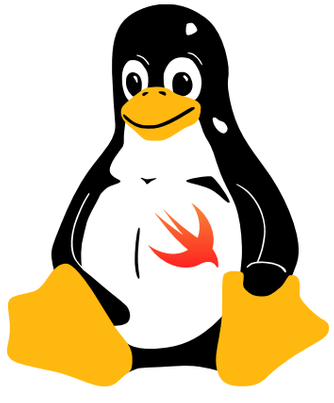
Server Side Swift

Swift在服务器端的应用程序开发

Kyle Jessup

PerfectlySoft Inc.
www.perfect.org
@kylej

What is Server Side Swift?

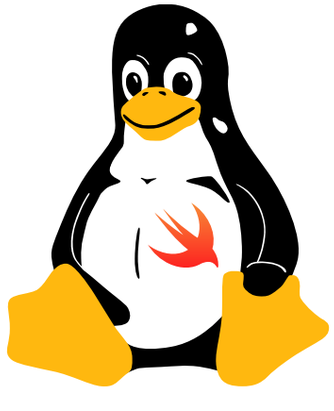


Swift application deployed (primarily) on Linux

目前Swift服务器端应用（主要）适用于Linux系统部署

- Located on cloud provider, co-located, in-house
- 其形式可以是云部署、主机托管或者自建服务器
- Responding to network clients
- 为网络客户提供服务

What is Server Side Swift?



API servers, workers, web sites

- 作用主要是为移动端提供后台服务接口支持
- iOS or Android backends

Peer-to-peer

- 或者是点对点，即服务器和客户机为同一设备
- Client & server on same device

Why Swift on the server?

Static compilation - 静态编译而非执行解释脚本

- Performance 性能更加卓越
- Shifts work to compile-time 将工作转移到编译阶段完成
- Work dynamic languages do at run-time
- 同时实现与其他脚本语言运行时同样的功能

Why Swift on the server?

Strong type checking 增强类型检查

- Detect many errors before running 在运行前检查错误
- Fewer surprises at run-time 避免服务器运行时异常
- Better for maintainable systems?
- 长期维护方面是否更好呢?
  - Impacts of Swift version changes getting better
  - Swift升级版本只会变得更好
  - Same problem for Swift on iOS/macOS
  - iOS / macOS上的Swift问题是一样的


Why Swift on the server?

Common language for clients & servers
服务器和客户机能够共享相同的计算机语言

- Standardize development teams on one language
- 开发团队的所有工程师能够使用同一种计算机语言
  - Leverage existing iOS/macOS Swift knowledge
  - 继承、沿袭现有iOS / macOS Swift语言知识体系

Why Swift on the server?

Common language for clients & servers
服务器和客户机能够共享相同的计算机语言

- Share code 共享代码
- Server & client use same base Swift code
- 服务器和客户端可以采用相同的Swift源程序
- Common structs, classes, core algorithms
- 共享数据结构、对象类型和核心算法
- Swift extensions provide specific behaviour
- 为特定行为提供类型扩展

Why Swift on the server?

Common language for clients & servers
服务器和客户机能够共享相同的计算机语言

- Simplified initial client/server debugging 简化服务器/客户端调试
  - Developing & running both in Xcode 整个过程都在Xcode实现

Why Swift on the server?

Good C bridging 良好的C语言桥接系统

- Use 3rd party libraries, drivers
- 可以直接使用第三方函数库或驱动程序
- Leverage existing functionality
- 直接调用现有C语言函数库中的函数
- No need to write C in most cases
- 多数情况下不需要写C语言程序

Why ~~Swift~~ on the server?
服务器端不采用Swift的理由

Drawbacks 缺点

- Young language - 这是一门年轻的计算机语言
- Shorter history of Q/A
- 历史不长，所以很多问题缺乏现成答案
- Fewer 3rd party contributions - 第三方贡献比较少
  - Compiling with SPM on newest Swift toolchain
  - 最新的工具链采用Swift 软件包管理器编译

Why ~~Swift~~ on the server?
服务器端不采用Swift的理由

Drawbacks 缺点

- Evolving - 该语言仍然在演进
- Compared to more dynamic languages 与脚本类语言相比
  - More complicated for prototyping - 做原型时相对复杂
  - More complicated deployment - 部署时也相对复杂

What does it look like?
典型应用场景

Many clients 各类形形色色的客户端

- Browsers 浏览器
- Mobile, Apps 移动应用
- Devices (IoT) 物联网设备

One or more servers 一个或多个服务器

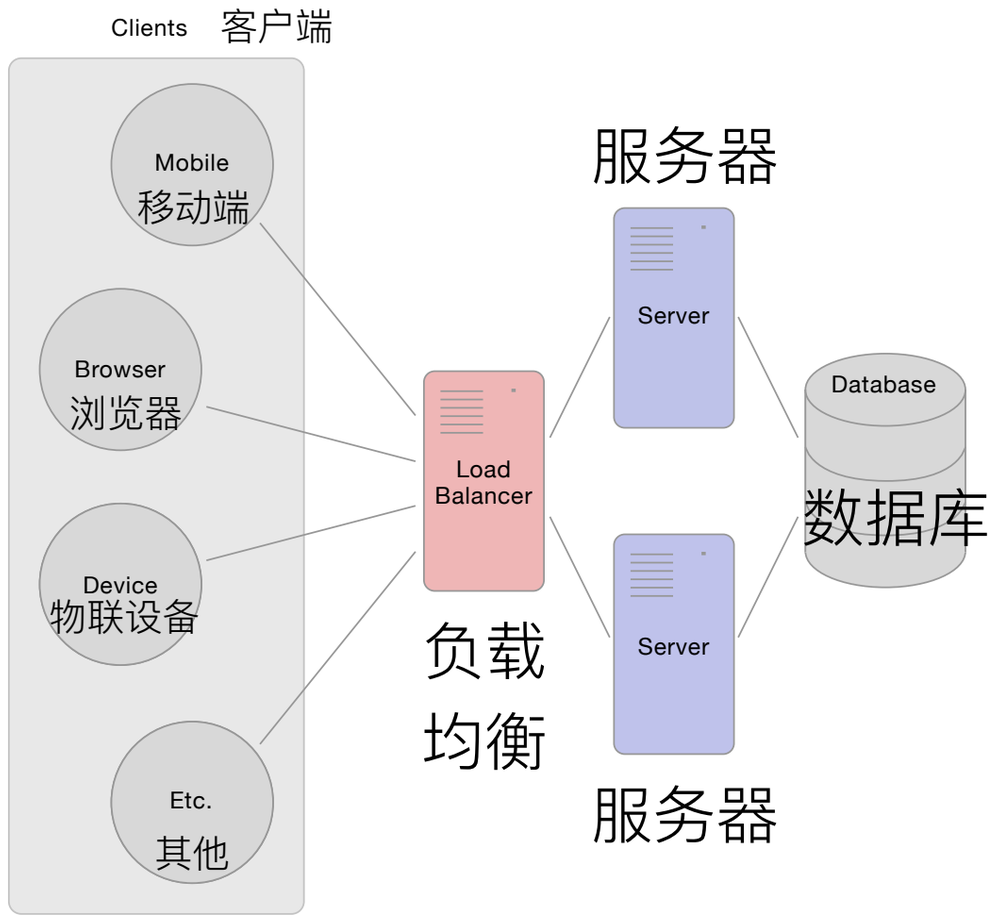
- Running Swift processes
- 运行Swift应用程序

Load balancer 负载均衡器

- Redundancy 实现冗余配置负载均衡

Database servers 数据库服务器

- Postgres, MySQL, MongoDB, caches



What is the workflow?
开发流程

SPM - Swift Package Manager 软件包管理器

- Part of Swift toolchain 工具链的组成部分
- Not used in usual iOS development
- ⚠️注意⚠️一般在iOS开发中时不用用到SPM
- Command line tool
- 命令行工具

What is the workflow?

SPM - Swift Package Manager

- Simple package definition
- 首先需要定义依存关系文件
  - Package.swift

```
import PackageDescription
let package = Package(
      name: "ExampleProject",
      dependencies: [
            .Package(url: "https://github.com/PerfectlySoft/Perfect-HTTPServer.git",
                majorVersion: 2)
      ])
```

What is the workflow?

SPM - Swift Package Manager

- Dependencies through git 采用git进行依存关系管理
- Versions through git tags 组件版本通过git标签管理
  - 1.0.0, 2.1.3
  - Version ranges - 或者定义有效版本的范围

What is the workflow?

SPM - Swift Package Manager

- Sources/, Tests/ - 标准目录结构
- 典型命令行：
- `swift package update` - 更新依存关系
  - clean, fetch, generate-xcodeproj
  - - 清除/读取依存关系，或者创建Xcode项目
- `swift build` - 编译当前文件夹下的项目

What is the workflow?

macOS with Xcode

- SPM generates Xcode project
  - `swift package generate-xcodeproj`
- Coding & initial testing on mac
- 在创建Xcode工程之后可以直接编码和测试
- Client & server development in one IDE
  - Xcode workspace very helpful
  - 集成开发环境的“工作空间”有助于服务器和客户机实现整体调试

What is the workflow?

- Linux (Ubuntu)
- Running on local or cloud VM
- 建议在本地或云端虚拟机，或者用Docker开发
  - Docker
- Choose correct toolchain version & install
- 下载安装版本适当的工具链
  - https://swift.org/download

What is the workflow?

Linux (Ubuntu)

- Build, run, test 编译、运行和测试
- Docker very helpful
- Can code on Linux (Atom)
- 也可以直接在Linux上编辑源程序，比如用Atom
- Eventually push to real server
- 编译完成后最终推送到生产服务器

What is the workflow?

Deployment options 部署目标选项

- Tencent, Alibaba, Baidu, AWS
- 腾讯云、阿里云、百度云或者亚马逊弹性计算云
- Docker
  - Anything that can run containers
  - 只要是能够运行容器的云统统可以!
- Google Cloud, Heroku

What is the role of a Server Side Swift framework?
SSS框架体系有何作用?

Simplify a huge amount of complexity
Provide what you get with iOS/macOS

简化了大量复杂工作,
实现iOS/macOS交叉平台编译

What is the role of a Server Side Swift framework?

Request->App->Response

- Simple but complicated
- 服务器接收请求——执行调度——返回响应
- 听起来很简单，但是过程很繁琐

Server configuration 比如服务器配置

- Ports, SSL 端口和证书加密

TCP, Threading 传输套接字、线程

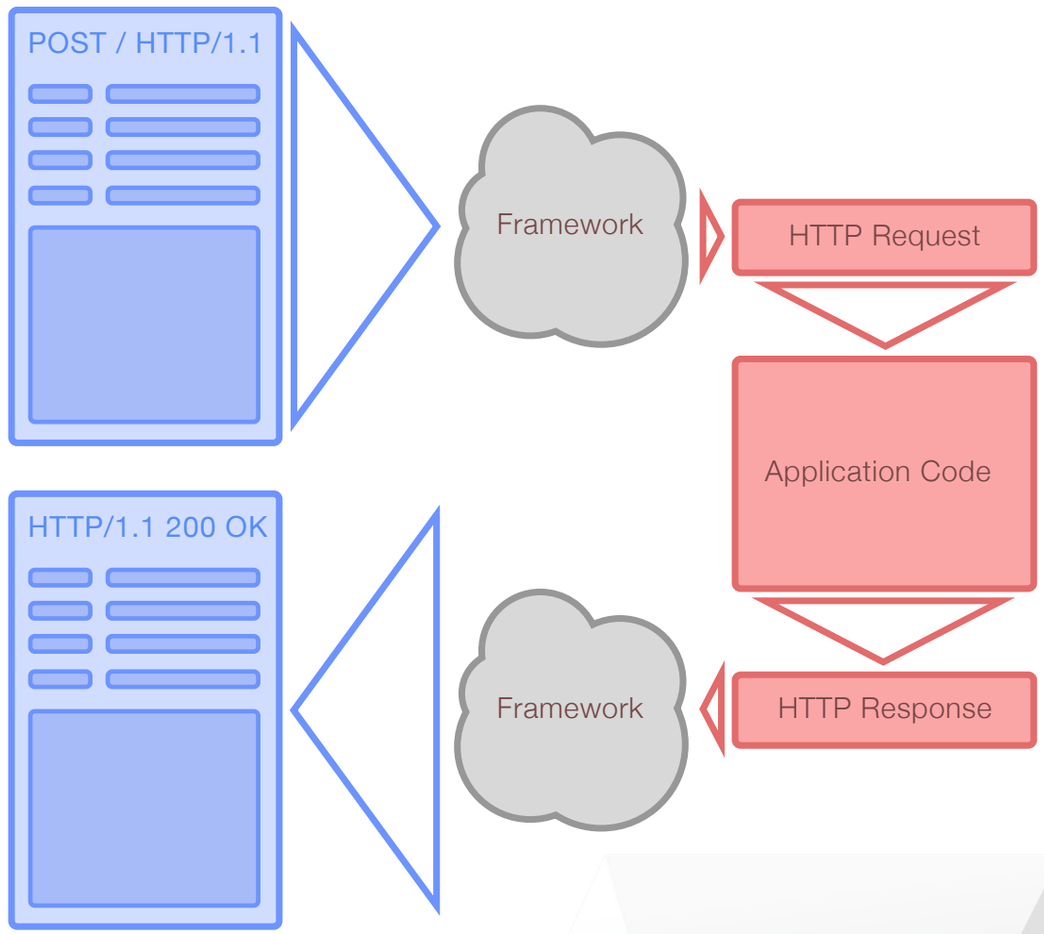
What is the role of a SSS framework?

Request parsing 请求解析

- Routing
- 编程接口路由分配
- Handlers
- 路由处理器（句柄）

Response formulation

- JSON, HTML
- XML, text, files
- Templating
  - Mustache, Markdown



Perfect

Light weight framework 轻量函数库体系

Focus on performance with async core

关注于异步内核性能

- More scalable —— 更好的可扩展性
- Uses fewer resources —— 使用更少的资源

Open source on Github —— 源代码完全开发

- Apache 2.0 license 许可证

Perfect

Core 核心函数库（线程、网络、加密、协议和服务器）

- PerfectLib, PerfectThread, PerfectNet, PerfectCrypto
- HTTP, HTTPServer

Companion packages 扩展函数库

- MySQL, PostgreSQL, SQLite 关系数据库
- MongoDB, Redis, Hadoop 非关系数据库及大数据
- StORM 数据库对象关系管理自动化
- Utility 工具类
  - Logging, CURL, authentication, WebSockets, Notifications (APNS)
  - 日志、网络访问、认证、网页套接字和消息

Examples, templates 代码示范和程序模板

Routing and Handlers
路由和句柄

A Route is a combination of three things:
每个路由都是以下三个组成部分的组合：

- HTTP request method 响应方法
  - GET, POST, etc. 获取或者提交
- URI 资源路径
  - /foo/bar/baz.html
- Handler 处理器句柄程序
  - Your code 您的源代码

Routing and Handlers
路由和句柄

Server matches the request to the handler
服务器将收到的网络请求匹配到处理句柄，然后调用句柄
Handler is called
- Given the parsed request 接收解析后的请求
- Sets response data 返回响应数据

Response data sent to client 服务器将响应数据最终发回客户

Routing and Handlers

main.swift
http://localhost:8080/hello

```
import PerfectHTTP
import PerfectHTTPServer

let myRoute = Route(uri: "/hello") {
    request, response in
    response.appendBody(string: "你好，世界!")
        .completed()
}

try HTTPServer.launch(name: "my.server.com",
                      port: 8080,
                      routes: [myRoute])
```

Routing and Handlers

http://myserver/v1/adduser?username=foo&password=bar

```
var routes = Routes(baseUri: "/v1")
routes.add(uri: "/adduser") {
    request, response in
    guard let userName = request.param(name: "username"),
        let password = request.param(name: "password") else {
            response.status = .badRequest
            return response.completed()
    }
    // TODO: Create the user
    response.completed()
}
```

Routing and Handlers

http://myserver/v1/user/kylej

```
routes.add(uri: "/user/{name}") {
	request, response in
	guard let userName = request.urlVariables["name"] else {
		response.status = .badRequest
		return response.completed()
	}
	guard let user = getTheUser(named: userName) else {
		response.status = .notFound
		return response.completed()
	}
	do {
		try response.setBody(json: user)
	} catch {
		// handling the error like a good developer
		print("\(error)")
		response.status = .internalServerError
	}
	response.completed()
}
try HTTPServer.launch(name: "my.server.cn", port: 8080, routes: routes)
```

Multiple Servers

```
// start a single server serving static files 启动一个服务器用于处理静态文件
try HTTPServer.launch(name: "localhost",
                      port: 8080,
                      documentRoot: "/path/to/webroot")
```

```
// start two servers. have one serve static files and the other handle API requests 两个服务器：静态文本和路由处理
try HTTPServer.launch(
    .server(name: "localhost", port: 8080, documentRoot:  "/path/to/webroot"),
    .server(name: "localhost", port: 8181, routes: [apiRoutes]))
```

```
// start a single server which handles API and static files 启动一个服务器但是可以同时处理静态文本和动态路由
try HTTPServer.launch(name: "localhost", port: 8080, routes: [
    Route(method: .get, uri: “/foo/bar") {
        req, resp in
        //do stuff
    },
    Route(method: .get, uri: "/foo/bar", handler:
        HTTPHandler.staticFiles(documentRoot: "/path/to/webroot"))
    ])
```

```
// start a secure server 启动一个服务器并支持证书加密
try HTTPServer.launch(.secureServer(TLSConfiguration(certPath: "/path/to/cert"),
                      name: "localhost",
                      port: 443,
                      routes: [apiRoutes]))
```

Best Practices
最佳攻略

Test on Linux often —— 使用Linux测试

No Cocoa, no UIKit ——不要用这些iOS上的东西

Be careful with Foundation —— 小心使用基础函数库

- Some aspects are not implemented on Linux
- 很多Foundation 函数库的内容并未在Linux上实现
- Name changes
- 命名规范（NSXxxx）变化很大
- No Obj-C bridge, class introspection, modification
- Linux上无法桥接Objective-C函数库、类声明和对象修改

Best Practices

Error is the default —— 服务器应用有风险，编程须谨慎

- Anything on the net is a target
- 只要上了互联网，所有东西都会被变成攻击目标
- Do not force unwrap! 不要强制拆包！
  - It will kill your server 会导致服务器崩溃

Security 安全性

- Secure SSH access 一定要采用证书加密的远程控制
- Restrict non-HTTP(S) ports 避免非证书加密端口使用
- OWASP 开放互联网社区
  - https://www.owasp.org

Best Practices

If you store data: 如果服务器上保存数据，那么一定要

- Backups
  - Backups
  - Backups

Use managed database system 数据库系统分开管理

- Most clouds have it
- Aliyun, Tencent, Baidu, AWS

Best Practices

Avoid storing dynamic data on the server

避免在服务器上存储动态数据

- File uploads, database
- 比如上载的文件、数据
- Use shared file system, S3
- 使用共享文件系统，比如亚马逊S3
- Permits horizontal scaling
- 使用水平扩展系统

Best Practices

Do not share mutable data between threads
不要在线程之间共享可修改的数据

- Structs are helpful
- 使用Struct结构
- Threading.Lock
- Threading.Event
- 使用线程锁和线程事件

Best Practices

Load testing 负载测试

- You are writing an app for many simultaneous users
- 您的程序将面对大量并发用户
- Test under load, even for small site
- 即使很小的网站也应该进行负载测试
  - Bots 自动脚本访问
  - DoS attacks 攻击
- seige, wrk, ab
- 可以使用这些工具完成负载测试

Test clients

- Paw, Postman

Other SSS Frameworks

Kitura

- Many similarities in request/response model
- 与Perfect很类似的请求/响应模型
- Tight integration with IBM tech, Bluemix, Watson
- 与其他IBM技术深度集成，如Bluemix和Watson

Vapor

- More high level
- Maybe better for beginners
- 较适用于极端初学者
- More constraints 存在更多的限制

Resources

Features & Benefits: Top Server-Side Swift Frameworks

- https://goo.gl/qdtTtA

Linux Benchmarks for Server Side Swift vs Node.js

- https://goo.gl/Qe3mGL

Resources

Swift on Linux

- https://swift.org/download

Join us on Slack #中文频道

- http://www.perfect.ly

Github

- https://github.com/PerfectlySoft

Documentation

- https://www.perfect.org/docs

Perfect Assistant

- https://www.perfect.org/en/assistant

Thank you!

Kyle Jessup

PerfectlySoft Inc.
www.perfect.org
@kylej