Swoole底层原理与应用实践

@hantianfeng Rango-韩天峰

关于我

- 车轮互联总架构师
- Swoole 开源项目创始人
- 微博：@hantianfeng
- Github: https://github.com/matyhtf

分享内容

一．网络 IO 基础，Socket 通信、同步 IO、异步 IO

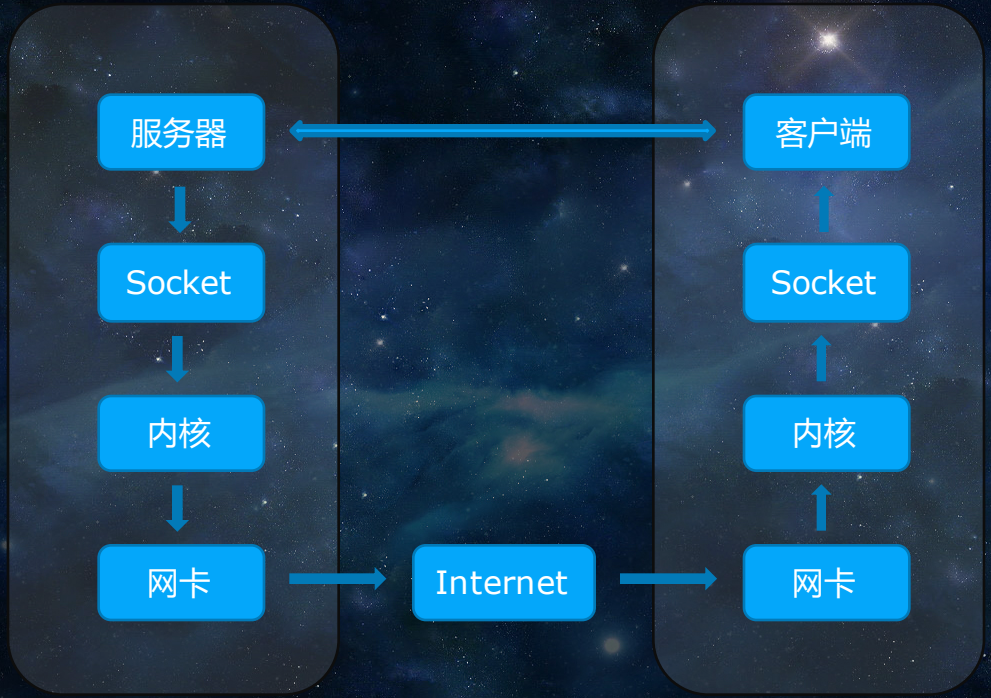
二．Swoole 模块，实现原理、应用

三．Swoole 2.0 协程，实现原理、应用

01 Socket

```c
int socket(int domain, int type, int protocol);
```

Type	Stream(TCP) DGRAM (UDP) Raw(ICMP、ARP)
Domain	AF_INET(IPv4: 192.168.1.122) AF_INET6(IPv6: C0A8:1E01::) AF_LOCAL(UnixSocket: /tmp/fpm.sock)
Status	Block Nonblock


服务器
Socket
内核
网卡
Internet
客户端
Socket
内核
网卡

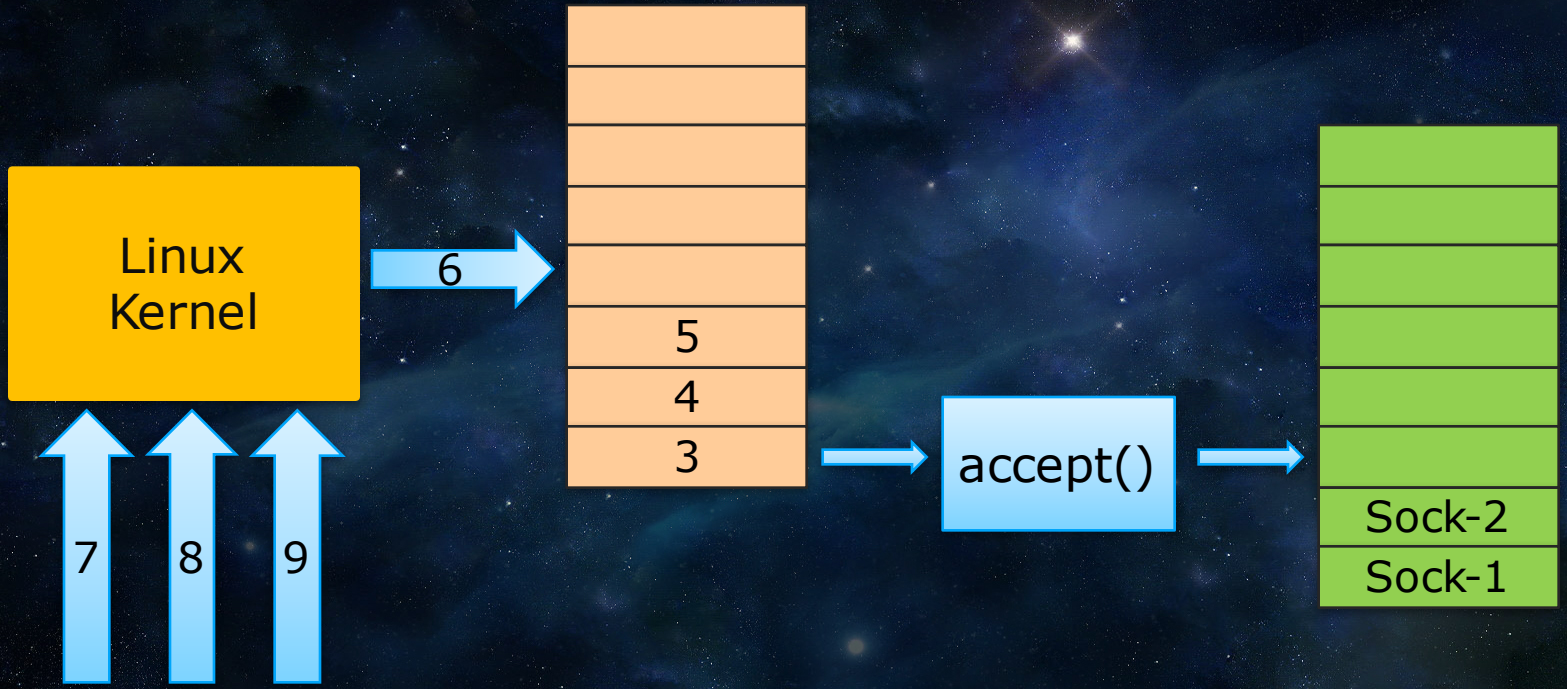
服务器	客户端
socket	socket
bind	[bind]
listen	-
accept	connect
recv	send
send	recv
close	close

HTTP：80
SMTP：25
SSH：22

Listen队列
Backlog设置

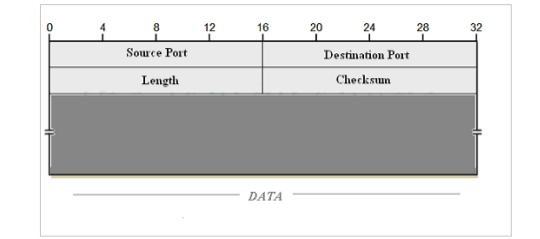
网卡1：192.168.1.X
网卡2：10.10.2.X

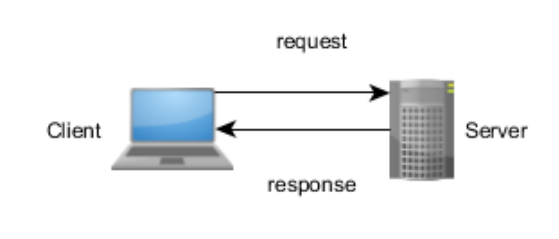
TCP三次握手


Linux
Kernel
6
5
4
3
accept()
Sock-2
Sock-1
7
8
9

Socket ：send & recv

1. STMP、FTP（\r\n）
2. HTTP、Redis
3. 二进制协议（长度+数据）
4. Request/Response





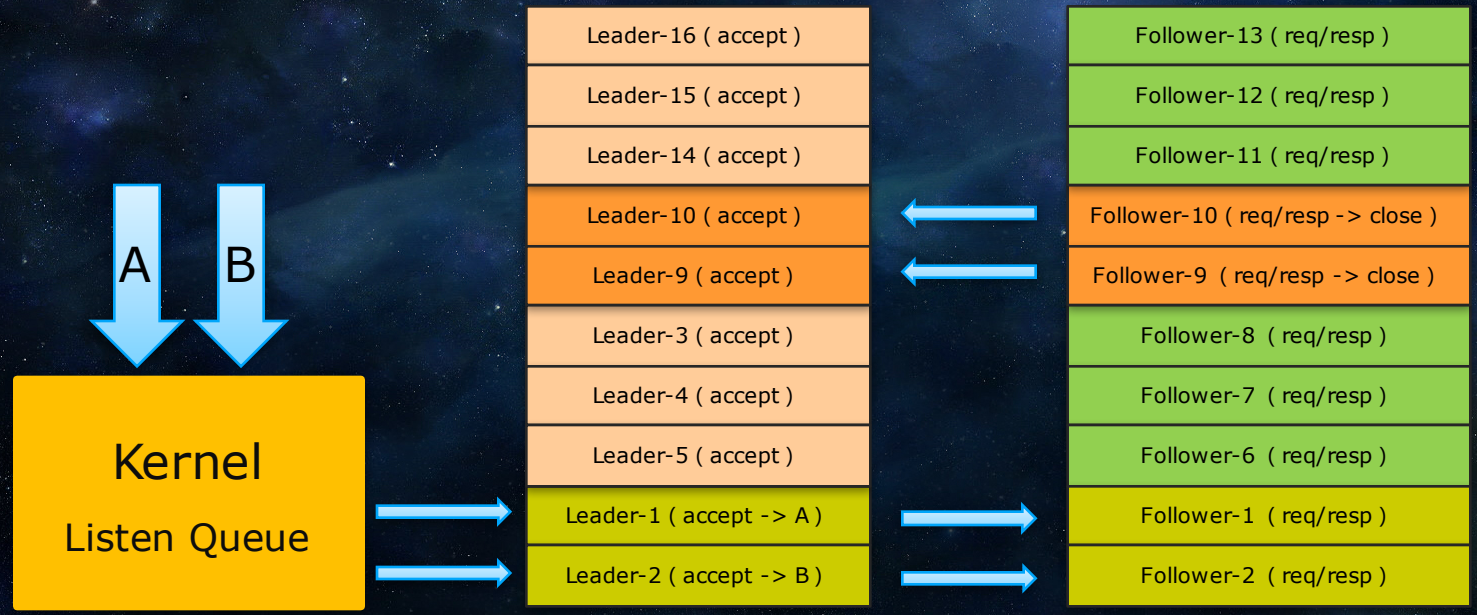
```php
<?php
$socket = stream_socket_server("tcp://0.0.0.0:8000",
    $errno, $errstr);
while ($conn = stream_socket_accept($socket)) {
    if (pcntl_fork() == 0) {
        $request = fread($conn);
        fwrite($conn, "hello world\n");
        fclose($conn);
        exit;
    }
}
```

Fork-On-Accept

1. 创建销毁进程开销极大
2. 高并发下会创建过多进程导致系统崩溃

```php
<?php
$socket = stream_socket_server("tcp://0.0.0.0:8000",
    $errno, $errstr);
for($i = 0; $i < 16; $i++) {
    if (pcntl_fork() > 0)  {
        continue;
    }
    while ($conn = stream_socket_accept($socket)) {
        $request = fread($conn);
        fwrite($conn, "hello world\n");
        fclose($conn);
    }
}
//pcntl_wait
```

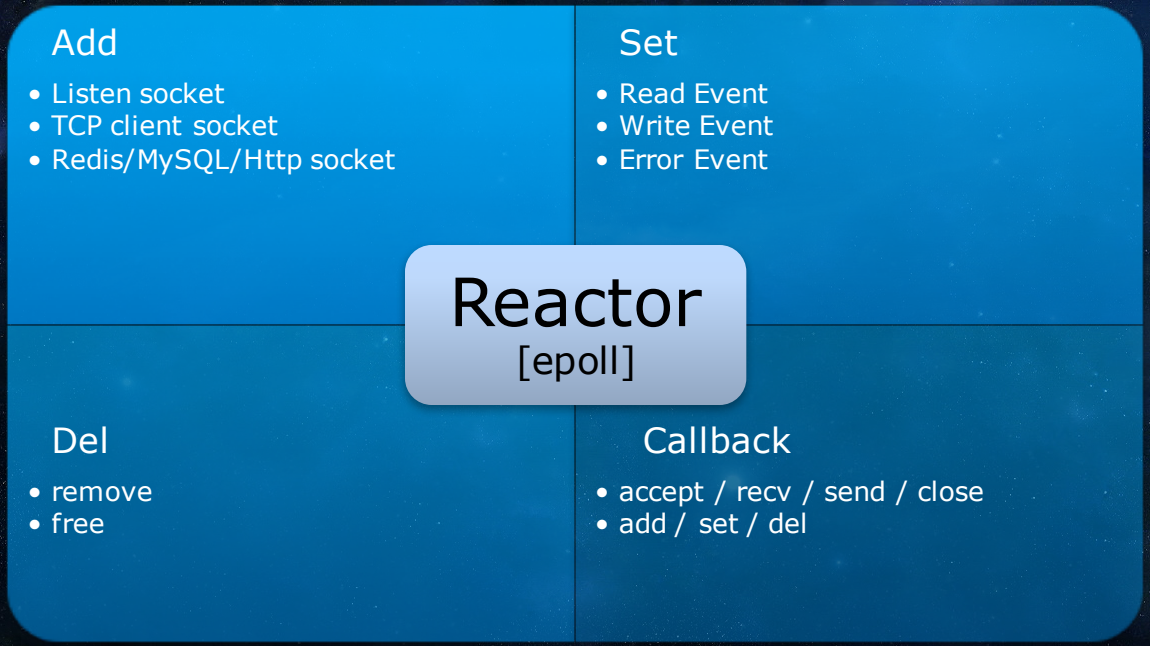
Leader-Follower 模型



新问题

1. 长连接服务：C1000K 问题，如 WebIM、推送服务
2. 外部 Http API 调用：平均耗时可能达到 100ms – 30s
3. 高性能场景：QPS 1万，如秒杀、抢红包
4. 慢速网络：某些客户端网络极差，发送 1M 数据需要 1分钟

异步 IO ：Reactor 模型



异步 IO

- Linux Epoll / FreeBSD Kqueue / Windows IOCP
- Libevent、Event 扩展
- Nginx / Netty / Swoole / Go / Node.js

```php
$reactor = new Reactor;
$socket = stream_socket_server("tcp://.0.0.0:8000", $errno, $errstr);
$reactor->add($socket, EVENT_READ, function ($sock) use ($reactor) {
    $conn = strem_socket_accept($sock);
    $reactor->add($conn, EVENT_READ, function ($sock) use ($reactor) {
        $request = fread($sock, 8192);
        $response = handleRequest($request);
        $reactor->add($conn, EVENT_WRITE, function ($sock) use ($reactor, $response) {
            fwrite($sock, $response);
            $reactor->del($sock);
            fclose($sock);
        });
    });
});
$reactor->wait();
```

02 Swoole

PHP 扩展

- stream、sockets
- strem_select、socket_select
- libevent、event
- pcntl、posix、pthreads [不稳定]
- sysvmsg、sysvshm、sysvsem、shmod

单进程单线程

- 只使用 单进程单线程：如 Node.js（未内置，需要第三方库）
- 无法利用多核，在24核CPU硬件上只能发挥1个核的作用

Swoole 内置多进程

```php
$serv = new Swoole\Server("127.0.0.1", 9501);
$serv->set(array(
    'worker_num' => 8,    //工作进程数量
));
$serv->on('connect', function ($serv, $fd) {
    echo "Client:Connect.\n";
});
$serv->on('receive', function ($serv, $fd, $from_id, $data) {
    $serv->send($fd, 'Swoole: '.$data);
    $serv->close($fd);
});
$serv->on('close', function ($serv, $fd) {
    echo "Client: Close.\n";
});
$serv->start();
```

配置：worker_num = 8
建议：设置为 CPU核数，可利用到所有 CPU核

现实问题

- 惊群效应：发生事件时会唤醒所有 epoll 的进程
- 处理不均衡：Reactor 处理事件发生停顿

  1) 某个请求 encode 很大的 json 消耗 10ms

  2) 心跳检测，遍历 100 万连接

  3) 广播，遍历并向 100 万连接发送消息
- 业务逻辑异常/致命错误：Worker进程挂掉，导致所有连接中断，大规模TCP客户端重连，产生网络风暴

现实问题

- 连接之间通信：A连接在进程1中，B连接在进程2中
- 进程1和进程2同时向连接A发送2M数据
- 逻辑代码中存在阻塞IO

Master进程

- Master线程，Accept连接
- Reactor线程，收发数据
- 心跳线程，定期检测坏连接

Worker进程

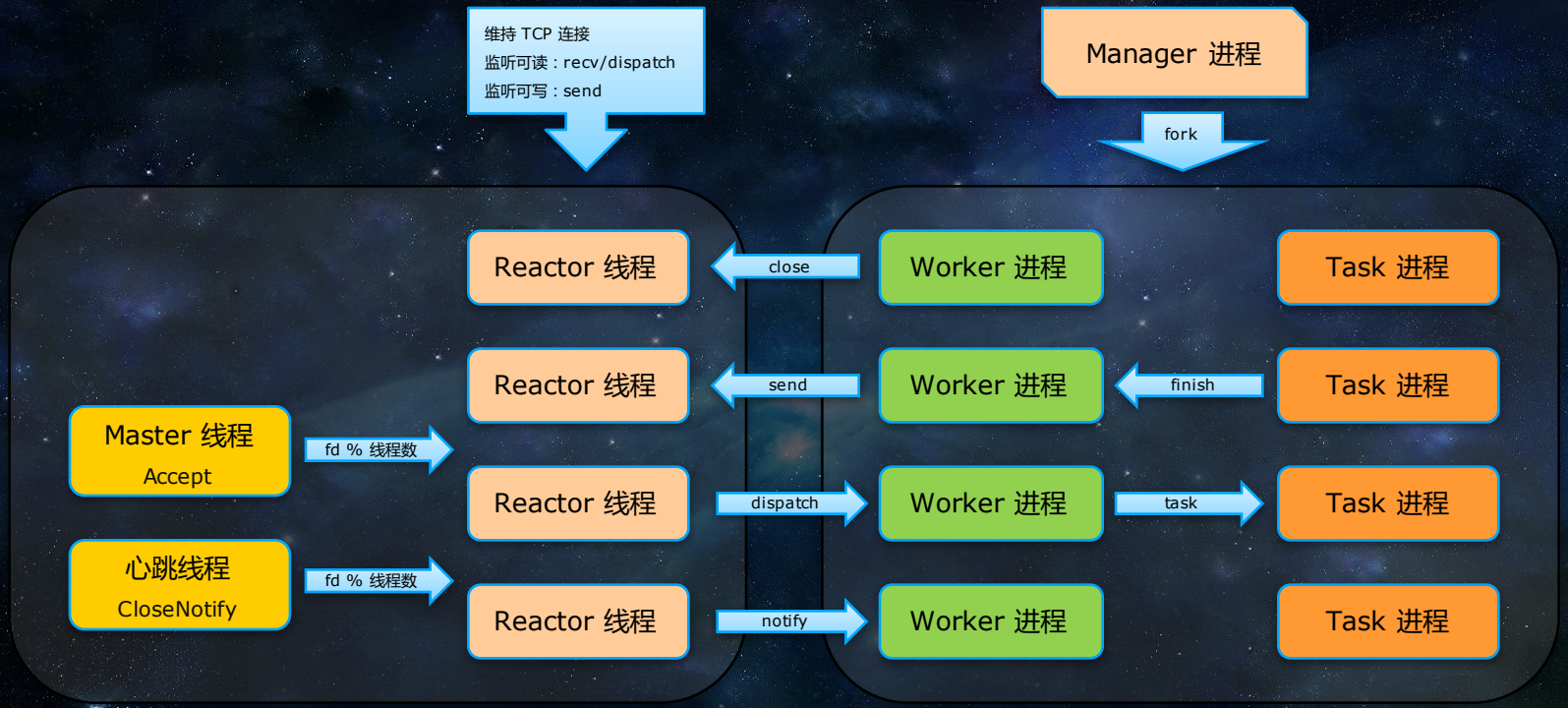
- 执行PHP代码，处理请求
- 业务逻辑

Manager进程

- 管理Worker和Task进程
- Reload

Task进程

- 处理慢速任务


维持 TCP 连接
监听可读：recv/dispatch
监听可写：send
Manager 进程
fork
Reactor 线程
Reactor 线程
Reactor 线程
Reactor 线程
close
send
dispatch
notify
Worker 进程
Worker 进程
Worker 进程
Worker 进程
finish
task
Task 进程
Task 进程
Task 进程
Task 进程
Master 线程
Accept
心跳线程
CloseNotify
fd % 线程数
fd % 线程数

心跳线程

```
$serv->set(array(
    'heartbeat_idle_time' => 600,
    'heartbeat_check_interval' => 60,
));
```

Task 进程池

```php
$serv->set(array('task_worker_num' => 4));

$serv->on('receive', function($serv, $fd, $rid, $data) {
    $task_id = $serv->task($data, -1, function ($serv, $task_id, $result) {
        $serv->send($fd, json_encode($result));
    });
});

$serv->on('task', function ($serv, $task_id, $wid, $data) {
    return array('code' => 1, 'data' => $data);
});
```

dispatch_mode

1：轮询分配，每个请求随机分配给 Worker

2：固定分配，同一个TCP连接的请求分配给固定的 Worker

3：忙闲分配，请求只分配给空闲状态的 Worker

4：按 IP 固定分配，同一个来源IP的请求分配给固定的 Worker

5：需要 PHP 代码层使用 bind 方法设置绑定关系

	有状态	无状态
同步	2/4/5	1/3
异步		1

BASE 模式

- 等于 Node.js
- BASE 仍然支持 Task 功能 和 worker_num 的设置
- 某些情况下，需要一个简单轻量的程序，可以用BASE模式

```
$serv = new swoole_server("127.0.0.1", 9502, SWOOLE_BASE);
```

连接迭代器

- 用于遍历所有 TCP 连接
- 可在任意进程内（Task/Worker）向任意 TCP 连接发送数据
- 多个进程向同一连接安全地并发发送 2M 数据

```
foreach($server->connections as $fd)
{
    $server->send($fd, "hello");
}
```

SessionId

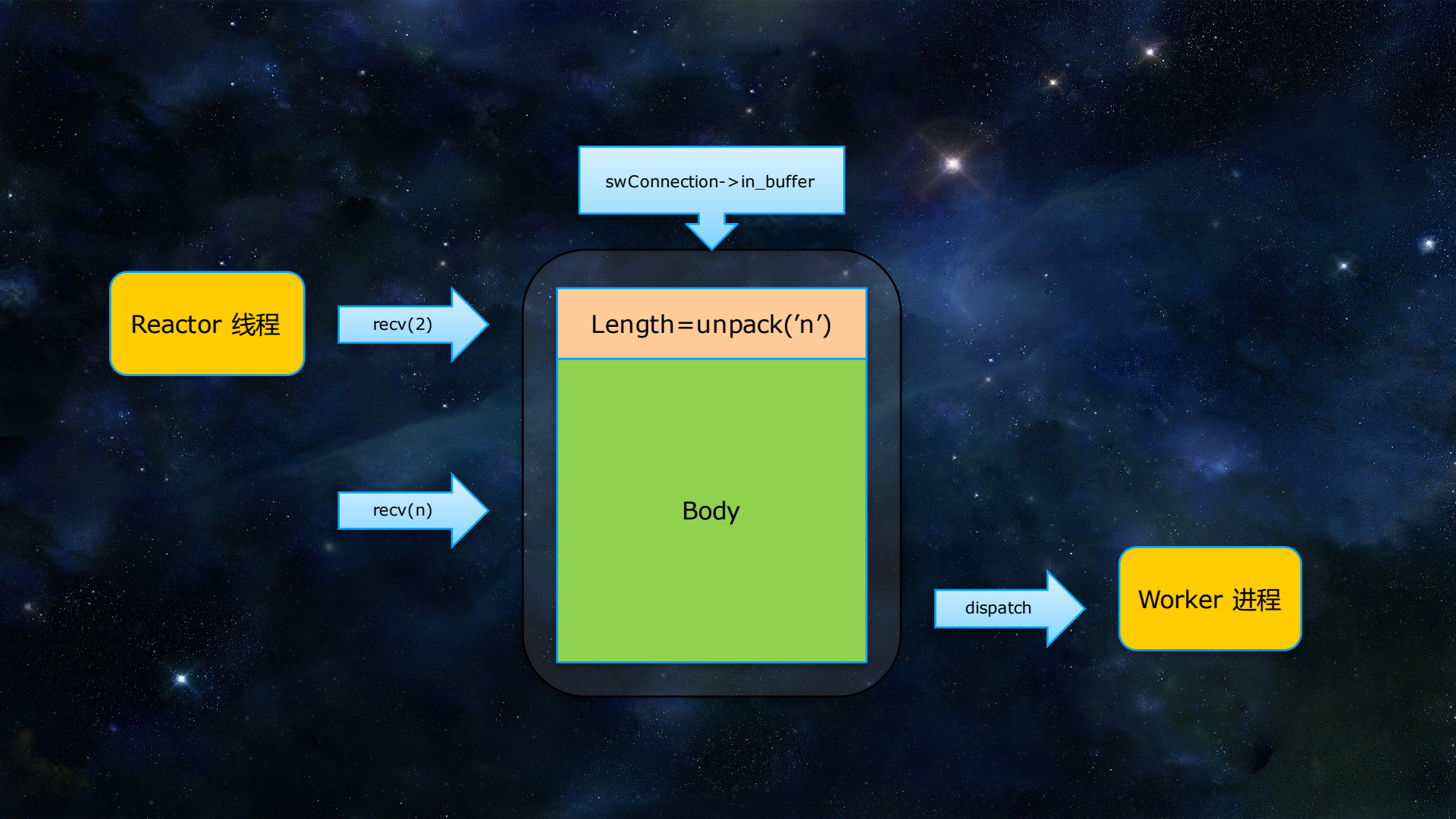
- $fd 并不是实际的 Socket Fd，而是 SessionId
- $fd 在短时间内不会复用， 1 - 1600万
- 底层会双向检测，SessionId 和 SocketFd 是否一致
- 不会出现向已关闭连接发送数据的问题

长度协议

```php
$serv->set([
    'open_length_check' => true,
    'package_max_length' => 81920,
    'package_length_type' => 'n',
    'package_length_offset' => 0,
    'package_body_offset' => 2,
]);
```

EOF协议

```
$server->set(array(
    'open_eof_split' => true,
    'package_eof' => "\r\n",
));
```


swConnection->in_buffer
Reactor 线程
recv(2)
Length=unpack('n')
recv(n)
Body
dispatch
Worker 进程

数据发送

```
$server->set(array(
    'buffer_output_size' => 1024 * 1024 * 8,
));
```

```
$server->send($fd, str_repeat('A', 1024 * 1024 * 8));
```

数据发送

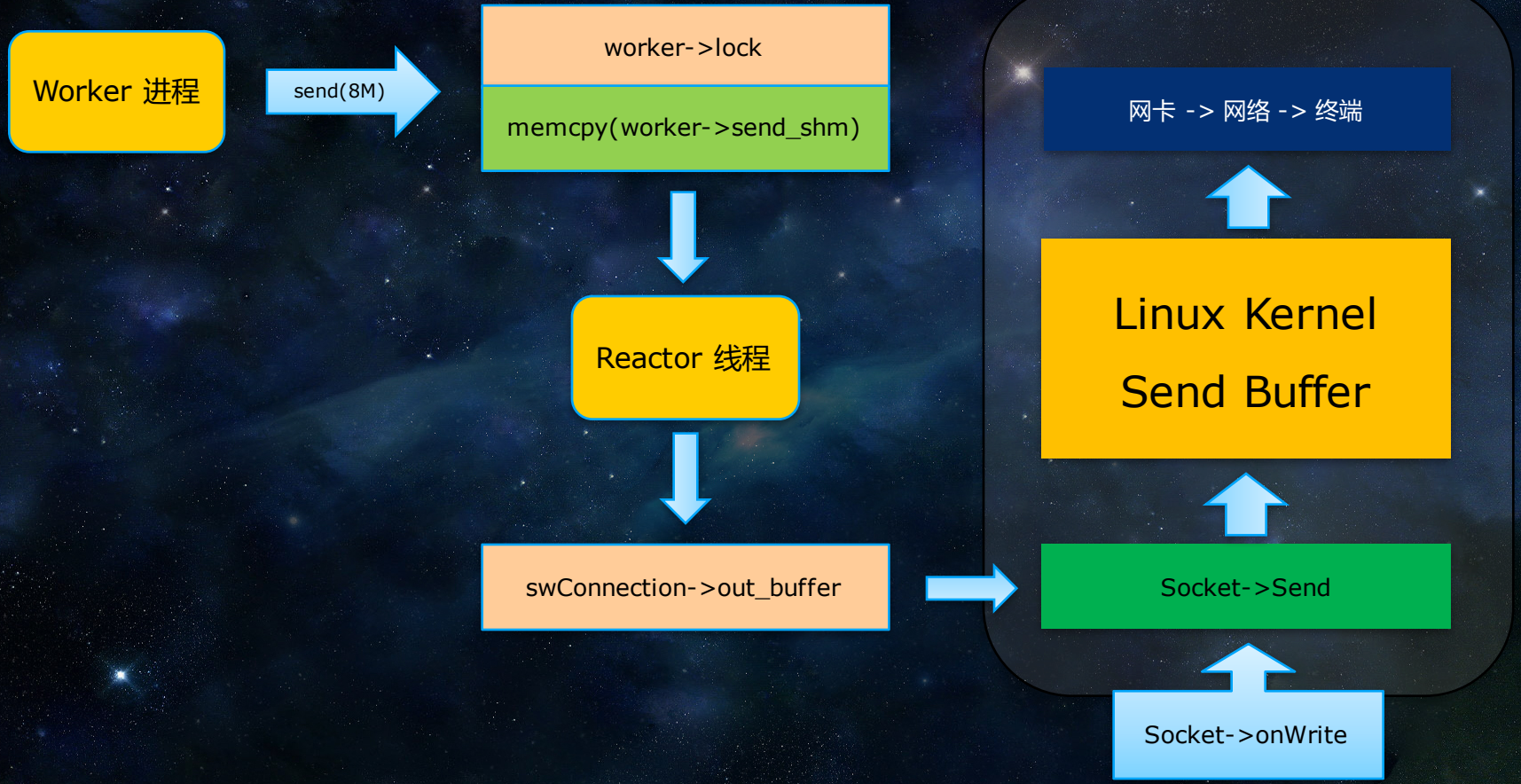
```
$server->set(array(
    'buffer_output_size' => 1024 * 1024 * 8,
));
```

```
$server->send($fd, str_repeat('A', 1024 * 1024 * 8));
```

```
struct _swWorker
{
    swLock lock;
    void *send_shm;
};
```

```
swWorker *worker = swServer_get_worker(serv, SwooleWG.id);

swPackage_response response;
response.length = resp->length;
response.worker_id = SwooleWG.id;

worker->lock.lock(&worker->lock);
memcpy(worker->send_shm, resp->data, resp->length);

ret = swWorker_send2reactor(&ev_data, sendn, session_id);
```

```
memcpy(&pkg_resp, resp.data, sizeof(pkg_resp));
worker = swServer_get_worker(SwooleG.serv, pkg_resp.worker_id);

_send.data = worker->send_shm;
_send.length = pkg_resp.length;

swReactorThread_send(&_send);
worker->lock.unlock(&worker->lock);
```


Worker 进程
send(8M)
worker->lock
memcpy(worker->send_shm)
Reactor 线程
swConnection->out_buffer
网卡 -> 网络 -> 终端
Linux Kernel
Send Buffer
Socket->Send
Socket->onWrite

Swoole 定时器

```php
$server->tick(1000, function ($timerId) use ($server, $fd) {
    $server->send($fd, "hello world");
});

$server->after(5000, function () use ($server, $fd) {
    $server->close($fd);
});

$server->defer(function () use ($mysql) {
    $mysql->close();
});
```

异步Http服务器

```php
$serv = new Swoole\Http\Server("127.0.0.1", 9502);

$serv->on('Request', function ($request, $response) {
    var_dump($request->get);
    var_dump($request->post);
    var_dump($request->cookie);
    var_dump($request->files);
    var_dump($request->header);
    var_dump($request->server);

    $response->cookie("User", "Swoole");
    $response->header("X-Server", "Swoole");
    $response->end("Hello Swoole!");
});

$serv->start();
```

异步WebSocket服务器

```php
$serv = new Swoole\WebSocket\Server("127.0.0.1", 9502);

$serv->on('Open', function ($server, $req)
{
    echo "connection open: " . $req->fd;
});

$serv->on('Message', function ($server, $frame)
{
    echo "message: " . $frame->data;
    $server->push($frame->fd, json_encode(["hello", "world"]));
});

$serv->on('Close', function ($server, $fd)
{
    echo "connection close: " . $fd;
});

$serv->start();
```

异步MySQL客户端

```php
$db = new Swoole\MySQL;
$server = array(
    'host' => '127.0.0.1',
    'user' => 'test',
    'password' => 'test',
    'database' => 'test',
);

$db->connect($server, function ($db, $result)
{
    $db->query("show tables", function (Swoole\MySQL $db, $result)
    {
        var_dump($result);
        $db->close();
    });
});
```

异步Redis客户端

```php
$redis = new Swoole\Redis;
$redis->connect('127.0.0.1', 6379, function ($redis, $result)
{
    $redis->set('test_key', 'value', function ($redis, $result)
    {
        $redis->get('test_key', function ($redis, $result)
        {
            var_dump($result);
        });
    });
});
```

异步Http/WebSocket客户端

```php
$cli = new Swoole\Http\Client('127.0.0.1', 80);
$cli->setHeaders(array('User-Agent' => 'swoole-http-client'));
$cli->setCookies(array('test' => 'value'));

$cli->post('/dump.php', array("test" => 'abc'), function ($cli)
{
    var_dump($cli->body);
    $cli->get('/index.php', function ($cli)
    {
        var_dump($cli->cookies);
        var_dump($cli->headers);
    });
});
```

并行编程的难题

- **多线程编程：**1) 读写全局需要加锁，一旦忘记加锁会出现数据同步问题，大量并发时出现严重错误。2) 锁的粒度过大，导致同时只有一个线程在跑（Python的GIL）。3) 复杂的锁逻辑，可能出现重复加锁，导致线程死锁。
- **多进程编程：**1) 进程是隔离的，A进程无法读取B进程的变量。2) 使用管道进行进程间通信，有一定开销，不方便。3) 使用共享内存，会出现和多线程一样的难题。
- **Lock-Free编程：**非常复杂，容易出错，如同走钢丝。只有极少数熟悉底层CPU硬件原理的人能掌握。

原子计数器

```php
<?php
use Swoole\\Atomic;

$atomic = new Atomic(0);
if (pcntl_fork()) {
    for($i=0; $i<10000; $i++) {
        $atomic->add(1); //+1
    }
} else {
    for($i=0; $i<10000; $i++) {
        $atomic->sub(1); //-1
    }
    $atomic->cmpset(5, 0); //等于5时，设置为0，
    $atomic->get(); //获取当前计数
}
```

并发Table

```php
<?php
use Swoole\\Table;

$table = new Table(1024*1024);
$table->column('id', swoole_table::TYPE_INT, 4);       //1,2,4,8
$table->column('name', swoole_table::TYPE_STRING, 64);
$table->create();

if (pcntl_fork()) {
    for($i=0; $i<10000; $i++) {
        $table->set('test_key_'.$i, array('id' => $i, 'name' => 'v'.$i));
    }
} else {
    for($i=0; $i<10000; $i++) {
        $table->get('test_key_'.$i);
    }
}
```

管道通信

```php
<?php
$serv = new swoole_server("0.0.0.0", 9501);

$serv->set(array(
    'worker_num' => 2,
));

$serv->on('pipeMessage', function($serv, $src_worker_id, $data) {
    echo "#{$serv->worker_id} message from #$src_worker_id: $data\\n";
});

$serv->on('receive', function ($serv, $fd, $reactor_id, $data) {
    $dst_worker_id = $serv->setting['worker_num'] - ($serv->worker_id + 1);
    $serv->sendMessage("hello", $dst_worker_id);
});

$serv->start();
```

03 Swoole 2.0

Callback Hell

- 复杂项目中，异步回调嵌套层次多，维护困难
- Yield / Generator 实现协程

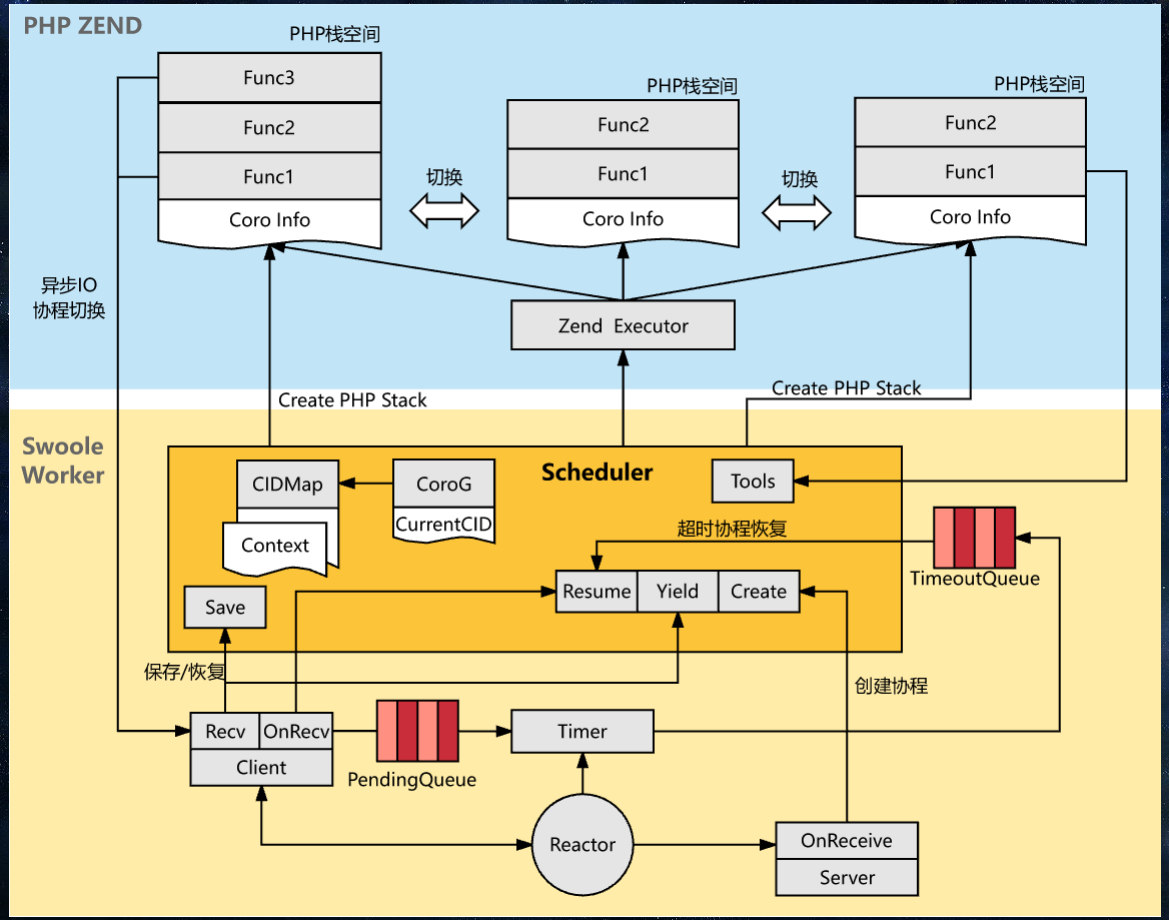
```php
$tcpReturn = (yield $this->tcpTest());
public function tcpTest(){
    $ip = '127.0.0.1';
    $port = '9905';
    $data = 'test';
    $timeout = 0.5; //second
    yield new Swoole\Client\TCP($ip, $port, $data, $timeout);
}
```

Swoole2.0 原生协程

```php
$server = new Swoole\Http\Server('127.0.0.1', 9501);
$server->on('Request', function($request, $response) {
    $mysql = new Swoole\Coroutine\MySQL();
    $mysql->connect([
        'host' => '127.0.0.1',
        'user' => 'user',
        'password' => 'test',
        'database' => 'test',
    ]);
    $res = $mysql->query('select * from userinfo limit 10');
    $response->end(json_encode($res));
});
$server->start();
```

原生协程优势

- 没有任何额外的关键词，（yield/await/async）
- 同步的写法，与传统LAMP编程完全一致
- 底层协程调度，实现异步IO，可提供超大并发能力
- Swoole 2.0 提供了协程版本的 TCP/UDP 客户端、Redis客户端、MySQL客户端、Http客户端


PHP ZEND
PHP栈空间
Func3
Func2
Func1
Coro Info
切换
PHP栈空间
Func2
Func1
Coro Info
切换
PHP栈空间
Func2
Func1
Coro Info
异步IO
协程切换
Zend Executor
Create PHP Stack
Create PHP Stack
Swoole
Worker
Scheduler
CIDMap
CoroG
CurrentCID
Context
Tools
超时协程恢复
TimeoutQueue
Resume
Yield
Create
Save
保存/恢复
创建协程
Recv
OnRecv
Client
PendingQueue
Timer
Reactor
OnReceive
Server

Swoole2.0 压力测试

Http长连接

	Swoole2.0	callback	PHP yield
Echo	540000	530000	80000
1次后端	83900	82400	41000
3次后端	31000	31000	20000

Http短连接

	Swoole2.0	callback	PHP yield
Echo	56000	55000	43000
1次后端	43600	43300	30000
3次后端	27000	27000	17000

04

Swoole 未来

Swoole 领域

- 互联网应用
- 网络游戏
- 物联网
- 移动网络

Swoole 生态

- 工具
- 框架
- 分布式
- 通信协议

Swoole 扩展

- 1.0分支
- 2.0分支

Swoole 2017

- 质量控制：单元测试率覆盖率 100%，RFC 投票机制
- 文档建设：中文出版物 2本，英文出版物 1本，英文文档
- 商业化：

  1) 全职开发者

  2) Swoole Compiler，编译PHP程序为二进制代码，保护源码，优化性能，解决 Opcache BUG

THANK YOU

- 期待再见-

Q & A