not fulfill the required lifetime [E0477]
src/main.rs:1253
  let builder = thread::spawn(move || {
                                     ^~~~~~~~~~~~~
note: type must outlive the static lifetime
error: aborting due to previous error
Could not compile `shurik`.
```

Глубоко вдохните и прогуляйтесь



Если серьёзно

Приходите в чат — помогут разобраться

`https://gitter.im/ruRust/general`

Почему я люблю компиляторы, которые умнее меня

- Компилятор всегда прав*
  - В отличие от людей
  - В отличие от тестов
  - *Кроме багов

Сообщество

Сообщество

- `http://rustycrate.ru`

Шурик на GitHub

Шурик на GitHub

`https://github.com/mkpankov/shurik`

Ссылки

Ссылки

- «Язык программирования Rust»
  `http://rurust.github.io/rust_book_ru/`
- «Растономикон»
  `https://github.com/ruRust/rustonomicon`
- «Маленькая книжка о макросах Rust»
  `https://github.com/ruRust/tlborm`
- Чат
  `https://gitter.im/ruRust/general`
- Reddit
  `http://reddit.com/r/rust/`
- `http://rustycrate.ru`

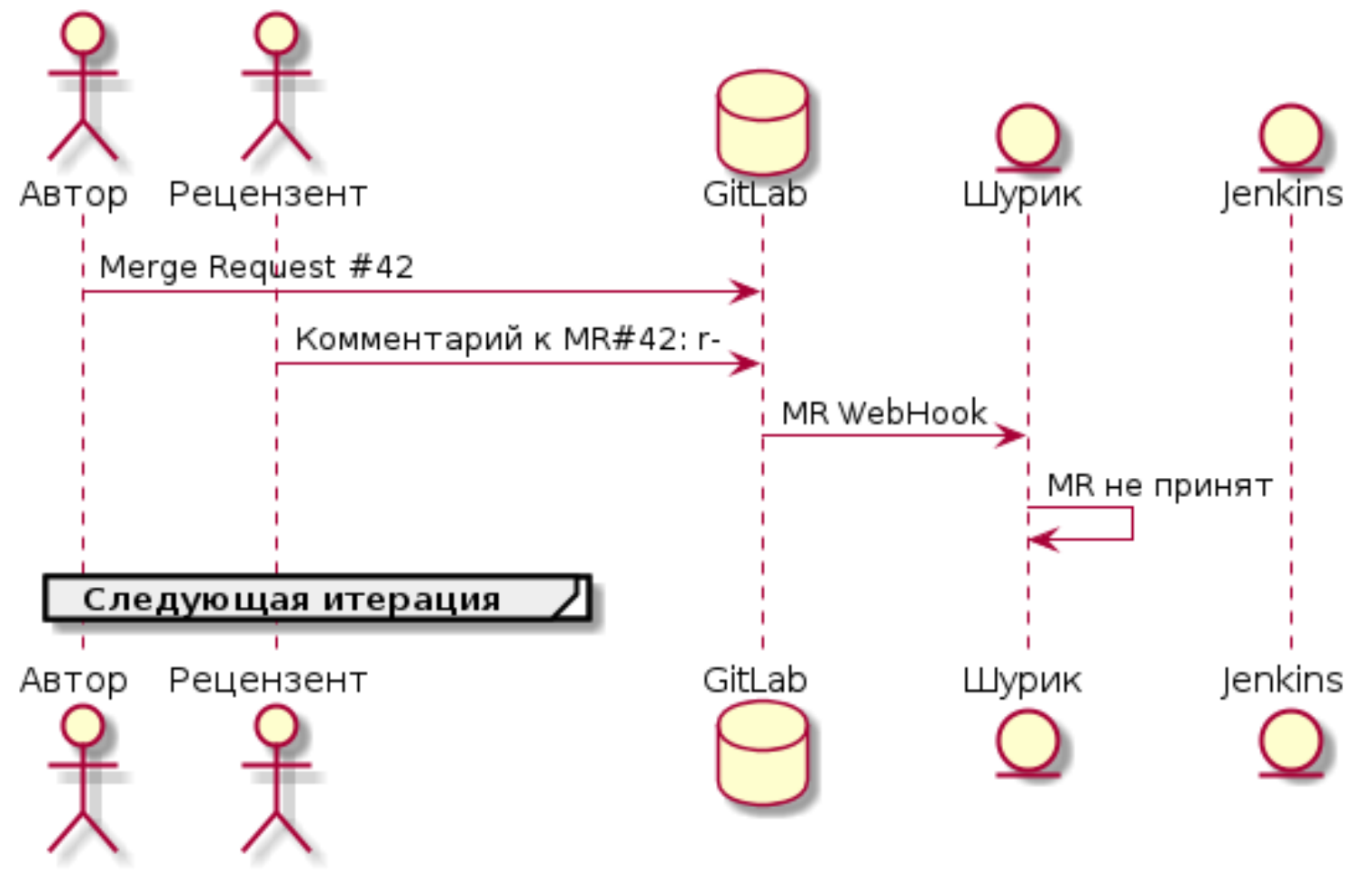
Где найти этот доклад

`http://mkpankov.github.io/rust-web-dev-2016/talk.pdf`

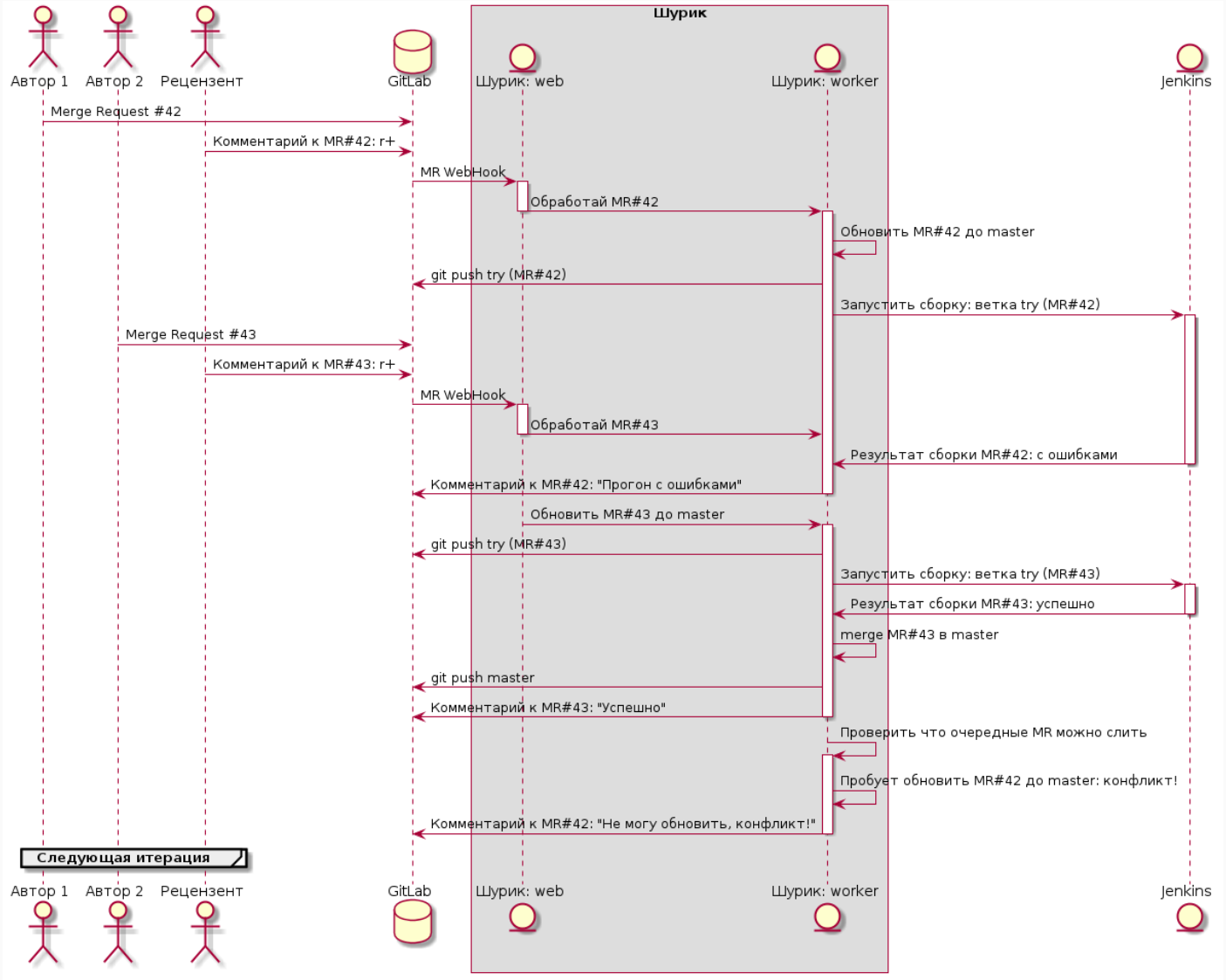
Спасибо!

Backup

Рецензент не одобрил



Прогон не успешен, а затем merge conflict



Связанные изменения

- Изменения в нескольких репозиториях
- Если применить эти изменения не ко всем нужным репозиториям, это приведёт к поломке сборки или тестов
- Если не учитывать связанность изменений, сторож никогда их не пропустит, т.к. тестирование каждого отдельного MR завершится ошибкой
- Большой пласт функциональности
- Пока не реализовано, будущая работа

Макрос try!

```
// try(f);

match f {
  Ok(o) => o,
  Err(e) => return From::from(e),
}
```

Deadlock

```
let mr_storage_locked_1 =
  &*mr_storage.lock().unwrap();
mr_storage_locked_1[...] = ... ;
...
let mr_storage_locked_2 =
  &*mr_storage.lock().unwrap();
mr_storage_locked_2[...] = ... ;
```

Сериализация в JSON (вручную) /1

```
impl Encodable for MrUid {
    fn encode<S: Encoder>(&self, s: &mut S)
       -> Result<(), S::Error> {
        format!(
          "{},{}", self.id, self.target_project_id)
        .encode(s)
    }
}
```

Сериализация в JSON (вручную) /2

```
impl Decodable for MrUid {
    fn decode<D: Decoder>(d: &mut D)
       -> Result<Self, D::Error> {
        let s = try!(d.read_str());
        let s_v: Vec<_> = s.split(",").collect();
        let mut v: Vec<u64> =
          s_v.iter().map(|x| x.parse().unwrap())
                    .collect();
        let mr_uid = MrUid {
            target_project_id: v.pop().unwrap(),
            id: v.pop().unwrap(),
        };
        Ok(mr_uid)
    }
```

GitLab API/5: Сложная обработка ошибок

```
...     let mut text = String::new();
        try!(res.read_to_string(&mut text));
        let json: ::serde_json::value::Value =
          ::serde_json::from_str(&text).unwrap();
        let obj =
          try!(json.as_object().ok_or(JsonError));
        let private_token_value =
          try!(obj
               .get("private_token")
               .ok_or(JsonObjectError));
        let private_token =
          try!(private_token_value
               .as_string()
               .ok_or(JsonObjectStringError));
```

GitLab API/6: Успех

```
            Ok(
                Session {
                    root: self.root.clone(),
                    private_token: private_token
                                       .to_owned(),
                })
        }
    }
```