Паттерны Серверных COMET-решений

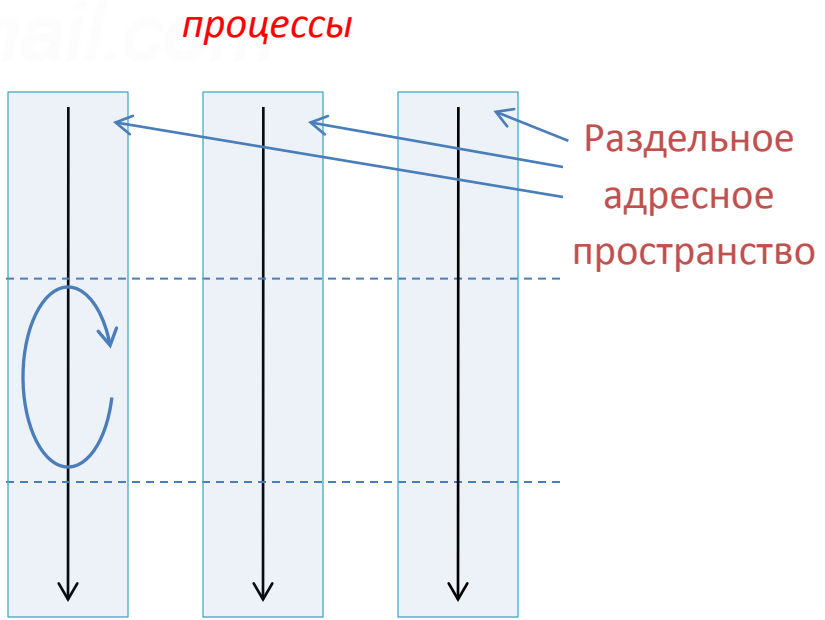
Илья Кантор

http://javascript.ru

Классические серверные паттерны

- Процессы OS

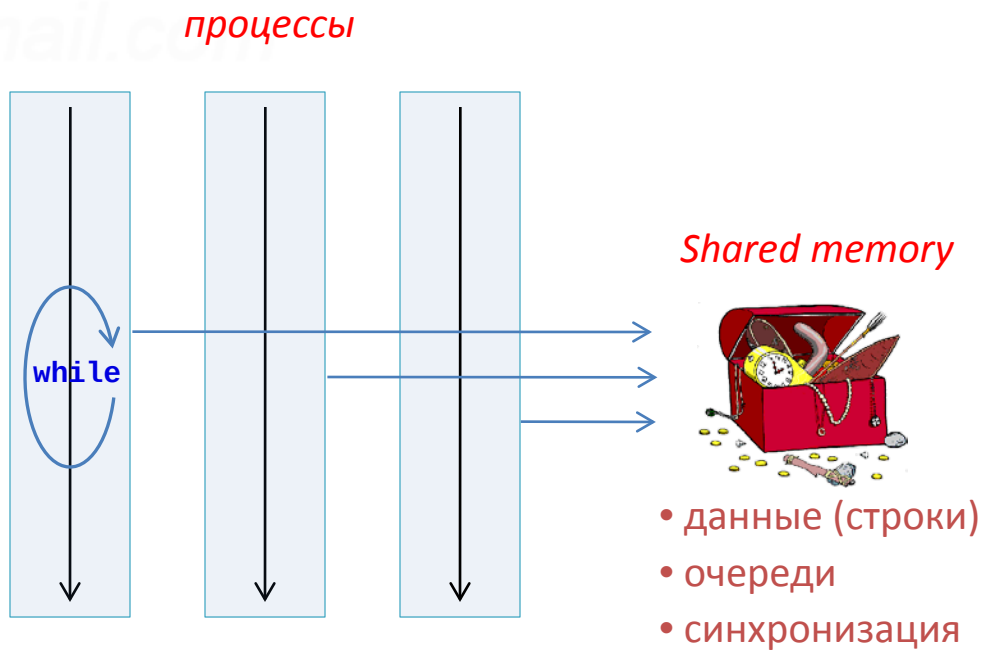
```
// инициализация фреймворка
include MyFramework;

MyFramework::init();
db = DB::connect();

// while ждет очередной запрос
while(request = FastCGI::accept()) {
    MyFramework::process(request);
}

MyFramework::finalize();
```



Классические серверные паттерны

- Процессы OS
  - Pre-fork
  - Обмен данными
    - Shared memory
    - Переменные процесса

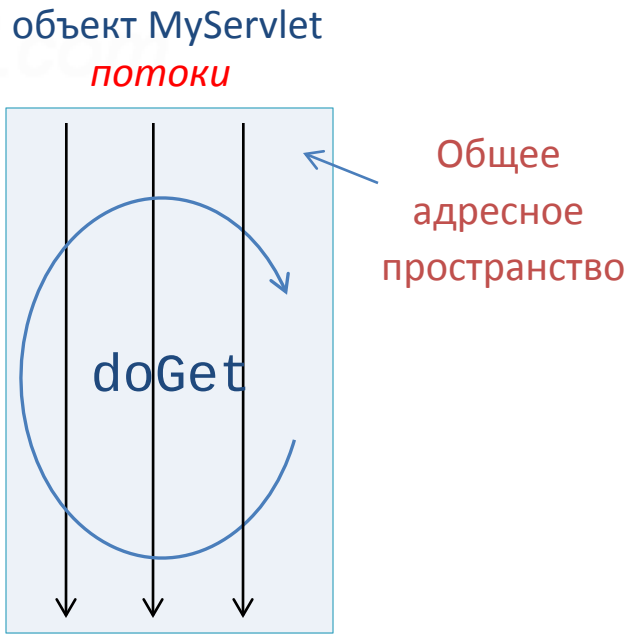
```
MyFramework::init();
db = DB::connect();

while(accept…) {
    …
}
```



Классические серверные паттерны

- Потоки OS
  - пул потоков

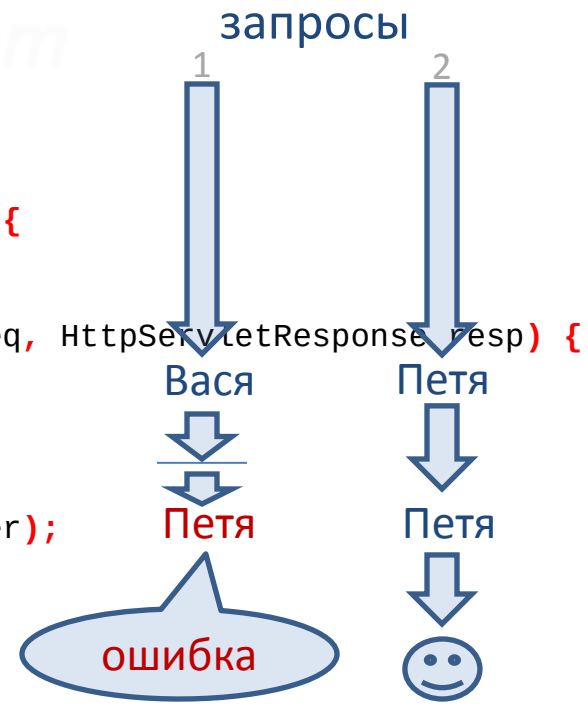
```
public class MyServlet extends HttpServlet {
    private User currentUser;

    public void doGet(…req,…resp) {
        currentUser = authorizeUser(req);

        // ...

        resp.getWriter().println(currentUser)
    }
}
```



Классические серверные паттерны

- Потоки OS

```
public class MyServlet extends HttpServlet {
    private User currentUser;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) {
        currentUser = authorizeUser(req);

        // ...

        resp.getWriter().println(currentUser);
    }
}
```



Сравнение

	Процессы OS	Потоки OS
Управление	Пул процессов (pre-fork)	Пул потоков
Переменные	Только независимые	Общие + независимые
Передача информации	Только строки	Структуры данных и объекты
Доступ к «чужим» методам	RPC, сигналы, очереди…	Прямой вызов
Надежность	Процессы разделены на уровне OS	Один поток может уронить все (кроме VM-потоков)
Масштабируемость	Слабая	NPTL: 5000+ потоков
Сложность поддержки	Сравнима с обычной программой	Синхронизация доступа к ресурсам, RACE conditions

Гибридные паттерны

- Процессы + потоки
  - Несколько процессов (Worker MPM)
    - В каждом процессе много потоков
    - Увеличенная надежность
  - Event MPM
    - отдельный поток для Keep-Alive
    - позволяет «рабочим» потокам работать дальше, пока висит Keep-Alive

Сравнение

- Обычный цикл

1. Принять запрос
2. Сгенерировать страничку
3. Выдать страницу

Сравнение

COMET

1. Принять запрос
   - посетитель подключился
   - ждать события
2. Сгенерировать ответ
3. Выдать ответ

Особенности

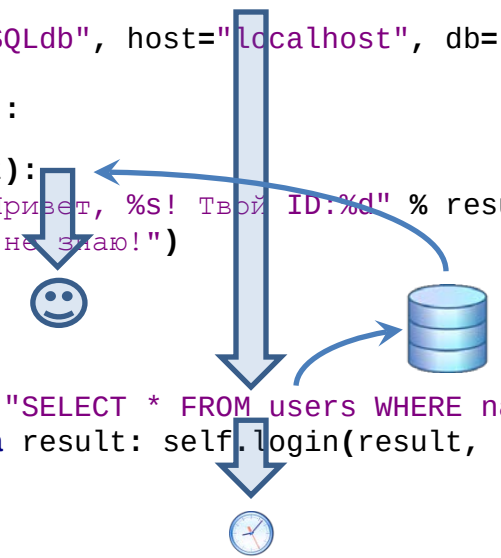
- Большое количество ожидающих соединений
  - Потоки/процессы съедают ресурсы
- Активная межпроцессная коммуникация
- Небольшой размер ответа

События в едином потоке

simple.py

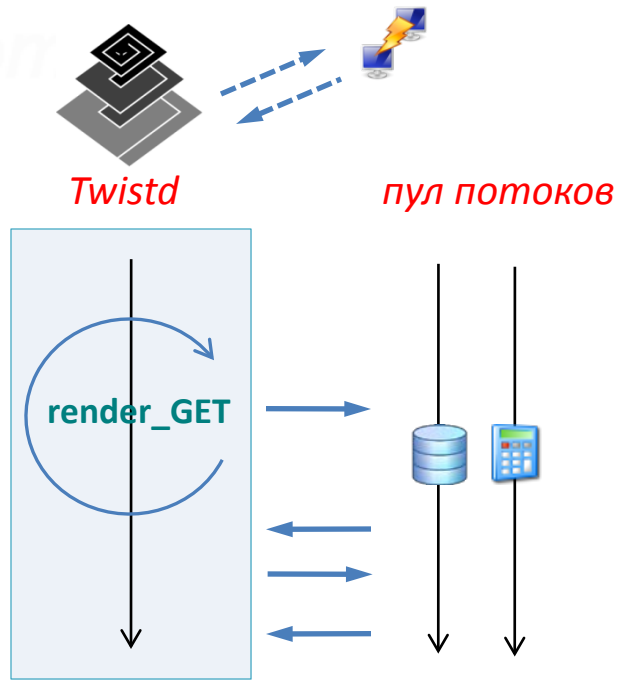
- Twisted, node.js, EventMachine, POE, phpDaemon
  - все действия, которые требуют ожидания, совершаются асинхронно

```
dbpool = adbapi.ConnectionPool("MySQLdb", host="localhost", db="test")

class MyResource(resource.Resource):

    def login(self, result, request):
        if result: request.write("Привет, %s! Твой ID:%d" % result[0])
        else: request.write("Я Вас не знаю!")
        request.finish()

    def render_GET(self, request):
        name = request.args['name']
        deferred = dbpool.runQuery("SELECT * FROM users WHERE name = '%s' " % name)
        deferred.addCallback(lambda result: self.login(result, request))
        return server.NOT_DONE_YET
```



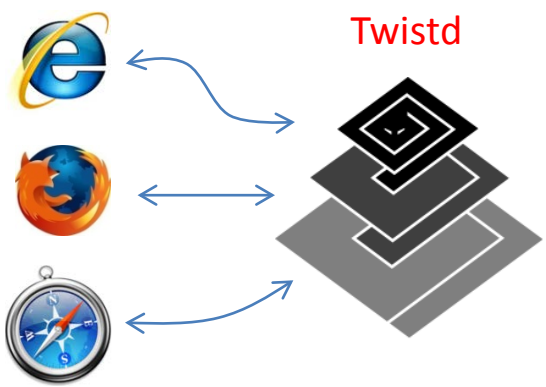
Модель работы

```
class MyResource(resource.Resource):

    def login(self, result, request):
        request.write("…" % result)
        request.finish()

    def render_GET(self, request):
        name = request.args['name']
        deferred = dbpool.runQuery("…" % name)
        deferred.addCallback(self.login)
        return server.NOT_DONE_YET
```



Общие данные

- Пример COMET-приложения
  - comet.py comet.js
  - все клиенты в одном массиве




```
class ClientManager:

    clients = []

    def registerClient(self, client):
        self.clients.append(client)

    def broadcastMessage(self, message):
        for client in self.clients:
            client.write(message)
            client.finish()
        self.clients = []
```

Мини-чат

- Демо
- server.js
  - POST для publish
- http://chat.nodejs.org

Отладка

- node --debug (--debug-brk)
  - В коде *debugger*
- node-inspector для отладки node в браузере
  - http://howtonode.org/debugging-with-node-inspector
  - Safari/Chrome
- PHPStorm plugin

Много клиентов? Экономим память!

1. прочитать все данные (DB/файл)
2. отдавать их

обычный паттерн
много медленных клиентов –
съест всю память

1. прочитать кусок данных
2. отослать

... стримить все, по возможности...

Мультипоточность

- GIL: Только один поток кода одновременно
  - Что делать при 100% загрузке CPU?
  - Запускать несколько процессов сервера
    - данные не в едином адресном пространстве
  - Выделять «тяжелый» код в расширения на C
    - Могут убирать GIL
  - jRuby: GIL частично убран

…Just Say No to the combined evils of locking, deadlocks, lock granularity, livelocks, nondeterminism and race conditions…
© Guido v.Rossum

Характеристики метода

- Для удобного асинхронного вызова нужно Async API
  - Хорошо когда оно есть
  - Плохо, что оно не всегда есть
- Нет синхронизации
- GIL: Только один поток кода одновременно
  - Что делать при 100% загрузке CPU?
    1. Выделять сложные операции в потоки (Python – GIL, сетевые операции не thread-safe)
    2. Запускать несколько экземпляров (процессов) сервера (данные не в едином адресном пространстве)

Fibers

- https://github.com/laverdet/node-fibers
- http://habrahabr.ru/blogs/nodejs/116124/
- http://en.wikipedia.org/wiki/Fiber_%28computer_science%29
- http://ruby-doc.org/core-1.9/classes/Fiber.html

Почитать

- Twisted
  - Официальная документация и учебник
  - Twisted.Web In 60 Seconds
- Node.js
  - nodejs.ru
  - Streaming file uploads with node.js
  - Socket.IO
  - Now.JS (RPC)

Continuations (Jetty, GlassFish…)

- Запрос 1 ожидает сообщение

```
public class SubscribeServlet extends HttpServlet {

    protected void doGet(HttpServletRequest req, HttpServletResponse resp) {
        // 1. запрос «упаковывается» в объект continuation
        Continuation continuation = ContinuationSupport.getContinuation(req);

        if (continuation.isInitial()) {  // true
            // 2. сообщить серверу, что запрос неокончен
            continuation.suspend();

            // 3. передать continuation асинхронному обработчику
            ClientManager.getInstance().registerClient(continuation);
        }
}
```

Continuation (неоконченный запрос)
«подвис» на сервере

Continuations

- Запрос 2 инициирует событие

```
public class ClientManager {
    // 1. массив клиентов / запросов
    ArrayList<Continuation> continuations = new ArrayList<Continuation>();

    public synchronized void broadcastMessage(String message) {
        for(Continuation continuation: continuations) {
            // 2. передать информацию о событии в continuation
            continuation.setAttribute("message", message);

            // 3. попросить сервер продолжить обработку
            continuation.resume();

        }
    }
}
```

2

можно в цикле выдавать ответы в запросы

Continuations

- Сервер перезапускает запрос 1

```
public class SubscribeServlet extends HttpServlet {

    protected void doGet(HttpServletRequest req, HttpServletResponse resp) {

        // 1(2). возвратит объект continuation, соответствующий запросу
        Continuation continuation = ContinuationSupport.getContinuation(req);

        if (continuation.isInitial()) {    // true
            // 1(1). сообщить серверу, что запрос неокончен
            continuation.suspend();

            // 1(1). передать continuation асинхронному обработчику
            ClientManager.getInstance().registerClient(continuation);
        } else {                           // false
            String message = (String)continuation.getAttribute("message");
            resp.getWriter().print(message);
        }
    }
}
```

Характеристики метода

- Основан на потоках
  - Используются все ядра
  - Синхронизация, многопоточное программирование

```
public synchronized void broadcastMessage
```

- Многопоточная обработка оживших continuations

```
for(Continuation continuation: continuations) {
   // разбросает обработку continuations по потокам
   continuation.resume()
}
```

Почитать

- Continuations to Continue
- Continuations (статья старовата)
- JSR-000315 Java Servlet 3.0 (Спецификация)
  - Glassfish

Микронити

- Erlang, Stackless Python(GIL)…
- Предыдущие подходы:
  - Сохранить запрос в памяти и освободить поток для нового запроса
- Микронити:
  - Сделать поток максимально «легким» - и можно не освобождать
  - Решение: отказ от потоков и стека OS, свои потоки
    - Green threads

Микронити

- Erlang
  - почему?
    - Структуры данных никогда не меняются

      +
    - Green Threads
  - chat_web.erl

Почитать

- A Million-user Comet Application with Mochiweb: Part 1, Part 2, Part 3
- Building an Erlang chat server with Comet: Part1, Part 2, Part 3
- Comet web chat (with MochiWeb)
  - Using the mochiweb project skeleton
- Mochiweb source
  - Документация к API
- Erlang
  - Официальный сайт
  - Erlang in Practice (screencast)