^2\_3^2workroom3工作室

--

//// 服务端\_PHP\_框架\_Swoft\_Swoft2开发文档(2.0.8)

https://www.swoft.org/documents/index.html

//

//// 完版日期

2020-04-22

--

--

第一卷 框架概述 SWOFT

https://www.swoft.org/documents/v2/

--

1、SWOFT简述

Swoft 是一款基于 Swoole 扩展实现的 PHP 微服务协程框架。Swoft 能像 Go 一样，内置协程网络服务器及常用的协程客户端且常驻内存，不依赖传统的 PHP-FPM。有类似 Go 语言的协程操作方式，有类似 Spring Cloud 框架灵活的注解、强大的全局依赖注入容器、完善的服务治理、灵活强大的 AOP、标准的 PSR 规范实现等等。

--

2、SWOFT特性

Swoft 通过长达三年的积累和方向的探索，把 Swoft 打造成 PHP 界的 Spring Cloud, 它是 PHP 高性能框架和微服务治理的最佳选择。

内置高性能网络服务器（Http/Websocket/RPC/TCP）；

灵活的组件功能；

强大的注解功能；

多样化的命令终端（控制台）；

强大的面向切面编程（AOP）；

容器管理，依赖注入（DI）；

灵活的事件机制；

基于PSR-7的HTTP消息的实现；

基于PSR-14的事件管理；

基于PSR-15的中间件；

国际化（i18n）支持；

简单有效的参数验证器；

高性能连接池（Mysql/Redis/RPC），自动重新连接；

数据库高度兼容Laravel的使用方式；

Redis高度兼容Laravel的使用方式；

秒级定时任务；

Session 管理；

进程池；

秒级定时任务；

Session 管理；

进程池；

服务注册与发现；

配置中心；

服务限流；

服务降级；

服务熔断；

Apollo；

Consul；

--

3、关于 Swoft

Swoft 是基于 Swoole 的企业级 PHP 应用程序开发框架，大量开发人员使用 Swoft Framework 来创建高性能、易于测试和可重用的代码。

Swoft 开源框架自 2018 年 3 月 6 日发布以来，我们秉承简单、高效、稳定的宗旨持续迭代升级。伴随着 1.x 的开发迭代和 Swoole 4 全协程化，1.x 的底层架构已经不再合适，所以在 2018 年 11 月开始规划 2.x。新版本底层借鉴 1.x 的经验全部重写，采用 Swoole Hook 全协程化，整体相比 1.x 更易上手，稳定性也更高。

Swoft 是一款轻量级的框架，所有组件均可以自定义且支持按需加载使用。

Swoft 框架可用于开发任何 Web 应用程序，构建高性能的 Web 系统、API、中间件、基础服务等。

--

4、优势

//// Swoft Framework 的优点

以组件化方式开发，开发者可以自定义组件并按需加载使用；

良好 Web MVC 设计，它为 Web 开发框架提供了一个很好的选择；

PHP 开发者可快速上手，数据库、缓存的使用均高度兼容 Laravel；

所有组件严格通过单元测试及压力测试；

--

5、社区

Swoft 官方 QQ 群 1：548173319（点击添加）

Swoft 官方 QQ 群 2：778656850（点击添加）

Swoft 社区：https://learnku.com/swoft

--

6、支持

您的支持是对开发组的最大鼓励，您可以通过以下方式来支持我们。

赞助 https://www.swoft.org/donation/

参与贡献 https://www.swoft.org/documents/v2/contribute/sub-questions/

--

--

第二卷 开发指南

--

--

一、版本更新

http://swoft.org/documents/v2/dev-guide/update/

--

1、v2.0.8[WIP]

(1)、修复(Fixed)

修复 ws server 的 message response.finish 兼容 swoole 4.4.12 问题 d724a949；

修复 console 组件中拥有短选项的选项获取值不同步问题 ac41eca；

修复 部分字符串操作不兼容 php 7.4 的问题 b4d0e90；

修复 输入未知命令时，显示http server 命令帮助信息有误 684807e；

修复 去除 process 组件中无效的调试语句 2bc84aa；

打包为phar后不能正确的加载同目录下的 .env 文件问题 8046c15；

(2)、更新(Update)

调整 路由没找到时抛出的异常将会携带错误 code 7e06afa；

(3)、增强(Enhancement)

优化 phar 包构建速度，现在打包时间将会大幅度缩短 1055fff；

优化 console 命令参数和选项的解析绑定，设置的类型的将会格式化为对应的类型 7dc0d58；

--

2、v2.0.7(2019-11-18)

升级提示：Swoole\WebSocket\Server::push 第四个参数 4.4.12 后改为了 int 类型。

(1)、修复(Fixed)

修复 config 注入时，没有找到值也会使用对应类型的默认值覆盖属性，导致属性默认值被覆盖 d84d50a7；

修复 ws server 中使用message调度时，没有过滤空数据，导致多发出一个响应。避免方法swoft-cloud/swoft#1002 d84d50a7；

修复 tcp server 中使用message调度时，没有过滤空数据，导致多发出一个响应。07a01ba1；

修复 独立使用console组件时缺少 swoft/stdlib 库依赖 c569c81a；

修复 ArrayHelper::get 传入key为 integer 时，报参数错误 a44dcad；

修复 console 渲染使用table，有int值时，计算宽度报类型错误 74a835ab；

修复 error 组件中用户无法自定义设置默认的错误处理级别 4c78aeb；

修复 启用和禁用 组件设置 isEnable() 不生效的问题 da8c51e56；

修复 在 cygwin 环境使用 uniqid() 方法必须将第二个参数设置为 true c7f688f；

修复 在 cygwin 环境不能够设置进程title而导致报错 c466f6a；

修复 使用 http response->delCookie() 无法删除浏览器的cookie数据问题 8eb9241；

修复 ws server消息调度时，接收到的ext数据不一定是数组导致报错 ff45b35；

修复 日志文件按时间拆分问题c195413；

修复 日志 JSON 格式小问题a3fc6b9；

修复 rpc 服务提供者 getList 调用两次问题fd03e71；

修复 redis cluster 不支持 auth 参数7a678f；

修复 模型查询 json 类型, 不支持 array 6023a9；

修复 redis multi 操作没有及时是否连接 e5f698；

修复 redis 不支持 expireAt, geoRadius 749241；

修复 crontab 时间戳检测偏差问题 eb08a46；

(2)、更新(Update)

更新 console 在渲染 help信息之前也会发出事件 ConsoleEvent::SHOW\_HELP\_BEFORE d3f7bc3；

简化和统一 http, ws, tcp, rpc server管理命令逻辑 f202c826；

更新 ws 和 tcp Connection类添加 newFromArray 和 toArray 方法，方便通过第三方存储(redis)时导出信息和恢复连接 a8b0b7c；

优化 server 添加统一的 swoole pipe message 事件处理，在 ws, tcp 中使用swoft事件来处理进程间消息 1c51a8c；

(3)、增强(Enhancement)

现在 tcp 请求支持添加全局或对应的方法中间件，流程和使用跟http中间件类似。仅当使用系统调度时有用 6b593877；

现在 websocket message 请求支持添加全局或对应的方法中间件，流程和使用跟http中间件类似。仅当使用系统调度时有用 9739815；

事件管理允许设置 destroyAfterFire 在每次事件调度后清理事件中携带的数据 50bf43d3；

数据库错误异常新增 code 返回fd306f4；

协程文件操作 writeFile 新增写失败异常08c4244；

RPC 新增参数验证8646fc5；

--

3、v2.0.6(2019-09-21)

(1)、升级提示

通过参数注入接收websocket原始数据时，现在需要加上类型 string。例如： public function echo(string $data)；

ws server 中使用message调度时，没有过滤空数据，导致多发出一个响应。避免方法swoft-cloud/swoft#1002；

(2)、修复(Fixed)

修复 ws server收到消息时，默认使用当前fd作为响应消息的客户端 3a8890cf；

修复 ws server通过swoole server拿到的全部连接里，fd的类型可能不是整型导致报错。0fa79d7；

修复 stdlib SystemHelper 在 windows 10 中会被 Defender 删除 71c2893；

修复 increment/decrement 多拿了一个连接 94c7171；

修复 模型属性 $modelTimestamps 关闭了无效 94c7171；

修复 在初始化前调用了log，导致可能会在启动阶段报context丢失错误 5a27704f；

修复 ws server的握手和open使用同一个顶级协程ID，如果有一个先执行完成，可能会导致后面的出现session丢失 c2cd17a；

修复 after 定时器传递参数 bug 8ed609a；

修复定时任务内存泄漏和日志打印数据丢失问题 aa6bc32；

优化 Http server sgo 处理流程 e60d92e；

修复数据库在事物情况下重连导致数据不一致问题 c2e87f2；

修复 Aop 极端情况返回 0 问题 85bc1b1；

修复 BeanFactory::createBean() 方法创建问题c2e2c89；

修复 RPC 错误数据是字符串问题 3a34658；

修复 redisCluster 初始化问题3a60c06；

修复 DbException 没有抛出 Database 的异常 Code3a60c06；

修复 ModelAttribute 的 getter 和 setter 驼峰判断错误2e2f0f1；

修复 Aggregate 聚合, 返回字符串强行转换 int错误7aa98a6；

修复模型 updateCounters 方法更新之后没有同步实体数据967d6f8；

(3)、更新(Update)

调整 ws server在worker 退出时自动关闭连接，添加开关配置，可以选择是否开启。3a8890c；

ws server 新增事件： MESSAGE\_PUSH` OPEN\_BEFORE MESSAGE\_RECEIVE` e1b0583；

(4)、增强(Enhancement)

新增日志按时间切割 f649df2；

新增日志支持传递额外参数 55d5533；

调整和新增了一些ws server相关事件，方便用户使用 e1b0583；

模型对 Prop 支持 630aad8；

新增连接连接池(mysql/redis/rpc)启动初始化连接功能 a367173；

tcp server 使用长度分包时，支持自定义头部数据打包、解包格式 a6fbe932；

RPC 调用错误信息调整 d9226b1；

新增操作 Database 异常, 打印控制台 错误 SQL3a60c06；

新增操作数据迁移 int 系列 长度支持967d6f8；

新增获取原始 sql 方法getRawSql967d6f8；

扩展updateOrInsert和updateOrCreate 支持自增参数967d6f8；

(5)、扩展(Extra)

swoftcli 支持类生成，创建新的应用、组件结构。使用请查看文档；

swoftTracker 支持方便开发者调试；

--

4、v2.0.5-beta(2019-08-06)

(1)、升级提示

请使用 context() 代替 Context::mustGet()；

这个版本开始验证器默认关闭，需要开发者开启 validator setting；

(2)、修复(Fixed)

修复提前中断请求逻辑可能导致无法正确的格式化对应的Content-Type f031398；

修复使用sgo创建子协程，在结束时没有清理与顶级协程的映射关系 de11ae5b；

修复 Xml 格式解析方法 24d0038；

修复类代理 Unique 字符串问题 444ddeb；

修复 context()->get() 存在问题以及废弃 Context::mustGet() 方法 6acc1b5；

修复接口注入，类名访问错误 2d9a31f；

修复 模型使用 paginate 方法, 结果集没有映射字段3027287；

修复 websocket server 里，当有多个worker，主动关闭非当前worker连接时报错 7666969；

(3)、更新(Update)

默认的 worker num 设置通过 swoole\_cpu\_num 函数获取 553f6500；

验证器调整成，默认非必须，用户自己开启 e5b258d；

不再抛出swoft内部的 ContainerException 到用户使用层 1e74893；

console output 新增 confirm` select` 等交互方法 0181138a；

(4)、增强(Enhancement)

console 命令选项设置 CommandOption，现在允许选项描述为多行信息 e5914983；

websocket server 的全部消息发送方法，支持传入 opcode 参数 dc164ffe9；

websocket 模块允许设置当前模块的默认 opcode，用于自动处理返回数据设置 opcode 9e3e9672；

增强 websocket 相关类，现在消息处理方法允许注入更多数据对象类型 Request` Response` 3d6c60b4；

当worker退出时，websocket server 会自动关闭所有连接 5ea10ecc；

Add Timer 定时器封装 eb92ee6；

Add Coroutine 方法封装 d8c2b88；

(5)、扩展(Extra)

swoft/crontab 秒级别定时任务；

swoft/whoops 对 filp/whoops 在swoft中使用的简单封装，用于渲染并显示更加利于阅读的错误信息；

--

5、v2.0.4-beta(2019-07-21)

(1)、升级注意

请去掉 bin/swoft 里的 Runtime::enanbleCoroutine() 设置；

请确保 swoole 的 swoole.use\_shortname 的值为 On；

从这个版本开始要求 swoole 4.4.1+；

(2)、增强(Enhancement)

Swoft\Http\Message\Request 新增 getHeaderLines() (74a2a91)；

Aop 新增 getArgsMap() 和 getClassName() 方法 (c47e785)；

新增 srun() 函数，用于协程调度 (3c4a6a4)；

优化 server 事件(onStart / onWorkStart / onWorkStop / onShutdown)，事件自带支持协程 (a8d5a8d)；

新增投递同步阻塞任务(ec938e5)；

新增 Redis call 方法, 用于使用同一连接操作(92456987)；

兼容 Swoole 4.4.x；

(3)、修复(Fixed)

修复 迁移类名太长导致记录类名不全(58314b8)；

修复 实体查询之后使用Setter更新字段值之后update更新无效(caadf0e)；

修复 stop 后删除pid文件的结果返回错误，导致restart失败 (2be450bf11)；

修复 i18n 设置默认语言不生效的问题 (b401a504e)；

修复 ws server在有多个worker时，无法主动关闭其他worker的连接(271b6398)；

修复 http server接收xml请求时，content type 不能正确匹配(2ff9a4e61)；

修复 使用 Database, json 操作无效(92456987)；

修复 limiter 限速器 Redis 加前缀无法使用问题(7b54d4c)；

(4)、更新(Update)

更新 ws server 可以通过配置 disabledModules 来禁用 ws 模块(fa31111d)；

(5)、扩展(Extra)

在官网增加案例展示，欢迎大家提交案例到官方案例仓库 swoft-cloud/swoft-case；

在GitHub上对文档的修改，将会自动更新到官网文档，不再需要手动刷新；

(6)、新增(New)

进程管理(c61b8f7)；

自定义用户进程(c328d54)；

进程池(d2e6290)；

TCP server 基本可用；

--

6、v2.0.3-beta(2019-07-08)

(1)、不兼容(Incompatible)

移除 request->json() 方法(c9e8f04) 替代方法 getParserdBody()；

应用日志和控制台日志 levels 由数组配置改成，字符串配置，详细见文档；

(2)、新增(Enhancement)

新增接口依赖注入(6169f84)；

新增 getFile 方法获取文件上传保存之后的信息(fe7e3a6)；

新增 restart() 服务新增重启方法(2ffec37)；

新增调用 1.x RPC 服务支持(30d73c3)；

新增 AOP 类名匹配支持正则表达式(bc5e479)；

新增 RPC Server /Http Server 中间件命名空间 use 错误提示(b1cec04)；

新增 验证器排除属性字段 unfields(b1bf44f)；

新增 自动写入时间戳(dc58011)；

新增 模型动作事件(dc58011)；

新增 数据库迁移(26bb464)；

新增 实体自动与 json 和数组互转(dc58011)；

新增 模型批量更新方法batchUpdateByIds(dc58011)；

(3)、修复(Fixed)

修复 cookies 设置时的一些问题，增加一些 withCookie 相关方法(b05afbb01)；

修复 在console使用协程方式运行命令时，没有捕获处理错误(8a5418bf)；

修复 websocket server 重启命令没有先停止旧server问题(db2d935)；

修复任务返回值为 null 问题(a69347c)；

修复 RPC Server 只有类中间件无法使用问题()204bc7f；

修复 RPC Server 返回值为 null 问题(4d091be)；

修复 Logger 和 CLog 日志等级无法覆盖和无效问题(8eb8aba)；

修复 模型里面的属性不支持自定义表达式(dc58011)；

(4)、更新(Update)

验证器优化，支持自定义验证规则(d959a4f)；

重命名错误处理管理类 ErrorHanlders 为 ErrorManager (f3a8f04b)；

console组件的异常处理改为由error组件提供的统一处理风格 (4f47204)；

console组件允许设置禁用命令组(c5a0269)；

在默认的错误处理中，允许设置错误捕获级别。默认级别是 E\_ALL | E\_STRICT (afff9029)；

优化 启动ws server时同时启用了http处理功能，信息面板添加提示(83a81170)；

优化 启动ws server 并同时添加rpc server启动，信息面板没有显示 rpc server信息(3d1d0d848)；

(5)、扩展(Extra)

文档添加支持通过google进行搜索；

新增 apollo 组件；

新增 consul 组件；

新增 breaker 组件；

新增 limter 组件；

新增 tcp-server 组件；

--

7、v2.0.2-beta(2019-06-21)

(1)、新增(Enhancement)

新增 $request->parsedQuery() 方法 (ab45089)；

新增 Bean 属性注入基础数据类型(string/int/bool/float/array)，自动根据注释类型转换。(e3d4085)；

新增 db(), 使 Model/Query/DB 支持切库(f3b12c9)；

新增 DbSelectorInterface 用于根据业务自动切库(b36ca03)；

新增 Http server 增加Cookies响应支持 (1a024bf)；

新增 devtool 组件的实体生成(14b4d39c)；

新增 允许注册 swoole 的 pipeMessage, packet 事件[afec0e3f]；

新增 添加更多启动时的相关事件[8aaa38c4]；

新增 validate() 不依赖注解验证方式(cc35560)；

新增 RPC 调用返回错误 RpcResponseException 异常(baa482f)；

新增 允许注册 swoole 的 pipeMessage, packet 事件(afec0e3f)；

新增 添加更多启动时的相关事件(8aaa38c4)；

(2)、修复(Fixed)

修复 paginate 无法指定查询字段(308d330)；

修复 实体 join 操作，显示不出关联表数据 (446a3a2)；

修复 实体属性名attributes等与系统定义属性冲突问题 (1a9d25b)；

修复 AOP 切面重复执行问题 (54e00ac)；

修复 Http 服务全局异常 contentType 失效问题(e704116)；

修复 Http server在启用https 时，type 检查无法通过 (3e9b431)；

修复 Http server通过data响应html时可能出现错误(e5513df)；

修复 控制器 table 数据显示 (4d27718)；

修复 devtool 生成的 phpstorm meta文件key重复(1838552)；

修复 ws-server 参数路由匹配失败(4d3392e6)；

修复 Http 服务 content-type 多值解析错误问题(1657979)；

修复 Co::multi 抛出异常调用错误问题(d0ff0f7)；

修复 body 字符串无解析调用错误问题(32abf60)；

修复 通过 setter操作实体 toArray失效问题(9c0f9505)；

现在停止server(包含Ctrl+C)时总是会删除pid文件，修复没有删除pid文件导致docker重启失败问题(6a96b2c)；

修复 通过 @Inject 注入Redis连接池没有释放以及连接池全部空闲判断问题(feac3700)；

修复 数据库开启事物多连接池操作错误(3f80474)；

修复 init() 方法中使用 config() 函数问题(fe9b92b)；

修复 getParsedQuery() 方法初始化问题(8e05836)；

修复 一些bean会出现重复初始化(例如 config)问题 (f8ef5a9a)；

修复 Autoloader 文件与其它composer包文件冲突问题 (ba19af363)；

(3)、更新(Update)

修改模型方法insert,insertGetId,updateOrInsert只会插入定义了@Column字段(9c0f9505)；

启动swoft时加入冲突扩展相关的运行环境检查(7fa7188f)；

调整 count,sum,avg,max,min 返回值类型(c9b5e17a)；

Http server 路由匹配顺序优化(d8fb588)；

完善 RPC Client 调用错误信息提示(a1e56eb)；

调整 Redis zAdd 成员和分数的位置⚠️不向下兼容(1a3275b3)；

调整 如果没有添加验证注解，会跳过验证解析相关流程(582c806cc)；

去除swoole中已经废弃的 buffer full/empty 事件相关代码 (873a97bed)；

优化 Http server 验证器流程(0e73a8c)；

(4)、扩展(Extra)

swoft/view 添加通过 @View 渲染视图支持；

--

8、v2.0.1-beta (2019-05-29)

(1)、新增(Enhancement)

新增验证器不存在错误提示(5ba8682)；

新增 AOP 代理类名称标识 \_PROXY\_(5ba8682)；

新增实体字段值类型转错误异常提示(ab3cfea)；

新增 BeanFactory::getBeans() 根据类名查询使用实例(999c10f)；

新增 Model getAttributeValue 获取字段值方法(642d791e)；

新增 Get 方式参数验证(6452a37)；

新增 Redis 通过 Inject 方式注入使用(fc0a88e)；

新增 RRC 服务全局异常处理(1653b24)；

(2)、修复(Fixed)

修复 CLog 日志调用栈错误(868102f)；

修复 Co::multi 调用失败错误数据格式(5ba8682)；

修复 Function.php 重复加载(b063dd1)；

修复 Logger 日志调用栈错误(ab3cfea)；

修复 worker 进程退出未释放(close)数据库、缓存、RPC 连接池(5b1baa5)；

修复 开发者自定义 content-type 失效问题(22cf706)；

修复 request/session 级别销毁失效问题(e0d1493)；

修复 bean 名称包含 . 点号失效问题(dc37c92)；

修复连接池 maxIdleTime 参数无效问题(0e05591)；

修复 文件上传 (10bc7fa)；

修复 默认输出编码为UTF8 (755c8cf)；

优化 Redis 配置类型转换(533b592)；

打包为phar运行时路径错误(533b592)；

(3)、Swoft-cli

修复一些问题 swoft-cli；

发布新版本时自动通过travis部署phar到GitHub release (d1503b2)；

(4)、Devtool

新 bean 方法代码智能提示(bean()`/BeanFactory::getBean()/Swoft::getBean()`)(b63fe07)；

--

9、v2.0.0-beta (2019-05-21)

Swoft 2 重构归来，基于 Swoole hook 方式，底层完全重构，相比之前更佳简单、稳定、易上手。

(1)、功能特性

内置高性能网络服务器(Http/Websocket)；

灵活的组件化；

灵活的注解功能；

多样化命令终端(Console)；

强大的切面编程（AOP）；

完善的容器管理、依赖注入 (DI)；

灵活的事件机制；

基于 PSR-7 的 HTTP 消息实现；

基于 PSR-14 的事件管理器；

基于 PSR-15 的中间件；

可扩展的高性能 RPC；

国际化(i18n)支持；

简单高效的参数验证器；

高性能连接池(Mysql/Redis/RPC)，自带断线重连；

数据库高度兼容 laravel；

缓存 Redis 高度兼容 Laravel；

高效任务处理；

灵活的异常处理；

强大的日志系统；

--

--

二、贡献规范

https://www.swoft.org/documents/v2/dev-guide/contribute-standard/

Swoft 开源框架，非商业项目，由开源组织开发与维护，这意味着任何人都可以为其开发和进步贡献力量。参与 Swoft 有多种方式：代码贡献、文档贡献、官网维护、社区维护。

--

1、代码贡献

Fork 组件代码库 swoft-component ( https://github.com/swoft-cloud/swoft-component )；

发送 Pull Request 修改请求；

等待 Swoft 开发组审核和合并；

所有官方维护的组件均由swoft-component ( https://github.com/swoft-cloud/swoft-component )项目统一合并后分发到各个组件。

--

2、参与 swoft 的开发是否很难

//// 通过对比 swoft 和传统框架, 来回答这个问题

(1)、框架定位

传统框架: 适合开发「巨石应用」, 所有功能都包含在同一份代码库中, 适合初创团队快速实现原型；

swoft: 致力于实现 PHP 下的微服务, 功能都由「组件」提供, 可以按照实际需求按需加载组装；

(2)、开发难度

///1/ 传统框架

熟悉经典的 lnmp 体系, 只用考虑 PHP 语法；

通常需要完整熟悉整个框架, 才可能参与开发, 老旧的框架提 PR 可能很久才会处理；

///2/ swoft

熟悉 swoole 以及网络编程的基础知识, 就能轻松用 PHP 写出并发能力远超 fpm；

开发组提供开发指南, 组件说明, 快速响应, 开发者可以选取一个简单的组件上手, 快速完成一个 PR；

难? 我亦无他，唯手熟尔。

--

3、开发步骤

(1)、使用 docker

推荐使用 docker 解决开发环境的困扰, 开发组维护了 swoft/swoft ( https://www.swoft.org/docs/2.x/zh-CN/introduction/update.html ) 和 swoft/alphp ( https://github.com/swoft-cloud/alphp ) 两个镜像, 简单示例。

//

//// 使用 docker-compose 进行服务编排

version: '3.1'

services: # 定义服务

swoft: # 服务的名称

image: swoft/alphp:cli

volumes:

- ../:/data # 挂载本地的源码到容器的 /data 目录下

ports:

- "80:9501" # 映射本地 80 端口到容器 9501 端口

links: # 其他基础服务也可以用 docker 解决, 用不到可以先去掉

- redis

- mysql

- rabbitmq

tty: true # 开启后, 可以使用 exec 进入到容器中

//

//// 简单使用

docker-compose up -d swoft # 启动服务

docker-compose exec swoft # 进入容器中

(2)、swoft 项目说明

//// swoft 包含2个重要项目

swoft-cloud/swoft-component: ( https://github.com/swoft-cloud/swoft-component ) swoft 源码, 组件都在此项目的 src/ 目录下；

swoft-cloud/swoft: ( https://github.com/swoft-cloud/swoft ) swoft 脚手架, 加载了全部组件并提供了所有组件的使用 demo；

//

//// 换一个说法

参与 swoft，其实在做应用层的事，业务中要怎么写，参考 swoft 中的 demo 就行，通俗说 – 造飞机。

参与 swoft-component，其实是参与其中具体的某个/某几个组件，通俗说 – 造飞机引擎。

//

//// swoft 的架构设计, 体现在 swoft 的组件化实现中

框架组件 framework: 框架核心工作, 以及框架核心配套的组件；

swoft 功能组件: 按照功能进行划分, 依赖框架组件, 可以按需加载, 常见的有rpc 全家桶( rpc rpc-server rpc-client )；cli 引用( console )；

如果只是想 造飞机, 只用给 swoft 项目提 PR 即可; 如果想 造飞机引擎, 需要给 swoft-component 提 PR, 本地还需要一份类似 swoft 项目的脚手架, 用来验证

(3)、举个例子: 参与 RPC 组件开发

如果对 RPC 的基本概念不熟悉, 参考 RPC 组件相关文档。新建一个空文件夹, 比如 swoft-test, 充当 swoft 项目的角色, 作为脚手架, 加载需要的组件。

//

//// 需要的基础文件

➜ swoft-test tree -L 2

.

├── app

│ ├── Application.php

│ └── bean.php

├── bin

│ ├── bootstrap.php

│ └── test.php

├── composer.json

├── config

是的, 只需要这么多, 从 swoft 复制过来即可。

//

//// 使用 composer 加载 swoft 组件

{

"require": {

"swoft/rpc": "dev-master",

"swoft/rpc-server": "dev-master",

"swoft/rpc-client": "dev-master"

},

"autoload": {

"psr-4": {

"App\\": "app/"

},

},

"minimum-stability": "dev",

"repositories": {

"hyperf": {

"type": "path",

"url": "../swoft-component/src/\*"

},

"packagist": {

"type": "composer",

"url": "https://mirrors.aliyun.com/composer/"

}

}

}

}

这里涉及 composer 从本地加载依赖文件的用法, 不熟悉可以参考 官方文档 composer-repository ( https://getcomposer.org/doc/04-schema.md#repositories )。

//

//// 好了, 更新

composer u --no-dev

之后就可以执行 php bin/swoft 来进行相应测试了, 如果遇到报错, 根据报错添加缺少的组件即可。

//

//// 修改 RPC 相关文件

修改 swoft-component 下的组件, 建议多和开发组交流, 有开发范式和 CI 来保证组件的开发质量和稳定性。

--

4、需要了解的更多知识

docker 基础知识。

github 如何提 PR. 推荐一个好用的工具, github desktop, 有快捷键快速提 PR。

composer 文档, 组件开发和生产项目有所差异, composer 提供了一些有意思的功能, 方便开发组件。

推荐使用 mac + phpstorm + docker, 构建全套开发环境. 有相关问题, 欢迎和开发组交流。

--

--

三、优质教程

https://www.swoft.org/documents/v2/dev-guide/course/

本章收集 Swoft 优质教程，方便开发者快速学习上手。如果您有好的优质教程，可以联系我们 (Swoft@qq.com)。

--

1、1.x 系列

(1)、视频

//// Sunny系列

// [收费]\_Swoft高性能协程框架

http://www.yzmedu.com/course/263

//

//// 老司机系列

// Swoft速学基础篇

http://www.jtthink.com/course/65#1325

// Swoft基础进阶篇

http://www.jtthink.com/course/73

--

2、2.x 系列

(1)、文章

//// Swoft2.x 学习社区

https://learnku.com/swoft

//

//// [收费]\_Swoft2.x从入门到微服务

https://www.kancloud.cn/sphynx/swoft

//

//// Swoft2.x 小白系列教程

https://8code.net/index/index/blog/type/back

(2)、视频

//// Sunny 系列

// [收费]\_Swoft2.x入门到微服务

https://study.163.com/course/introduction.htm?courseId=1006515006

//

//// [收费]\_老司机系列

// Swoft2.x 基础篇

http://www.jtthink.com/course/65

//

// Swoft2.x HTTP API 篇

http://www.jtthink.com/course/87

//

// Swoft2.x 微服务基础篇

http://www.jtthink.com/course/93

//

// swoft2.x 实战第一季

http://www.jtthink.com/course/98

--

--

四、开发注意事项

https://www.swoft.org/documents/v2/dev-guide/dev-note/

--

1、注意事项

(1)、禁止使用$\_GET、$\_POST、$GLOBALS、$\_SERVER、$\_FILES、$\_COOKIE、$\_SESSION、$\_REQUEST、$\_ENV等超全局变量。

(2)、谨慎使用global、static关键字。

(3)、不要在代码非协程环境中执行sleep以及其他睡眠函数，这样会导致整个进程阻塞，exit/die 是危险的，会导致 worker 进程退出。

(4)、不要在业务代码中使用 swoole 不支持的 hook, 例如 MongoDB`,pgsql client`, 如果需要使用这些扩展需要单独开用户进程执行。

(5)、无法 hook 的 io 都会同步阻塞进程, 导致协程无法切换, 直接的影响就是服务器大规模超时。

(6)、不建议使用 curl 扩展 类似的GuzzleHttp, 推荐使用 swoft 封装的网络请求包。

(7)、Swoole 短名开启 ,在 php.ini 中配置swoole.use\_shortname = 'on'。

--

2、常见问题

(1)、The HTTP server have been running!(PID: xx)

出现这种问题，是因为服务已经启动了。通常两种方式解决，第一种方式，stop 服务。第二种方式，kill pid。

(2)、Could not scan for classes inside xxx which does not appear to be a file nor a folder

出现这种问题是composer源的问题，一般会在创建项目`(composer create swoft/swoft swoft)的时候出现，解决办法很简单只需要切换源全局设置即可。如：composer config -g repo.packagist composer https://mirrors.aliyun.com/composer/`。

(3)、代码格式

推荐参照 PSR2 ( https://www.php-fig.org/psr/psr-2/ ) 代码规范。

--

--

五、常见内存溢出

https://www.swoft.org/documents/v2/dev-guide/memory-overflow/

--

1、内存溢出

内存泄漏指由于疏忽或错误造成程序未能释放已经不再使用的内存。内存泄漏并非指内存在物理上的消失，而是应用程序分配某段内存后，由于设计错误，导致在释放该段内存之前就失去了对该段内存的控制，从而造成了内存的浪费。

内存泄漏通常情况下只能由获得程序源代码的程序员才能分析出来, 也是一个比较难以排查的问题，所以需要在开发前知道一些规范。内存溢出一直向一个 属性/变量 写数据, 写入超过内存最大限额, php 会抛出致命错误可能导致进程退出。要避免内存溢出，首先要知道常驻内存开发，局部变量处理完毕之后会被GC，静态变量/静态属性的值GC不会标记回收，需要用户自己管理。不建议使用 静态变量/静态属性 来共享内存, 因为用户很难管理自己的内存空间。

不同的进程中PHP变量不是共享，即使是全局变量，在A进程内修改了它的值，在B进程内是无效的，如果需要在不同的Worker进程内共享数据，可以用Redis、MySQL等工具实现Worker进程内共享数据。

(1)、为什么Config Bean不支持动态写入？

下面是 Config Bean 部分源码，你可能好奇，写入为啥会抛异常呢？下面我们就来分析这个吧！

//

<?php declare(strict\_types=1);

namespace Swoft\Config;

use InvalidArgumentException;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Config\Contract\ParserInterface;

use Swoft\Config\Exception\ConfigException;

use Swoft\Config\Parser\PhpParser;

use Swoft\Config\Parser\YamlParser;

use Swoft\Stdlib\Collection;

use Swoft\Stdlib\Helper\ArrayHelper;

/\*\*

\* Class Config

\*

\* @Bean("config")

\*

\* @since 2.0

\*/

class Config

{

/\*\*

\* The items contained in the collection.

\*

\* @var array

\*/

protected $items = [];

// ......

/\*\*

\* Get value

\*

\* @param mixed $key

\*

\* @return mixed

\*/

public function offsetGet($key)

{

return ArrayHelper::get($this->items, $key);

}

/\*\*

\* @param mixed $key

\* @param mixed $value

\*

\* @throws ConfigException

\*/

public function offsetSet($key, $value): void

{

throw new ConfigException('Config is not supported offsetSet!');

}

/\*\*

\* @param string $key

\*

\* @throws ConfigException

\*/

public function offsetUnset($key): void

{

throw new ConfigException('Config is not supported offsetUnset!');

}

/\*\*

\* @param array|string $keys

\*

\* @return Collection

\* @throws ConfigException

\*/

public function forget($keys): Collection

{

throw new ConfigException('Config is not supported forget!');

}

// ......

}

?>

//

在 Config 中有一个 item 属性，虽然他不是静态的。但是我们需要明白 单例的 Bean 是全局的生命周期 在创建启动时已经创建创建好了。如果 Config 支持了写入，可能因为某段代码修改了配置导致脏读，请求写入的配置是不会被垃圾回收。

//

//// 所以我们需要明白 Swoft Bean 的生命周期。下面是一张简单的 生命周期 Bean。

php bin/swoft

singleton bean

------------------------------request

|

request bean |

|

------------------------------response

singleton bean

------------------------------

singleton bean 是全局存在的, 所以 singleton 类型的 bean 值是不允许动态修改的。需要借助属性来存储 结果值 请使用 request/prototype 类型的 bean，详情参考 Bean 章节。

(2)、非常驻内存框架常用代码

<?php declare(strict\_types=1);

namespace App\Model\Logic;

use App\Model\Data\UserData;

use Swoft\Bean\Annotation\Mapping\Inject;

class UserLogic

{

/\*\*

\* @Inject()

\*

\* @var UserData

\*/

private $userData;

/\*\*

\* @var array

\*/

private static $userCache = [];

/\*\*

\* @var string

\*/

public static $lastError = '';

/\*\*

\* Get user list

\*

\* @param array $userIds

\*

\* @return array

\*/

public function getUsers(array $userIds): array

{

$users = [];

foreach ($userIds as $k => $userId) {

if (isset(self::$userCache[$userId])) {

$users[] = self::$userCache[$userId];

unset($userIds[$k]);

}

}

if ( empty( $userIds ) ) {

return $users;

}

$list = $this->userData->getList($userIds);

if (empty($list)) {

self::$lastError = 'get user fail';

return [];

}

foreach ($list as $user) {

$userId = $user['uid'];

self::$userCache[$userId] = $user;

$users[] = self::$userCache[$userId];

}

return $users;

}

}

?>

//

//// 上面代码有以下两个问题

使用 $userCache 静态属性来存储查询结果集，在同一进程下的请求数据是共享的，多个请求都写入当超过内存限额就会导致进程退出，也就是内存泄露。

使用 $lastError 静态属性来，存储上一次错误信息，在同一进程下的请求数据是共享的可能会导致数据错乱，这也就是swoft 没有类似 get\_last\_sql() 这样的函数。

//

说明：由于基于 php-fpm 的多进程模式开发,每个请求的数据是独立的, 使用静态变量共享本次请求数据也是常用的方法。但是在常驻内存下开发，请不要使用 静态属性/全局变量/静态变量 来共享数据，因为swoft 是基于 协程模式，一个进程下面的协程的数据都是共享。并发修改静态属性的情况就会导致数据错乱，使用 static，global 关键字定义变量也是同理，请谨慎使用。

--

--

第三卷、快速开始

--

一、框架安装

https://www.swoft.org/documents/v2/quick-start/install/

--

1、系统要求

Swoft 框架支持 Linux、macOS 以及 Windows 10 (通过 Windows 10 安装 WSL(Windows Subsystem for Linux，windows下的Linux子系统) 后可以使用大部分 Linux 功能，推荐安装 Ubuntu LTS 作为子系统)。

--

2、环境要求

(1)、必要部分

PHP，版本 >=7.1

PHP 包管理器 Composer

PCRE 库

PHP 扩展 Swoole，版本 >=4.3

额外扩展：PDO、Redis

(2)、冲突部分

下方列出部分已知与 Swoole 存在冲突的 PHP 扩展，请在使用 Swoft 时移除安装或禁用：Xdebug、Xhprof、Blackfire、Zend、trace、Uopz。

--

3、安装方式

(1)、通过 Docker 安装

docker run -p 18306:18306 --name swoft swoft/swoft

(2)、通过 Docker Compose 安装

git clone https://github.com/swoft-cloud/swoft

cd swoft

docker-compose up

(3)、通过 Composer 安装

composer create-project swoft/swoft Swoft

提示：建议全局配置 Composer 国内镜像以加速下载、更新包，参考：阿里云 Composer 全量镜像( https://developer.aliyun.com/composer )。

(4)、手动安装

git clone https://github.com/swoft-cloud/swoft

cd swoft

composer install

cp .env.example .env

# 编辑 .env 文件，根据需要调整相关环境配置

(5)、通过 Swoft CLI 创建

关于 Swoft CLI 工具，请查阅 Swoft CLI 文档( https://www.swoft.org/documents/v2/dev-tools/swoft-cli )，该工具支持从不同模板项目中快速创建一个干净的 Swoft 应用。

php swoftcli.phar create:app --type full Swoft-Full

php swoftcli.phar create:app --type ws Swoft-WebSocket

php swoftcli.phar create:app --type tcp Swoft-TCP

--

4、IDE 插件

通过文本编辑器进行 Swoft 开发时，Swoft 的注解功能虽然便捷，但仍需要 use 注解相对应的命名空间，这显然不是一个高效的做法。我们推荐在使用集成开发环境 IDEA 或 PhpStorm 时，通过插件市场搜索并安装 PHP Annotations 插件以提供注解命名空间自动补全、注解属性代码提醒、注解类跳转等非常有助于提升开发效率的功能。

--

--

二、目录结构

https://www.swoft.org/documents/v2/quick-start/directory-struct/

--

1、简介

//// 一个完整的 Swoft 应用包含

Console 应用；

HTTP 服务（类似传统框架）；

WebSocket 服务；

RPC 服务；

TCP 服务；

说明：app 下的类目录为了避免部分文件夹名称没有复数单词而导致命名不统一，所以所有文件夹名称 统一使用单数。

--

2、完整结构

├── app/ ----- 应用代码目录

│ ├── Annotation/ ----- 定义注解相关

│ ├── Aspect/ ----- AOP 切面

│ ├── Common/ ----- 一些具有独立功能的 class bean

│ ├── Console/ ----- 命令行代码目录

│ ├── Exception/ ----- 定义异常类目录

│ │ └── Handler/ ----- 定义异常处理类目录

│ ├── Http/ ----- HTTP 服务代码目录

│ │ ├── Controller/

│ │ └── Middleware/

│ ├── Helper/ ----- 助手函数

│ ├── Listener/ ----- 事件监听器目录

│ ├── Model/ ----- 模型、逻辑等代码目录(这些层并不限定，根据需要使用)

│ │ ├── Dao/

│ │ ├── Data/

│ │ ├── Logic/

│ │ └── Entity/

│ ├── Rpc/ ----- RPC 服务代码目录

│ │ └── Service/

│ │ └── Middleware/

│ ├── WebSocket/ ----- WebSocket 服务代码目录

│ │ ├── Chat/

│ │ ├── Middleware/

│ │ └── ChatModule.php

│ ├── Tcp/ ----- TCP 服务代码目录

│ │ └── Controller/ ----- TCP 服务处理控制器目录

│ ├── Application.php ----- 应用类文件继承自swoft核心

│ ├── AutoLoader.php ----- 项目扫描等信息(应用本身也算是一个组件)

│ └── bean.php

├── bin/

│ ├── bootstrap.php

│ └── swoft ----- Swoft 入口文件

├── config/ ----- 应用配置目录

│ ├── base.php ----- 基础配置

│ └── db.php ----- 数据库配置

├── public/ ----- 公共目录

├── resource/ ----- 应用资源目录

│ ├── language/ ----- 语言资源目录

│ └── view/ ----- 视图资源目录

├── runtime/ ----- 临时文件目录（日志、上传文件、文件缓存等）

├── test/ ----- 单元测试目录

│ └── bootstrap.php

├── composer.json

├── phar.build.inc

└── phpunit.xml.dist

--

--

三、运行服务

https://www.swoft.org/documents/v2/quick-start/run-server/

--

1、简述

Swoft 拥有便捷的命令行工具和相应的服务管理命令，参阅 Swoft Devtool( https://www.swoft.org/documents/v2/dev-tools/swoft-devtool ) 了解更多。

--

2、HTTP 服务

# 启动 HTTP 服务

$ php ./bin/swoft http:start

#

# 以守护进程模式启动

$ php ./bin/swoft http:start -d

#

# 重启 HTTP 服务

$ php ./bin/swoft http:restart

#

# 重新加载 HTTP 服务

$ php ./bin/swoft http:reload

#

# 停止 HTTP 服务

$ php ./bin/swoft http:stop

--

3、WebSocket 服务

# 启动 WS 服务

$ php ./bin/swoft ws:start

#

# 以守护进程模式启动

$ php ./bin/swoft ws:start -d

#

# 重启 WS 服务

$ php ./bin/swoft ws:restart

#

# 重新加载 WS 服务

$ php ./bin/swoft ws:reload

#

# 关闭 WS 服务

$ php ./bin/swoft ws:stop

--

4、RPC 服务

# 启动 RPC 服务

$ php ./bin/swoft rpc:start

#

# 以守护进程模式启动

$ php ./bin/swoft rpc:start -d

#

# 重启 RPC 服务

$ php ./bin/swoft rpc:restart

#

# 重新加载 RPC 服务

$ php ./bin/swoft rpc:reload

#

# 关闭 RPC 服务

$ php ./bin/swoft rpc:stop

--

5、启动信息

默认情况下，在任何服务启动时控制台会显示相应的启动信息。如果在 .env 文件中开启了调试 SWOFT\_DEBUG=1 将会在控制台中显示更多详细的信息。

//

//// 如果需要关闭这些信息，只需要编辑 app/Application.php 文件

public function getCLoggerConfig(): array

{

$config = parent::getCLoggerConfig();

// False: 关闭控制台日志打印

$config['enable'] = true;

// 日志等级

$config['levels'] = 'error,warning';

return $config;

}

编辑后重启 Swoft 相关服务生效。

--

--

第四卷 开发工具

--

一、PhpStorm

https://www.swoft.org/documents/v2/dev-tools/phpstorm/

--

1、Annotations 插件

通过文本编辑器的环境进行 Swoft 开发，在使用 Swoft 便捷的注解时，仍需要 use 注解相对应的命名空间， 这显然不是一个高效的做法。我们推荐在 PHPStorm 环境下，并安装 Jetbrain 自带的 PHP Annotations 插件，可提供注解命名空间自动补全，注解属性代码提醒，注解类跳转等非常有助于提升开发效率的功能。

--

--

二、Swoft CLI

https://www.swoft.org/documents/v2/dev-tools/swoft-cli/

--

1、简介

GitHub: https://github.com/swoft-cloud/swoft-cli

Swoft CLI 是一个独立的命令行应用，提供了一些内置的功能方便开发者使用：生成 Swoft 应用类文件，例如 HTTP 控制器，WebSocket 模块类等；监视用户 Swoft 项目的文件更改并自动重新启动服务器；快速创建新应用或组件；将一个 Swoft 应用打包成 Phar 包。

欢迎提供意见、贡献代码，Swoft CLI 是基于 Swoft 2.0 框架构建的应用，运行时同样需要安装 Swoole。

--

2、运行预览

$ php swoftcli.phar -h

🛠️ Command line tool application for quick use swoft (Version: 0.1.3)

Usage:

swoftcli.phar COMMAND [arg0 arg1 arg2 ...] [--opt -v -h ...]

Options:

--debug Setting the application runtime debug level(0 - 4)

--no-color Disable color/ANSI for message output

-h, --help Display this help message

-V, --version Show application version information

--expand Expand sub-commands for all command groups

Available Commands:

client Provide some commands for quick connect tcp, ws server

gen Generate some common application template classes(alias: generate)

new Provide some commads for quick create new application or component(alias: create)

phar There are some command for help package application

self-update Update the swoft-cli to latest version from github(alias: selfupdate, update-self, updateself)

serve Provide some commands for manage and watch swoft server project

system Provide some system information commands[WIP](alias: sys)

tool Some internal tool commands, like ab, update-self

More command information, please use: swoftcli.phar COMMAND -h

--

3、安装

安装 Swoft CLI 非常简单，我们已经提供打包好的 Phar 放在 GitHub 上，只需从 Swoft CLI Releases - GitHub ( https://github.com/swoft-cloud/swoft-cli/releases ) 下载打包好的 swoftcli.phar 文件即可。当然你也可以通过 wget 命令下载。

//

//// 通过 wget 命令下载

wget https://github.com/swoft-cloud/swoft-cli/releases/download/{VERSION}/swoftcli.phar

wget https://github.com/swoft-cloud/swoft-cli/releases/download/v0.1.5/swoftcli.phar

注意：你需要替换 {VERSION} 部分为最新版本。

//

//// 检查包是否可用，打印版本信息

php swoftcli.phar -V

//

//// 显示帮助信息

php swoftcli.phar -h

--

4、全局使用

mv swoftcli.phar /usr/local/bin/swoftcli && chmod a+x /usr/local/bin/swoftcli

# 完成后检查是否可用

swoftcli -V

--

5、手动构建

//// 如果你需要通过最新的 Swoft CLI 或修改后的代码构建 Phar 包

// 克隆项目

git clone https://github.com/swoft-cloud/swoft-cli

cd swoft-cli

composer install

// 构建

php -d phar.readonly=0 ./bin/swoftcli phar:pack -o=swoftcli.phar

--

--

三、Swoft Devtool

https://www.swoft.org/documents/v2/dev-tools/swoft-devtool/

--

1、介绍

GitHub: https://github.com/swoft-cloud/swoft-devtool

Swoft 提供了一套内置的命令工具以方便开发者使用。

--

2、安装

//// 默认情况下开发工具已包含在 Swoft 框架中，如需重新安装只需执行下方命令

composer require swoft/devtool

--

3、帮助命令

命令行中所有项均可以通过加上 -h 参数显示更多可选参数提示，例如 php ./bin/swoft http:start -h。

$ php ./bin/swoft -h

Console application description (Version: 2.0.0)

Usage:

./bin/swoft COMMAND [arg0 arg1 arg2 ...] [--opt -v -h ...]

Options:

--debug Setting the application runtime debug level(0 - 4)

--no-color Disable color/ANSI for message output

-h, --help Display this help message

-V, --version Show application version information

--expand Expand sub-commands for all command groups

Available Commands:

agent This is an agent for Apllo config center

app There are some help command for application[by devtool]

dclient Provide some simple tcp, ws client for develop or testing[by devtool]

demo Class DemoCommand

dinfo There are some commands for application dev[by devtool]

entity Generate entity class by database table names[by devtool]

http Provide some commands to manage the swoft HTTP server(alias: httpsrv)

issue There are some commands for application dev[by devtool]

migrate Manage swoft project database migration commands[by devtool](alias: mig)

process Provides some command for manage process pool

rpc Provide some commands to manage swoft RPC server

tcp Provide some commands to manage swoft TCP server(alias: tcpsrv)

test Class TestCommand

ws Provide some commands to manage swoft websocket server(alias: wsserver,websocket

More command information, please use: ./bin/swoft COMMAND -h

--

4、更多功能

后续会根据用户需要增加更多功能，欢迎用户提供意见、贡献代码。

--

--

第五卷 基础组件

--

--

一、注解

https://www.swoft.org/documents/v2/basic-components/annotation/

--

1、概念介绍

注解（Annotation）又称标注，Java 最早在 2004 年的 JDK 5 中引入的一种注释机制。目前 PHP 官方版本并未提供内置元注解和注解概念，但我们通过 ReflectionClass 反射类解析 PHP 代码注释从而实现了自己的一套注解机制。

--

2、如何使用

就像我们日常开发写注释一样，只需在类、方法或成员变量上方按规则添加注释即可。

//

//// 如定义一个控制器及其路由

use Swoft\Http\Message\Request;

use Swoft\Http\Server\Annotation\Mapping\Controller;

use Swoft\Http\Server\Annotation\Mapping\RequestMapping;

/\*\*

\* Class Home

\*

\* @Controller(prefix="home")

\*/

class Home

{

/\*\*

\* 该方法路由地址为 /home/index

\*

\* @RequestMapping(route="/index", method="post")

\*

\* @param Request $request

\*/

public function index(Request $request)

{

// TODO:

}

}

//

注意需引入相关注解（Annotation）类，且必须以 /\*\* 开始并以 \*/ 结束，否则会导致无法解析！

--

3、开发规范

//// Swoft 框架注解规范如下

类注解：所有类注释后面；

属性注解：属性描述之后，其它注释之前；

方法注解：方法描述之后，其它注释之前；

请 严格 按照规范使用注解，避免产生错误。如有更佳方式，欢迎提供建议。

--

4、开发工具

参阅：快速开始 - 框架安装 - IDE 插件 ( https://www.swoft.org/documents/v2/quick-start/install/#ide-%E6%8F%92%E4%BB%B6 ) 在 IDEA 或 PhpStorm 中安装 PHP Annotation 插件以提供开发帮助。

--

--

二、容器

https://www.swoft.org/documents/v2/basic-components/container/

--

1、简介

Swoft 基于 PSR-11 规范设计了自己容器，并基于 注解 增强了它的功能。容器是 Swoft 最重要的设计，称得上是 Swoft 的核心精髓，也是 Swoft 各模块的实现基础。本章节将会对容器的相关基础知识做一个介绍，以便大家更好的理解容器。

--

2、前置知识 - IoC

IoC 即控制反转（Inversion of Control），它不是一门技术而是一种设计思想。利用 IoC 将你设计好的对象交给容器控制，而非传统地在你的对象内直接控制、处理。简单来说就是如此，理论知识总是如此，我们继续下文。

--

3、前置知识 - DI

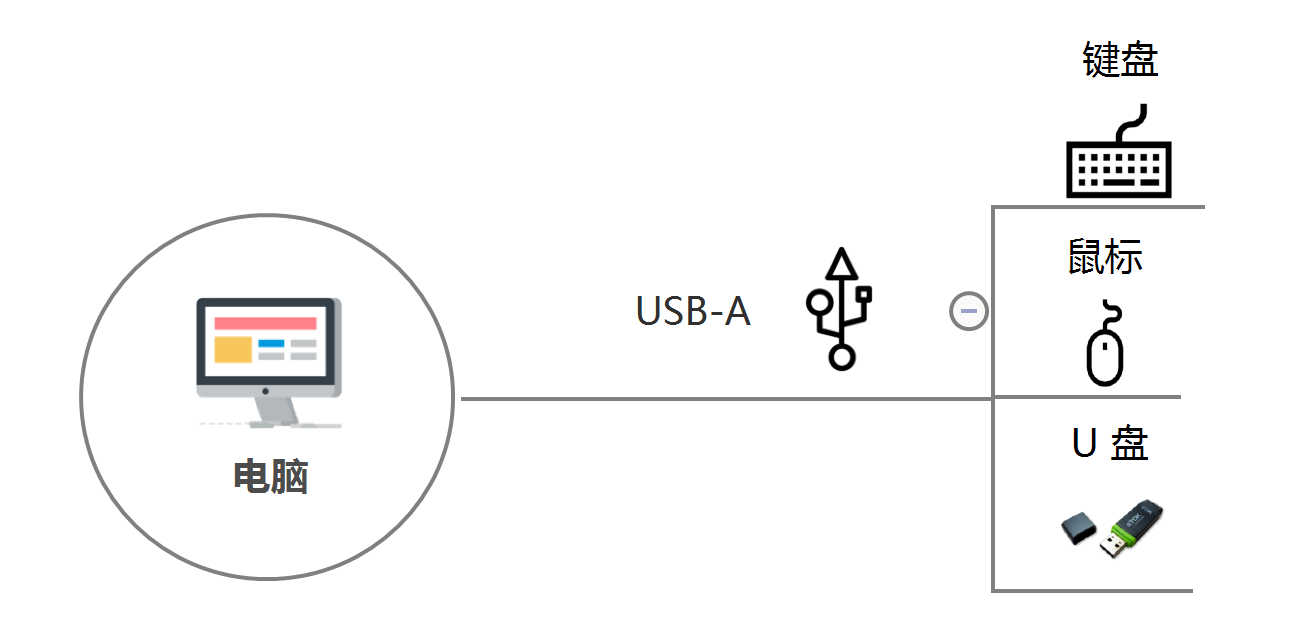
DI 为依赖注入（Dependency Injection）的缩写，DI 其实也不是一门技术，它是一种实现 IoC 的方式。同样的，我们先初步认识 IoC 和 DI 的概念，然后继续阅读下方图文深入理解。

--

4、概念知识

我们知道一台标准的电脑拥有 USB 接口，通过 USB 接口可以扩展电脑功能，如：键盘、鼠标、U 盘、蓝牙、无线网卡等等。只要是拥有标准 USB 接口的设备，连接到电脑上就能使用。现在我们有一台没有 USB 接口的电脑，我们也拥有键盘、鼠标和 U 盘（均为标准 USB 接口设备），当我们需要使用键鼠、U 盘的功能时，就必须 依赖 USB 接口来实现我们的目标。

( 图 https://www.swoft.org/img/computer.png )



电脑 --USB-A--> 键盘/鼠标/U盘

说明：这里的 USB 接口指传统 USB 接口，即 USB-A 接口。

--

5、理解 IoC 与 DI

让我们继续，我们通过一些伪代码来理解上述的 IoC 及 DI。

//

///1/ 定义电脑

class Computer

{

/\*\* @var 电脑型号 \*/

private $model;

/\*\* @var 永不断电！ \*/

private $powerOn = true;

public function \_\_construct($model)

{

$this->model = $model;

}

public function useKeyboard()

{

// 使用键盘（写代码 yes~）

}

public function useMouse()

{

// 使用鼠标

}

public function useUDisk()

{

// 使用 U 盘

}

}

//

///2/ USB接口定义

前文提到，我们现在没有 USB，所以我们得依赖 USB 接口实现键鼠、U 盘的使用。

class USBAHub

{

public function writeCode($computer\_model)

{

// 果真写代码！

echo "电脑 {$computer\_model} 已开机，我正在疯狂写代码<br>";

}

public function paintWithMouse($computer\_model)

{

// 鼠标画图

echo "电脑 {$computer\_model} 已开机，我正在构建灵感美学<br>";

}

public function storageWithUDisk($computer\_model)

{

// U 盘存储

echo "电脑 {$computer\_model} 已开机，我用 U 盘存了...（不可描述）<br>";

}

}

//

///3/ USB接口实现

自从有了这个强大的 USB-A Hub 后，我们就能愉快的使用了。

class Computer

{

private $model;

private $powerOn = true;

public function \_\_construct($model)

{

$this->model = $model;

}

public function useKeyboard()

{

// 使用键盘写代码

(new USBAHub)->writeCode($this->$model);

}

public function useMouse()

{

// 使用鼠标画图

(new USBAHub)->paintWithMouse($this->$model);

}

public function useUDisk()

{

// 使用 U 盘存储

(new USBAHub)->storageWithUDisk($this->$model);

}

}

//

///4/ 当我们使用这台携带 USB 接口的电脑一天后

$computer = new Computer('神机 i999 + GXX9080Tx');

$computer->useKeyboard();

$computer->useMouse();

$computer->useUDisk();

//

/// 输出内容

电脑 神机 i999 + GXX9080Tx 已开机，我正在疯狂写代码

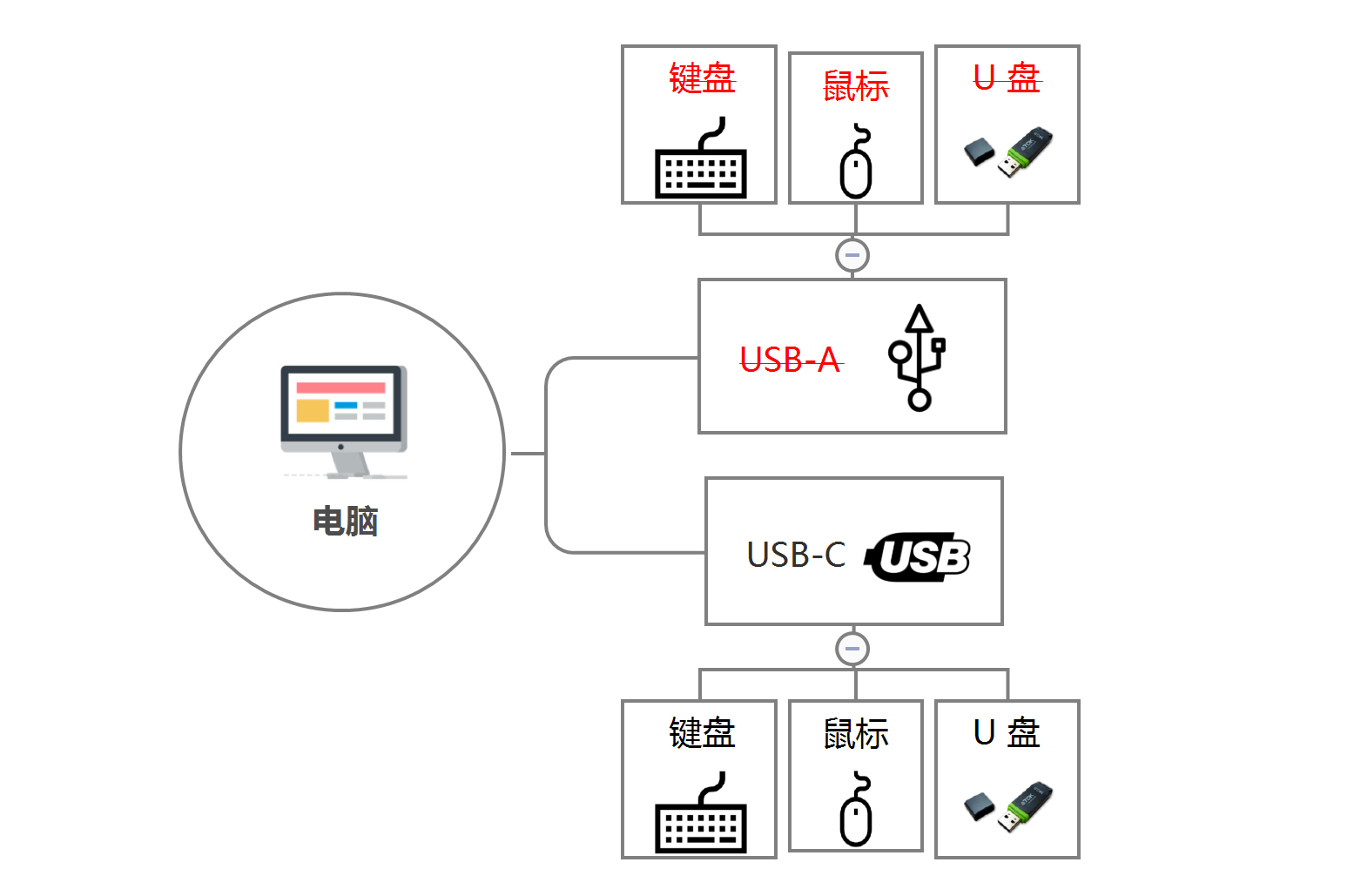
电脑 神机 i999 + GXX9080Tx 已开机，我正在构建灵感美学~

电脑 神机 i999 + GXX9080Tx 已开机，我用 U 盘存了...（不可描述）

//

可是即便是当年强大的神机也会面临岁月的折磨，我们的 USB 接口在某天变得老化，速度颇慢，跟不上时代的步伐，更不兼容最新的 Type-C 接口，于是我们准备更换到新一代的 USB-C（Type-C）。

( 图 https://www.swoft.org/img/upgrade.png )



电脑 --(旧)USB-A--> 键盘/鼠标/U盘

电脑 --(新)USB-C--> 键盘/鼠标/U盘

但是，USB 接口的更换等同于更换主板（忽略处理器、内存等接口版本差异），拆机、装机、插跳线、配置 BIOS 等工作极其麻烦，我们是否可以将这个工作交给 别人 来完成，从而我们当一个电脑的使用者就好？将这个复杂的工作、控制权交给所谓的“别人”替我们完成的思想就叫做 控制反转。

这个“别人”可以是电脑商贩，也可以是维修店或者朋友，我们现在将“别人”称之为“帮手”。我们将这项工作移交给 帮手 来完成，交给帮手完成的操作实现就是 依赖注入。有了帮手之后，我们通过 构造函数 将帮手 注入 到电脑中，就能轻松使用 USB 设备了。

//

///5/ 控制反转，依赖注入

use Components\Helper;

class Computer

{

private $model;

private $powerOn = true;

private $helper;

public function \_\_construct($model, Helper $helper)

{

$this->model = $model;

$this->helper = $helper;

}

public function useKeyboard()

{

// 使用键盘写代码

$this->helper->writeCode($this->$model);

}

public function useMouse()

{

// 使用鼠标画图

$this->helper->paintWithMouse($this->$model);

}

public function useUDisk()

{

// 使用 U 盘存储

$this->helper->storageWithUDisk($this->$model);

}

}

注入的方式有三种，分别是：基于构造函数、基于 setter 方法、基于接口。其中基于构造函数是最常见的注入方式。

//

///6/ 多版本使用

现在我们手中除了旧的 USB-A 的键鼠、U 盘外，也新购置了 USB-C 的键鼠、U 盘，所以我们在使用这台电脑时并不关心 USB 版本。

// 实例化 USB-A

$usbHub = new USBAHub();

// 如果 USB-A 老化，更新使用 USB-C

if($usbHub->isOld()) {

$usbHub = new USBCHub();

}

// 使用 USB

$computer = new Computer('超级神机 i999 + XXX9080Ti', $usbHub);

为什么不用 USB-B 呢？USB -B 目前我们见得最多的是在打印机上，而 USB-B 和 USB-A 同样是单面插口，而 USB-C（Type-C）是双面插口（不区分正反面）。所以 USB-C 是更为合理的升级方案。

--

6、接口约束

前文我们通过电脑 USB 的例子理解了 IoC 和 DI，但缺少实际开发中最重要的一环——利用接口进行约束。我们应当通过接口类约束 USB-A 以及 USB-C 需要实现的功能，前文中的 帮手 即为接口。

//

///1/ 接口定义

interface HelperInterface()

{

public function writeCode(string $model);

public function paintWithMouse(string $model);

public function storageWithUDisk(string $model);

public function isOld(): bool;

}

//

///2/ 接口实现

而无论 USB-A 还是 USB-C 均需要实现该接口，以下用 USB-C 举例。

use Components\Interfaces\HelperInterface;

class USBCHub implements HelperInterface

{

public function writeCode($computer\_model)

{

// 键盘写代码

echo "电脑 {$computer\_model} 已开机，我正在疯狂写代码<br>";

}

public function paintWithMouse($computer\_model)

{

// 鼠标画图

echo "电脑 {$computer\_model} 已开机，我正在构建灵感美学<br>";

}

public function storageWithUDisk($computer\_model)

{

// U 盘存储

echo "电脑 {$computer\_model} 已开机，我用 U 盘存了...（不可描述）<br>";

}

/\*\*

\* 是否老化

\*/

public function isOld()

{

return false;

}

}

USB-C 也将在某天老化，那时可能会有 USB-D、USB-E、USB-F 存在，届时我们仅需根据 HelperInterface 实现相应的 USB-D、USB-E、USB-F 类，并将 USB-C 中 isOld 方法返回 true 即可实现更新，从而平滑过渡到新版本 USB 接口。

--

7、IoC 容器

我们通过前文充分了解了如何利用 DI 实现 IoC，而实际项目中我们可能会面临几十上百个类去处理依赖关系，且类之间还会存在嵌套等问题，那么我们手动实现 IoC 就变得非常困难，效率也低。IoC 容器的出现极其便捷的替我们解决了这一烦恼。

(1)、IoC 容器的主要功能

自动管理依赖关系，避免手工管理存在缺陷。

需要使用依赖时自动注入所需依赖。

管理对象生命周期。

为了更好理解 IoC 容器，我们通过一个简单的示例演示：（代码仅作演示用，Swoft 已提供完善强大且易用的 IoC 容器）。

--

--

二、AOP

https://www.swoft.org/documents/v2/basic-components/aop/

--

1、概念介绍

AOP（Aspect-oriented programming）即 面向切面的程序设计，AOP 和 OOP（Object-oriented programming）面向对象程序设计一样都是一种程序设计思想，AOP 只是 OOP 的补充和延伸，可以更方便的对业务代码进行解耦，从而提高代码质量和增加代码的可重用性。后面的篇章会结合实际案例，教会大家理解并学会使用 AOP。AOP 并不规定必须使用某种方式来实现。

--

2、场景

我们就日志记录功能入手带领大家理解 AOP 切面编程的优势。首先假设我们有以下需求：需要对某个接口做调用记录，记录接口的调用详情（调用参数，返回结果等）；当调用接口出现异常时，做特殊处理，比如记录异常日志、邮件通知运维小伙伴。

如上，我们定义了一个简单的需求，接下来我们去实现它，我们先不要考虑 AOP ，就先用你所熟悉的 OOP 思想来完成以上的需求。

(1)、传统 OOP 实现

///1/ 订单服务类

<?php declare(strict\_types=1);

/\*\*

\* 订单服务

\*/

class OrderService

{

/\*\*

\* 生成订单

\*

\* @param string $product\_name 商品名字

\* @param int $quantity 购买数量

\* @param int $user\_id 用户 ID

\*

\* @return array

\*/

public function generateOrder($product\_name, $quantity, $user\_id): array

{

$price = 1000;

$amount = $price \* $quantity;

$order = [

'order\_no' => uniqid($user\_id . time() . $amount), // 订单号

'product\_name' => $product\_name, // 商品名称

'price' => $price, // 商品单价

'quantity' => $quantity, // 商品数量

'amount' => $amount // 订单总额

];

return $order;

}

}

?>

//

///2/ 添加调用日志

在传统的 OOP 思想下，我们要实现所需的功能，可以创建一个类继承 OrderService，然后重写 generateOrder 方法，最后在所有使用 OrderService 的地方替换成新建的类。示例。

<?php declare(strict\_types=1);

/\*\*

\* Class InheritOrderService

\*

\*/

class InheritOrderService extends OrderService

{

/\*\*

\* @param string $product\_name

\* @param int $quantity

\* @param int $user\_id

\*

\* @return array

\* @throws Exception

\*/

public function generateOrder($product\_name, $quantity, $user\_id): array

{

// TODO: 记录接口调用参数

try {

$result = parent::generateOrder($product\_name, $quantity, $user\_id);

// TODO: 处理父方法返回结果，根据返回结果进行相关处理

return $result;

} catch (Exception $e) {

// TODO: 如有异常，记录异常日志并发送邮件，然后继续将异常抛出

throw $e;

}

}

}

?>

//

///3/ 修改调用

最后在我们调用 OrderService 的地方，将 OrderService 替换为 InheritOrderService。

<?php declare(strict\_types=1);

// $order = new OrderService(); // 注释掉旧业务调用

$order = new InheritOrderService();// 替换

$order->generateOrder('MacBook Pro', 1, 10000);

?>

//

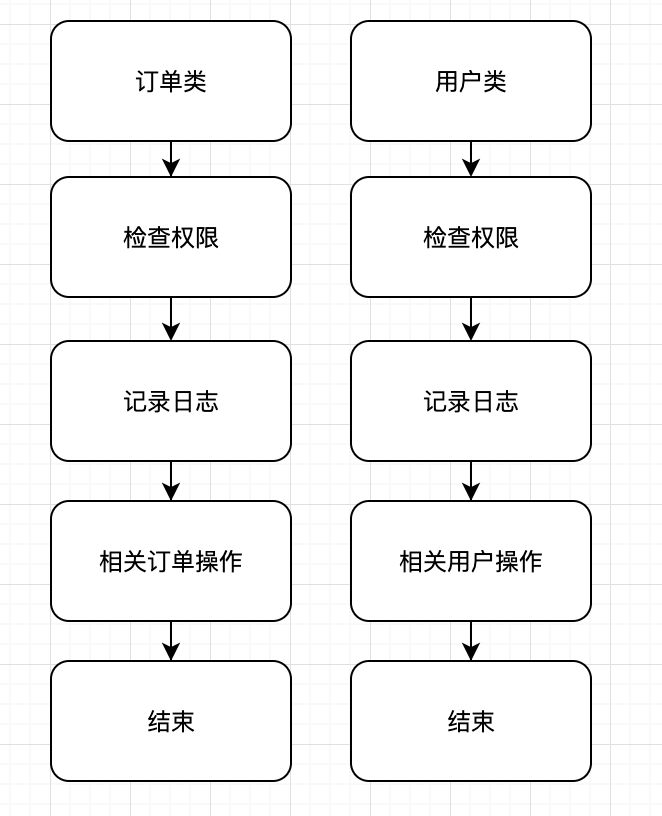
至此，经过上方的调整后，满足了我们的业务需求。现在我们回顾一下，过程似乎非常繁琐，耦合严重，甚至污染了 generateOrder 方法。如果项目中存在 100 处 OrderService 类的调用，我们就得找到这 100 个地方进行修改、替换，这就是 OOP 的思想。

这时你或许会想到使用中间件、拦截器等类似的方法来解决，其实这些方法本质上也是基于 AOP 的设计思想而来的。前文已经提到，AOP 是基于 OOP 的补充和延伸。

(2)、AOP 实现

AOP 的主要作用是在不侵入原有代码的情况下添加新的功能。我们知道 OOP 实际上就是对我们的功能属性、方法做一个抽象封装，能够清晰的划分逻辑单元。但是 OOP 只能够进行纵向的抽象封装，无法很好的解决 横向 的重复代码，而 AOP 则很好的解决了这一问题。

( 请看图示 https://www.swoft.org/img/OOP.png )

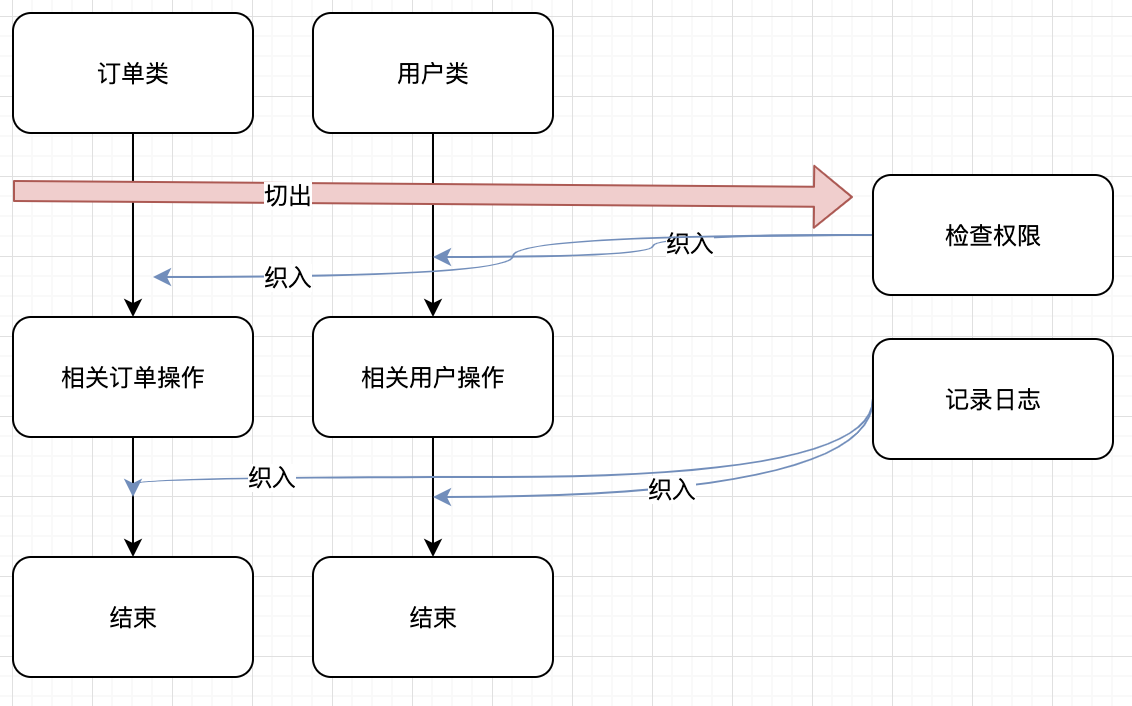


订单类 --> 检查权限 --> 记录日志 --> 相关订单操作 --> 结束

用户类 --> 检查权限 --> 记录日志 --> 相关用户操作 --> 结束

如上图所示，我们有两个类：订单类 和 用户类，我们对其相关功能做了封装。但是，权限检查、日志记录等功能就是在重复的编码，而利用 AOP 思想就可以将这些功能 横向切 出去，然后在适当的时候再将这些功能织入进来。

( 请看图示 https://www.swoft.org/img/AOP.png )



订单类 --(切出/织入)检查权限--> 相关订单操作 --(切出/织入)记录日志--> 结束

用户类 --(切出/织入)检查权限--> 相关用户操作 --(切出/织入)记录日志--> 结束

这就是 AOP。

--

3、相关术语

(1)、Advice（通知）

通知是织入到目标类连接点上的一段代码，例如上方的权限检查及记录日志。

(2)、Aspect（切面）

切面由通知和切点组成，通知明确了目的，而切点明确目的地。

(3)、Introduction（引介）

引介指向一个现有类增加方法或字段属性。引介还可以在不改变现有类代码的情况下实现新的接口。

(4)、Joinpoint（连接点）

程序执行的某个特定位置，将通知放置的地方。如方法调用前、方法调用后、返回、抛出异常等。允许使用通知的地方都可称为连接点。

(5)、Pointcut（切点）

切点指需织入目标的方法。假设一个目标对象（类）中拥有 10 个方法，需要在其中 3 个方法中织入通知，这 3 个方法称为切点。

(6)、Proxy（代理）

应用通知的对象，详细内容参见设计模式里面的代理模式。代理实现了切面的业务， Swoft 使用了 PHP-Parser 库实现 AOP。

(7)、Target（目标对象）

被通知的对象类。目标含有真正的业务逻辑，可被无感知的织入。

(8)、Weaving（织入）

将切面应用至目标对象以创建新代理对象的过程。

--

4、运用声明

(1)、声明切面

//// 通过注解 @Aspect 将类定义为切面类

<?php declare(strict\_types=1);

use Swoft\Aop\Annotation\Mapping\Aspect;

/\*\*

\* @Aspect(order=1)

\*/

class DemoAspect

{

// TODO: ...

}

?>

order：用于指定优先级，数字越小则优先执行。

(2)、声明切点

目标类必须指定为携带 namespace 的完整路径或如示例代码中在顶部 use。

//

//// 示例代码

<?php declare(strict\_types=1);

namespace App\Aspect;

use App\Services\OrderService;

use App\Services\UserService;

use Swoft\Aop\Annotation\Mapping\Aspect;

use Swoft\Aop\Annotation\Mapping\PointAnnotation;

use Swoft\Aop\Annotation\Mapping\PointBean;

use Swoft\Aop\Annotation\Mapping\PointExecution;

use Swoft\Http\Server\Annotation\Mapping\RequestMapping;

/\*\*

\* @Aspect(order=1)

\*

\* @PointBean(

\* include={OrderService::class,UserService::class},

\* exclude={}

\* )

\*

\* @PointAnnotation(

\* include={RequestMapping::class},

\* exclude={}

\* )

\*

\* @PointExecution(

\* include={OrderService::createOrder,UserService::getUserBalance},

\* exclude={OrderService::generateOrder}

\* )

\*/

class DemoAspect

{

// TODO: ...

}

?>

//

//// PointBean：定义目标类切点

include：需被 指定 为切点的目标类集合；

exclude：需被 排除 为切点的目标类集合；

//

//// PointAnnotation：定义 注解类 切点，所有使用对应注解的方法均会通过该切面类代理

inlucde：需被 织入 的注解类集合；

exclude：需被 排除 的注解类集合；

//

//// PointExecution：定义确切的目标类方法。

include：需被 织入 的目标类方法集合，支持正则表达式；

exclude：需被 排除 的目标类方法集合，支持正则表达式；

使用正则表达式时，参数内容 必须 使用双引号 " " 包裹；命名空间分隔符 必须 使用 \ 转义，同时双引号内 必须 是类的完整路径。以上注解定义的关系为并集，定义的排除为并集后的结果。建议为了便于理解和使用，一个切面类尽量只使用其中一个注解。

(3)、声明通知

<?php declare(strict\_types=1);

namespace App\Aspect;

use App\Services\OrderService;

use App\Services\UserService;

use Swoft\Aop\Annotation\Mapping\After;

use Swoft\Aop\Annotation\Mapping\AfterReturning;

use Swoft\Aop\Annotation\Mapping\AfterThrowing;

use Swoft\Aop\Annotation\Mapping\Around;

use Swoft\Aop\Annotation\Mapping\Aspect;

use Swoft\Aop\Annotation\Mapping\Before;

use Swoft\Aop\Annotation\Mapping\PointExecution;

use Swoft\Aop\Point\JoinPoint;

use Swoft\Aop\Point\ProceedingJoinPoint;

use Throwable;

/\*\*

\* @Aspect(order=1)

\*

\* @PointExecution(

\* include={OrderService::createOrder,UserService::getUserBalance},

\* exclude={OrderService::generateOrder}

\* )

\*/

class DemoAspect

{

/\*\*

\* 前置通知

\*

\* @Before()

\*/

public function beforeAdvice()

{

}

/\*\*

\* 后置通知

\*

\* @After()

\*/

public function afterAdvice()

{

}

/\*\*

\* 返回通知

\*

\* @AfterReturning()

\*

\* @param JoinPoint $joinPoint

\*

\* @return mixed

\*/

public function afterReturnAdvice(JoinPoint $joinPoint)

{

$ret = $joinPoint->getReturn();

// 返回

return $ret;

}

/\*\*

\* 异常通知

\*

\* @AfterThrowing()

\*

\* @param Throwable $throwable

\*/

public function afterThrowingAdvice(Throwable $throwable)

{

}

/\*\*

\* 环绕通知

\*

\* @Around()

\*

\* @param ProceedingJoinPoint $proceedingJoinPoint

\*

\* @return mixed

\* @throws Throwable

\*/

public function aroundAdvice(ProceedingJoinPoint $proceedingJoinPoint)

{

// 前置通知

$ret = $proceedingJoinPoint->proceed();

// 后置通知

return $ret;

}

}

?>

@Before：前置通知。在目标方法之前执行

@After：后置通知。在目标方法之后执行

@AfterReturing：返回通知

@AfterThrowing：异常通知。目标方法异常时执行

@Around：环绕通知。等同于前置通知加上后置通知，在目标方法之前及之后执行

--

5、业务示例

为了更好的理解，我们使用前置、后置通知实现一个用于计算代码执行时长的例子。

(1)、控制器

///1/ 在控制器中测试更为直观

<?php declare(strict\_types=1);

namespace App\Http\Controller;

use Swoft\Http\Server\Annotation\Mapping\Controller;

use Swoft\Http\Server\Annotation\Mapping\RequestMapping;

/\*\*

\* @Controller(prefix="test")

\*/

class TestRunTimeController

{

/\*\*

\* 闭包递归，计算阶乘

\*

\* @RequestMapping(route="factorial/{number}")

\*

\* @param int $number

\*

\* @return array

\*/

public function factorial(int $number): array

{

$factorial = function ($arg) use (&$factorial) {

if ($arg == 1) {

return $arg;

}

return $arg \* $factorial($arg - 1);

};

return [$factorial($number)];

}

/\*\*

\* 计算 1-1000 和，最后休眠 1s

\*

\* @RequestMapping(route="sum")

\*/

public function sumAndSleep(): array

{

$sum = 0;

for ($i = 1; $i <= 1000; $i++) {

$sum = $sum + $i;

}

sleep(1);

return [$sum];

}

}

?>

//

//// 启动 HTTP 服务后，我们可以通过访问下方两个地址执行代码

http://localhost:18306/test/factorial/100

http://localhost:18306/test/sum

(2)、切面类

前文已强调，通过 AOP 我们可以在不侵入原有代码的情况下实现额外的操作。请看示例。

<?php declare(strict\_types=1);

namespace App\Aspect;

use App\Http\Controller\TestRunTimeController;

use Swoft\Aop\Annotation\Mapping\After;

use Swoft\Aop\Annotation\Mapping\Aspect;

use Swoft\Aop\Annotation\Mapping\Before;

use Swoft\Aop\Annotation\Mapping\PointBean;

use Swoft\Aop\Point\JoinPoint;

/\*\*

\* @Aspect(order=1)

\*

\* @PointBean(include={TestRunTimeController::class})

\*/

class CalculateRunTimeAspect {

/\*\* @var float 执行开始 \*/

private $time\_begin;

/\*\*

\* 前置通知

\*

\* @Before()

\*/

public function beforeAdvice()

{

// 起点时间

$this->time\_begin = microtime(true);

}

/\*\*

\* 后置通知

\*

\* @After()

\*

\* @param JoinPoint $joinPoint

\*/

public function afterAdvice(JoinPoint $joinPoint)

{

/\*\* @var float 执行结束 \*/

$timeFinish = microtime(true);

$method = $joinPoint->getMethod();

$runtime = round(($timeFinish - $this->time\_begin) \* 1000, 3);

echo "{$method} 方法，本次执行时间为: {$runtime}ms \n";

}

}

?>

//

//// 编写完成后重启 HTTP 服务，然后再次访问

http://localhost:18306/test/factorial/100

http://localhost:18306/test/sum

//

//// 得到结果后返回控制台查看执行时间

factorial 方法，本次执行时间为: 0.107ms

sumAndSleep 方法，本次执行时间为: 1000.319ms

--

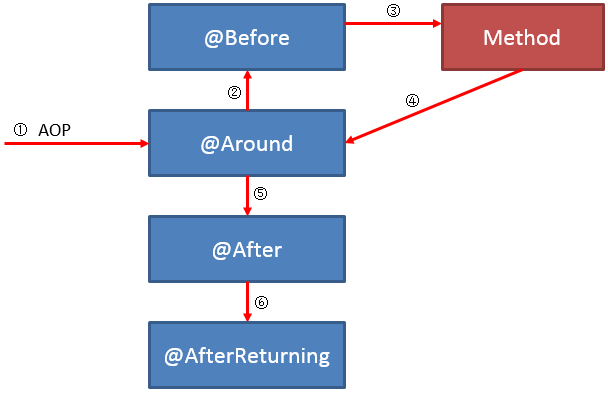
6、通知执行顺序

前文已提到，多个切面按照 order 属性值进行优先级划分，数字越小优先执行。而在一个切面中的多个通知同样也按照顺序执行。

(1)、单切面

///1/ 正常顺序

( 图 https://www.swoft.org/img/singular-aspect-normal.jpg )



AOP --1--> @Around --2--> @Before --3--> Method --4--> @Around --5--> @After --6--> @AfterReturning

<1>、@Around 环绕通知 前 置部分

<2>、@Before 前置通知

<3>、目标对象方法

<4>、@Around 环绕通知 后 置部分

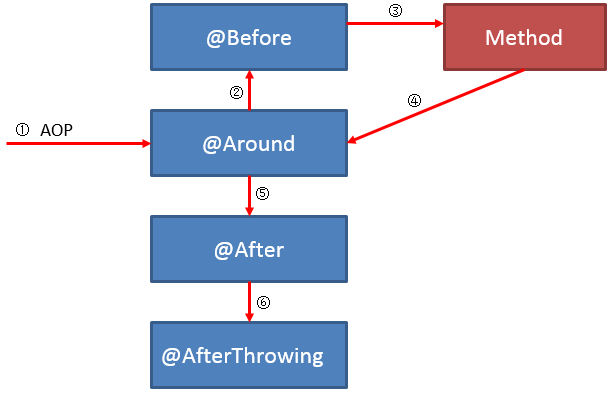
<5>、@After 后置通知

<6>、@AfterReturn 返回通知

//

///2/ 异常顺序

( 图 https://www.swoft.org/img/singular-aspect-exception.jpg )



AOP --1--> @Around --2--> @Before --3--> Method --4--> @Around --5--> @After --6--> @AfterThrowing

<1>、@Around 环绕通知 前 置部分

<2>、@Before 前置通知

<3>、目标对象方法

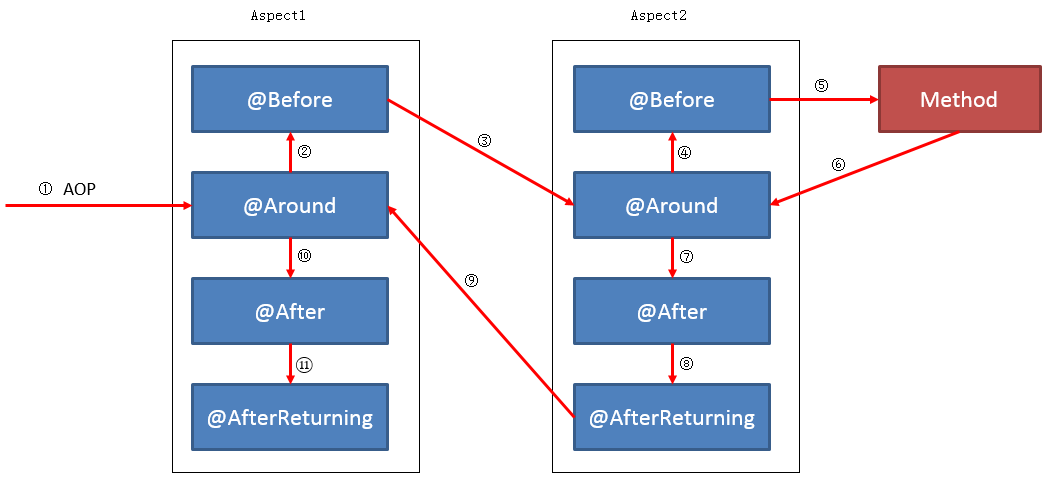
<4>、@Around 环绕通知 后 置部分

<5>、@After 后置通知

<6>、@AfterThrowing 返回通知

(2)、多切面

( 图 https://www.swoft.org/img/multiple-aspects.jpg )



//// 以正常情况为例

<1>、切面 1 @Around 环绕通知 前置部分

<2>、切面 1 @Before 前置通知

<3>、切面 2 @Around 环绕通知 前置部分

<4>、切面 2 @Before 前置通知

<5>、目标对象方法

<6>、切面 2 @Around 环绕通知 后置部分

<7>、切面 2 @After 后置通知

<8>、切面 2 @AfterReturn 返回通知

<9>、切面 1 @Around 环绕通知 后置通知

<10>、切面 1 @After 后置通知

<11 >、切面 1 @AfterReturn 返回通知

--

7、注意事项

AOP 仅拦截 public 及 protected 修饰的方法，不会拦截 private 方法。

此外，在 Swoft AOP 中，如果某个方法内调用了另一个 被织入 的方法时，AOP 也会向该方法织入通知。例如我们定义了一个类 A，它有两个 public 方法 func1 和 func2，然后我们定义一个切面，使用 @PointBean(include={A::class}) 注解将 A 类（所有方法）进行织入，示例代码。

//

//// 定义了一个类 A，它有两个 public 方法 func1 和 func2，然后我们定义一个切面，使用 @PointBean(include={A::class}) 注解将 A 类（所有方法）进行织入

<?php

class A

{

function func1()

{

echo "func1 \n";

}

function func2()

{

$this->func1();

echo "func2 \n";

}

}

?>

当我们执行 func2 时，我们的切面会被执行 两次，两次执行的顺序相同。切面会先对 func2 织入通知，其次对 func1 织入通知。

--

--

四、事件

https://www.swoft.org/documents/v2/basic-components/event/

//

Swoft 2 对事件进行了更加清晰和严谨的规划，提供了基本的事件注册与触发管理。

GitHub: https://github.com/swoft-cloud/swoft-event

--

1、安装

默认情况下事件组件已包含在 Swoft 框架中，如需单独安装只需执行下方命令。

composer require swoft/event

--

2、简介

//// 在 Swoft 中，我们将事件分为三大类。

Swoole Server 回调事件；

Swoft Server 事件，基于 Swoole 回调处理，扩展了可用事件以方便自定义；

应用级自定义事件管理；

--

3、事件分组

除部分特殊事件外，在一个应用中，大多数事件存在关联性，此时我们可以对事件进行分组以方便识别和管理。建议根据实际情况在名称设计上对事件进行分组。

//

//// 示例

swoft.server.\*

swoft.process.\*

swoft.pool.\*

//

swoft.http.request.before

swoft.http.request.after

//

swoft.db.query.start

swoft.db.query.after

//

swoft.redis.start

swoft.redis.after

//

swoft.ws.start

swoft.ws.after

//

swoft.tcp.start

swoft.tcp.after

//

swoft.udp.start

swoft.udp.after

--

4、通配符

//// 事件支持使用通配符 \* 对一组相关事件进行监听，具体分为两种

|1>、\* 全局事件通配符。直接对 \* 添加监听器 @Listener("\*")，此时所有触发的事件都会被此监听器所监听。

|2>、{prefix}.\* 指定分组事件的监听。例如 @Listener("swoft.db.\*")，此时所有以 swoft.db. 为前缀的事件（例如 swoft.db.query、swoft.db.connect）都会被此监听器所监听。

在事件到达监听器前停止本次事件的传播 $event->stopPropagation(true) 时，后面的监听器将不会收到该事件。

--

5、如何使用

(1)、监听器

//// 事件监听器类通过注解 @Listener 指定，其拥有两个属性

event：监听事件名称；

priority：监听器的优先级，数字越大优先执行；

注意：监听类必须实现接口 Swoft\Event\EventHandlerInterface。

(2)、消费者

事件消费者类通过注解 @Subscriber 指定。

(3)、监听器示例

<?php declare(strict\_types=1);

namespace SwoftTest\Event\Testing;

use Swoft\Event\Annotation\Mapping\Listener;

use Swoft\Event\EventHandlerInterface;

use Swoft\Event\EventInterface;

/\*\*

\* Class TestHandler

\*

\* @Listener("test.evt")

\*/

class TestHandler implements EventHandlerInterface

{

/\*\*

\* @param EventInterface $event

\*/

public function handle(EventInterface $event): void

{

$pos = \_\_METHOD\_\_;

echo "handle the event '{$event->getName()}' on the: $pos\n";

}

}

?>

(4)、消费者示例

<?php declare(strict\_types=1);

namespace SwoftTest\Event\Testing;

use Swoft\Event\Annotation\Mapping\Subscriber;

use Swoft\Event\EventInterface;

use Swoft\Event\EventSubscriberInterface;

use Swoft\Event\Listener\ListenerPriority;

/\*\*

\* Class TestSubscriber

\*

\* @Subscriber()

\*/

class TestSubscriber implements EventSubscriberInterface

{

public const EVENT\_ONE = 'test.event1';

public const EVENT\_TWO = 'test.event2';

/\*\*

\* Configure events and corresponding processing methods (you can configure the priority)

\* @return array

\* [

\* 'event name' => 'handler method'

\* 'event name' => ['handler method', priority]

\* ]

\*/

public static function getSubscribedEvents(): array

{

return [

self::EVENT\_ONE => 'handleEvent1',

self::EVENT\_TWO => ['handleEvent2', ListenerPriority::HIGH],

];

}

public function handleEvent1(EventInterface $evt): void

{

$evt->setParams(['msg' => 'handle the event: test.event1 position: TestSubscriber.handleEvent1()']);

}

public function handleEvent2(EventInterface $evt): void

{

$evt->setParams(['msg' => 'handle the event: test.event2 position: TestSubscriber.handleEvent2()']);

}

}

?>

(5)、触发事件

事件名称建议放置在一个专用类的常量中，方便进行管理和维护。

//// 方式一，使用便捷，多个参数按顺序存入，因此只能通过索引获取

Swoft::trigger('event name', 'target', $arg0, $arg1);

// 获取

$target = $event->getTarget();

$arg0 = $event->getParam(0);

$arg1 = $event->getParam(1);

//

//// 方式二，多个参数按键值对的方式存入，可根据 key 获取

Swoft::triggerByArray('event name', 'target', [

'arg0' => $arg0,

'arg1' => $arg1

]);

// 获取

$target = $event->getTarget();

$arg0 = $event->getParam('arg0');

$arg1 = $event->getParam('arg1');

--

6、Swoft 事件

Swoft 事件基于 Swoole 的回调处理扩展了一些可用 Server 事件，提供更加精细的操作空间。完整事件列表请参阅 ServerEvent.php ( https://github.com/swoft-cloud/swoft-server/blob/master/src/ServerEvent.php ) 文件。

//

//// ServerEvent.php

<?php declare(strict\_types=1);

namespace Swoft\Server;

/\*\*

\* Class ServerEvent

\*

\* @since 2.0

\*/

final class ServerEvent

{

/\*\*

\* Before set swoole settings

\*/

public const BEFORE\_SETTING = 'swoft.server.setting.before';

/\*\*

\* Before add swoole events

\*/

public const BEFORE\_ADDED\_EVENT = 'swoft.server.added.event.before';

/\*\*

\* After add swoole events

\*/

public const AFTER\_ADDED\_EVENT = 'swoft.server.added.event.after';

/\*\*

\* Before add listener(s)

\*/

public const BEFORE\_ADDED\_LISTENER = 'swoft.server.added.listener.before';

/\*\*

\* After each listener is successfully added

\*/

public const AFTER\_ADDED\_LISTENER = 'swoft.server.added.listener.after';

/\*\*

\* Before add process(es)

\*/

public const BEFORE\_ADDED\_PROCESS = 'swoft.server.added.process.before';

/\*\*

\* Add process(es)

\*/

public const ADDED\_PROCESS = 'swoft.server.added.process';

/\*\*

\* After each process is successfully added

\*/

public const AFTER\_ADDED\_PROCESS = 'swoft.server.added.process.after';

/\*\*

\* Swoft before start server event

\*/

public const BEFORE\_START = 'swoft.server.start.before';

/\*\*

\* On task process start event

\*/

public const TASK\_PROCESS\_START = 'swoft.process.task.start';

/\*\*

\* On work process start event

\*/

public const WORK\_PROCESS\_START = 'swoft.process.work.start';

/\*\*

\* on user process start event

\*/

public const USER\_PROCESS\_START = 'swoft.process.user.start';

/\*\*

\* Server pipe-message. please {@see \Swoft\Server\Swoole\PipeMessageListener}

\*/

public const PIPE\_MESSAGE = 'swoft.server.pipe.message';

/\*\*

\* after event

\*/

public const AFTER\_EVENT = 'swoft.server.event.after';

/\*\*

\* Before shutdown event

\*/

public const BEFORE\_SHUTDOWN\_EVENT = 'swoft.server.event.shutdown.before';

/\*\*

\* Before start event

\*/

public const BEFORE\_START\_EVENT = 'swoft.server.event.start.before';

/\*\*

\* Before worker error event

\*/

public const BEFORE\_WORKER\_ERROR\_EVENT = 'swoft.server.event.worker.error.before';

/\*\*

\* Before worker start event

\*/

public const BEFORE\_WORKER\_START\_EVENT = 'swoft.server.event.worker.start.before';

/\*\*

\* Before worker stop event

\*/

public const BEFORE\_WORKER\_STOP\_EVENT = 'swoft.server.event.worker.stop.before';

/\*\*

\* Before bind listener(s)

\*

\* @deprecated

\*/

public const BEFORE\_BIND\_LISTENER = 'swoft.server.added.listener.before';

/\*\*

\* Before bind swoole events

\*

\* @deprecated

\*/

public const BEFORE\_BIND\_EVENT = 'swoft.server.added.event.before';

}

?>

(1)、基础事件

基础事件定义请参阅 SwoftEvent.php ( http://github.com/swoft-cloud/swoft-framework/blob/master/src/SwoftEvent.php ) 文件。

//

//// SwoftEvent.php

// http://github.com/swoft-cloud/swoft-framework/blob/master/src/SwoftEvent.php

<?php declare(strict\_types=1);

namespace Swoft;

/\*\*

\* Class SwoftEvent

\*

\* @since 2.0

\*/

final class SwoftEvent

{

/\*\*

\* Swoft init complete

\*/

public const APP\_INIT\_COMPLETE = 'swoft.init.complete';

/\*\*

\* Session complete

\* - webSocket connection close

\* - tcp connection close

\*/

public const SESSION\_COMPLETE = 'swoft.session.complete';

/\*\*

\* Coroutine complete

\*/

public const COROUTINE\_COMPLETE = 'swoft.co.complete';

/\*\*

\* Coroutine exception

\*/

public const COROUTINE\_EXCEPTION = 'swoft.co.exception';

/\*\*

\* Coroutine destroy

\*/

public const COROUTINE\_DESTROY = 'swoft.co.destroy';

/\*\*

\* Coroutine defer

\*/

public const COROUTINE\_DEFER = 'swoft.co.defer';

/\*\*

\* Worker shutdown

\*/

public const WORKER\_SHUTDOWN = 'swoft.worker.shutdown';

/\*\*

\* Timer after before

\*/

public const TIMER\_AFTER\_BEFORE = 'swoft.timer.after.before';

/\*\*

\* Timer after after

\*/

public const TIMER\_AFTER\_AFTER = 'swoft.timer.after.after';

/\*\*

\* Timer tick before

\*/

public const TIMER\_TICK\_BEFORE = 'swoft.timer.tick.before';

/\*\*

\* Timer tick after

\*/

public const TIMER\_TICK\_AFTER = 'swoft.timer.tick.after';

}

(2)、使用示例

我们可以在 Swoole Server 启动前注册一个自定义进程，这样可以让进程由 Server 托管。

|1>、不需要执行 Start，在 Server 启动时会自动创建进程，并执行指定的子进程函数；

|2>、在 Shutdown 关闭服务器时，会向用户进程发送 SIGTERM 信号以关闭用户进程；

|3>、自定义进程会托管到 Manager 进程，如果发生致命错误，Manager 进程会自动重建；

//

///1/ AttachMyProcessHandler

<?php declare(strict\_types=1);

namespace App\Listener;

use App\Process\MyProcess;

use Swoft\Event\Annotation\Mapping\Listener;

use Swoft\Event\EventHandlerInterface;

use Swoft\Event\EventInterface;

use Swoft\Server\ServerEvent;

/\*\*

\* Class AttachMyProcessHandler

\*

\* @Listener(ServerEvent::BEFORE\_START)

\*/

class AttachMyProcessHandler implements EventHandlerInterface

{

/\*\*

\* @param EventInterface $event

\*/

public function handle(EventInterface $event): void

{

$swooleServer = $event->target->getSwooleServer();

$process = bean(MyProcess::class);

$swooleServer->addProcess($process->create());

}

}

?>

//

///2/ MyProcess

<?php declare(strict\_types=1);

namespace App\Process;

use Swoft\Event\Annotation\Mapping\Listener;

use Swoole\Process;

/\*\*

\* Class MyProcess

\*

\* @Bean()

\*/

class MyProcess

{

public function create(): Process

{

return new Process([$this, 'handle']);

}

public function handle(Process $process)

{

CLog::info('MyProcess started.');

// 用户进程实现了广播功能，循环接收管道消息，并发给服务器的所有连接

while (true) {

$msg = $process->read();

foreach($server->connections as $conn) {

$server->send($conn, $msg);

}

}

}

}

?>

--

7、Swoole 事件

Swoole 文档中的每个事件，在 Swoft 里面均可监听，并且可以存在多个监听器。

(1)、事件常量

事件常量请参阅 SwooleEvent.php ( https://github.com/swoft-cloud/swoft-server/blob/master/src/SwooleEvent.php ) 文件。

//

//// SwooleEvent.php

// https://github.com/swoft-cloud/swoft-server/blob/master/src/SwooleEvent.php

<?php declare(strict\_types=1);

namespace Swoft\Server;

use Swoft\Server\Contract\CloseInterface;

use Swoft\Server\Contract\ConnectInterface;

use Swoft\Server\Contract\FinishInterface;

use Swoft\Server\Contract\HandshakeInterface;

use Swoft\Server\Contract\MessageInterface;

use Swoft\Server\Contract\PacketInterface;

use Swoft\Server\Contract\PipeMessageInterface;

use Swoft\Server\Contract\ReceiveInterface;

use Swoft\Server\Contract\RequestInterface;

use Swoft\Server\Contract\SyncTaskInterface;

use Swoft\Server\Contract\TaskInterface;

/\*\*

\* Class SwooleEvent

\*

\* @since 2.0

\*/

class SwooleEvent

{

/\*\*

\* Start

\*/

public const START = 'start';

/\*\*

\* Shutdown

\*/

public const SHUTDOWN = 'shutdown';

/\*\*

\* WorkerStart

\*/

public const WORKER\_START = 'workerStart';

/\*\*

\* WorkerStop

\*/

public const WORKER\_STOP = 'workerStop';

/\*\*

\* WorkerError

\*/

public const WORKER\_ERROR = 'workerError';

/\*\*

\* ManagerStart

\*/

public const MANAGER\_START = 'managerStart';

/\*\*

\* ManagerStop

\*/

public const MANAGER\_STOP = 'managerStop';

/\*\*

\* Task

\*/

public const TASK = 'task';

/\*\*

\* Finish

\*/

public const FINISH = 'finish';

/\*\*

\* PipeMessage

\*/

public const PIPE\_MESSAGE = 'pipeMessage';

/\*\*

\* Handshake

\*/

public const HANDSHAKE = 'handshake';

/\*\*

\* Message

\*/

public const MESSAGE = 'message';

/\*\*

\* Open

\*/

public const OPEN = 'open';

/\*\*

\* Request

\*/

public const REQUEST = 'request';

/\*\*

\* Packet

\*/

public const PACKET = 'packet';

/\*\*

\* Receive

\*/

public const RECEIVE = 'receive';

/\*\*

\* Connect

\*/

public const CONNECT = 'connect';

/\*\*

\* Close

\*/

public const CLOSE = 'close';

/\*\*

\* Event interface listener mapping

\*/

public const LISTENER\_MAPPING = [

// For http server

self::REQUEST => RequestInterface::class,

// For websocket server

self::HANDSHAKE => HandshakeInterface::class,

self::MESSAGE => MessageInterface::class,

// For tcp server

self::CLOSE => CloseInterface::class,

self::RECEIVE => ReceiveInterface::class,

self::CONNECT => ConnectInterface::class,

// For udp server

self::PACKET => PacketInterface::class,

// For task

self::TASK => [

SyncTaskInterface::class,

TaskInterface::class

],

self::FINISH => FinishInterface::class,

// For process

self::PIPE\_MESSAGE => PipeMessageInterface::class,

];

}

(2)、使用示例

<?php declare(strict\_types=1);

namespace App\Listener;

use ReflectionException;

use Swoft\Bean\Exception\ContainerException;

use Swoft\Event\Annotation\Mapping\Listener;

use Swoft\Event\EventHandlerInterface;

use Swoft\Event\EventInterface;

use Swoft\Log\Helper\CLog;

use Swoft\Server\Swoole\SwooleEvent;

/\*\*

\* Class MasterStartListener

\*

\* @Listener(SwooleEvent::START)

\*/

class MasterStartListener implements EventHandlerInterface

{

/\*\*

\* @param EventInterface $event

\*

\* @throws ReflectionException

\* @throws ContainerException

\*/

public function handle(EventInterface $event): void

{

CLog::info('Master started');

}

}

?>

--

8、自定义事件

更多自定义事件介绍参考：https://github.com/inhere/php-event-manager/blob/master/README.md 。

//

//// Event Dispatcher

// https://github.com/inhere/php-event-manager/blob/master/README.md

//

//// 简洁、功能完善的事件管理调度实现

实现自 Psr 14 - 事件调度器；

支持对一个事件添加多个监听器；

支持设置事件优先级；

支持快速的事件组注册；

支持根据事件名称来快速的对事件组监听，eg 触发 app.run、app.end 都将同时会触发 app.\* 事件；

支持通配符事件的监听；

//

(1)、项目地址

github https://github.com/inhere/php-event-manager.git

(2)、安装

//// composer 命令

composer require inhere/event

//

//// composer.json

{

"require": {

"inhere/event": "dev-master"

}

}

(3)、事件调度器

事件调度器, 也可称之为事件管理器。事件的注册、监听器注册、调度(触发)都是由它管理的。

use Inhere\Event\EventManager;

$em = new EventManager();

(4)、事件监听器

监听器允许是：function 函数；一个闭包；一个监听器类(可以有多种方式)。

//

///1/ function函数

// ...(省略)

$em->attach(Mailer::EVENT\_MESSAGE\_SENT, 'my\_function');

//

///2/ 闭包

// ...(省略)

$em->attach(Mailer::EVENT\_MESSAGE\_SENT, function(Event $event) {

// $message = $event->message;

// ... some logic

});

//

///3/ 监听器类(有多种方式)

// 类里面存在跟事件相同名称的方法，此种方式可以在类里面写多个事件的处理方法

class ExamListener1

{

public function messageSent(EventInterface $event)

{

echo "handle the event {$event->getName()}\n";

}

}

// 一个类(含有 \_\_invoke 方法)，此时这个类对象就相当于一个闭包

class ExamListener2

{

public function \_\_invoke(EventInterface $event)

{

echo "handle the event {$event->getName()}\n";

}

}

// 实现接口 EventHandlerInterface，触发时会自动调用 handle() 方法。

class ExamHandler implements EventHandlerInterface

{

/\*\*

\* @param EventInterface $event

\* @return mixed

\*/

public function handle(EventInterface $event)

{

// TODO: Implement handle() method.

}

}

// 实现接口 EventSubscriberInterface，可以在一个类里面自定义监听多个事件

/\*\*

\* Class EnumGroupListener

\* @package Inhere\Event\Examples

\*/

class EnumGroupListener implements EventSubscriberInterface

{

const TEST\_EVENT = 'test';

const POST\_EVENT = 'post';

/\*\*

\* 配置事件与对应的处理方法

\* @return array

\*/

public static function getSubscribedEvents(): array

{

return [

self::TEST\_EVENT => 'onTest',

self::POST\_EVENT => ['onPost', ListenerPriority::LOW], // 还可以配置优先级

];

}

public function onTest(EventInterface $event)

{

$pos = \_\_METHOD\_\_;

echo "handle the event {$event->getName()} on the: $pos\n";

}

public function onPost(EventInterface $event)

{

$pos = \_\_METHOD\_\_;

echo "handle the event {$event->getName()} on the: $pos\n";

}

}

(5)、快速使用

///1/ 绑定事件触发

// 事件定义(a pre-defined event)

class MessageEvent extends Event

{

// append property ...

public $message;

}

//

// 事件触发(in the business)

class Mailer

{

use EventManagerAwareTrait;

const EVENT\_MESSAGE\_SENT = 'messageSent';

public function send($message)

{

// ...发送 $message 的逻辑...

$event = new MessageEvent(self::EVENT\_MESSAGE\_SENT);

$event->message = $message;

// 事件触发

$this->eventManager->trigger($event);

}

}

//

///2/ 触发事件

$em = new EventManager();

// 绑定事件

$em->attach(Mailer::EVENT\_MESSAGE\_SENT, 'exam\_handler');

$em->attach(Mailer::EVENT\_MESSAGE\_SENT, function (EventInterface $event)

{

$pos = \_\_METHOD\_\_;

echo "handle the event {$event->getName()} on the: $pos\n";

});

// 这里给它设置了更高的优先级

$em->attach(Mailer::EVENT\_MESSAGE\_SENT, new ExamListener1(), 10);

$em->attach(Mailer::EVENT\_MESSAGE\_SENT, new ExamListener2());

$em->attach(Mailer::EVENT\_MESSAGE\_SENT, new ExamHandler());

$mailer = new Mailer();

$mailer->setEventManager($em);

// 执行，将会触发事件

$mailer->send('hello, world!');

//

///3/ 运行示例

完整的实例代码在 examples/demo.php 中。

运行: php examples/demo.php

输出：

$ php examples/exam.php

handle the event 'messageSent' on the: ExamListener1::messageSent // 更高优先级的先调用

handle the event 'messageSent' on the: exam\_handler

handle the event 'messageSent' on the: {closure}

handle the event 'messageSent' on the: ExamListener2::\_\_invoke

handle the event 'messageSent' on the: Inhere\Event\Examples\ExamHandler::handle

(6)、一组事件的监听器

除了一些特殊的事件外，在一个应用中，大多数事件是有关联的，此时我们就可以对事件进行分组，方便识别和管理使用。

//

///1/ 事件分组 推荐将相关的事件，在名称设计上进行分组

// 模型相关：

model.insert

model.update

model.delete

// DB相关：

db.connect

db.disconnect

db.query

// 应用相关：

app.start

app.run

app.stop

//

///1-1/ 一个简单的示例应用类

/\*\*

\* Class App

\* @package Inhere\Event\Examples

\*/

class App

{

use EventManagerAwareTrait;

const ON\_START = 'app.start';

const ON\_STOP = 'app.stop';

const ON\_BEFORE\_REQUEST = 'app.beforeRequest';

const ON\_AFTER\_REQUEST = 'app.afterRequest';

public function \_\_construct(EventManager $em)

{

$this->setEventManager($em);

$this->eventManager->trigger(new Event(self::ON\_START, [

'key' => 'val'

]));

}

public function run()

{

$sleep = 0;

$this->eventManager->trigger(self::ON\_BEFORE\_REQUEST);

echo 'request handling ';

while ($sleep <= 3) {

$sleep++;

echo '.';

sleep(1);

}

echo "\n";

$this->eventManager->trigger(self::ON\_AFTER\_REQUEST);

}

public function \_\_destruct()

{

$this->eventManager->trigger(new Event(self::ON\_STOP, [

'key1' => 'val1'

]));

}

}

//

///1-2/ 此应用的监听器类

将每个事件的监听器写一个类，显得有些麻烦。我们可以只写一个类用里面不同的方法来处理不同的事件。

// 方式一： 类里面存在跟事件名称相同的方法(app.start -> start())。这种方式简单快捷，但是有一定的限制 - 事件名与方法的名称必须相同。

/\*\*

\* Class AppListener

\* @package Inhere\Event\Examples

\*/

class AppListener

{

public function start(EventInterface $event)

{

$pos = \_\_METHOD\_\_;

echo "handle the event {$event->getName()} on the: $pos\n";

}

public function beforeRequest(EventInterface $event)

{

$pos = \_\_METHOD\_\_;

echo "handle the event {$event->getName()} on the: $pos\n";

}

public function afterRequest(EventInterface $event)

{

$pos = \_\_METHOD\_\_;

echo "handle the event {$event->getName()} on the: $pos\n";

}

public function stop(EventInterface $event)

{

$pos = \_\_METHOD\_\_;

echo "handle the event {$event->getName()} on the: $pos\n";

}

}

//

// 方式二：实现接口 EventSubscriberInterface，有时候我们并不想将处理方法定义成事件名称一样，想自定义。此时我们可以实现接口 EventSubscriberInterface，通过里面的 getSubscribedEvents() 来自定义事件和对应的处理方法。运行示例请看 examples/enum-group.php。

/\*\*

\* Class EnumGroupListener

\* @package Inhere\Event\Examples

\*/

class EnumGroupListener implements EventSubscriberInterface

{

const TEST\_EVENT = 'test';

const POST\_EVENT = 'post';

/\*\*

\* 配置事件与对应的处理方法

\* @return array

\*/

public static function getSubscribedEvents(): array

{

return [

self::TEST\_EVENT => 'onTest',

self::POST\_EVENT => ['onPost', ListenerPriority::LOW], // 还可以配置优先级

];

}

public function onTest(EventInterface $event)

{

$pos = \_\_METHOD\_\_;

echo "handle the event {$event->getName()} on the: $pos\n";

}

public function onPost(EventInterface $event)

{

$pos = \_\_METHOD\_\_;

echo "handle the event {$event->getName()} on the: $pos\n";

}

}

//

///1-3/ 添加监听

// 这里使用方式一

$em = new EventManager();

// register a group listener

$em->attach('app', new AppListener());

// create app

$app = new App($em);

// run

$app->run();

//

///1-4/ 运行示例

完整的示例代码在 examples/named-group.php 中。

运行: php examples/named-group.php

输出：

$ php examples/named-group.php

handle the event 'app.start' on the: Inhere\Event\Examples\AppListener::start

handle the event 'app.beforeRequest' on the: Inhere\Event\Examples\AppListener::beforeRequest

request handling ....

handle the event 'app.afterRequest' on the: Inhere\Event\Examples\AppListener::afterRequest

handle the event 'app.stop' on the: Inhere\Event\Examples\AppListener::stop

(7)、事件通配符 \*

///1/ 支持使用事件通配符 \* 对一组相关的事件进行监听, 分两种。

\* 全局的事件通配符。直接对 \* 添加监听器($em->attach('\*', 'global\_listener')), 此时所有触发的事件都会被此监听器接收到。

{prefix}.\* 指定分组事件的监听。eg $em->attach('db.\*', 'db\_listener'), 此时所有触发的以 db. 为前缀的事件(eg db.query db.connect)都会被此监听器接收到。

//

当然，你在事件到达监听器前停止了本次事件的传播$event->stopPropagation(true);，就不会被后面的监听器接收到了。

//

///2/ 示例，在上面的组事件监听器改下，添加一个 app.\* 的事件监听。

// AppListener 新增一个方法

class AppListener

{

// ...

public function allEvent(EventInterface $event)

{

$pos = \_\_METHOD\_\_;

echo "handle the event '{$event->getName()}' on the: $pos\n";

}

}

// ...

$em = new EventManager();

$groupListener = new AppListener();

// register a group listener

$em->attach('app', $groupListener);

// all `app.` prefix events will be handled by `AppListener::allEvent()`

$em->attach('app.\*', [$groupListener, 'allEvent']);

// create app

$app = new App($em);

// run

$app->run();

//

///3/ 运行示例

运行: php examples/named-group.php 输出：(可以看到每个事件都经过了AppListener::allEvent()的处理)

$ php examples/named-group.php

handle the event 'app.start' on the: Inhere\Event\Examples\AppListener::start

handle the event 'app.start' on the: Inhere\Event\Examples\AppListener::allEvent

handle the event 'app.beforeRequest' on the: Inhere\Event\Examples\AppListener::beforeRequest

handle the event 'app.beforeRequest' on the: Inhere\Event\Examples\AppListener::allEvent

request handling ....

handle the event 'app.afterRequest' on the: Inhere\Event\Examples\AppListener::afterRequest

handle the event 'app.afterRequest' on the: Inhere\Event\Examples\AppListener::allEvent

handle the event 'app.stop' on the: Inhere\Event\Examples\AppListener::stop

handle the event 'app.stop' on the: Inhere\Event\Examples\AppListener::allEvent

(8)、事件对象

事件对象 - 装载了在触发事件时相关的上下文信息，用户自定义的。

//

///1/ 预先创建一个事件

/// 直接简单的使用类 Event

$myEvent = new Event('name', 'target', [ 'some params ...' ]);

//

/// 使用继承了 Event 的子类，这样你可以追加自定义数据。

// create event class

class MessageEvent extends Event

{

protected $name = 'messageSent';

// append property ...

public $message;

}

--

9、参与贡献

欢迎参与贡献，您可以 Fork 我们的组件仓库( swoft/component )；修改代码然后发送 PR；阅读 提交代码 的注意事项。

--

--

五、配置

https://www.swoft.org/documents/v2/basic-components/config/

Swoft 配置由两部分组成，env 环境配置和 config 应用配置。

env 一般配置一些和环境相关的一些参数，比如运行模式、资源地址

config 一般用于配置应用级别的配置以及业务级别的配置

--

1、安装

composer require swoft/config

--

2、Git仓库

Github https://github.com/swoft-cloud/swoft-config

--

3、环境配置

(1)、文件配置

项目根目录配置一个名称为 .env 文件，采用 KV 格式配置，此文件配置的数据，可以加载到内存里面，供业务使用。

APP\_DEBUG = 1

SWOFT\_DEBUG = 1

(2)、系统变量

除文件方式配置外，还可以把一些参数配置到系统变量，系统变量的参数也会加载到内存，供业务使用。

(3)、如何使用

swoft 提供了函数读取以上两种方式配置的数据。

//

//// 获取一个环境变量的值或所有环境变量参数

env(string $key = null, mixed $default = null): mixed

返回环境变量 key 的值， 如果环境变量 key 不存在则返回默认值。 如果省略 key 参数，则所有环境变量都将作为关联数组 array 返回。

default 默认值，可以是任何类型，也可以是一个 闭包。

返回值默认做了转换。比如配置 true 字符串，返回的转换成一个 bool 类型。

(4)、特殊规则

配置(string) 转换类型结果

true/false/(true)/(false) bool

empty|string(空字符串)

null|null

A\_B|如果是存在的常量，转成对应的值

--

4、应用配置

(1)、bean配置

应用配置数据是由一个bean对象管理的，可以在 app/bean.php 文件设置应用配置参数。

return [

'config' => [

'path' => \_\_DIR\_\_ . '/../config',

],

];

//// 可配置项

path 自定配置文件路径。

base 主文件名称，默认 base (其他文件的数据都会按文件名为key合并到主文件数据中)。

type 配置文件类型，默认 php 同时也支持 yaml 格式。

parser 配置解析器，默认已经配置 php/yaml 解析器。

env 配置当前环境比如 dev/test/pre/pro。

(2)、数据格式

配置目录所有配置文件会解析成一个数组，但是不会递归合并数据，只会合并当前目录文件数据，以它的文件名称为数组 key 进行合并数组。 比如 config 目录配置文件如下:

|-- base.php

|-- data.php

`-- pro

|-- base.php

`-- data.php

只会解析当前目录文件数据，不会递归解析数据。当前使用 env 配置时，环境目录里面的配置信息会覆盖最外层文件名称相同的数据。提醒：配置文件里面可以使用 env()函数读取环境配置。

// config/base.php

return [

'key' => 'value'

];

// config/data.php

return [

'dkey' => [

'dvalue'

],

'key' => 'value'

];

// config/pro/base.php

return [

'key' => 'valuePro'

];

// config/pro/data.php

return [

'dkey' => [

'dvalue'

],

'key' => 'valuePro'

];

//// 如上配置文件，当不配置 config 的 env 参数，合并的数据格式如下

return [

'key' => 'value',

'data' => [

'dkey' => [

'dvalue'

],

'key' => 'value'

]

];

//

//// 当配置 config 对象的 'env' => 'pro' 参数，合并的数据格式如下

return [

'key' => 'valuePro',

'data' => [

'dkey' => [

'dvalue'

],

'key' => 'valuePro'

]

];

(3)、获取配置

框架提供全局函数、注解、config 对象多种方式，使用应用配置数据。

///1/ 函数 全局函数使用 config()

config(string $key = null, mixed $default = null):mixed

key 配置参数 key，子数组可以使用 . 分割，比如上面的例子 data.dkey 可以获取到 ["dvalue"], 当key=null 获取所有配置参数。

default 默认参数，如果 key 参数不存在，返回默认值，默认值可以是任意类型。

//

///2/ 注解

通过容器使用注解的方式，注入配置到属性值。

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Config\Annotation\Mapping\Config;

/\*\*

\* @Bean()

\*/

class Demo

{

/\*\*

\* @Config("data.dkey")

\*/

private $dvalue = [];

// ...

}

此例子和上面功能一样，都是读取相同的数据，两种不同的方式。使用注解，一定要保证类是一个bean对象(通过其它注解注入到容器)。

//

///3/ 对象

如果上面两种方式还不能满足你的业务需求，你可以从容器里面获取配置对象，里面自带很多方式操作配置数据。

$config = \Swoft::getBean('config');

/// config 对象常用方法

get(string $key, $default = null) 获取参数；

offsetGet($key) 获取参数；

…

--

5、参与贡献

欢迎参与贡献，您可以 fork 我们的开发仓库 swoft/component ( https://github.com/swoft-cloud/swoft-component )；修改代码然后发起 PR；关于发起PR的注意事项( https://github.com/swoft-cloud/swoft/issues/829 )。

--

--

六、公共方法

https://www.swoft.org/documents/v2/basic-components/public-function/

--

0、简介

框架中内置封装了一些公共函数，开发者在实际业务中可以直接使用，无需重复封装。其中包括：协程函数、数组函数、目录（文件夹）函数、环境函数、文件函数、文件系统函数、对象函数、PHP 助手函数、字符串函数、系统函数、XML 函数、通用函数。

--

1、协程函数

(1)、创建协程

Swoft 框架中不能使用 Swoole 提供的 go 函数创建协程，否则会造成请求和上下文丢失最终导致一些不可预估的问题。Swoft 拥有两种方式创建协程。

//

///1/ 方式一

use Swoft\Co;

Co::create(function () {

// to do

});

//

///2/ 方式二，通过助手函数 sgo 创建

sgo(function () {

// todo

});

sgo 函数使用与 Swoole 中的 go 函数完全一致，再次强调，框架中只能使用 sgo 函数创建协程。

(2)、协程 ID

// 获取当前协程 ID，-1 为非协程环境

use Swoft\Co;

$id = Co::id();

(3)、顶级协程 ID

// 获取顶级（最外层）协程 ID

use Swoft\Co;

$id = Co::tid();

(4)、读文件

读取成功返回字符串内容，读取失败返回 false，可使用 swoole\_last\_error 获取错误信息。`readFile` 方法没有文件大小限制，读取的内容会存放在内存中，因此读取大文件时会占用大量内存。

// 使用示例

use Swoft\Co;

$fileName = 'file.txt'; //$filename: 带完整路径的文件名称;

$data = Co::readFile($fileName);

(5)、写文件

// 写入成功返回 true，失败返回 false。

use Swoft\Co;

$fileName = 'file.txt';

$data = Co::writeFile($fileName, 'Swoft Framework');

$filename：带完整路径的文件的名称，须有可写权限，文件不存在时会自动创建，打开文件失败会直接返回 false。

$data：写入内容，大小限制为 4M。

$flags：写入选项，默认值为 FILE\_USE\_INCLUDE\_PATH 常量。可选值参考：文件系统 - 预定义常量 - PHP Manual ( https://www.php.net/manual/zh/filesystem.constants.php )。

(6)、并发

框架底层通过协程通道，封装了一套混合 IO 并发操作的方法，一般用于没有依赖的多个流程。

<?php

use Co\Http\Client;

use Swoft\Co;

/\*\*

\* Class CoTest

\*

\* @since 2.0

\*/

class CoTest

{

public function testMulti()

{

$requests = [

'method' => [$this, 'requestMethod'], //对象方法；

'staticMethod' => self::requestMehtodByStatic(), //对象静态方法;

'closure' => function () { //闭包函数;

$cli = new Client('www.baidu.com', 80);

$cli->get('/');

$result = $cli->body;

$cli->close();

return $result;

}

];

$response = Co::multi($requests);

}

public function requestMethod()

{

$cli = new Client('www.baidu.com', 80);

$cli->get('/');

$result = $cli->body;

$cli->close();

return $result;

}

public static function requestMehtodByStatic()

{

$cli = new Client('www.baidu.com', 80);

$cli->get('/');

$result = $cli->body;

$cli->close();

return $result;

}

}

?>

//// 参数说明

$requests：多个操作集合，requests 支持多种格式，对象方法、对象静态方法、闭包函数。

$timeout：超时时间，默认永不超时。

并发执行结果按照 requests 数组中 key 对应关系返回。如果某个 key 对应的值为 false，意味着该操作执行失败。requests 内每个操作可执行的业务不存在上限，根据实际业务而定。

--

2、数组函数

数组函数需引入 Swoft\Stdlib\Helper\Arr 或 Swoft\Stdlib\Helper\ArrayHelper，详情参阅：ArrayHelper.php 文件( https://github.com/swoft-cloud/swoft-stdlib/blob/master/src/Helper/ArrayHelper.php )。

<?php declare(strict\_types=1);

namespace Swoft\Stdlib\Helper;

use ArrayAccess;

use Closure;

use InvalidArgumentException;

use Iterator;

use stdClass;

use Swoft\Stdlib\Collection;

use Swoft\Stdlib\Contract\Arrayable;

use Traversable;

use function array\_pop;

use function count;

use function func\_get\_args;

use function get\_class;

use function in\_array;

use function is\_array;

use function is\_float;

use function is\_int;

use function is\_numeric;

use function is\_object;

use function is\_string;

use function mb\_strlen;

use function method\_exists;

use function similar\_text;

use function value;

/\*\*

\* Array helper

\*

\* @since 2.0

\*/

class ArrayHelper

{

/\*\*

\* Converts an object or an array of objects into an array.

\*

\* @param object|array|string $object the object to be converted into an array

\* @param array $properties a mapping from object class names to the properties that need to put into the resulting arrays.

\* The properties specified for each class is an array of the following format:

\*

\* @param boolean $recursive whether to recursively converts properties which are objects into arrays.

\*

\* @return array the array representation of the object

\*/

public static function toArray($object, $properties = [], $recursive = true): array

{

if (is\_array($object)) {

if ($recursive) {

/\*\* @var array $object \*/

foreach ($object as $key => $value) {

if (is\_array($value) || is\_object($value)) {

$object[$key] = static::toArray($value, $properties, true);

}

}

}

return $object;

}

if (is\_object($object)) {

if (!empty($properties)) {

$className = get\_class($object);

if (!empty($properties[$className])) {

$result = [];

foreach ($properties[$className] as $key => $name) {

if (is\_int($key)) {

$result[$name] = $object->$name;

} else {

$result[$key] = static::getValue($object, $name);

}

}

return $recursive ? static::toArray($result, $properties) : $result;

}

}

if ($object instanceof Arrayable) {

$result = $object->toArray();

} else {

$result = [];

/\*\* @var array $object \*/

foreach ($object as $key => $value) {

$result[$key] = $value;

}

}

return $recursive ? static::toArray($result, $properties) : $result;

}

return [$object];

}

/\*\*

\* Merges two or more arrays into one recursively.

\* If each array has an element with the same string key value, the latter

\* will overwrite the former (different from array\_merge\_recursive).

\* Recursive merging will be conducted if both arrays have an element of array

\* type and are having the same key.

\* For integer-keyed elements, the elements from the latter array will

\* be appended to the former array.

\*

\* @param array $a array to be merged to

\* @param array $b array to be merged from. You can specify additional

\* arrays via third argument, fourth argument etc.

\*

\* @return array the merged array (the original arrays are not changed.)

\*/

public static function merge($a, $b): array

{

$args = func\_get\_args();

$res = array\_shift($args);

while (!empty($args)) {

$next = array\_shift($args);

foreach ($next as $k => $v) {

if (is\_int($k)) {

if (isset($res[$k])) {

$res[] = $v;

} else {

$res[$k] = $v;

}

} elseif (is\_array($v) && isset($res[$k]) && is\_array($res[$k])) {

$res[$k] = self::merge($res[$k], $v);

} else {

$res[$k] = $v;

}

}

}

return $res;

}

/\*\*

\* Retrieves the value of an array element or object property with the given key or property name.

\* If the key does not exist in the array or object, the default value will be returned instead.

\*

\* The key may be specified in a dot format to retrieve the value of a sub-array or the property

\* of an embedded object. In particular, if the key is `x.y.z`, then the returned value would

\* be `$array['x']['y']['z']` or `$array->x->y->z` (if `$array` is an object). If `$array['x']`

\* or `$array->x` is neither an array nor an object, the default value will be returned.

\* Note that if the array already has an element `x.y.z`, then its value will be returned

\* instead of going through the sub-arrays. So it is better to be done specifying an array of key names

\* like `['x', 'y', 'z']`.

\*

\* Below are some usage examples,

\*

\* ```php

\* // working with array

\* $username = \Swoft\Helper\ArrayHelper::getValue($\_POST, 'username');

\* // working with object

\* $username = \Swoft\Helper\ArrayHelper::getValue($user, 'username');

\* // working with anonymous function

\* $fullName = \Swoft\Helper\ArrayHelper::getValue($user, function ($user, $defaultValue) {

\* return $user->firstName . ' ' . $user->lastName;

\* });

\* // using dot format to retrieve the property of embedded object

\* $street = \Swoft\Helper\ArrayHelper::getValue($users, 'address.street');

\* // using an array of keys to retrieve the value

\* $value = \Swoft\Helper\ArrayHelper::getValue($versions, ['1.0', 'date']);

\* ```

\*

\* @param array|object $array array or object to extract value from

\* @param string|Closure|array $key key name of the array element, an array of keys or property name of the object,

\* or an anonymous function returning the value. The anonymous function signature should be:

\* `function($array, $defaultValue)`.

\* @param mixed $default the default value to be returned if the specified array key does not exist. Not used when

\* getting value from an object.

\*

\* @return mixed the value of the element if found, default value otherwise

\*/

public static function getValue($array, $key, $default = null)

{

if ($key instanceof Closure) {

return $key($array, $default);

}

if (is\_array($key)) {

$lastKey = array\_pop($key);

/\*\* @var array $key \*/

foreach ($key as $keyPart) {

$array = static::getValue($array, $keyPart);

}

$key = $lastKey;

}

if (is\_array($array) && (isset($array[$key]) || array\_key\_exists($key, $array))) {

return $array[$key];

}

if (is\_string($key) && ($pos = strrpos($key, '.')) !== false) {

$array = static::getValue($array, substr($key, 0, $pos), $default);

$key = (string)substr($key, $pos + 1);

}

if (is\_object($array)) {

// this is expected to fail if the property does not exist, or \_\_get() is not implemented

// it is not reliably possible to check whether a property is accessable beforehand

return $array->$key;

}

if (is\_array($array)) {

return (isset($array[$key]) || array\_key\_exists($key, $array)) ? $array[$key] : $default;

}

return $default;

}

/\*\*

\* Removes an item from an array and returns the value. If the key does not exist in the array, the default value

\* will be returned instead.

\*

\* Usage examples,

\*

\* ```php

\* // $array = ['type' => 'A', 'options' => [1, 2]];

\* // working with array

\* $type = \Swoft\Helper\ArrayHelper::remove($array, 'type');

\* // $array content

\* // $array = ['options' => [1, 2]];

\* ```

\*

\* @param array $array the array to extract value from

\* @param string $key key name of the array element

\* @param mixed $default the default value to be returned if the specified key does not exist

\*

\* @return mixed|null the value of the element if found, default value otherwise

\*/

public static function remove(&$array, $key, $default = null)

{

if (is\_array($array) && (isset($array[$key]) || array\_key\_exists($key, $array))) {

$value = $array[$key];

unset($array[$key]);

return $value;

}

return $default;

}

/\*\*

\* Get all of the given array except for a specified array of keys.

\*

\* @param array $array

\* @param array|string $keys

\*

\* @return array

\*/

public static function except($array, $keys): array

{

static::forget($array, $keys);

return $array;

}

/\*\*

\* Remove one or many array items from a given array using "dot" notation.

\*

\* @param array $array

\* @param array|string $keys

\*

\* @return void

\*/

public static function forget(&$array, $keys): void

{

$original = &$array;

$keys = (array)$keys;

if (count($keys) === 0) {

return;

}

foreach ($keys as $key) {

// if the exact key exists in the top-level, remove it

if (static::exists($array, $key)) {

unset($array[$key]);

continue;

}

$parts = explode('.', $key);

// clean up before each pass

$array = &$original;

while (count($parts) > 1) {

$part = array\_shift($parts);

if (isset($array[$part]) && is\_array($array[$part])) {

$array = &$array[$part];

} else {

continue 2;

}

}

unset($array[array\_shift($parts)]);

}

}

/\*\*

\* Get a value from the array, and remove it.

\*

\* @param array $array

\* @param string $key

\* @param mixed $default

\*

\* @return mixed

\*/

public static function pull(&$array, $key, $default = null)

{

$value = static::get($array, $key, $default);

static::forget($array, $key);

return $value;

}

/\*\*

\* Indexes and/or groups the array according to a specified key.

\* The input should be either multidimensional array or an array of objects.

\*

\* The $key can be either a key name of the sub-array, a property name of object, or an anonymous

\* function that must return the value that will be used as a key.

\*

\* $groups is an array of keys, that will be used to group the input array into one or more sub-arrays based

\* on keys specified.

\*

\* If the `$key` is specified as `null` or a value of an element corresponding to the key is `null` in addition

\* to `$groups` not specified then the element is discarded.

\*

\* For example:

\*

\* ```php

\* $array = [

\* ['id' => '123', 'Data' => 'abc', 'device' => 'laptop'],

\* ['id' => '345', 'Data' => 'def', 'device' => 'tablet'],

\* ['id' => '345', 'Data' => 'hgi', 'device' => 'smartphone'],

\* ];

\* $result = ArrayHelper::index($array, 'id');

\* ```

\*

\* The result will be an associative array, where the key is the value of `id` attribute

\*

\* ```php

\* [

\* '123' => ['id' => '123', 'Data' => 'abc', 'device' => 'laptop'],

\* '345' => ['id' => '345', 'Data' => 'hgi', 'device' => 'smartphone']

\* // The second element of an original array is overwritten by the last element because of the same id

\* ]

\* ```

\*

\* An anonymous function can be used in the grouping array as well.

\*

\* ```php

\* $result = ArrayHelper::index($array, function ($element) {

\* return $element['id'];

\* });

\* ```

\*

\* Passing `id` as a third argument will group `$array` by `id`:

\*

\* ```php

\* $result = ArrayHelper::index($array, null, 'id');

\* ```

\*

\* The result will be a multidimensional array grouped by `id` on the first level, by `device` on the second level

\* and indexed by `Data` on the third level:

\*

\* ```php

\* [

\* '123' => [

\* ['id' => '123', 'Data' => 'abc', 'device' => 'laptop']

\* ],

\* '345' => [ // all elements with this index are present in the result array

\* ['id' => '345', 'Data' => 'def', 'device' => 'tablet'],

\* ['id' => '345', 'Data' => 'hgi', 'device' => 'smartphone'],

\* ]

\* ]

\* ```

\*

\* The anonymous function can be used in the array of grouping keys as well:

\*

\* ```php

\* $result = ArrayHelper::index($array, 'Data', [function ($element) {

\* return $element['id'];

\* }, 'device']);

\* ```

\*

\* The result will be a multidimensional array grouped by `id` on the first level, by the `device` on the second one

\* and indexed by the `Data` on the third level:

\*

\* ```php

\* [

\* '123' => [

\* 'laptop' => [

\* 'abc' => ['id' => '123', 'Data' => 'abc', 'device' => 'laptop']

\* ]

\* ],

\* '345' => [

\* 'tablet' => [

\* 'def' => ['id' => '345', 'Data' => 'def', 'device' => 'tablet']

\* ],

\* 'smartphone' => [

\* 'hgi' => ['id' => '345', 'Data' => 'hgi', 'device' => 'smartphone']

\* ]

\* ]

\* ]

\* ```

\*

\* @param array $array the array that needs to be indexed or grouped

\* @param string|Closure|null $key the column name or anonymous function which result will be used to index the array

\* @param string|string[]|Closure[]|null $groups the array of keys, that will be used to group the input array

\* by one or more keys. If the $key attribute or its value for the particular element is null and $groups is not

\* defined, the array element will be discarded. Otherwise, if $groups is specified, array element will be added

\* to the result array without any key.

\*

\* @return array the indexed and/or grouped array

\*/

public static function index($array, $key, $groups = []): array

{

$result = [];

$groups = (array)$groups;

foreach ($array as $element) {

$lastArray = &$result;

foreach ($groups as $group) {

$value = static::getValue($element, $group);

if (!array\_key\_exists($value, $lastArray)) {

$lastArray[$value] = [];

}

$lastArray = &$lastArray[$value];

}

if ($key === null) {

if (!empty($groups)) {

$lastArray[] = $element;

}

} else {

$value = static::getValue($element, $key);

if ($value !== null) {

if (is\_float($value)) {

$value = (string)$value;

}

$lastArray[$value] = $element;

}

}

unset($lastArray);

}

return $result;

}

/\*\*

\* Returns the values of a specified column in an array.

\* The input array should be multidimensional or an array of objects.

\*

\* For example,

\*

\* ```php

\* $array = [

\* ['id' => '123', 'Data' => 'abc'],

\* ['id' => '345', 'Data' => 'def'],

\* ];

\* $result = ArrayHelper::getColumn($array, 'id');

\* // the result is: ['123', '345']

\*

\* // using anonymous function

\* $result = ArrayHelper::getColumn($array, function ($element) {

\* return $element['id'];

\* });

\* ```

\*

\* @param array $array

\* @param string|Closure $name

\* @param boolean $keepKeys whether to maintain the array keys. If false, the resulting array

\* will be re-indexed with integers.

\*

\* @return array the list of column values

\*/

public static function getColumn($array, $name, $keepKeys = true): array

{

$result = [];

if ($keepKeys) {

foreach ($array as $k => $element) {

$result[$k] = static::getValue($element, $name);

}

} else {

foreach ($array as $element) {

$result[] = static::getValue($element, $name);

}

}

return $result;

}

/\*\*

\* Builds a map (key-value pairs) from a multidimensional array or an array of objects.

\* The `$from` and `$to` parameters specify the key names or property names to set up the map.

\* Optionally, one can further group the map according to a grouping field `$group`.

\*

\* For example,

\*

\* ```php

\* $array = [

\* ['id' => '123', 'name' => 'aaa', 'class' => 'x'],

\* ['id' => '124', 'name' => 'bbb', 'class' => 'x'],

\* ['id' => '345', 'name' => 'ccc', 'class' => 'y'],

\* ];

\*

\* $result = ArrayHelper::map($array, 'id', 'name');

\* // the result is:

\* // [

\* // '123' => 'aaa',

\* // '124' => 'bbb',

\* // '345' => 'ccc',

\* // ]

\*

\* $result = ArrayHelper::map($array, 'id', 'name', 'class');

\* // the result is:

\* // [

\* // 'x' => [

\* // '123' => 'aaa',

\* // '124' => 'bbb',

\* // ],

\* // 'y' => [

\* // '345' => 'ccc',

\* // ],

\* // ]

\* ```

\*

\* @param array $array

\* @param string|Closure $from

\* @param string|Closure $to

\* @param string|Closure $group

\*

\* @return array

\*/

public static function map($array, $from, $to, $group = null): array

{

$result = [];

foreach ($array as $element) {

$key = static::getValue($element, $from);

$value = static::getValue($element, $to);

if ($group !== null) {

$result[static::getValue($element, $group)][$key] = $value;

} else {

$result[$key] = $value;

}

}

return $result;

}

/\*\*

\* Checks if the given array contains the specified key.

\* This method enhances the `array\_key\_exists()` function by supporting case-insensitive

\* key comparison.

\*

\* @param string $key the key to check

\* @param array $array the array with keys to check

\* @param boolean $caseSensitive whether the key comparison should be case-sensitive

\*

\* @return boolean whether the array contains the specified key

\*/

public static function keyExists($key, $array, $caseSensitive = true): ?bool

{

if ($caseSensitive) {

return array\_key\_exists($key, $array);

} else {

foreach (array\_keys($array) as $k) {

if (strcasecmp($key, $k) === 0) {

return true;

}

}

return false;

}

}

/\*\*

\* Sorts an array of objects or arrays (with the same structure) by one or several keys.

\*

\* @param array $array the array to be sorted. The array will be modified after calling this method.

\* @param string|Closure|array $key the key(s) to be sorted by. This refers to a key name of the sub-array

\* elements, a property name of the objects, or an anonymous function returning the values for comparison

\* purpose. The anonymous function signature should be: `function($item)`.

\* To sort by multiple keys, provide an array of keys here.

\* @param integer|array $direction the sorting direction. It can be either `SORT\_ASC` or `SORT\_DESC`.

\* When sorting by multiple keys with different sorting directions, use an array of sorting directions.

\* @param integer|array $sortFlag the PHP sort flag. Valid values include

\* `SORT\_REGULAR`, `SORT\_NUMERIC`, `SORT\_STRING`, `SORT\_LOCALE\_STRING`, `SORT\_NATURAL` and `SORT\_FLAG\_CASE`.

\* Please refer to [PHP manual](http://php.net/manual/en/function.sort.php)

\* for more details. When sorting by multiple keys with different sort flags, use an array of sort flags.

\*

\* @throws InvalidArgumentException if the $direction or $sortFlag parameters do not have

\* correct number of elements as that of $key.

\*/

public static function multisort(&$array, $key, $direction = SORT\_ASC, $sortFlag = SORT\_REGULAR): void

{

$keys = is\_array($key) ? $key : [$key];

if (empty($keys) || empty($array)) {

return;

}

$n = count($keys);

if (is\_scalar($direction)) {

$direction = array\_fill(0, $n, $direction);

} elseif (count($direction) !== $n) {

throw new InvalidArgumentException('The length of $direction parameter must be the same as that of $keys.');

}

if (is\_scalar($sortFlag)) {

$sortFlag = array\_fill(0, $n, $sortFlag);

} elseif (count($sortFlag) !== $n) {

throw new InvalidArgumentException('The length of $sortFlag parameter must be the same as that of $keys.');

}

$args = [];

foreach ($keys as $i => $k) {

$flag = $sortFlag[$i];

$args[] = static::getColumn($array, $k);

$args[] = $direction[$i];

$args[] = $flag;

}

// This fix is used for cases when main sorting specified by columns has equal values

// Without it it will lead to Fatal Error: Nesting level too deep - recursive dependency?

$args[] = range(1, count($array));

$args[] = SORT\_ASC;

$args[] = SORT\_NUMERIC;

$args[] = &$array;

array\_multisort(...$args);

}

/\*\*

\* Returns a value indicating whether the given array is an associative array.

\*

\* An array is associative if all its keys are strings. If `$allStrings` is false,

\* then an array will be treated as associative if at least one of its keys is a string.

\*

\* Note that an empty array will NOT be considered associative.

\*

\* @param array $array the array being checked

\* @param boolean $allStrings whether the array keys must be all strings in order for

\* the array to be treated as associative.

\*

\* @return boolean whether the array is associative

\*/

public static function isAssociative($array, $allStrings = true): ?bool

{

if (!is\_array($array) || empty($array)) {

return false;

}

if ($allStrings) {

foreach ($array as $key => $value) {

if (!is\_string($key)) {

return false;

}

}

return true;

} else {

foreach ($array as $key => $value) {

if (is\_string($key)) {

return true;

}

}

return false;

}

}

/\*\*

\* Returns a value indicating whether the given array is an indexed array.

\*

\* An array is indexed if all its keys are integers. If `$consecutive` is true,

\* then the array keys must be a consecutive sequence starting from 0.

\*

\* Note that an empty array will be considered indexed.

\*

\* @param array $array the array being checked

\* @param boolean $consecutive whether the array keys must be a consecutive sequence

\* in order for the array to be treated as indexed.

\*

\* @return boolean whether the array is associative

\*/

public static function isIndexed($array, $consecutive = false): ?bool

{

if (!is\_array($array)) {

return false;

}

if (empty($array)) {

return true;

}

if ($consecutive) {

return array\_keys($array) === range(0, count($array) - 1);

} else {

foreach ($array as $key => $value) {

if (!is\_int($key)) {

return false;

}

}

return true;

}

}

/\*\*

\* Check whether an array or [[\Traversable]] contains an element.

\*

\* This method does the same as the PHP function [in\_array()](http://php.net/manual/en/function.in-array.php)

\* but additionally works for objects that implement the [[\Traversable]] interface.

\*

\* @param mixed $needle The value to look for.

\* @param array|Traversable $haystack The set of values to search.

\* @param boolean $strict Whether to enable strict (`===`) comparison.

\*

\* @return boolean `true` if `$needle` was found in `$haystack`, `false` otherwise.

\* @throws InvalidArgumentException if `$haystack` is neither traversable nor an array.

\* @see http://php.net/manual/en/function.in-array.php

\*/

public static function isIn($needle, $haystack, $strict = false): bool

{

if ($haystack instanceof Traversable) {

foreach ($haystack as $value) {

if ($needle == $value && (!$strict || $needle === $value)) {

return true;

}

}

} elseif (is\_array($haystack)) {

return in\_array($needle, $haystack, $strict);

} else {

throw new InvalidArgumentException('Argument $haystack must be an array or implement Traversable');

}

return false;

}

/\*\*

\* Checks whether a variable is an array or [[\Traversable]].

\*

\* This method does the same as the PHP function [is\_array()](http://php.net/manual/en/function.is-array.php)

\* but additionally works on objects that implement the [[\Traversable]] interface.

\*

\* @param mixed $var The variable being evaluated.

\*

\* @return boolean whether $var is array-like

\* @see http://php.net/manual/en/function.is\_array.php

\*/

public static function isTraversable($var): bool

{

return is\_array($var) || $var instanceof Traversable;

}

/\*\*

\* Checks whether an array or [[\Traversable]] is a subset of another array or [[\Traversable]].

\*

\* This method will return `true`, if all elements of `$needles` are contained in

\* `$haystack`. If at least one element is missing, `false` will be returned.

\*

\* @param array|Traversable $needles The values that must \*\*all\*\* be in `$haystack`.

\* @param array|Traversable $haystack The set of value to search.

\* @param boolean $strict Whether to enable strict (`===`) comparison.

\*

\* @return boolean `true` if `$needles` is a subset of `$haystack`, `false` otherwise.

\* @throws InvalidArgumentException if `$haystack` or `$needles` is neither traversable nor an array.

\*/

public static function isSubset($needles, $haystack, $strict = false): ?bool

{

if (is\_array($needles) || $needles instanceof Traversable) {

foreach ($needles as $needle) {

if (!static::isIn($needle, $haystack, $strict)) {

return false;

}

}

return true;

}

throw new InvalidArgumentException('Argument $needles must be an array or implement Traversable');

}

/\*\*

\* Filters array according to rules specified.

\*

\* For example:

\* ```php

\* $array = [

\* 'A' => [1, 2],

\* 'B' => [

\* 'C' => 1,

\* 'D' => 2,

\* ],

\* 'E' => 1,

\* ];

\*

\* $result = \Swoft\Helper\ArrayHelper::Filter($array, ['A']);

\* // $result will be:

\* // [

\* // 'A' => [1, 2],

\* // ]

\*

\* $result = \Swoft\Helper\ArrayHelper::Filter($array, ['A', 'B.C']);

\* // $result will be:

\* // [

\* // 'A' => [1, 2],

\* // 'B' => ['C' => 1],

\* // ]

\* ```

\*

\* $result = \Swoft\Helper\ArrayHelper::Filter($array, ['B', '!B.C']);

\* // $result will be:

\* // [

\* // 'B' => ['D' => 2],

\* // ]

\* ```

\*

\* @param array $array Source array

\* @param array $filters Rules that define array keys which should be left or removed from results.

\* Each rule is:

\* - `var` - `$array['var']` will be left in result.

\* - `var.key` = only `$array['var']['key'] will be left in result.

\* - `!var.key` = `$array['var']['key'] will be removed from result.

\*

\* @return array Filtered array

\*/

public static function filter($array, $filters): array

{

$result = [];

$forbiddenVars = [];

foreach ($filters as $var) {

$keys = explode('.', $var);

$globalKey = $keys[0];

$localKey = $keys[1] ?? null;

if ($globalKey[0] === '!') {

$forbiddenVars[] = [

substr($globalKey, 1),

$localKey,

];

continue;

}

if (empty($array[$globalKey])) {

continue;

}

if ($localKey === null) {

$result[$globalKey] = $array[$globalKey];

continue;

}

if (!isset($array[$globalKey][$localKey])) {

continue;

}

if (!array\_key\_exists($globalKey, $result)) {

$result[$globalKey] = [];

}

$result[$globalKey][$localKey] = $array[$globalKey][$localKey];

}

foreach ($forbiddenVars as $var) {

[$globalKey, $localKey] = $var;

if (array\_key\_exists($globalKey, $result)) {

unset($result[$globalKey][$localKey]);

}

}

return $result;

}

/\*\*

\* Determine whether the given value is array accessible.

\*

\* @param mixed $value

\*

\* @return bool

\*/

public static function accessible($value): bool

{

return is\_array($value) || $value instanceof ArrayAccess;

}

/\*\*

\* Determine if the given key exists in the provided array.

\*

\* @param ArrayAccess|array $array

\* @param string|int $key

\*

\* @return bool

\*/

public static function exists($array, $key): bool

{

if (is\_array($array)) {

return array\_key\_exists($key, $array);

}

return $array->offsetExists($key);

}

/\*\*

\* Get an item from an array using "dot" notation.

\*

\* @param ArrayAccess|array $array

\* @param string|int $key

\* @param mixed $default

\*

\* @return mixed

\*/

public static function get($array, $key = null, $default = null)

{

if ($array instanceof stdClass) {

$array = (array)$array;

}

if (null === $key) {

return $array;

}

if (isset($array[$key])) {

return $array[$key];

}

// Fix: If is int, stop continue find.

if (!is\_string($key)) {

return $default;

}

foreach (explode('.', $key) as $segment) {

if (static::accessible($array) && static::exists($array, $segment)) {

$array = $array[$segment];

} else {

return value($default);

}

}

return $array;

}

/\*\*

\* Check if an item exists in an array using "dot" notation.

\*

\* @param ArrayAccess|array $array

\* @param string $key

\*

\* @return bool

\*/

public static function has($array, $key): bool

{

if (empty($array) || null === $key) {

return false;

}

if (array\_key\_exists($key, $array)) {

return true;

}

foreach (explode('.', $key) as $segment) {

if ((is\_array($array) && array\_key\_exists($segment,

$array)) || ($array instanceof ArrayAccess && $array->offsetExists($segment))

) {

$array = $array[$segment];

} else {

return false;

}

}

return true;

}

/\*\*

\* Set an array item to a given value using "dot" notation.

\* If no key is given to the method, the entire array will be replaced.

\*

\* @param array $array

\* @param string $key

\* @param mixed $value

\*

\* @return array

\*/

public static function set(&$array, $key, $value): array

{

if (null === $key) {

return $array = $value;

}

$keys = explode('.', $key);

while (count($keys) > 1) {

$key = array\_shift($keys);

// If the key doesn't exist at this depth, we will just create an empty array

// to hold the next value, allowing us to create the arrays to hold final

// values at the correct depth. Then we'll keep digging into the array.

if (!isset($array[$key]) || !is\_array($array[$key])) {

$array[$key] = [];

}

$array = &$array[$key];

}

$array[array\_shift($keys)] = $value;

return $array;

}

/\*\*

\* Insert one array to another array

\*

\* @param array $array

\* @param int $index

\* @param array $insert

\*/

public static function insert(array &$array, int $index, ...$insert): void

{

$firstArray = array\_splice($array, 0, $index);

$array = array\_merge($firstArray, $insert, $array);

}

/\*\*

\* If the given value is not an array and not null, wrap it in one.

\*

\* @param mixed $value

\*

\* @return array

\*/

public static function wrap($value): array

{

if ($value === null) {

return [];

}

return is\_array($value) ? $value : [$value];

}

/\*\*

\* @param mixed $value

\*

\* @return bool

\*/

public static function isArrayable($value): bool

{

return is\_array($value) || $value instanceof Arrayable;

}

/\*\*

\* Flatten a multi-dimensional array into a single level.

\*

\* @param array $array

\* @param int $depth

\*

\* @return array

\*/

public static function flatten(array $array, int $depth = PHP\_INT\_MAX): array

{

$result = [];

foreach ($array as $item) {

$item = $item instanceof Collection ? $item->all() : $item;

if (!is\_array($item)) {

$result[] = $item;

} elseif ($depth === 1) {

$result = array\_merge($result, array\_values($item));

} else {

$result = array\_merge($result, static::flatten($item, $depth - 1));

}

}

return $result;

}

/\*\*

\* find similar text from an array|Iterator

\*

\* @param string $need

\* @param Iterator|array $iterator

\* @param int $similarPercent

\*

\* @return array

\*/

public static function findSimilar(string $need, $iterator, int $similarPercent = 45): array

{

if (!$need) {

return [];

}

// find similar command names by similar\_text()

$similar = [];

foreach ($iterator as $name) {

similar\_text($need, $name, $percent);

if ($similarPercent <= (int)$percent) {

$similar[] = $name;

}

}

return $similar;

}

/\*\*

\* get key Max Width

\*

\* @param array $data

\* [

\* 'key1' => 'value1',

\* 'key2-test' => 'value2',

\* ]

\* @param bool $expectInt

\*

\* @return int

\*/

public static function getKeyMaxWidth(array $data, bool $expectInt = false): int

{

$keyMaxWidth = 0;

foreach ($data as $key => $value) {

// key is not a integer

if (!$expectInt || !is\_numeric($key)) {

$width = mb\_strlen((string)$key, 'UTF-8');

$keyMaxWidth = $width > $keyMaxWidth ? $width : $keyMaxWidth;

}

}

return $keyMaxWidth;

}

/\*\*

\* Return the first element in an array passing a given truth test.

\*

\* @param array $array

\* @param callable|null $callback

\* @param mixed $default

\*

\* @return mixed

\*/

public static function first($array, callable $callback = null, $default = null)

{

if ($callback === null) {

if (empty($array)) {

return value($default);

}

foreach ($array as $item) {

return $item;

}

}

foreach ($array as $key => $value) {

if (call\_user\_func($callback, $value, $key)) {

return $value;

}

}

return value($default);

}

/\*\*

\* Filter the array using the given callback.

\*

\* @param array $array

\* @param callable $callback

\*

\* @return array

\*/

public static function where($array, callable $callback)

{

return array\_filter($array, $callback, ARRAY\_FILTER\_USE\_BOTH);

}

/\*\*

\* Convert the array into a query string.

\*

\* @param array $array

\*

\* @return string

\*/

public static function query(array $array): string

{

return http\_build\_query($array, '', '&', PHP\_QUERY\_RFC3986);

}

/\*\*

\* Get a subset of the items from the given array.

\*

\* @param array $array

\* @param array $keys

\*

\* @return array

\*/

public static function only(array $array, array $keys): array

{

return array\_intersect\_key($array, array\_flip((array)$keys));

}

/\*\*

\* Return the last element in an array passing a given truth test.

\*

\* @param array $array

\* @param callable|null $callback

\* @param mixed $default

\*

\* @return mixed

\*/

public static function last($array, callable $callback = null, $default = null)

{

if (is\_null($callback)) {

return empty($array) ? value($default) : end($array);

}

return static::first(array\_reverse($array, true), $callback, $default);

}

/\*\*

\* Pluck an array of values from an array.

\*

\* @param array $array

\* @param string|array $value

\* @param string|array|null $key

\*

\* @return array

\*/

public static function pluck($array, $value, $key = null)

{

$results = [];

foreach ($array as $item) {

$itemValue = static::get($item, $value);

// If the key is "null", we will just append the value to the array and keep

// looping. Otherwise we will key the array using the value of the key we

// received from the developer. Then we'll return the final array form.

if (is\_null($key)) {

$results[] = $itemValue;

} else {

$itemKey = static::get($item, $key);

if (is\_object($itemKey) && method\_exists($itemKey, '\_\_toString')) {

$itemKey = (string)$itemKey;

}

$results[$itemKey] = $itemValue;

}

}

return $results;

}

/\*\*

\* Collapse an array of arrays into a single array.

\*

\* @param array $array

\*

\* @return array

\*/

public static function collapse($array)

{

$results = [];

foreach ($array as $values) {

if ($values instanceof Collection) {

$values = $values->all();

} elseif (!is\_array($values)) {

continue;

}

$results = array\_merge($results, $values);

}

return $results;

}

/\*\*

\* Cross join the given arrays, returning all possible permutations.

\*

\* @param array ...$arrays

\*

\* @return array

\*/

public static function crossJoin(...$arrays)

{

$results = [[]];

foreach ($arrays as $index => $array) {

$append = [];

foreach ($results as $product) {

foreach ($array as $item) {

$product[$index] = $item;

$append[] = $product;

}

}

$results = $append;

}

return $results;

}

/\*\*

\* Push an item onto the beginning of an array.

\*

\* @param array $array

\* @param mixed $value

\* @param mixed $key

\*

\* @return array

\*/

public static function prepend($array, $value, $key = null)

{

if (is\_null($key)) {

array\_unshift($array, $value);

} else {

$array = [$key => $value] + $array;

}

return $array;

}

/\*\*

\* Get one or a specified number of random values from an array.

\*

\* @param array $array

\* @param int|null $number

\*

\* @return mixed

\*

\* @throws InvalidArgumentException

\*/

public static function random($array, $number = null)

{

$requested = is\_null($number) ? 1 : $number;

$count = count($array);

if ($requested > $count) {

throw new InvalidArgumentException("You requested {$requested} items, but there are only {$count} items available.");

}

if (is\_null($number)) {

return $array[array\_rand($array)];

}

if ((int)$number === 0) {

return [];

}

$keys = array\_rand($array, $number);

$results = [];

foreach ((array)$keys as $key) {

$results[] = $array[$key];

}

return $results;

}

/\*\*

\* array to string

\*

\* @param array $array

\* @param string $glue

\*

\* @return string

\*/

public static function toString(array $array, string $glue = '\_'): string

{

return implode($glue, $array);

}

/\*\*

\* Shuffle the given array and return the result.

\*

\* @param array $array

\* @param int|null $seed

\*

\* @return array

\*/

public static function shuffle($array, $seed = null)

{

if (is\_null($seed)) {

shuffle($array);

} else {

mt\_srand($seed);

shuffle($array);

mt\_srand();

}

return $array;

}

}

?>

--

3、目录函数

目录函数包括创建、遍历目录内容等，需引入 Swoft\Stdlib\Helper\Dir 或 Swoft\Stdlib\Helper\DirHelper 或 Swoft\Stdlib\Helper\DirectoryHelper。目录函数继承自文件系统函数，详情参阅：DirectoryHelper.php 文件( https://github.com/swoft-cloud/swoft-stdlib/blob/master/src/Helper/DirectoryHelper.php )。

<?php declare(strict\_types=1);

namespace Swoft\Stdlib\Helper;

use DirectoryIterator;

use FilesystemIterator;

use InvalidArgumentException;

use IteratorIterator;

use RecursiveCallbackFilterIterator;

use RecursiveDirectoryIterator;

use RecursiveIteratorIterator;

use RuntimeException;

use SplFileInfo;

use function file\_exists;

use function is\_dir;

use function mkdir;

use function sprintf;

use function strpos;

use function substr;

/\*\*

\* Directory helper

\*

\* @since 2.0

\*/

class DirectoryHelper extends FSHelper

{

/\*\*

\* Create directory with recursive

\*

\* @param string $dir

\* @param integer $mode

\*

\* @return void

\*/

public static function make(string $dir, int $mode = 0755): void

{

if (!file\_exists($dir) && !mkdir($dir, $mode, true) && !is\_dir($dir)) {

throw new RuntimeException(sprintf('Directory "%s" was not created', $dir));

}

}

/\*\*

\* Directory recursive iterator

\*

\* @param string $path

\* @param int $mode

\* @param int $flags

\*

\* @return RecursiveIteratorIterator

\*/

public static function recursiveIterator(

string $path,

int $mode = RecursiveIteratorIterator::LEAVES\_ONLY,

int $flags = 0

): RecursiveIteratorIterator {

if (empty($path) || !file\_exists($path)) {

throw new InvalidArgumentException('File path is not exist! Path: ' . $path);

}

$directoryIterator = new RecursiveDirectoryIterator($path);

return new RecursiveIteratorIterator($directoryIterator, $mode, $flags);

}

/\*\*

\* Directory iterator

\*

\* @param string $path

\*

\* @return IteratorIterator

\*/

public static function iterator(string $path): IteratorIterator

{

if (empty($path) || !file\_exists($path)) {

throw new InvalidArgumentException('File path is not exist! Path: ' . $path);

}

$directoryIterator = new DirectoryIterator($path);

return new IteratorIterator($directoryIterator);

}

/\*\*

\* Find all php files in the dir-path.

\*

\* @param string $dirPath

\*

\* @return RecursiveIteratorIterator

\*/

public static function phpFilesIterator(string $dirPath): RecursiveIteratorIterator

{

$filter = function (SplFileInfo $f): bool {

$name = $f->getFilename();

// Skip hidden files and directories.

if (strpos($name, '.') === 0) {

return false;

}

// Goon read sub-dir

if ($f->isDir()) {

return true;

}

// Only find php file

return $f->isFile() && substr($name, -4) === '.php';

};

return self::filterIterator($dirPath, $filter);

}

/\*\*

\* Directory iterator but support filter files.

\*

\* @param string $dirPath

\* @param callable $filter

\* eg: only find php file

\* $filter = function (\SplFileInfo $f): bool {

\* $name = $f->getFilename();

\*

\* // Skip hidden files and directories.

\* if (\strpos($name, '.') === 0) {

\* return false;

\* }

\*

\* // go on read sub-dir

\* if ($f->isDir()) {

\* return true;

\* }

\*

\* // php file

\* return $f->isFile() && \substr($name, -4) === '.php';

\* }

\* @param int $flags

\*

\* @return RecursiveIteratorIterator

\* @throws InvalidArgumentException

\*/

public static function filterIterator(

string $dirPath,

callable $filter,

$flags = FilesystemIterator::KEY\_AS\_PATHNAME | FilesystemIterator::CURRENT\_AS\_FILEINFO

): RecursiveIteratorIterator {

if (!$dirPath || !file\_exists($dirPath)) {

throw new InvalidArgumentException('Please provide a exists source directory. Path:' . $dirPath);

}

$directory = new RecursiveDirectoryIterator($dirPath, $flags);

$filterIterator = new RecursiveCallbackFilterIterator($directory, $filter);

return new RecursiveIteratorIterator($filterIterator);

}

}

?>

--

4、环境函数

环境函数主要用于获取当前运行环境，其中包括：

use Swoft\Stdlib\Helper\EnvHelper;

// 是否运行在 CLI 模式

EnvHelper::isCli();

// 是否运行在 PHP Debug 模式

EnvHelper::isPhpDbg();

// 当前运行环境是否为 Windows

EnvHelper::isWin();

EnvHelper::isWindows();

// 当前运行环境是否为 macOS

EnvHelper::isMac();

--

5、文件函数

se Swoft\Stdlib\Helper\FileHelper;

$fileName = '/opt/file.txt';

// 获取文件后缀，$clearPoint 为 true 时不返回带“.”的后缀

FileHelper::getSuffix($fileName, true);

// getExt 方法为 getExtension 方法别名，该方法与 getSuffix 方法作用相同

FileHelper::getExt($fileName, true);

FileHelper::getExtension($fileName, true);

// 获取文件 MIME 类型，文本会返回 text/plain

FileHelper::mimeType($fileName);

--

6、文件系统函数

文件系统函数需引入 Swoft\Stdlib\Helper\FSHelper。详情参阅：FSHelper 文件( https://github.com/swoft-cloud/swoft-stdlib/blob/master/src/Helper/FSHelper.php )。

<?php

namespace Swoft\Stdlib\Helper;

use function array\_pop;

use function explode;

use function implode;

use function preg\_match;

use function preg\_replace;

use function str\_replace;

use function strpos;

use function substr;

use const DIRECTORY\_SEPARATOR;

/\*\*

\* Class FSHelper - file system helper

\*

\* @since 2.0

\*/

class FSHelper

{

/\*\*

\* @param string $path

\*

\* @return string

\*/

public static function formatPath(string $path): string

{

if (DIRECTORY\_SEPARATOR === '\\') {

return str\_replace('\\', '/', $path);

}

return $path;

}

/\*\*

\* @param $path

\*

\* @return bool

\*/

public static function isAbsPath(string $path): bool

{

if (!$path) {

return false;

}

if (strpos($path, '/') === 0 // linux/mac

|| 1 === preg\_match('#^[a-z]:[\/|\\\]{1}.+#i', $path) // windows

) {

return true;

}

return false;

}

/\*\*

\* @param string $path e.g phar://E:/workenv/xxx/yyy/app.phar/web

\*

\* @return string

\*/

public function clearPharPath(string $path): string

{

if (strpos($path, 'phar://') === 0) {

$path = (string)substr($path, 7);

if (strpos($path, '.phar')) {

return preg\_replace('//[\w-]+\.phar/', '', $path);

}

}

return $path;

}

/\*\*

\* Returns canonicalized absolute pathname

\* Convert 'this/is/../a/./test/.///is' to 'this/a/test/is'

\*

\* @param string $path

\* @param bool $filter

\*

\* @return string

\*/

public static function conv2abs(string $path, bool $filter = true): string

{

$path = str\_replace('\\', '/', $path);

if (strpos($path, '..') === false) {

return $path;

}

$first = '';

$parts = explode('/', $path);

if ($filter) {

$first = $path[0] === '/' ? '/' : '';

$parts = array\_filter($parts, 'strlen');

}

$absolutes = [];

foreach ($parts as $part) {

if ('.' === $part) {

continue;

}

if ('..' === $part) {

array\_pop($absolutes);

} else {

$absolutes[] = $part;

}

}

return $first . implode('/', $absolutes);

}

}

?>

--

7、对象函数

对象函数用于相关对象操作，其中包括：

use Swoft\Stdlib\Helper\ObjectHelper;

// 获取对象哈希值

ObjectHelper::hash($object);

// 向对象设置或新增属性

ObjectHelper::init($object, $options = ['name' => 'Swoft']);

// 解析参数类型

ObjectHelper::parseParamType('int', $object);

// 获取属性基本数据类型

ObjectHelper::getPropertyBaseType($property);

// 获取数据类型默认值

ObjectHelper::getDefaultValue('int');

--

8、PHP 助手函数

PHP 助手函数扩展了一些原生函数，需引入 Swoft\Stdlib\Helper\PhpHelper。详情参阅：PhpHelper.php ( https://github.com/swoft-cloud/swoft-stdlib/blob/master/src/Helper/PhpHelper.php )文件。

<?php declare(strict\_types=1);

namespace Swoft\Stdlib\Helper;

use Throwable;

use function explode;

use function function\_exists;

use function get\_class;

use function is\_array;

use function is\_object;

use function is\_string;

use function method\_exists;

use function ob\_get\_clean;

use function ob\_start;

use function preg\_replace;

use function serialize;

use function sprintf;

use function strpos;

use function unserialize;

use function var\_dump;

use function var\_export;

use const PHP\_EOL;

/\*\*

\* Php helper

\*

\* @since 2.0

\*/

class PhpHelper

{

/\*\*

\* Call by callback

\*

\* @param callable|array $cb callback

\* @param array $args arguments

\*

\* @return mixed

\*/

public static function call($cb, ...$args)

{

if (is\_string($cb)) {

// className::method

if (strpos($cb, '::') > 0) {

$cb = explode('::', $cb, 2);

// function

} elseif (function\_exists($cb)) {

return $cb(...$args);

}

} elseif (is\_object($cb) && method\_exists($cb, '\_\_invoke')) {

return $cb(...$args);

}

if (is\_array($cb)) {

[$obj, $mhd] = $cb;

return is\_object($obj) ? $obj->$mhd(...$args) : $obj::$mhd(...$args);

}

return $cb(...$args);

}

/\*\*

\* Call by callback

\*

\* @param callable $cb

\* @param array $args

\*

\* @return mixed

\*/

public static function callByArray($cb, array $args = [])

{

return self::call($cb, ...$args);

}

/\*\*

\* @param $data

\*

\* @return string

\*/

public static function serialize($data): string

{

return serialize($data);

}

/\*\*

\* @param string $data

\* @param array $opts

\*

\* @return array|mixed

\*/

public static function unserialize(string $data, array $opts = ['allowed\_classes' => false])

{

return unserialize($data, $opts);

}

/\*\*

\* dump vars

\*

\* @param array ...$args

\*

\* @return string

\*/

public static function dumpVars(...$args): string

{

ob\_start();

var\_dump(...$args);

$string = ob\_get\_clean();

return preg\_replace("/=>\n\s+/", '=> ', $string);

}

/\*\*

\* print vars

\*

\* @param array ...$args

\*

\* @return string

\*/

public static function printVars(...$args): string

{

$string = '';

foreach ($args as $arg) {

$string .= print\_r($arg, 1) . PHP\_EOL;

}

return preg\_replace("/Array\n\s+\(/", 'Array (', $string);

}

/\*\*

\* @param mixed $var

\*

\* @return string

\*/

public static function exportVar($var): string

{

$string = var\_export($var, true);

return preg\_replace('/=>\s+\n\s+array \(/', '=> array (', $string);

}

/\*\*

\* @param Throwable $e

\* @param string $title

\* @param bool $debug

\*

\* @return string

\*/

public static function exceptionToString(Throwable $e, string $title = '', bool $debug = false): string

{

$errClass = get\_class($e);

if (false === $debug) {

return sprintf('%s %s(code:%d) %s', $title, $errClass, $e->getCode(), $e->getMessage());

}

return sprintf('%s%s(code:%d): %s At %s line %d', $title ? $title . ' - ' : '', $errClass, $e->getCode(),

$e->getMessage(), $e->getFile(), $e->getLine());

}

/\*\*

\* @param Throwable $e

\* @param bool $debug

\*

\* @return array

\*/

public static function exceptionToArray(Throwable $e, bool $debug = false): array

{

if (false === $debug) {

return [

'code' => $e->getCode(),

'error' => $e->getMessage(),

];

}

return [

'code' => $e->getCode(),

'error' => sprintf('(%s) %s', get\_class($e), $e->getMessage()),

'file' => sprintf('At %s line %d', $e->getFile(), $e->getLine()),

'trace' => $e->getTraceAsString(),

];

}

}

?>

--

9、字符串函数

字符串函数内容过多，需引入 Swoft\Stdlib\Helper\Str 或 Swoft\Stdlib\Helper\StringHelper。详情参阅：StringHelper.php ( https://github.com/swoft-cloud/swoft-stdlib/blob/master/src/Helper/StringHelper.php )文件。

<?php declare(strict\_types=1);

namespace Swoft\Stdlib\Helper;

use Swoft\Stdlib\Concern\RandomStringTrait;

use function explode;

use function is\_string;

use function lcfirst;

use function mb\_convert\_case;

use function mb\_strlen;

use function mb\_strtoupper;

use function preg\_match;

use function preg\_replace;

use function str\_pad;

use function str\_repeat;

use function str\_replace;

use function strlen;

use function strpos;

use function strrpos;

use function substr;

use function substr\_replace;

use function trim;

use const STR\_PAD\_LEFT;

use const STR\_PAD\_RIGHT;

/\*\*

\* String helper

\*

\* @since 2.0

\*/

class StringHelper

{

use RandomStringTrait;

/\*\*

\* The cache of snake-cased words.

\*

\* @var array

\*/

protected static $snakeCache = [];

/\*\*

\* The cache of camel-cased words.

\*

\* @var array

\*/

protected static $camelCache = [];

/\*\*

\* The cache of studly-cased words.

\*

\* @var array

\*/

protected static $studlyCache = [];

/\*\*

\* Transliterate a UTF-8 value to ASCII.

\*

\* @param string $value

\*

\* @return string

\*/

public static function ascii($value): string

{

foreach (StringChars::getChars() as $key => $val) {

$value = str\_replace($val, $key, $value);

}

return preg\_replace('/[^\x20-\x7E]/u', '', $value);

}

/\*\*

\* Convert a value to camel case.

\*

\* @param string $value

\* @param bool $lcfirst

\*

\* @return string

\*/

public static function camel($value, bool $lcfirst = true): string

{

if (isset(static::$camelCache[$value])) {

return static::$camelCache[$value];

}

return static::$camelCache[$value] = ($lcfirst ? lcfirst(static::studly($value)) : static::studly($value));

}

/\*\*

\* @param string $string

\* @param string $delimiter

\* @param int $limit

\*

\* @return array

\*/

public static function toArray(string $string, string $delimiter = ',', int $limit = 0): array

{

$string = trim($string, "$delimiter ");

if ($string === '') {

return [];

}

$values = [];

$rawList = $limit < 1 ? explode($delimiter, $string) : explode($delimiter, $string, $limit);

foreach ($rawList as $val) {

if (($val = trim($val)) !== '') {

$values[] = $val;

}

}

return $values;

}

/\*\*

\* @param string $str

\* @param string $separator

\* @param int $limit

\*

\* @return array

\*/

public static function explode(string $str, string $separator = ',', int $limit = 0): array

{

return static::toArray($str, $separator, $limit);

}

/\*\*

\* Determine if a given string contains a given substring.

\*

\* @param string $haystack

\* @param string|array $needles

\*

\* @return bool

\*/

public static function contains(string $haystack, $needles): bool

{

foreach ((array)$needles as $needle) {

if ($needle !== '' && strpos($haystack, $needle) !== false) {

return true;

}

}

return false;

}

/\*\*

\* Determine if a given string ends with a given substring.

\*

\* @param string $haystack

\* @param string|array $needles

\*

\* @return bool

\*/

public static function endsWith(string $haystack, $needles): bool

{

foreach ((array)$needles as $needle) {

if ((string)$needle === substr($haystack, -strlen($needle))) {

return true;

}

}

return false;

}

/\*\*

\* @param string $str

\*

\* @return string

\*/

public static function firstLine(string $str): string

{

if (!$str = trim($str)) {

return '';

}

if (strpos($str, "\n") > 0) {

return explode("\n", $str)[0];

}

return $str;

}

/\*\*

\* Cap a string with a single instance of a given value.

\*

\* @param string $value

\* @param string $cap

\*

\* @return string

\*/

public static function finish($value, $cap): string

{

$quoted = preg\_quote($cap, '/');

return preg\_replace('/(?:' . $quoted . ')+$/', '', $value) . $cap;

}

/\*\*

\* Determine if a given string matches a given pattern.

\*

\* @param string $pattern

\* @param string $value

\*

\* @return bool

\*/

public static function is(string $pattern, string $value): bool

{

if ($pattern === $value) {

return true;

}

$pattern = preg\_quote($pattern, '#');

// Asterisks are translated into zero-or-more regular expression wildcards

// to make it convenient to check if the strings starts with the given

// pattern such as "library/\*", making any string check convenient.

$pattern = str\_replace('\\*', '.\*', $pattern);

return (bool)preg\_match('#^' . $pattern . '\z#', $value);

}

/\*\*

\* Return the length of the given string.

\*

\* @param string|int $value

\* @param string $encode

\*

\* @return int

\*/

public static function length($value, string $encode = 'utf-8'): int

{

return mb\_strlen((string)$value, $encode);

}

/\*\*

\* Return the length of the given string.

\*

\* @param string|int $value

\* @param string|null $encode

\*

\* @return int

\*/

public static function len($value, string $encode = 'utf-8'): int

{

return mb\_strlen((string)$value, $encode);

}

/\*\*

\* @param string|int $string

\* @param int|float $padLen

\* @param string $padStr

\* @param int $padType

\*

\* @return string

\*/

public static function pad($string, $padLen, string $padStr = ' ', int $padType = STR\_PAD\_RIGHT): string

{

$string = (string)$string;

return $padLen > 0 ? str\_pad($string, (int)$padLen, $padStr, $padType) : $string;

}

/\*\*

\* @param string|int $string

\* @param int $padLen

\* @param string $padStr

\*

\* @return string

\*/

public static function padLeft($string, int $padLen, string $padStr = ' '): string

{

$string = (string)$string;

return $padLen > 0 ? str\_pad($string, $padLen, $padStr, STR\_PAD\_LEFT) : $string;

}

/\*\*

\* @param string|int $string

\* @param int $padLen

\* @param string $padStr

\*

\* @return string

\*/

public static function padRight($string, int $padLen, string $padStr = ' '): string

{

$string = (string)$string;

return $padLen > 0 ? str\_pad($string, $padLen, $padStr) : $string;

}

/\*\*

\* @param string|int $string

\* @param int $length

\*

\* @return string

\*/

public static function repeat($string, $length): string

{

return str\_repeat((string)$string, (int)$length);

}

/\*\*

\* Limit the number of characters in a string.

\*

\* @param string $value

\* @param int $limit

\* @param string $end

\*

\* @return string

\*/

public static function limit($value, $limit = 100, $end = '...'): string

{

if (mb\_strwidth($value, 'UTF-8') <= $limit) {

return $value;

}

return rtrim(mb\_strimwidth($value, 0, $limit, '', 'UTF-8')) . $end;

}

/\*\*

\* Convert the given string to lower-case.

\*

\* @param string $value

\*

\* @return string

\*/

public static function lower(string $value): string

{

return mb\_strtolower($value, 'UTF-8');

}

/\*\*

\* Limit the number of words in a string.

\*

\* @param string $value

\* @param int $words

\* @param string $end

\*

\* @return string

\*/

public static function words($value, $words = 100, $end = '...'): string

{

preg\_match('/^\s\*+(?:\S++\s\*+){1,' . $words . '}/u', $value, $matches);

if (!isset($matches[0]) || strlen($value) === strlen($matches[0])) {

return $value;

}

return rtrim($matches[0]) . $end;

}

/\*\*

\* Parse a `Class@method` style callback into class and method.

\*

\* @param string $callback

\* @param string $default

\*

\* @return array

\*/

public static function parseCallback(string $callback, string $default): array

{

return static::contains($callback, '@') ? explode('@', $callback, 2) : [$callback, $default];

}

/\*\*

\* Compares two strings using a constant-time algorithm.

\* Note: This method will leak length information.

\* Note: Adapted from Symfony\Component\Security\Core\Util\StringUtils.

\*

\* @param string $knownString

\* @param string $userInput

\*

\* @return bool

\* @deprecated since version 5.2. Use hash\_equals instead.

\*/

public static function equals(string $knownString, string $userInput): bool

{

return hash\_equals($knownString, $userInput);

}

/\*\*

\* Replace the first occurrence of a given value in the string.

\*

\* @param string $search

\* @param string $replace

\* @param string $subject

\*

\* @return string

\*/

public static function replaceFirst($search, $replace, $subject): string

{

$position = strpos($subject, $search);

if ($position !== false) {

return substr\_replace($subject, $replace, $position, strlen($search));

}

return $subject;

}

/\*\*

\* Replace the last occurrence of a given value in the string.

\*

\* @param string $search

\* @param string $replace

\* @param string $subject

\*

\* @return string

\*/

public static function replaceLast(string $search, $replace, string $subject): string

{

$position = strrpos($subject, $search);

if ($position !== false) {

return substr\_replace($subject, $replace, $position, strlen($search));

}

return $subject;

}

/\*\*

\* Convert the given string to upper-case.

\*

\* @param string $value

\*

\* @return string

\*/

public static function upper(string $value): string

{

return mb\_strtoupper($value, 'UTF-8');

}

/\*\*

\* Convert the given string to title case.

\*

\* @param string $value

\*

\* @return string

\*/

public static function title($value): string

{

return mb\_convert\_case($value, MB\_CASE\_TITLE, 'UTF-8');

}

/\*\*

\* Generate a URL friendly "slug" from a given string.

\*

\* @param string $title

\* @param string $separator

\*

\* @return string

\*/

public static function slug(string $title, string $separator = '-'): string

{

$title = static::ascii($title);

// Convert all dashes/underscores into separator

$flip = $separator === '-' ? '\_' : '-';

$title = preg\_replace('![' . preg\_quote($flip) . ']+!u', $separator, $title);

// Remove all characters that are not the separator, letters, numbers, or whitespace.

$title = preg\_replace('![^' . preg\_quote($separator) . '\pL\pN\s]+!u', '', mb\_strtolower($title));

// Replace all separator characters and whitespace by a single separator

$title = preg\_replace('![' . preg\_quote($separator) . '\s]+!u', $separator, $title);

return trim($title, $separator);

}

/\*\*

\* Convert a string to snake case.

\*

\* @param string $value

\* @param string $delimiter

\*

\* @return string

\*/

public static function snake(string $value, string $delimiter = '\_'): string

{

$key = $value . $delimiter;

if (isset(static::$snakeCache[$key])) {

return static::$snakeCache[$key];

}

if (!ctype\_lower($value)) {

$value = preg\_replace('/\s+/', '', $value);

$value = strtolower(preg\_replace('/(.)(?=[A-Z])/', '$1' . $delimiter, $value));

}

return static::$snakeCache[$key] = $value;

}

/\*\*

\* Determine if a given string starts with a given substring.

\*

\* @param string $haystack

\* @param string|array $needles

\*

\* @return bool

\*/

public static function startsWith(string $haystack, $needles): bool

{

foreach ((array)$needles as $needle) {

if ($needle !== '' && strpos($haystack, $needle) === 0) {

return true;

}

}

return false;

}

/\*\*

\* Convert a value to studly caps case.

\*

\* @param string $value

\*

\* @return string

\*/

public static function studly($value): string

{

$key = $value;

if (isset(static::$studlyCache[$key])) {

return static::$studlyCache[$key];

}

$value = ucwords(str\_replace(['-', '\_'], ' ', $value));

return static::$studlyCache[$key] = str\_replace(' ', '', $value);

}

/\*\*

\* Returns the portion of string specified by the start and length parameters.

\*

\* @param string $string

\* @param int $start

\* @param int|null $length

\*

\* @return string

\*/

public static function substr(string $string, $start, $length = null): string

{

return mb\_substr($string, $start, $length, 'UTF-8');

}

/\*\*

\* Make a string's first character uppercase.

\*

\* @param string $string

\*

\* @return string

\*/

public static function ucfirst(string $string): string

{

return static::upper(static::substr($string, 0, 1)) . static::substr($string, 1);

}

public static function trim($str, $prefix = '', $suffix = ''): void

{

}

public static function strSplit($str, $splitLength = 1)

{

$splitLength = (int)$splitLength;

if (self::isAscii($str)) {

return str\_split($str, $splitLength);

}

if ($splitLength < 1) {

return false;

}

if (mb\_strlen($str) <= $splitLength) {

return [$str];

}

preg\_match\_all('/.{' . $splitLength . '}|[^\x00]{1,' . $splitLength . '}$/us', $str, $matches);

return $matches[0];

}

/\*\*

\* @param string $str

\*

\* @return bool

\*/

public static function isAscii($str): bool

{

return is\_string($str) && !preg\_match('/[^\x00-\x7F]/S', $str);

}

/\*\*

\* Get class name without suffix. eg: HomeController -> home

\*

\* @param string $class full class name, with namespace.

\* @param string $suffix class suffix

\*

\* @return string

\*/

public static function getClassName(string $class, string $suffix): string

{

if (empty($suffix)) {

return $class;

}

// \\(\w+)Helper$

if (strpos($class, $suffix) > 0) {

$regex = '/\\\(\w+)' . $suffix . '$/';

$ok = preg\_match($regex, $class, $match);

} else {

$ok = true;

$match = [1 => $class];

}

return $ok ? lcfirst($match[1]) : '';

}

/\*\*

\* @param string $path

\*

\* @return string

\*/

public static function formatPath(string $path): string

{

$path = '/' . trim($path, '/ ');

return $path ?: '/';

}

/\*\*

\* @param string $path

\*

\* @return string

\*/

public static function rmPharPrefix(string $path): string

{

if (0 === strpos($path, 'phar://')) {

return preg\_replace('/[\w-]+\.phar\//', '', substr($path, 7));

}

return $path;

}

/\*\*

\* @param array|string $queryData

\*

\* @return string

\*/

public static function buildQuery($queryData): string

{

// is string

if (is\_string($queryData)) {

return $queryData;

}

// array: k-v map

return preg\_replace('/%5B(?:\d|[1-9]\d+)%5D=/', '=', http\_build\_query($queryData));

}

}

?>

--

10、系统函数

系统函数用于运行命令及获取系统相关信息等操作，需引入 Swoft\Stdlib\Helper\Sys 或 Swoft\Stdlib\Helper\SystemHelper。详情参阅：SystemHelper.php ( https://github.com/swoft-cloud/swoft-stdlib/blob/master/src/Helper/SystemHelper.php )文件。

<?php declare(strict\_types=1);

namespace Swoft\Stdlib\Helper;

use RuntimeException;

use function chdir;

use function exec;

use function fclose;

use function function\_exists;

use function getcwd;

use function getmyuid;

use function implode;

use function is\_resource;

use function ob\_start;

use function preg\_match;

use function preg\_replace;

use function proc\_close;

use function proc\_open;

use function shell\_exec;

use function stream\_get\_contents;

use function sys\_get\_temp\_dir;

use function system;

use function trim;

/\*\*

\* Class SystemHelper

\*

\* @since 2.0

\*/

class SystemHelper extends EnvHelper

{

/\*\*

\* Set cli process title

\*

\* @param string $title

\*

\* @return bool

\*/

public static function setProcessTitle(string $title): bool

{

if (self::isMac() || self::isCygwin()) {

return false;

}

if (function\_exists('cli\_set\_process\_title')) {

return @cli\_set\_process\_title($title);

}

return true;

}

/\*\*

\* Run a command. It is support windows

\*

\* @param string $command

\* @param string|null $cwd

\*

\* @return array [$code, $output, $error]

\* @throws RuntimeException

\*/

public static function run(string $command, string $cwd = null): array

{

$descriptors = [

0 => ['pipe', 'r'], // stdin - read channel

1 => ['pipe', 'w'], // stdout - write channel

2 => ['pipe', 'w'], // stdout - error channel

3 => ['pipe', 'r'], // stdin - This is the pipe we can feed the password into

];

$pipes = [];

$process = proc\_open($command, $descriptors, $pipes, $cwd);

if (!is\_resource($process)) {

throw new RuntimeException("Can't open resource with proc\_open.");

}

// Nothing to push to input.

fclose($pipes[0]);

$output = stream\_get\_contents($pipes[1]);

fclose($pipes[1]);

$error = stream\_get\_contents($pipes[2]);

fclose($pipes[2]);

// TODO: Write passphrase in pipes[3].

fclose($pipes[3]);

// Close all pipes before proc\_close! $code === 0 is success.

$code = proc\_close($process);

return [$code, $output, $error];

}

/\*\*

\* Method to execute a command in the sys

\* Uses :

\* - system

\* - exec

\* - shell\_exec

\*

\* @param string $command

\* @param bool $returnStatus

\* @param string|null $cwd

\*

\* @return array|string

\*/

public static function execute(string $command, bool $returnStatus = true, string $cwd = null)

{

$exitStatus = 1;

if ($cwd) {

chdir($cwd);

}

// system

if (function\_exists('system')) {

ob\_start();

system($command, $exitStatus);

$output = ob\_get\_clean();

//exec

} elseif (function\_exists('exec')) {

exec($command, $output, $exitStatus);

$output = implode("\n", $output);

//shell\_exec

} elseif (function\_exists('shell\_exec')) {

$output = shell\_exec($command);

} else {

$output = 'Command execution not possible on this system';

$exitStatus = 0;

}

if ($returnStatus) {

return [

'output' => trim($output),

'status' => $exitStatus

];

}

return trim($output);

}

/\*\*

\* Get unix user of current process.

\*

\* @return array

\*/

public static function getCurrentUser(): array

{

if (function\_exists('posix\_getpwuid')) {

/\*\* @noinspection PhpComposerExtensionStubsInspection \*/

return posix\_getpwuid(getmyuid());

}

return [];

}

/\*\*

\* @return string

\*/

public static function tempDir(): string

{

return self::getTempDir();

}

/\*\*

\* @return string

\*/

public static function getTempDir(): string

{

// @codeCoverageIgnoreStart

if (function\_exists('sys\_get\_temp\_dir')) {

$tmp = sys\_get\_temp\_dir();

} elseif (!empty($\_SERVER['TMP'])) {

$tmp = $\_SERVER['TMP'];

} elseif (!empty($\_SERVER['TEMP'])) {

$tmp = $\_SERVER['TEMP'];

} elseif (!empty($\_SERVER['TMPDIR'])) {

$tmp = $\_SERVER['TMPDIR'];

} else {

$tmp = getcwd();

}

// @codeCoverageIgnoreEnd

return $tmp;

}

/\*\*

\* get bash is available

\*

\* @return bool

\*/

public static function shIsAvailable(): bool

{

// $checkCmd = "/usr/bin/env bash -c 'echo OK'";

// $shell = 'echo $0';

$checkCmd = "sh -c 'echo OK'";

return self::execute($checkCmd, false) === 'OK';

}

/\*\*

\* get bash is available

\*

\* @return bool

\*/

public static function bashIsAvailable(): bool

{

// $checkCmd = "/usr/bin/env bash -c 'echo OK'";

// $shell = 'echo $0';

$checkCmd = "bash -c 'echo OK'";

return self::execute($checkCmd, false) === 'OK';

}

/\*\*

\* get screen size of the terminal

\*

\* ```php

\* list($width, $height) = Sys::getScreenSize();

\* ```

\*

\* @from Yii2

\*

\* @param boolean $refresh whether to force checking and not re-use cached size value.

\* This is useful to detect changing window size while the application is running but may

\* not get up to date values on every terminal.

\*

\* @return array|boolean An array of ($width, $height) or false when it was not able to determine size.

\*/

public static function getScreenSize(bool $refresh = false)

{

static $size;

if ($size !== null && !$refresh) {

return $size;

}

if (self::shIsAvailable()) {

// try stty if available

$stty = [];

if (exec('stty -a 2>&1', $stty)) {

$sttyText = implode(' ', $stty);

// linux: speed 38400 baud; rows 97; columns 362; line = 0;

$pattern = '/rows\s+(\d+);\s\*columns\s+(\d+);/mi';

// mac: speed 9600 baud; 97 rows; 362 columns;

if (self::isMac()) {

$pattern = '/(\d+)\s+rows;\s\*(\d+)\s+columns;/mi';

}

if (preg\_match($pattern, $sttyText, $matches)) {

return ($size = [$matches[2], $matches[1]]);

}

}

// fallback to tput, which may not be updated on terminal resize

if (($width = (int)exec('tput cols 2>&1')) > 0 && ($height = (int)exec('tput lines 2>&1')) > 0) {

return ($size = [$width, $height]);

}

// fallback to ENV variables, which may not be updated on terminal resize

if (($width = (int)getenv('COLUMNS')) > 0 && ($height = (int)getenv('LINES')) > 0) {

return ($size = [$width, $height]);

}

}

if (self::isWindows()) {

$output = [];

exec('mode con', $output);

if (isset($output[1]) && strpos($output[1], 'CON') !== false) {

return ($size = [

(int)preg\_replace('~\D~', '', $output[3]),

(int)preg\_replace('~\D~', '', $output[4])

]);

}

}

return ($size = false);

}

}

?>

--

11、XML 函数

XML 函数用于编解码 XML 数据、与数组互转、解析等操作，需引入 Swoft\Stdlib\Helper\XmlHelper。详情参阅 XmlHelper.php ( https://github.com/swoft-cloud/swoft-stdlib/blob/master/src/Helper/XmlHelper.php )文件。

decode(string $xml): array

{

return self::xmlToArray($xml);

}

/\*\*

\* @param array $data

\*

\* @return string

\*/

public static function encode(array $data): string

{

$xml = '<xml>';

$xml .= self::arrayToXml($data);

$xml .= '</xml>';

return $xml;

}

/\*\*

\* @param string $xml

\*

\* @return array

\*/

public static function xmlToArray(string $xml): array

{

$string = simplexml\_load\_string($xml, 'SimpleXMLElement', LIBXML\_NOCDATA | LIBXML\_NOBLANKS);

$jsonStr = JsonHelper::encode($string);

$data = JsonHelper::decode($jsonStr, true);

if ($data === false) {

return [];

}

return $data;

}

/\*\*

\* @param $data

\*

\* @return array

\*/

protected static function parseToArray($data): array

{

$res = null;

if (is\_object($data)) {

$data = (array)$data;

}

if (is\_array($data)) {

foreach ($data as $key => $val) {

if (is\_iterable($val)) {

$res[$key] = self::parseToArray($val);

} else {

$res[$key] = $val;

}

}

}

return $res;

}

/\*\*

\* @param array $data

\*

\* @return string

\*/

public static function arrayToXml(array $data): string

{

$xml = '';

if (!empty($data)) {

foreach ($data as $key => $val) {

$xml .= "<$key>";

if (is\_iterable($val)) {

$xml .= self::arrayToXml($val);

} elseif (is\_numeric($val)) {

$xml .= $val;

} else {

$xml .= self::characterDataReplace($val);

}

$xml .= "</$key>";

}

}

return $xml;

}

/\*\*

\* @param string $string

\*

\* @return string

\*/

protected static function characterDataReplace(string $string): string

{

return sprintf('<![CDATA[%s]]>', $string);

}

}

?>

--

12、通用函数

(1)、alias

获取路径别名。

function alias(string $key): string

(2)、bean

从容器中获取 bean 对象，等同于 Swoft::getBean(); 。

function bean(string $key): object

(3)、config

获取应用配置值。

function config(string $key, $default = null): mixed

(4)、container

获取容器对象。

function container(): Container

(5)、context

获取上下文对象。

function context(): ContextInterface

(6)、env

获取环境变量值。

function env(string $key = null, $default = null): mixed

(7)、event

获取事件管理器。

function event(): EventManager

(8)、fnmatch

文件名正则匹配。

function fnmatch(string $pattern, string $string): bool

(9)、printr

打印数据，类似 print\_r，允许多参数。

function printr(...$vars): void

(10)、server

获取服务实例，包括 HTTP Server 和 WebSocket Server。

function server(): Server

(11)、sgo

开启新协程。

function sgo(callable $callable, bool $wait = true): void

(12)、srun

启动协程并等待执行结束。

function srun(callable $callable): bool

(14)、tap

向指定值调用闭包函数，然后返回该值。

function tap($value, Closure $callback = null): mixed

(15)、validate

数据验证。

function validate(array $data, string $validatorName, array $fields = [], array $userValidators = []): array

(16)、value

获取回调结果。

function value($value): mixed

(17)、vdump

转储数据，与 var\_dump 类似。不同处在于 vdump 会增加一行打印位置提示。

--

--

七、错误处理

https://swoft.org/documents/v2/basic-components/error-dispose/

--

GitHub: https://github.com/swoft-cloud/swoft-error](https://github.com/swoft-cloud/swoft-error)

--

1、简介

Swoft 拥有完善的异常错误处理机制，与 FPM 模式有所不同。Swoft 根据不同的场景类型将错误进行了区分。因此你无需关心在不同场景下的错误如何处理，只需实现对应场景的错误处理逻辑即可。Swoft 会自动调度对应的错误处理器。

--

2、安装

默认情况下错误处理组件已包含在 Swoft 框架中，如需单独安装只需执行下方命令：

composer require swoft/error

--

3、场景

场景类型的划分主要根据 Swoole 的回调事件范围来划分。

比如，DB 中抛出一个 DbException，在 HTTP 服务运行场景里你需对异常处理后返回一个 Reponse 对象即可。而在 CLI 等环境里，你可以只对异常做出相应处理，无需返回任何结果。

为何要进行划分？在不同的场景里，即使是同一个地方抛出的异常错误，处理方式也可能是不同的。

--

4、场景类型

Swoft 在 Swoft\Error\ErrorType 中定义了场景类型，参考：ErrorType.php 文件( https://github.com/swoft-cloud/swoft-error/blob/master/src/ErrorType.php )。

<?php declare(strict\_types=1);

namespace Swoft\Error;

/\*\*

\* Class ErrorType

\*

\* @since 2.0

\*/

final class ErrorType

{

// Console application

public const CLI = 2;

public const RPC = 3;

public const UDP = 4;

public const SOCK = 7;

public const TASK = 8;

public const WORKER = 9;

// HTTP server

public const HTTP = 16;

// WebSocket server

public const WS\_HS = 21;

public const WS\_OPN = 22;

public const WS\_MSG = 23;

public const WS\_CLS = 24;

// Tcp server

public const TCP\_CNT = 31;

public const TCP\_RCV = 32;

public const TCP\_CLS = 33;

public const SYS = 85;

// Default error type

public const DEF = 90;

public const DEFAULT = 90;

}

?>

现支持的场景：控制台应用、HTTP 服务应用、RPC 服务应用、TCP 服务应用、WebSocket 服务应用。

--

5、异常处理器

(1)、注解

我们通过 @ExceptionHandler 将类指定为异常处理器。相关属性：exceptions：定义需进行处理的异常类，需提供完整类名或在顶部 use。

(2)、匹配逻辑

在同一个场景里，可以定义多个处理器来处理不同的异常：异常发生时，首先会使用异常类完整匹配进行处理器查找，匹配成功则由处理器处理；完整匹配失败，检查异常类是否为已注册异常类的子类，如果是则选择第一个匹配的处理器进行处理；仍然匹配失败，交由系统默认处理。

--

6、HTTP Request 异常

定义异常处理器类时需继承 Swoft\Http\Server\Exception\Handler\AbstractHttpErrorHandler。处理异常后必须返回 Swoft\Http\Message\Response 对象作为 HTTP 客户端响应。

(1)、使用示例

参考 swoft/swoft 项目内的 HttpExceptionHandler.php 文件( https://github.com/swoft-cloud/swoft/blob/master/app/Exception/Handler/HttpExceptionHandler.php )。

<?php declare(strict\_types=1);

/\*\*

\* This file is part of Swoft.

\*

\* @link https://swoft.org

\* @document https://swoft.org/docs

\* @contact group@swoft.org

\* @license https://github.com/swoft-cloud/swoft/blob/master/LICENSE

\*/

namespace App\Exception\Handler;

use Swoft\Error\Annotation\Mapping\ExceptionHandler;

use Swoft\Http\Message\Response;

use Swoft\Http\Server\Exception\Handler\AbstractHttpErrorHandler;

use Swoft\Log\Helper\CLog;

use Swoft\Log\Helper\Log;

use Throwable;

use function get\_class;

use function sprintf;

use const APP\_DEBUG;

/\*\*

\* Class HttpExceptionHandler

\*

\* @ExceptionHandler(\Throwable::class)

\*/

class HttpExceptionHandler extends AbstractHttpErrorHandler

{

/\*\*

\* @param Throwable $e

\* @param Response $response

\*

\* @return Response

\*/

public function handle(Throwable $e, Response $response): Response

{

// Log error message

Log::error($e->getMessage());

CLog::error('%s. (At %s line %d)', $e->getMessage(), $e->getFile(), $e->getLine());

// Debug is false

if (!APP\_DEBUG) {

return $response->withStatus(500)->withContent($e->getMessage());

}

$data = [

'code' => $e->getCode(),

'error' => sprintf('(%s) %s', get\_class($e), $e->getMessage()),

'file' => sprintf('At %s line %d', $e->getFile(), $e->getLine()),

'trace' => $e->getTraceAsString(),

];

// Debug is true

return $response->withData($data);

}

}

?>

--

7、RPC 异常

定义异常处理器类时需继承 Swoft\Rpc\Server\Exception\Handler\AbstractRpcServerErrorHandler。

处理异常后必须返回 Swoft\Rpc\Server\Response 对象作为 RPC 客户端响应。

(1)、使用示例

参考 swoft/swoft 项目内的 RpcExceptionHandler.php 文件( https://github.com/swoft-cloud/swoft/blob/master/app/Exception/Handler/RpcExceptionHandler.php )。

<?php declare(strict\_types=1);

/\*\*

\* This file is part of Swoft.

\*

\* @link https://swoft.org

\* @document https://swoft.org/docs

\* @contact group@swoft.org

\* @license https://github.com/swoft-cloud/swoft/blob/master/LICENSE

\*/

namespace App\Exception\Handler;

use Swoft\Error\Annotation\Mapping\ExceptionHandler;

use Swoft\Log\Debug;

use Swoft\Rpc\Error;

use Swoft\Rpc\Server\Exception\Handler\RpcErrorHandler;

use Swoft\Rpc\Server\Response;

use Throwable;

/\*\*

\* Class RpcExceptionHandler

\*

\* @since 2.0

\*

\* @ExceptionHandler(\Throwable::class)

\*/

class RpcExceptionHandler extends RpcErrorHandler

{

/\*\*

\* @param Throwable $e

\* @param Response $response

\*

\* @return Response

\*/

public function handle(Throwable $e, Response $response): Response

{

// Debug is false

if (!APP\_DEBUG) {

// just show error message

$error = Error::new($e->getCode(), $e->getMessage(), null);

} else {

$message = sprintf(' %s At %s line %d', $e->getMessage(), $e->getFile(), $e->getLine());

$error = Error::new($e->getCode(), $message, null);

}

Debug::log('Rpc server error(%s)', $e->getMessage());

$response->setError($error);

// Debug is true

return $response;

}

}

?>

--

8、WebSocket 异常

WebSocket 异常处理分为握手（Handshake）及消息（Message），分别需继承 Swoft\WebSocket\Server\Exception\Handler\AbstractHandshakeErrorHandler 和 Swoft\WebSocket\Server\Exception\Handler\AbstractMessageErrorHandle。

(1)、使用示例

参考 swoft/swoft 项目内的 WsHandshakeExceptionHandler.php ( https://github.com/swoft-cloud/swoft/blob/master/app/Exception/Handler/WsHandshakeExceptionHandler.php ) 及 WsMessageExceptionHandler.php 文件( https://github.com/swoft-cloud/swoft/blob/master/app/Exception/Handler/WsMessageExceptionHandler.php )。

//

//// WsHandshakeExceptionHandler.php

<?php declare(strict\_types=1);

/\*\*

\* This file is part of Swoft.

\*

\* @link https://swoft.org

\* @document https://swoft.org/docs

\* @contact group@swoft.org

\* @license https://github.com/swoft-cloud/swoft/blob/master/LICENSE

\*/

namespace App\Exception\Handler;

use Swoft\Error\Annotation\Mapping\ExceptionHandler;

use Swoft\Http\Message\Response;

use Swoft\WebSocket\Server\Exception\Handler\AbstractHandshakeErrorHandler;

use Throwable;

use function get\_class;

use function sprintf;

use const APP\_DEBUG;

/\*\*

\* Class HttpExceptionHandler

\*

\* @ExceptionHandler(\Throwable::class)

\*/

class WsHandshakeExceptionHandler extends AbstractHandshakeErrorHandler

{

/\*\*

\* @param Throwable $e

\* @param Response $response

\*

\* @return Response

\*/

public function handle(Throwable $e, Response $response): Response

{

// Debug is false

if (!APP\_DEBUG) {

return $response->withStatus(500)->withContent(sprintf(

'%s At %s line %d',

$e->getMessage(),

$e->getFile(),

$e->getLine()

));

}

$data = [

'code' => $e->getCode(),

'error' => sprintf('(%s) %s', get\_class($e), $e->getMessage()),

'file' => sprintf('At %s line %d', $e->getFile(), $e->getLine()),

'trace' => $e->getTraceAsString(),

];

// Debug is true

return $response->withData($data);

}

}

?>

//

//// WsMessageExceptionHandler.php

<?php declare(strict\_types=1);

/\*\*

\* This file is part of Swoft.

\*

\* @link https://swoft.org

\* @document https://swoft.org/docs

\* @contact group@swoft.org

\* @license https://github.com/swoft-cloud/swoft/blob/master/LICENSE

\*/

namespace App\Exception\Handler;

use Swoft\Error\Annotation\Mapping\ExceptionHandler;

use Swoft\Log\Helper\Log;

use Swoft\WebSocket\Server\Exception\Handler\AbstractMessageErrorHandler;

use Swoole\WebSocket\Frame;

use Throwable;

use function server;

use const APP\_DEBUG;

/\*\*

\* Class WsMessageExceptionHandler

\*

\* @since 2.0

\*

\* @ExceptionHandler(\Throwable::class)

\*/

class WsMessageExceptionHandler extends AbstractMessageErrorHandler

{

/\*\*

\* @param Throwable $e

\* @param Frame $frame

\*/

public function handle(Throwable $e, Frame $frame): void

{

$message = sprintf('%s At %s line %d', $e->getMessage(), $e->getFile(), $e->getLine());

Log::error('Ws server error(%s)', $message);

// Debug is false

if (!APP\_DEBUG) {

server()->push($frame->fd, $e->getMessage());

return;

}

server()->push($frame->fd, $message);

}

}

?>

--

9、使用说明

//// 示例代码

class BusinessLogic

{

public function doSomething()

{

throw new BusinessException("Error Processing Request", 500);

}

}

抛出异常时：在 HTTP Request 场景下，将由 HttpExceptionHandler 处理；在 RPC Server 场景下，将由 RpcExceptionHandler 处理。

上方处理器通过查找父类得到。我们也可以指定针对 BusinessException 异常的处理器，然后使用注解 @ExceptionHandler(BusinessException::class)。

通过示例我们可以看到，即使在同一地方抛出的异常，只要定义了不同场景的异常处理器就可以分别针对不同场景的请求（如上面的 HTTP 和 RPC）而作出不同的响应，无需进行额外的检查与判断。

--

10、参与贡献

欢迎参与贡献，您可以：Fork 我们的组件仓库，swoft/component；修改代码然后发送 PR；阅读 提交代码 的注意事项。

--

--

第六卷 核心组件

--

一、命令行

https://swoft.org/documents/v2/core-components/cli/

swoft 提供了功能强大的命令行应用处理功能，swoft 的 http server、websocket server 等都是通过命令行启动和管理的。

--

1、安装

(1)、Composer 安装

composer require swoft/console

(2)、Git 仓库

Github https://github.com/swoft-cloud/swoft-console

--

2、参与贡献

欢迎参与贡献，您可以：fork 我们的开发仓库 swoft/component( https://github.com/swoft-cloud/swoft-component )；修改代码然后发起 PR；阅读 提交代码( https://swoft.org/documents/v2/contribute/sub-codes/ ) 的注意事项。

--

3、功能特性

命令行的解析运行是基于注解 @Command @CommandMapping 自动收集注册；

支持给命令、命令组设置别名，一个命令可以有多个别名；

功能全面的命令行的选项参数解析(命名参数，短选项，长选项 …)，命令行下 input, output 管理、使用；

命令方法注释自动解析为帮助信息（支持 @CommandOption @CommandArgument @example 等注解）；

支持输出多种颜色风格的消息文本(info, comment, success, warning, danger, error 等)；

常用的特殊格式信息显示(section, panel, padding, helpPanel, table, title, list, multiList)；

丰富的动态信息显示(pending/loading, pointing, spinner, counterTxt, dynamicText, progressTxt, progressBar)；

常用的用户信息交互支持(select, multiSelect, confirm, ask/question, askPassword/askHiddenInput)；

颜色输出是 windows linux mac 兼容的，不支持颜色的环境会自动去除相关CODE；

支持 协程 和 非协程（传统同步阻塞）两种方式运行命令；

内置 Phar 打包工具类，可以方便的将应用打包成 phar 文件，方便作为工具分发和使用；

快速的为当前应用生成 bash/zsh 环境下的自动补全脚本(TODO)。

swoft/console 基于 inhere/php-console( https://github.com/inhere/php-console ) 改进，并参考了 symfony/console( https://github.com/symfony/console ) 部分特性

--

4、定义命令

一个命令由命令组和执行命令组成，一个类就是一个命令组，类里面的方法对应操作命令，一个命令的运行，是通过执行命令组对应的操作命令。命令逻辑里面可以使用 Swoft 所有功能。

(1)、命令结构

value of option: opt1

option: opt1 |

| |

php bin/swoft group:cmd john male 43 --opt1 value1 -y

| | | | | |

script command | | |\_\_\_\_\_ option: yes, it use shortcat: y, and it is a bool, so no value.

| |\_\_\_ |

argument: name | argument: age

argument: sex

(2)、参数与选项

不以 - 开头的都认为是参数；

以 - 开头的则是选项数据，-- 开头的是长选项(long-option)；一个 - 开头的是短选项(short-option)；

提示：支持混合式选项的赋值 --id=154 和 --id 154 是等效的。

注意: 输入如下的字符串将会认为是布尔值，on|yes|true – true，off|no|false – false。

--

5、注解

命令的定义主要通过 @Command 和 @CommandMapping 两个注解。 @Command 定义命令组名称，@CommandMapping 定义操作命令的映射关系。命令的命令使用帮助信息，也是通过注解完成定义。

(1)、@Command

@Command 定义命令组，标记一个类为 console 命令类。作用域：class。

//// 拥有属性

name string 定义命令组名称，如果缺省，根据类名称自动解析。

alias string 命令组别名，通过别名仍然可以访问它，允许多个，以逗号隔开即可。

desc string 命令组描述信息说明，支持颜色标签。

coroutine bool 定义是否为协程下运行，默认 true, 框架会启动一个协程运行此组里面的命令。

//

Tips: 若 desc 为空，将会自动解析类的第一行注释作为命令组描述。

(2)、@CommandMapping

@CommandMapping 定义操作命令映射关系，标明了一个具体的命令。作用域：method。

//// 拥有属性

name string 定义命令组名称，如果缺省，会执行使用方法名称。

alias string 命令别名，通过别名仍然可以访问它，允许多个，以逗号隔开即可。

desc string 命令的描述信息说明，支持颜色标签。

//

Tips: 若 desc 为空，将会自动解析类的第一行注释作为描述。

(3)、@CommandOption

@CommandOption 定义一个命令的选项。作用域：method|class。

//// 拥有属性

name string 必填项 定义命令选项名称 eg: opt。

short string 定义命令选项名称的短选项。

default mixed 命令选项的默认值。

desc string 命令选项的描述信息说明，支持颜色标签。

type string 命令选项的值类型。

mode int 命令选项的值输入限定：Command::OPT\_BOOLEAN (int 1 可选)，Command::OPT\_REQUIRED (int 2 必须)。

//

Tips: 特别提示 @CommandOption 可以用在 @Command 类注解上面，这样子相当于给里面所有的命令都加了公共选项。

(4)、@CommandArgument

@CommandArgument 定义一个命令的参数。作用域：method。

//// 拥有属性

name string 必填项 定义命令参数名称，eg: opt。

default mixed 命令参数的默认值。

desc string 命令参数的描述信息说明，支持颜色标签。

type string 命令参数的值类型。

mode int 命令选项的值输入限定：Command::OPT\_BOOLEAN (int 1 可选)，Command::OPT\_REQUIRED (int 2 必须)。

//

Tips: 命令参数是根据输入位置(有顺序的)来获取的，name 是代码里给这个位置的参数添加的命名。

(5)、example 注释

@example 注释会被特殊处理(不是注解)，如果你的命令方法上面有这个注释，它的内容也会被显示到命令帮助信息上面。

(6)、代码示例

/\*\*

\* Provide some commands for manage and watch swoft server project

\*

\* @Command()

\*/

class ServeCommand

{

/\*\*

\* Start the swoft server and monitor the file changes to restart the server

\*

\* @CommandMapping()

\* @CommandArgument("targetPath", type="path",

\* desc="Your swoft project path, default is current work directory"

\* )

\* @CommandOption("interval", type="integer", default=3,

\* desc="Interval time for watch files, unit is seconds"

\* )

\* @CommandOption(

\* "bin-file", short="b", type="string", default="bin/swoft",

\* desc="Entry file for the swoft project"

\* )

\* @CommandOption(

\* "start-cmd", short="c", type="string", default="http:start",

\* desc="the server startup command to be executed"

\* )

\* @CommandOption(

\* "watch", short="w", default="app,config", type="directories",

\* desc="List of directories you want to watch, relative the <cyan>targetPath</cyan>"

\* )

\* @example

\* {binFile} run -c ws:start -b bin/swoft /path/to/php/swoft

\* @param Input $input

\*/

public function run(Input $input): void

{

Show::aList([

'options' => $input->getOpts(),

'arguments' => $input->getArgs(),

]);

}

}

--

6、使用与运行

完成定义命令后，可以执行命令，处理对应业务逻辑。现在你可以执行 php bin/swoft, 命令列表中将会显示 serve 组命令；执行 php bin/swoft serve 或者 php bin/swoft serve -h 将会看到serve组里拥有的具体命令；执行 php bin/swoft serve:run -h 将会看到此命令的完整帮助信息。

如果不特殊设置，swoft 默认在 协程环境 运行命令。command 里是始终不可能直接操作 server，你每运行一次 command，都是在一个全新的进程里，除了使用一样的代码，其他毫无关系。

//

//// 运行 serve 下面的，某个 run 命令。

# 运行 http 命令组中的 start 操作

php bin/swoft serve:run -h

( 渲染效果图 https://swoft.org/img/cli-run-example.png )

(1)、更多命令

php bin/swoft [-h,–help] 查看当前已经定义的所有命令组

php bin/swoft -v | –version 查看当前 swoft 框架版本信息

php bin/swoft xxx [-h,–help] 查看 xxx 命令组帮助信息

php bin/swoft xxx:yyy -h |–help 查看 xxx 命令组下的 yyy 操作的帮助信息

php bin/swoft xxx:yyy 执行 xxx 命令组下的 yyy 操作

--

7、输入输出对象

输入对象 是 Swoft\Console\Input\Input 的实例，用于获取用户输入的命令参数选项等信息。命令逻辑里面，可以通过函数参数和全局函数获取输入输出对象。

输出对象 是 Swoft\Console\Output\Output 的实例，用于显示信息到控制台。命令逻辑里面，可以通过函数参数和全局函数获取输入输出对象。

(1)、获取输入输出对象

我们可以通过 方法参数注入 或 全局函数 两种方式来获取输入输出对象，使用方式如下。

//

//// 方法参数注入示例

/\*\*

\* Test command

\*

\* @Command(coroutine=true)

\*/

class TestCommand

{

/\*\*

\* 使用方法参数注入获取输入对象。

\* @param Input $input 输入对象

\* @param Output $output 输出对象

\*

\* @CommandMapping("test2")

\*/

public function test(Input $input, Output $output)

{

// ......

}

}

//

//// 全局函数示例

/\*\*

\* Test command

\*

\* @Command(coroutine=true)

\*/

class TestCommand

{

/\*\*

\* 使用全局方法获取输入对象。

\* @CommandMapping()

\*/

public function demo()

{

$input = \input(); //输入对象

$output = \output(); //输出对象

// ......

}

}

(2)、使用输入输出对象

通过前面 定义命令( https://swoft.org/documents/v2/core-components/cli/#%E5%AE%9A%E4%B9%89%E5%91%BD%E4%BB%A4 ) 这一部分的介绍，我们已经知道了命令参数、命令选项。现在，在终端中执行如下的命令，用于演示参数选项等信息的解析。

php bin/swoft demo:test status=2 name=john arg0 -s=test --page 23 --id=154 -e dev -v vvv -d -rf --debug --test=false

注意： 输入如下的字符串将会认为是布尔值。

on|yes|true = true

off|no|false = false

//

///1/ 获取基本信息

echo $input->getScriptFile(); // 'bin/swoft' 执行的入口脚本文件

echo $input->getCommand(); // 'demo:test' 命令名称 解析到的第一个参数将会被认为是命令名称，并且不会再存入到参数列表中

echo $input->getPwd(); // 当前工作目录

//

///2/ 命令参数信息

通常的参数如 arg0 只能根据 index key 来获取值。但是提供以等号(=)连接的方式来指定参数名(eg: status=2)

// 打印所有的参数信息

var\_dump($input->getArgs());

array(3) {

'status' => string(1) "2"

'name' => string(4) "john"

[0] => string(4) "arg0"

}

//

///3/ 获取命令参数值

// argument

$first = $input->getFirstArg(); // 'arg0'

$status = $input->getArg('status', 'default value'); // '2'

$status = $input->getInt('status'); // 2

// 获取一个必须的参数，若用户没有输入值，将会抛出错误信息

$id = $input->getRequiredArg('id');

//

///4/ 命令选项信息

// 获取解析后的选项信息

没有值的选项，将设置默认值为 bool(true)；

短选项不仅仅只是以一个 - 开头，而且名称只能是一个字符；

多个(默认值的)短选项可以合并到一起写，如 -rf 会被解析为两个短选项 'r' => bool(true) 'f' => bool(true)；

// 示例

var\_dump($input->getOpts());

// var\_dump($input->getLOpts()); // 只打印长选项信息

// var\_dump($input->getSOpts()); // 只打印短选项信息

// 输出

array(10) {

's' => string(4) "test"

'e' => string(3) "dev"

'v' => string(3) "vvv"

'd' => bool(true)

'r' => bool(true)

'f' => bool(true)

'page' => string(2) "23"

'id' => string(3) "154"

'debug' => bool(true)

'test' => bool(false)

}

//

///5/ 获取选项值

输入对象中提供了非常多的选项值获取方法，方便快速地获取需要的信息。

// option

$page = $input->getOpt('page') // '23'

$page = $input->getIntOpt('page') // 23

$debug = $input->getBoolOpt('debug') // True

$test = $input->getBoolOpt('test') // False

$d = $input->getBoolOpt('d') // True

// 获取到一个值就返回，对同一个含义的选项选项非常有用

$showHelp = $input->sameOpt(['h','help']);

// 获取一个必须的选项，若用户没有输入值，将会抛出错误信息

$id = $input->getRequiredOpt('id');

//

///6/ 读取用户输入

echo "Your name:";

$name = $input->read();

echo 'input is ' . $name; // 'inhere'

// 效果(in terminal)

Your name: inhere

input is inhere

--

8、数据展示

Console 数据展示 - 提供格式化信息的输出显示。

主要功能封装在命名空间 Swoft\Console\Advanced\Formatter 下，提供了 Swoft\Console\Helper\Show 辅助类来快速使用它们。

(1)、标题文本输出

使用 Show::title()/$output->title()。

public static function title(string $title, array $opts = [])

(2)、段落式文本输出

使用 Show::section()/$output->section()。

public static function section(string $title, string|array $body, array $opts = [])

(3)、列表数据展示输出

使用 Show::aList()/$output->aList()。

public static function aList(array $data, string $title, array $opts = [])

$data array 列表数据，可以是 key-value 形式，也可以只有 value，还可以两种混合。

$title string 列表标题，可选的。

$opts array 选项设置(同表格、面板的选项)：leftChar 左侧边框字符，默认两个空格，也可以是其他字符(eg: \* .)；keyStyle 当 key-value 形式时，渲染 key 的颜色风格，默认 info, 设为空即是不加颜色渲染；titleStyle 标题的颜色风格，默认 comment。

aList 的默认选项，可以渲染一个命令的帮助信息。

//

//// 示例

$title = 'list title';

$data = [

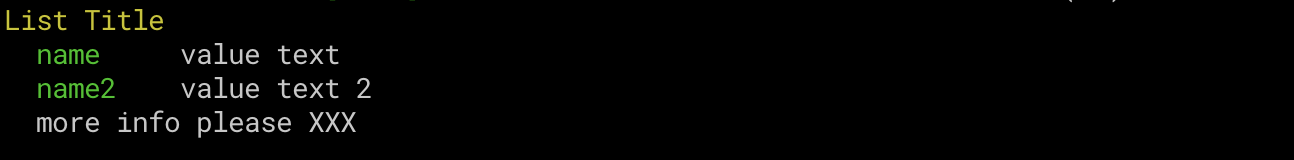
'name' => 'value text', // key-value

'name2' => 'value text 2',

'more info please XXX', // only value

];

Show::aList($data, $title);



( 渲染效果图 https://swoft.org/img/fmt-list.png )

(4)、多列表数据展示输出

使用 Show::mList()/$output->mList()，别名方法 Show::multiList()。

public static function mList(array $data, array $opts = [])

mList 的默认选项，可以渲染一组命令的帮助信息。效果与 helpPanel() 相同，并且自定义性更高。

$data = [

'list1 title' => [

'name' => 'value text',

'name2' => 'value text 2',

],

'list2 title' => [

'name' => 'value text',

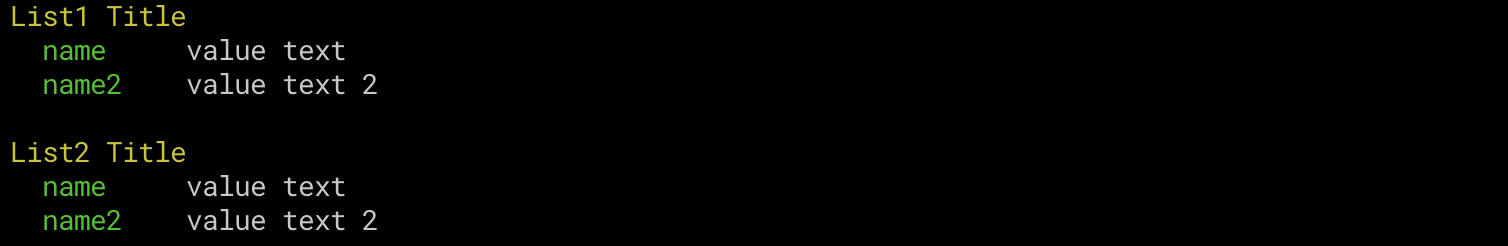
'name2' => 'value text 2',

],

// ... ...

];

Show::mList($data);



( 渲染效果 https://swoft.org/img/fmt-multi-list.png )

(5)、面板展示信息输出

使用 Show::panel()/$output->panel()。

public static function panel(mixed $data, string $title = 'Information Panel', array $opts = [])

展示信息面板。比如 命令行应用 开始运行时需要显示一些 版本信息、环境信息等等。

$data = [

'application version' => '1.2.0',

'system version' => '5.2.3',

'see help' => 'please use php bin/app -h',

'a only value message',

];

Show::panel($data, 'panel show', ['borderChar' => '#']);



( 渲染效果图 https://swoft.org/img/fmt-panel.png )

(6)、数据表格信息输出

使用 Show::table()/$output->table() 可直接渲染从数据库拉取的数据(会自动提取字段名作为表头)。

public static function table(array $data, $title = 'Data Table', array $opts = [])

/// like from database query's data.

$data = [

[ col1 => value1, col2 => value2, col3 => value3, ... ], // first row

[ col1 => value4, col2 => value5, col3 => value6, ... ], // second row

... ...

];

Show::table($data, 'a table');

//

/// 自己构造数据时，还要写字段名就有些麻烦了。可以通过选项配置 $opts 手动配置表头字段列表。

// use custom head

$data = [

[ value1, value2, value3, ... ], // first row

[ value4, value5, value6, ... ], // second row

// ... ...

];

$opts = [

'showBorder' => true,

'columns' => [col1, col2, col3, ...]

];

Show::table($data, 'a table', $opts);

//

/// 示例

$data = [

['1', 'john', '2', 'john@email.com'],

['2', 'tom', '0', 'tom@email.com'],

['3', 'jack', '1', 'jack@email.com'],

];

$opts = [

'showBorder' => true,

'columns' => ['id', 'name', 'status', 'email']

];

Show::table($data, 'Table Show', $opts);

$opts['showBorder'] = false;

Show::table($data, 'No Border Table Show', $opts);

$opts['bodyStyle'] = 'red';

$opts['showBorder'] = true;

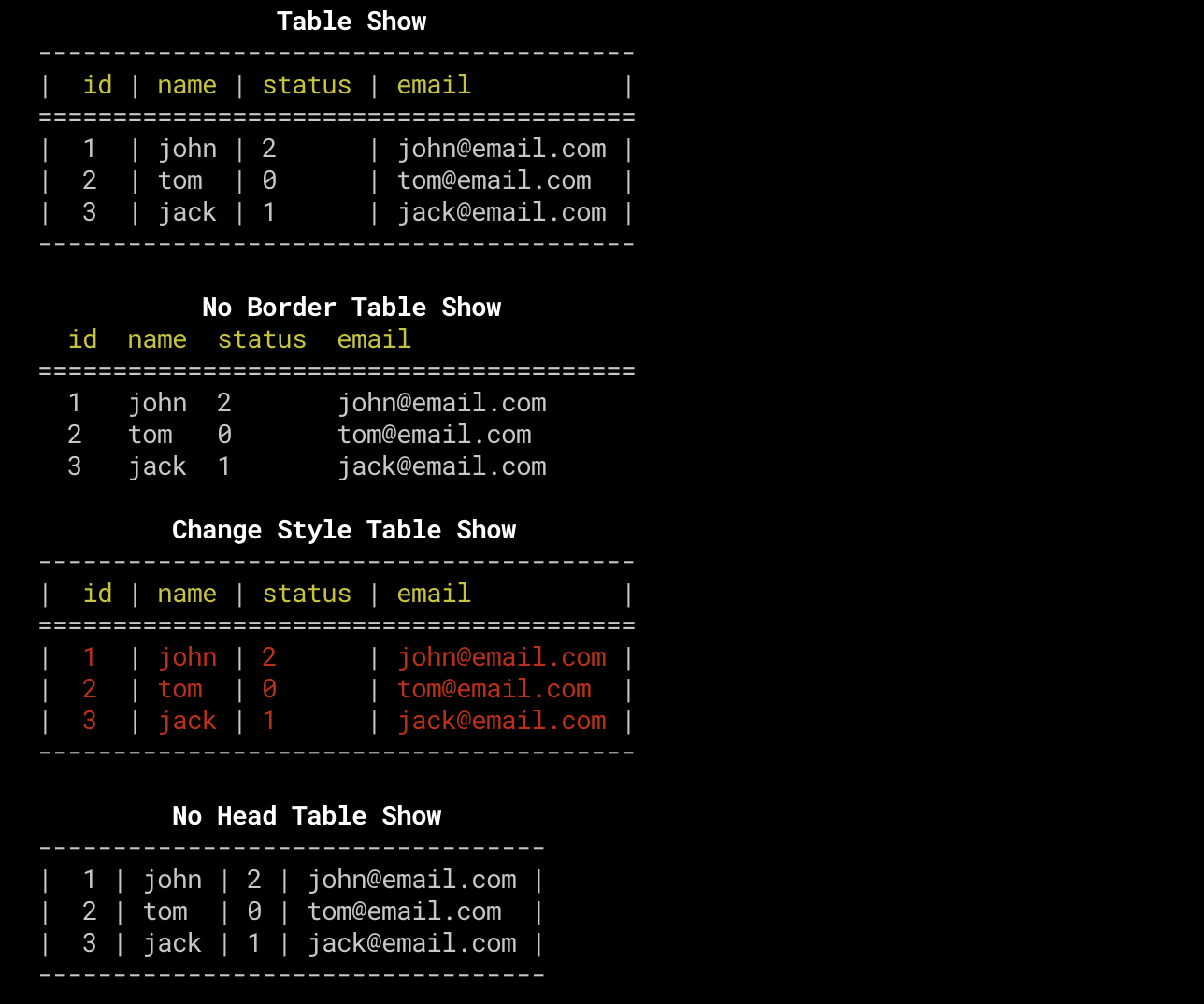
Show::table($data, 'Change Style Table Show', $opts);

$opts['bodyStyle'] = '';

$opts['showBorder'] = true;

$opts['columns'] = [];

Show::table($data, 'No Head Table Show', $opts);



( 渲染效果图 https://swoft.org/img/table-show.png )

(7)、渲染帮助信息面板

使用 Show::helpPanel()/$output->helpPanel()。

public static function helpPanel(array $config, $showAfterQuit = true)

Show::helpPanel([

"description" => 'a help panel description text. (help panel show)',

"usage" => 'a usage text',

"arguments" => [

'arg1' => 'arg1 description',

'arg2' => 'arg2 description',

],

"options" => [

'--opt1' => 'a long option',

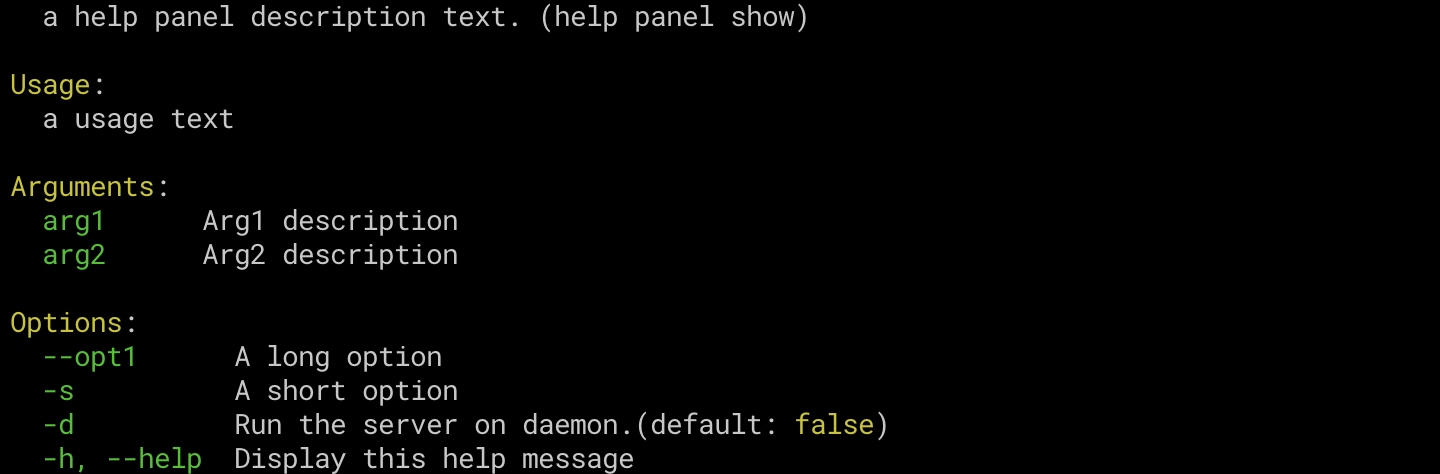
'-s' => 'a short option',

'-d' => 'Run the server on daemon.(default: <comment>false</comment>)',

'-h, --help' => 'Display this help message'

],

]);



( 渲染效果图 https://swoft.org/img/fmt-help-panel.png )

--

--

二、HTTP Server

https://swoft.org/documents/v2/core-components/http-server/

//

基于 \Swoole\Http\Server 实现的协程 HTTP 服务, 框架层做了很好的封装, 用户按照传统的 MVC 方式编写代码, 就能获得协程带来的超高性能。

--

1、安装

(1)、Composer 安装

composer require swoft/http-server

(2)、Git 仓库

Github https://github.com/swoft-cloud/swoft-http-server

--

2、参与贡献

欢迎参与贡献，您可以 fork 我们的开发仓库 swoft/component( https://github.com/swoft-cloud/swoft-component )；修改代码然后发起 PR；阅读 提交代码( https://swoft.org/documents/v2/contribute/sub-codes/ ) 的注意事项。

--

3、功能特色

基于 PSR-7 的 HTTP 消息实现；

基于 PSR-15 的中间件；

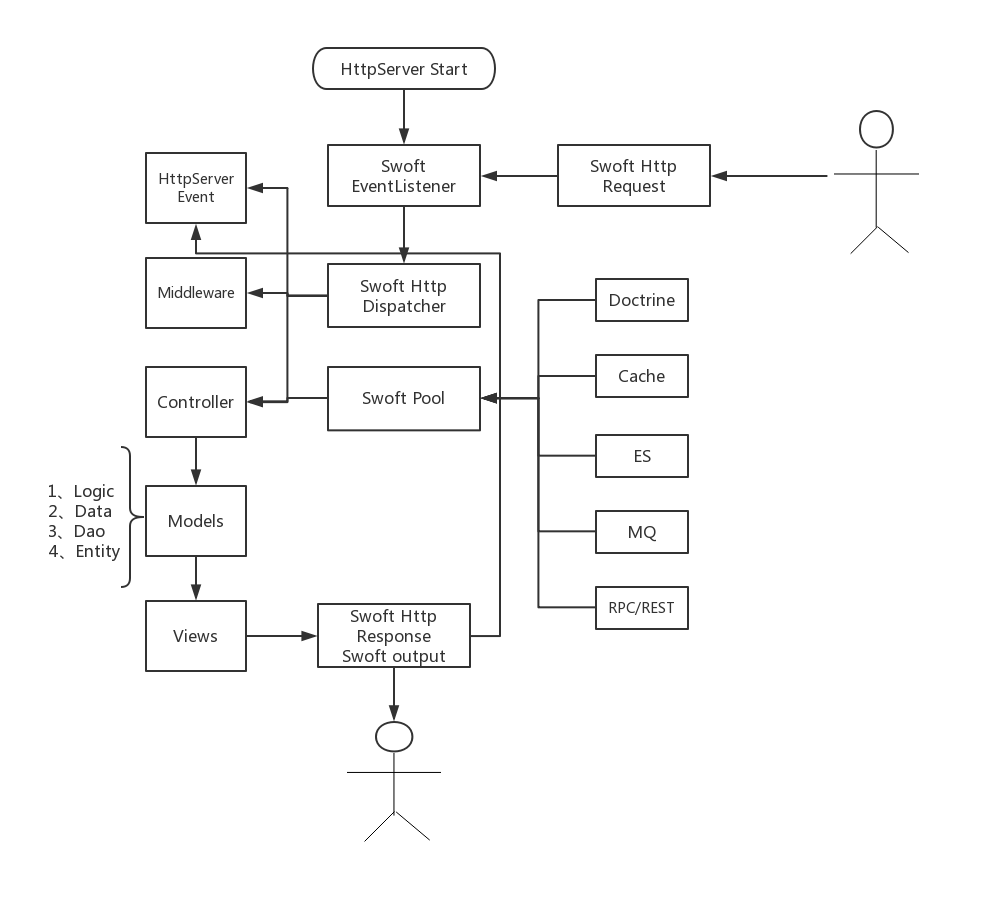
@Controller 灵活的控制器注解；

@RequestMapping 灵活的路由注解；

--

4、Http 生命周期

了解请求生命周期，有利于理解HTTP服务各组件，编写出更好代码。



( 生命周期图 https://swoft.org/img/http-life-cycle.png )

--

5、请求处理流程

( 流程图 https://swoft.org/img/http-request-flow.png )

--

6、Http Server 命令

$ php bin/swoft http

Provide some commands to manage the swoft HTTP server

//

Group: http (alias: httpsrv)

Usage:

bin/swoft http:COMMAND [--opt ...] [arg ...]

//

Global Options:

--debug Setting the application runtime debug level(0 - 4)

--no-color Disable color/ANSI for message output

-h, --help Display this help message

-V, --version Show application version information

//

Commands:

reload Reload worker processes

restart Restart the http server

start Start the http server

stop Stop the currently running server

//

Example:

bin/swoft http:start Start the http server

bin/swoft http:stop Stop the http server

//

View the specified command, please use: bin/swoft http:COMMAND -h

//

Http Server 的命令都在 Commands 中，reload 重新加载工作进程；restart 重新启动http服务器；start 启动http服务器；stop 停止当前正在运行的服务器。

(1)、使用

///1/ 前台运行

$ php bin/swoft http:start

Information Panel

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HTTP | Listen: 0.0.0.0:18306, type: TCP, mode: Process, worker: 1

\* rpc | Listen: 0.0.0.0:18307, type: TCP

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HTTP server start success !

//

///2/ 后台运行

$ php bin/swoft http:start -d

Information Panel

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HTTP | Listen: 0.0.0.0:18306, type: TCP, mode: Process, worker: 1

\* rpc | Listen: 0.0.0.0:18307, type: TCP

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HTTP server start success !

服务成功启动后访问 http://localhost:18306 即可访问 Http 服务

--

7、配置参数

在应用下的 app/bean.php 配置 server，在这个文件里，你可以看到 Http Server数组里面包含了 Http Server 的基本信息。下面列举了一些简单的配置，你也可以自由组合同时提供多种服务。

'httpServer' => [

'class' => HttpServer::class,

'port' => 18306,

'listener' => [

'rpc' => bean('rpcServer')

],

'process' => [

// 'monitor' => bean(MonitorProcess::class)

// 'crontab' => bean(CrontabProcess::class)

],

'on' => [

// Enable task must task and finish event

SwooleEvent::TASK => bean(TaskListener::class),

SwooleEvent::FINISH => bean(FinishListener::class)

],

/\* @see HttpServer::$setting \*/

'setting' => [

'task\_worker\_num' => 12,

'task\_enable\_coroutine' => true,

'worker\_num' => 6,

// Enable Https

'ssl\_cert\_file' => '/my/certs/2288803\_www.domain.com.pem',

'ssl\_key\_file' => '/my/certs/2288803\_www.domain.com.key',

]

],

//Enable Https

'type' => SWOOLE\_SOCK\_TCP | SWOOLE\_SSL,

//

可配置项用于 http server bean 配置，除了 class 其他都是 http server 的属性。

class 指定 Http Server 的处理类；

port 指定 Http Server 的端口；

listener 指定其他一同启动的服务，添加端口服务监听，可以多个，rpc 启动 RPC 服务；

process 启动自定义用户进程；

on 配置监听的事件，注册事件、设置对应事件的处理监听，事件触发组件调用，在任务里面使用；

setting 这里是参考 Swoole Server 配置选项( https://wiki.swoole.com/wiki/page/274.html )；

pidFile 设置进程 pid 文件 位置，默认值 @runtime/swoft.pid；

mode 运行的模式，参考 Swoole Server 构造函数 第三个参数( https://wiki.swoole.com/wiki/page/14.html )；

type 指定Socket的类型，支持TCP、UDP、TCP6、UDP6、UnixSocket Stream/Dgram 等 Swoole Server 构造函数 第四个参数( https://wiki.swoole.com/wiki/page/14.html )；

启用 Https 支持，注意：你必须安装 OpenSSL 库，并且确保安装 swoole 是启用了 ssl 选项的。同时，需要设置 'type' => SWOOLE\_SOCK\_TCP | SWOOLE\_SSL

--

8、Controller 控制器

控制器作为HTTP服务的核心组件，串接起一次请求的整个生命周期. 通过 注解 的方式，相较于传统的 Controller，代码更简洁，用户可以更关注业务逻辑。

(1)、创建控制器

主要通过 @Controller 注解实现。代码可以放置任意位置，不过为了统一标准，建议放在 app/Http/Controller 下。

// 可以通过 swoftcli 快速创建新的控制器

php swoftcli.phar gen:http-ctrl user --prefix /users

(2)、@Controller 注解

Http 控制器类注解 @Controller。

注解类：Swoft\Http\Server\Annotation\Mapping\Controller。

作用范围：CLASS。

拥有属性：prefix 指定路由前缀。

通常仅有 @Controller 是没有什么效果的，它需要配合接下来的 @RequestMapping 一起才能正确的工作。

(3)、路由规则

显式指定路由前缀：@Controller(prefix="/index") 或 @Controller("/index")。

隐式指定路由前缀：@Controller() 默认自动使用 小驼峰 格式解析 controller class 的名称，示例：class IndexController 对应路由 /index。

一个完整的路由规则是通过 @Controller + @RequestMapping 注解实现，通常前者定义前缀，后者定义后缀。关于 @RequestMapping 注解将在稍后 路由-@RequestMapping ( https://swoft.org/documents/v2/core-components/http-server/#requestmapping-%E6%B3%A8%E8%A7%A3 )章节将会详细介绍。

示例：根据下方的定义，对应的访问路由是 /v1/users/list (/v1/users + list)。

/\*\*

\* @Controller(prefix="/v1/users")

\*/

class UsersController

{

/\*\*

\* @RequestMapping(route="list")

\*/

public function list(){}

}

示例： 若 @Controller() 参数为空，则会使用隐式路由前缀绑定，例如下方的定义，对应的访问路由是 /user/list。

/\*\*

\* @Controller()

\*/

class UsersController

{

/\*\*

\* @RequestMapping(route="list")

\*/

public function list(){}

}

在 Swoft 里不要按照传统的 fpm 框架继承父类控制器的成员属性在其他控制器使用，这种做法是错误的。

//

//// 错误示范

/\*\*

\* @Controller()

\*/

class BaseController

{

protected $num;

}

/\*\*

\* @Controller(prefix="/v1/index")

\*/

class IndexController extends BaseController

{

/\*\*

\* @RequestMapping(route="index")

\*/

public function index()

{

$this->num++;

echo $this->num."\n";

}

}

--

9、路由

Swoft 与传统的 PHP 框架不一样，并没有采用配置文件的方式来配置路由，而采用了注解。在 Swoft 里我们可以使用 @RequestMapping 注解快速地添加路由。

(1)、路由配置

//// 这是默认的路由配置

// at file: vendor/swoft/http-server/src/AutoLoader.php

'httpRouter' => [

'name' => 'swoft-http-router',

// config

'ignoreLastSlash' => true,

'tmpCacheNumber' => 500,

// 'handleMethodNotAllowed' => false

],

//

/// 配置说明

ignoreLastSlash bool，默认true，是否忽略 URI path 最后的 /，如果设置为 false 不忽略， /home 与 /home/ 将是两个不同的路由。

tmpCacheNumber int，默认500，动态路由缓存数量，动态参数路由匹配后会缓存下来，下次相同的路由将会更快地匹配命中。

handleMethodNotAllowed bool 默认false，是否处理 MethodNotAllowed，为了加快匹配速度，默认 method 不匹配也是直接抛出 Route not found 错误，如有特殊需要可以开启此选项，开启后将会抛出 Method Not Allowed 错误。

//

// 若你需要自定义路由配置，直接在 app/bean.php 添加 httpRouter 项配置即可。

'httpRouter' => [

'handleMethodNotAllowed' => true

]

(2)、@RequestMapping 注解

Http 控制器类中方法路由注解 @RequestMapping，route 路由规则path；method 请求方式（GET、POST、PUT、PATCH、DELETE、OPTIONS、HEAD）；params 可以通过它为path变量添加正则匹配限制。每个方法上尽量只写一个 @RequestMapping 注解，以免出现紊乱。

//

//// 路由规则

通常情况，一个完整的路由 path 等于 @Controller 的 prefix + @RequestMapping 的 route，显示指定路由后缀 @RequestMapping("index") 或 @RequestMapping(route="index")；隐式指定路由后缀 使用 @RequestMapping() 默认解析方法名为后缀。

特殊的，当你的 @RequestMapping 上的路由以 / 开头时，那完整的路由就是它，即不会再将 prefix 添加到它的前面。

//

示例：在控制器方法中加入 @RequestMapping 注解。

/\*\*

\* @Controller()

\*/

class UserController

{

/\*\*

\* @RequestMapping()

\*/

public function index()

{}

/\*\*

\* @RequestMapping("index")

\*/

public function index()

{}

/\*\*

\* @RequestMapping(route="index")

\*/

public function index()

{}

}

代码执行后将会为 index 方法绑定路由为 /user/index，允许的请求方法为默认的 GET 和 POST。

//

///1/ 绑定路由 path 参数

指定路由参数 @RequestMapping(route="index/{name}")，Action 方法中可以直接使用 $name 作为方法参数。

当路由参数被 [] 包起来则 URL path 传递参数是可选的。注意，可选符只能用在最后面，示例1 @RequestMapping("/index[/{name}]") 这样 /index` /index/tom` 都可以访问到；示例2 @RequestMapping("/about[.html]") 相当于伪静态，/about` /about.html` 都可以访问到。

//

///2/ 设置路由请求方式

如果想要设置允许请求控制器的 HTTP 请求方式。可以使用方法在控制器中的 @RequestMapping 注解配置 method 参数，可以是 GET、POST、PUT、PATCH、DELETE、OPTIONS、HEAD 中的一个或多个。限定 HTTP 方法：@RequestMapping(method={RequestMethod::GET})。指定路由支持的 HTTP 方法，默认是支持 GET 和 POST。

请切记要引入相关的注解类，Swoft\Http\Server\Annotation\Mapping\RequestMapping，Swoft\Http\Server\Annotation\Mapping\RequestMethod。

//

///3/ 获取匹配结果

你可以在中间件或者 action 拿到路由匹配的结果信息。

[$status, $path, $route] = $request->getAttribute(Request::ROUTER\_ATTRIBUTE);

--

10、Http 请求对象

Swoft的请求与响应实现于 PSR-7 规范。请求与响应对象存在于每次 HTTP 请求。请求对象 Request 为 Swoft\Http\Message\Request，响应对象 Response 为 Swoft\Http\Message\Response。

PSR-7 接口为请求和响应对象提供了这些公共方法，withProtocolVersion($version)，withHeader($name, $value)，withAddedHeader($name, $value)，withoutHeader($name)，withBody(StreamInterface $body)。

PSR-7 接口为请求对象提供了这些方法，withMethod(string $method)，withUri(UriInterface $uri, $preserveHost = false)，withCookieParams(array $cookies)，withQueryParams(array $query)，withUploadedFiles(array $uploadedFiles)，withParsedBody($data)，withAttribute($name, $value)，withoutAttribute($name)。

更多请参考 PSR-7 和 查看 swoft/http-message ( https://github.com/swoft-cloud/swoft-http-message ) 中具体的实现类。

根据PSR-7对象的不可变性(immutable)，所有的 with\* 方法都是克隆对象然后返回，必须接收新对象来做进一步处理，或使用链式调用

(1)、获取请求对象

通过控制器方法参数注入 public function action(Request $request)。

通过请求上下文获取 Swoft\Context\Context::mustGet()->getRequest()。

// 示例：获取请求动作

$request = context()->getRequest();

$method = $request->getMethod();

//

示例：获取请求的URI

每个 HTTP 请求都有一个 URI 标识所请求的应用程序资源。HTTP 请求 URI 有几个部分：Scheme (e.g. http or https)，Host (e.g. example.com)，Port (e.g. 80 or 443)，Path (e.g. /users/1)，Query string (e.g. sort=created&dir=asc)。

你可以通过请求对象的 getUri() 方法获取 PSR-7 URI对象: $method = $request->getUri();

PSR-7 请求对象的 URI 本身就是一个对象，它提供了下列方法检查 HTTP 请求的 URL 部分，getScheme()，getAuthority()，getUserInfo()，getHost()，getPort()，getPath()，getQuery() (e.g. a=1&b=2)，getFragment()。

//

///1/ 示例：获取请求 Headers

// 全部的 Headers

$headers = $request->getHeaders();

foreach ($headers as $name => $values) {

echo $name . ": " . implode(", ", $values).PHP\_EOL;

}

//

// 指定的 Header

$headerValueArray = $request->getHeader('host');

print\_r($headerValueArray); // return Array

$host = $request->getHeaderLine("host");

print\_r($host); // return String

//

///2/ 示例：获取请求的数据

//

// GET 数据

$data = $request->query();

$some = $request->query('key', 'default value')

$data = $request->get();

$some = $request->get('key','default value');

// POST 数据

$data = $request->post();

$some = $request->post('key', 'default value');

无需关心请求的数据格式，json、xml请求都会自动解析为php的数组数据，都可以通过$request->post()获取。

// GET & POST 数据

$data = $request->input();

$some = $request->input('key', 'default value');

// RAW 数据

$data = $request->raw();

// SERVER 数据

$data = $request->getServerParams();

$some = $request->server('key', 'default value');

// 获取上传文件

$file = $request->getUploadedFiles();

获取的结果是一维数组或者二位数组，数据结构如下。若表单中上传的是单文件则返回的是一个一维数组，数组内容是 Swoft\Http\Message\Upload\UploadedFile 文件对象，例如文件字段名为 file 则数据结构如下。

array(1) {

["file"]=>

object(Swoft\Http\Message\Upload\UploadedFile)#6510 (7) {

["size":"Swoft\Http\Message\Upload\UploadedFile":private]=>

int(1319)

["errorCode":"Swoft\Http\Message\Upload\UploadedFile":private]=>

int(0)

["file":"Swoft\Http\Message\Upload\UploadedFile":private]=>

string(25) "/tmp/swoole.upfile.f7p2EL"

["clientFilename":"Swoft\Http\Message\Upload\UploadedFile":private]=>

string(6) "at.png"

["clientMediaType":"Swoft\Http\Message\Upload\UploadedFile":private]=>

string(9) "image/png"

["moved":"Swoft\Http\Message\Upload\UploadedFile":private]=>

NULL

["path":"Swoft\Http\Message\Upload\UploadedFile":private]=>

NULL

}

}

若表单中是一个字段数组上传多个文件如 file[] 则返回的是一个二维数组，数组内容依然是 Swoft\Http\Message\Upload\UploadedFile 文件对象，数据结构如下。

array(1) {

["file"]=>

array(2) {

[0]=>

object(Swoft\Http\Message\Upload\UploadedFile)#6516 (7) {

["size":"Swoft\Http\Message\Upload\UploadedFile":private]=>

int(1319)

["errorCode":"Swoft\Http\Message\Upload\UploadedFile":private]=>

int(0)

["file":"Swoft\Http\Message\Upload\UploadedFile":private]=>

string(25) "/tmp/swoole.upfile.TVKdOS"

["clientFilename":"Swoft\Http\Message\Upload\UploadedFile":private]=>

string(6) "at.png"

["clientMediaType":"Swoft\Http\Message\Upload\UploadedFile":private]=>

string(9) "image/png"

["moved":"Swoft\Http\Message\Upload\UploadedFile":private]=>

NULL

["path":"Swoft\Http\Message\Upload\UploadedFile":private]=>

NULL

}

...

}

}

// 文件操作方法

moveTo() 将上传的文件移动到新位置。

getSize() 获取文件大小，单位 byte。

getError() 获取上传文件相关的错误信息，若无错将必须返回 UPLOAD\_ERR\_OK 常量，若有错误将返回 UPLOAD\_ERR\_XXX 相关常量。

getClientFilename() 获取文件上传时客户端本地的文件名，不要相信此方法返回的值，客户端可能会发送恶意虚假文件名，意图破坏或破解您的应用程序。

getClientMediaType() 获取客户端中文件的 MediaType 类型，不要相信此方法返回的值，客户端可能会发送恶意虚假文件名，意图破坏或破解您的应用程序。

// 其他辅助方法

if ($request->isAjax()) {

// Do something

}

if ($request->isXmlHttpRequest()) {

// Do something

}

if ($request->isGet()) {

// Do something

}

if ($request->isPost()) {

// Do something

}

if ($request->isPut()) {

// Do something

}

if ($request->isDelete()) {

// Do something

}

if ($request->isPatch()) {

// Do something

}

$contentType = $request->getContentType();

--

11、Http 响应对象

Swoft的请求与响应实现于 PSR-7 规范。请求与响应对象存在于每次 HTTP 请求。请求对象 Request 为 Swoft\Http\Message\Request，响应对象 Response 为 Swoft\Http\Message\Response。

PSR-7 接口为请求和响应对象提供了这些公共方法，withProtocolVersion($version)，withHeader($name, $value)，withAddedHeader($name, $value)，withoutHeader($name)，withBody(StreamInterface $body)。

PSR-7 接口为响应对象提供了这些方法，withStatus($code, $reasonPhrase = '')。更多请参考 PSR-7 和 查看 swoft/http-message ( https://github.com/swoft-cloud/swoft-http-message ) 中具体的实现类。

根据 PSR-7 对象的不可变性(immutable)，所有的 with\* 方法都是克隆对象然后返回，必须接收新对象来做进一步处理，或使用链式调用。

(1)、获取响应对象

通过控制器方法参数注入 (Response $response)，通过请求上下文获取 context()->getResponse()，通过请求上下文获取 Swoft\Context\Context::mustGet()->getResponse() \_(已废弃)\_。

//

///1/ 示例：设置响应状态码

$response = \context()->getResponse();

return $response->withStatus(404);

//

///2/ 示例：输出字符串内容响应

return $response->withContent("Hello Swoft2.0");

//

///3/ 示例：输出数组内容响应

$data = ['name'=>'Swoft2.0'];

$response->withData($data);

//

///4/ 示例：设置响应头信息

return $response->withHeader("name","Swoft2.0");

//

///6/ 示例：重定向

// 302

return $response->redirect("http://www.swoft.org",302);

// 404 page

return $response->redirect('/404');

//

///7/ 示例：文件下载

return $response->file(\alias('@runtime/1.zip'), "application/octet-stream");

//

///8/ 示例：设置 Cookies

$response = $response->withCookie(’name', 'value');

$response = $response->withCookie(’name', [

'value' => 'value3',

'httpOnly' => true

]);

$cookies = [

'key1' => 'value1',

'key2' => [

'value' => 'value2',

],

'key3' => [

'value' => 'value3',

'httpOnly' => true

],

];

$response = $response->withCookies($cookies);

--

12、中间件

中间件是用于控制 请求到达 和 响应请求 的整个流程的，通常用于对请求进行过滤验证处理，当你需要对请求或响应作出对应的修改或处理，或想调整请求处理的流程时均可以使用中间件来实现。

(1)、定义中间件

只需要实现了 Swoft\Http\Server\Contract\MiddlewareInterface 接口均为一个合法的中间件，其中 process() 方法为该中间件逻辑处理方法。

namespace App\Http\Middleware;

use Psr\Http\Message\ResponseInterface;

use Psr\Http\Message\ServerRequestInterface;

use Psr\Http\Server\RequestHandlerInterface;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Http\Server\Contract\MiddlewareInterface;

/\*\*

\* @Bean()

\*/

class ControllerMiddleware implements MiddlewareInterface

{

/\*\*

\* Process an incoming server request.

\*

\* @param ServerRequestInterface $request

\* @param RequestHandlerInterface $handler

\*

\* @return ResponseInterface

\* @inheritdoc

\*/

public function process(ServerRequestInterface $request, RequestHandlerInterface $handler): ResponseInterface

{

$response = $handler->handle($request);

return $response;

}

}

(2)、配置全局中间件

当你的自定义中间件需要全局请求应用，则可以考虑将此中间件作为全局中间件去使用，只需在 Bean 配置文件内配置 httpDispatcher 的 middlewares 属性，在数组中加入你的自定义中间件的命名空间地址，相关配置通常在 app/bean.php 内。

return [

...

'httpDispatcher'=>[

'middlewares'=>[

AuthMiddleware::class,

ApiMiddleware::class

]

]

...

]

(3)、通过注解使用

通过 @Middleware 和 @Middlewares, 可以很方便的配置中间件到当前的 Controller 和 Action 内。

当将此注解应用于 Controller 上，则作用域为整个 Controller，将此注解应用于 Action 上，则作用域仅为当前的 Action。@Middleware 用于配置单个中间件，@Middlewares 是用于配置一组 @Middleware，按照定义顺序依次执行。

namespace App\Http\Controller;

use App\Http\Middleware\ApiMiddleware;

use App\Http\Middleware\IndexMiddleware;

use App\Http\Middleware\ControllerMiddleware;

use Swoft\Http\Server\Annotation\Mapping\Controller;

use Swoft\Http\Server\Annotation\Mapping\Middleware;

use Swoft\Http\Server\Annotation\Mapping\Middlewares;

use Swoft\Http\Server\Annotation\Mapping\RequestMapping;

/\*\*

\* @Controller()

\* @Middlewares({

\* @Middleware(ApiMiddleware::class),

\* @Middleware(ControllerMiddleware::class)

\* })

\*/

class MiddlewareController

{

/\*\*

\* @RequestMapping()

\* @Middleware(IndexMiddleware::class)

\*/

public function index(){

return "MiddlewareController";

}

}

注意：记得要引入对应的中间件类。

(4)、应用

///1/ 示例：提前拦截请求。拦截要在 $handler->handle($request) 之前。

namespace App\Http\Middleware;

use Psr\Http\Message\ResponseInterface;

use Psr\Http\Message\ServerRequestInterface;

use Psr\Http\Server\RequestHandlerInterface;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Context\Context;

use Swoft\Http\Server\Contract\MiddlewareInterface;

/\*\*

\* @Bean()

\*/

class SomeMiddleware implements MiddlewareInterface

{

/\*\*

\* Process an incoming server request.

\* @param ServerRequestInterface $request

\* @param RequestHandlerInterface $handler

\* @return ResponseInterface

\* @inheritdoc

\*/

public function process(ServerRequestInterface $request, RequestHandlerInterface $handler): ResponseInterface

{

$path = $request->getUri()->getPath();

if ($path === '/favicon.ico') {

$response = Context::mustGet()->getResponse();

return $response->withStatus(404);

}

return $handler->handle($request);

}

}

//

///2/ 示例：跨域设置

namespace App\Http\Middleware;

use Psr\Http\Message\ResponseInterface;

use Psr\Http\Message\ServerRequestInterface;

use Psr\Http\Server\RequestHandlerInterface;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Http\Server\Contract\MiddlewareInterface;

/\*\*

\* @Bean()

\*/

class CorsMiddleware implements MiddlewareInterface

{

/\*\*

\* Process an incoming server request.

\* @param ServerRequestInterface $request

\* @param RequestHandlerInterface $handler

\* @return ResponseInterface

\* @inheritdoc

\*/

public function process(ServerRequestInterface $request, RequestHandlerInterface $handler): ResponseInterface

{

if ('OPTIONS' === $request->getMethod()) {

$response = Context::mustGet()->getResponse();

return $this->configResponse($response);

}

$response = $handler->handle($request);

return $this->configResponse($response);

}

private function configResponse(ResponseInterface $response)

{

return $response

->withHeader('Access-Control-Allow-Origin', 'http://mysite')

->withHeader('Access-Control-Allow-Headers', 'X-Requested-With, Content-Type, Accept, Origin, Authorization')

->withHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, PATCH, OPTIONS');

}

}

//

///3/ 示例：JWT登录验证

namespace App\Http\Middleware;

use Psr\Http\Message\ResponseInterface;

use Psr\Http\Message\ServerRequestInterface;

use Psr\Http\Server\RequestHandlerInterface;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Context\Context;

use Swoft\Http\Server\Contract\MiddlewareInterface;

/\*\*

\* @Bean()

\*/

class AuthMiddleware implements MiddlewareInterface

{

/\*\*

\* Process an incoming server request.

\* @param ServerRequestInterface $request

\* @param RequestHandlerInterface $handler

\* @return ResponseInterface

\* @inheritdoc

\*/

public function process(ServerRequestInterface $request, RequestHandlerInterface $handler): ResponseInterface

{

//// before request handle

// 判断token

$token = $request->getHeaderLine("token");

$type = \config('jwt.type');

$public = \config('jwt.publicKey');

try {

$auth = JWT::decode($token, $public, ['type' => $type]);

$request->user = $auth->user;

} catch (\Exception $e) {

$json = ['code'=>0,'msg'=>'授权失败']

$response = Context::mustGet()->getResponse();

return $response->withData($json);

}

$response = $handler->handle($request);

return $response;

// after request handle

}

}

--

13、异常处理

通常我们把异常类放置 app/Exception 中，异常类处理器放置 app/Exception/Handler 中， 异常分为两部分。自定义的 Exception 异常类，异常处理类 ExceptionHandler。

(1)、定义异常类

在不同应用场景下，定义不同的异常类，例如需要一个控制器抛异常的类。

namespace App\Exception;

class ApiException extends \Exception

{

}

(2)、定义异常处理类

namespace App\Exception\Handler;

use App\Exception\ApiException;

use Swoft\Error\Annotation\Mapping\ExceptionHandler;

use Swoft\Http\Message\Response;

use Swoft\Http\Server\Exception\Handler\AbstractHttpErrorHandler;

/\*\*

\* @ExceptionHandler(ApiException::class)

\*/

class ApiExceptionHandler extends AbstractHttpErrorHandler

{

/\*\*

\* @param \Throwable $e

\* @param Response $response

\* @return Response

\* @throws \ReflectionException

\* @throws \Swoft\Bean\Exception\ContainerException

\*/

public function handle(\Throwable $e, Response $response): Response

{

$data = ['code'=>-1,'msg'=>$e->getMessage()];

return $response->withData($data);

}

}

(3)、@ExceptionHandler 注解

异常处理程序，指定这个处理器要处理当前异常，当程序抛出 ExceptionHandler 注解里有的异常时，将会自动执行 handle 方法。

指定异常：参数可以是字符串也可以是数组。

示例：处理一个异常 @ExceptionHandler(ApiException::class)。

示例：处理多个异常 @ExceptionHandler({ApiException::class,ServiceException::class})。

--

14、Http 客户端

官方建议使用 saber 和 Guzzle ，不再重复造轮子。

(1)、Saber

Swoole 官方封装的 Http client 库，已在多个大型项目中使用。

//

///1/ 安装

composer require swlib/saber

//

///2/ 使用

SaberGM::get('http://httpbin.org/get');

SaberGM::delete('http://httpbin.org/delete');

SaberGM::post('http://httpbin.org/post', ['foo' => 'bar']);

SaberGM::put('http://httpbin.org/put', ['foo' => 'bar']);

SaberGM::patch('http://httpbin.org/patch', ['foo' => 'bar']);

(2)、Guzzle

Guzzle 老牌 HTTP 封装库，很多依赖都是它封装的。swoole 版本必须 4.4 即以上。

///1/ 安装

composer require guzzlehttp/guzzle

//

///2/ 使用

$client = new \GuzzleHttp\Client();

$response = $client->request('GET', 'https://api.github.com/repos/guzzle/guzzle');

echo $response->getStatusCode(); # 200

echo $response->getHeaderLine('content-type'); # 'application/json; charset=utf8'

echo $response->getBody(); # '{"id": 1420053, "name": "guzzle", ...}'

--

--

三、Websocket

https://swoft.org/documents/v2/core-components/websocket/

WebSocket 服务基于现有 swoole ws server 上的进一步封装实现。即开启 websocket 服务的同时可以处理 http 请求。

--

1、安装

(1)、Composer 安装

composer require swoft/websocket-server

(2)、Git 仓库

Github https://github.com/swoft-cloud/swoft-websocket-server

--

2、参与贡献

欢迎参与贡献，您可以 fork 我们的开发仓库 swoft/component( https://github.com/swoft-cloud/swoft-component )，修改代码然后发起 PR，阅读 提交代码( https://swoft.org/documents/v2/contribute/sub-codes/ ) 的注意事项。

--

3、功能特色

快速地搭建使用 websocket server；

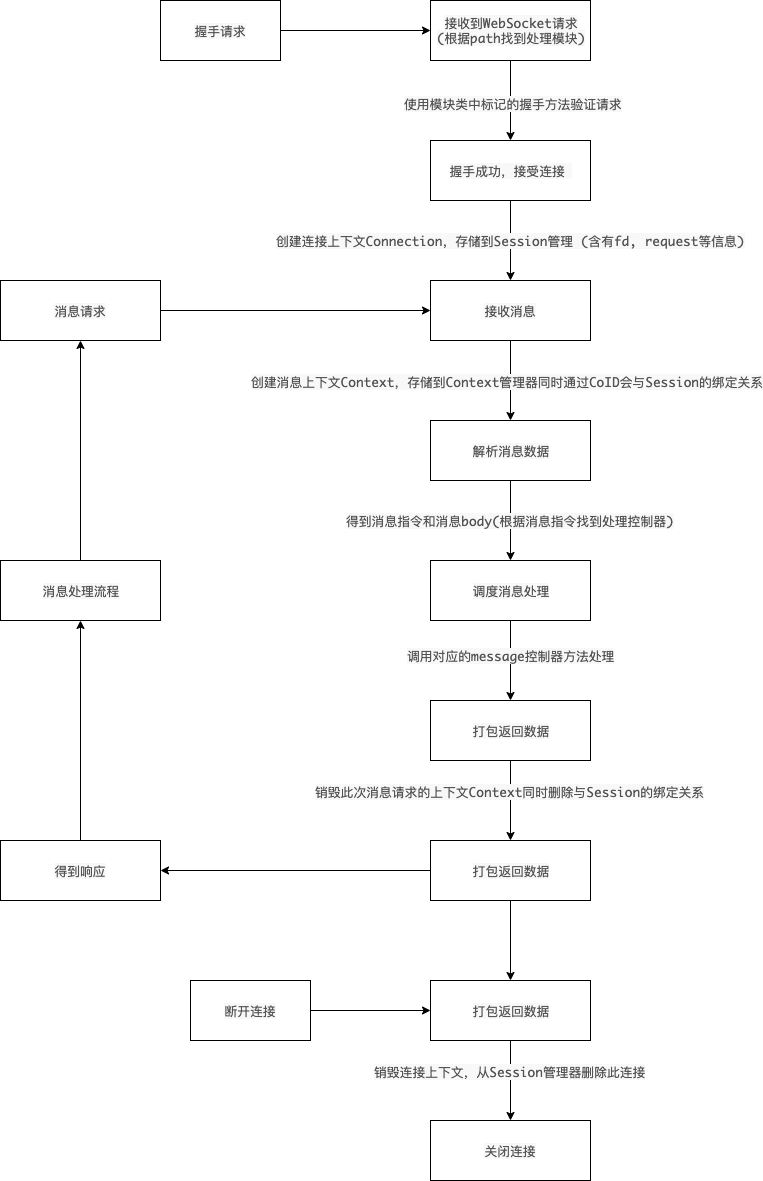
支持完全的自定义流程处理，如果你不想使用框架自带的处理；

支持消息阶段的数据解析和路由调度；

通用的消息发送方法封装(send, sendToSome, sendToAll, broadcast 等)。

--

4、连接处理流程



( 流程图 https://swoft.org/img/websocket-flow-chat.png )

--

5、配置参数

websocket 的 host, port 等配置是都是完全可以自定义的。 配置需要编辑 app/bean.php 文件，下面列举了一些简单的配置，你也可以自由组合同时提供多种服务。

'wsServer' => [

'class' => WebSocketServer::class,

'port' => 18307,

'on' => [

// 开启处理http请求支持

SwooleEvent::REQUEST => bean(RequestListener::class),

// 启用任务必须添加 task, finish 事件处理

SwooleEvent::TASK => bean(TaskListener::class),

SwooleEvent::FINISH => bean(FinishListener::class)

],

'listener' => [

// 引入 rpcServer

'rpc' => \bean('rpcServer')

],

'debug' => env('SWOFT\_DEBUG', 0),

/\* @see WebSocketServer::$setting \*/

'setting' => [

'log\_file' => alias('@runtime/swoole.log'),

// 任务需要配置 task worker

'task\_worker\_num' => 2,

'task\_enable\_coroutine' => true

],

],

'rpcServer' => [

'class' => ServiceServer::class,

'port' => 18308,

],

可配置项用于 ws server bean 配置，除了 class 其他都是 ws server 的属性。

class 指定 websocket server 的处理类；

port 指定 websocket server 的端口；

listener 指定其他一同启动的服务，添加端口服务监听，可以多个，rpc 启动 RPC 服务；

process 启动自定义用户进程；

on 配置监听的事件，注册事件、设置对应事件的处理监听，事件触发组件调用，在任务里面使用；

setting 这里是参考 Swoole Server 配置选项( https://wiki.swoole.com/wiki/page/274.html )；

pidFile 设置进程 pid 文件 位置，默认值 @runtime/swoft.pid；

mode 运行的模式，参考 Swoole Server 构造函数 第三个参数( https://wiki.swoole.com/wiki/page/14.html )；

type 指定Socket的类型，支持TCP、UDP、TCP6、UDP6、UnixSocket Stream/Dgram 等 Swoole Server 构造函数 第四个参数( https://wiki.swoole.com/wiki/page/14.html )，启用 WSS 支持，注意你必须安装 OpenSSL 库，并且确保安装 swoole 时是启用了 ssl 选项的，同时，需要设置 'type' => SWOOLE\_SOCK\_TCP | SWOOLE\_SSL。

--

6、Websocket Server 命令

在项目目录下执行如下命令可以看到 websocket server 的管理命令. 跟 http server 的管理命令一致。

$ php bin/swoft ws

Description:

There some commands for manage the webSocket server

//

Usage:

ws:{command} [arguments] [options]

//

Commands:

start Start the webSocket server

stop Stop the running server

restart Restart the running server

//

Options:

-h, --help Show help of the command group or specified command action

(1)、使用

//// 前台运行

$ php bin/swoft ws:start

//

//// 后台运行

$ php bin/swoft ws:start -d

websocket server 的默认端口是 18308，如果你注册了 ws 的路由处理模块，现在就可以通过浏览器等ws客户端连接上server了。

--

7、websocket 模块

在根据上面安装配置好 websocket 之后，就可以在 app/WebSocket 下创建需要的 websocket 模块来处理相关逻辑。在每个模块里允许用户处理的几个事件有 handshake、open、message、close。

(1)、@WsModule 注解

websocket 模块类注解 @WsModule。

注解类：Swoft\WebSocket\Server\Annotation\Mapping\WsModule；

作用范围：CLASS；

拥有属性：path string 标明了允许ws连接的 URI path；controllers array 绑定到此模块的 消息控制器( https://swoft.org/documents/v2/core-components/websocket/#%E6%B6%88%E6%81%AF%E6%8E%A7%E5%88%B6%E5%99%A8 ) 类；messageParser string 绑定到此模块的 消息解析器( https://swoft.org/documents/v2/core-components/websocket/#%E6%B6%88%E6%81%AF%E8%A7%A3%E6%9E%90%E5%99%A8 )；defaultOpcode integer 此模块默认的消息数据 opcode。

//

///1/ 示例

/\*\*

\* @WsModule("/echo", controllers={XXController::class, XYController::class})

\*/

上面的注解标明了允许 ws 连接的 URI path. 即客户端请求的ws连接类似：ws://IP:PORT/echo。你可以绑定多个控制器，请注意引入完整的控制器、消息解析器类。

(2)、@OnHandshake 注解

方法注解 @OnHandshake 标记处理握手的方法。

注解类：Swoft\WebSocket\Server\Annotation\Mapping\OnHandshake；

作用范围：METHOD；

这方法是可选的。如果没有特殊的需求，可以忽略它，框架会帮你握手并响应握手成功。

必须返回含有两个元素的 array，bool 第一个元素的值来决定是否进行握手，第二个元素是 response 对象 - 可以在 response 设置一些自定义 header，body 等信息。

(3)、@OnOpen 注解

在握手成功后，就会触发 open 事件。 方法注解 @OnOpen 标记对应方法。此时开始你就可以给客户端发消息了:)。

注解类：Swoft\WebSocket\Server\Annotation\Mapping\OnOpen；

作用范围：METHOD；

此方法也是可选的，可以没有。

(4)、@OnMessage

通过的方法注解 @OnMessage 标记一个消息处理方法。在此阶段你可以接收到客户端的消息和发送消息给对方。

注解类：Swoft\WebSocket\Server\Annotation\Mapping\OnMessage；

作用范围：METHOD；

当你没有绑定消息控制器时，表明你想自己处理消息阶段的逻辑，此方法是必须存在的。当你有绑定消息控制器时，框架会自动解析消息并路由到指定的消息处理方法。

(5)、@OnClose

通过的方法注解 @OnClose 标记一个关闭连接时的处理方法。注意：触发此事件时连接已被关闭，不能再给对方发消息。

当客户的关闭连接或者 server 在其他地方主动关闭连接时，就会触发此事件。你可以在这里做一些连接关闭后的工作, 比如：记录日志，解绑用户等。

注解类：Swoft\WebSocket\Server\Annotation\Mapping\OnClose；

作用范围：METHOD；

此方法也是可选的，可以没有；

(6)、快速创建模块类

// 可以使用 swoftcli 工具来快速创建一个 websocket 模块类，默认生成的是支持内置路由调度的模块类。

php swoftcli.phar gen:wsmod chat --prefix /chat

//

// 生成用户自定义调度的模块类。

php swoftcli.phar gen:wsmod chat --prefix /chat --tpl-file ws-module-user

//

///1/ 示例：这里面方法上的 server 对象都是 Swoole\WebSocket\Server 的实例。

<?php

namespace App\WebSocket;

use Swoft\Http\Message\Request;

use Swoft\Http\Message\Response;

use Swoft\WebSocket\Server\Annotation\Mapping\OnClose;

use Swoft\WebSocket\Server\Annotation\Mapping\OnHandshake;

use Swoft\WebSocket\Server\Annotation\Mapping\OnOpen;

use Swoft\WebSocket\Server\Annotation\Mapping\WsModule;

use Swoole\WebSocket\Frame;

use Swoole\WebSocket\Server;

/\*\*

\* Class EchoModule

\*

\* @WsModule("/echo")

\*/

class EchoModule

{

/\*\*

\* 在这里你可以验证握手的请求信息

\* @OnHandshake()

\* @param Request $request

\* @param Response $response

\* @return array [bool, $response]

\*/

public function checkHandshake(Request $request, Response $response): array

{

return [true, $response];

}

/\*\*

\* On connection has open

\*

\* @OnOpen()

\* @param Request $request

\* @param int $fd

\*/

public function onOpen(Request $request, int $fd): void

{

server()->push($fd, 'hello, welcome! :)');

}

/\*\*

\* @OnMessage()

\* @param Server $server

\* @param Frame $frame

\*/

public function onMessage(Server $server, Frame $frame)

{

$server->push($frame->fd, 'I have received message: ' . $frame->data);

}

/\*\*

\* On connection closed

\* - you can do something. eg. record log

\*

\* @OnClose()

\* @param Server $server

\* @param int $fd

\*/

public function onClose(Server $server, int $fd): void

{

// you can do something. eg. record log, unbind user...

}

}

?>

//

///2/ 示例：简易的客户端 js 代码连接示例。

// wsUrl = websocket host + module path

const wsUrl = 'ws://127.0.0.1:18308/echo';

// 连接

let ws = new WebSocket(wsUrl);

// 错误

ws.onerror = function (event){

console.log("error: " + event.data)

}

// 打开

ws.onopen = function (event){

console.log("open: connection opened");

}

// 信息

ws.onmessage = function (event){

console.log("message: " + event.data);

}

// 关闭

ws.onclose = function (event){

console.log("close: connection closed");

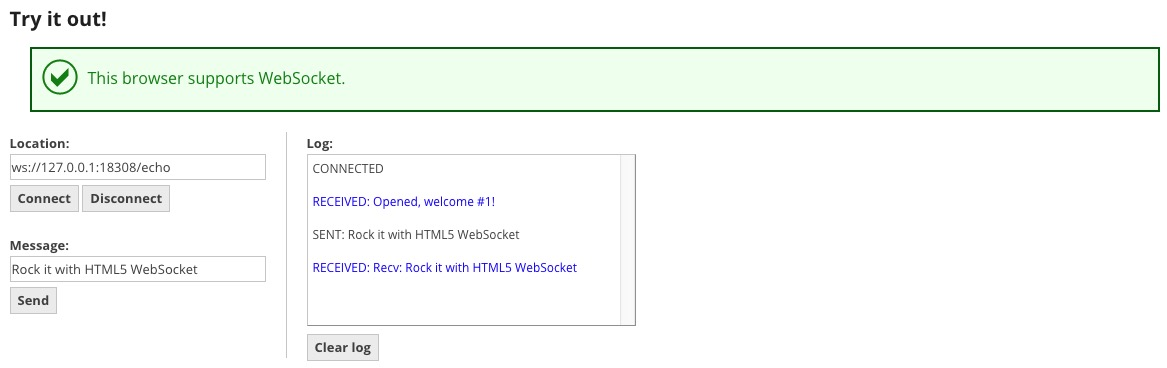
ws.close();

}

//

如果你安装并启用了 devtool, 那么你可以打开页面 IP:PORT/\_\_devtool/ws/test 来进行ws测试。填上你的 ws server 地址(注意不要忘了 URI path)(2.0 devtool 暂无 web UI)，然后就可以连接上 ws server 并收发消息了，如果你在前台运行的 server 你也能在运行 server 的 console 上看到 ws 连接与消息 log。

可在网上找一个 ws test 网页来进行测试。注意，请确保 server 是启动且地址没有填写错误。这里我们使用 http://www.websocket.org/echo.html 简单测试使用下。



( 图 https://swoft.org/img/ws-echo-test.jpg )

--

8、消息控制器

swoft 提供了灵活的 websocket 使用，支持自定义和由框架托管处理消息两种方式。

如果你在 ws 模块类没有添加 @OnMessage 处理方法，框架将会自动托管这个阶段，解析消息并根据路由分发到不同的方法执行。如果你在 ws 模块类里面绑定了 @OnMessage 处理方法，swoft 就认为你想自己处理这个阶段，框架就不会处理了。本篇文档的使用是建立在由框架托管消息路由的基础上。

(1)、@WsController 注解

websocket 消息控制器注解 @WsController。

注解类：Swoft\WebSocket\Server\Annotation\Mapping\WsController；

作用范围：CLASS；

拥有属性：prefix string 消息路由前缀；

(2)、@MessageMapping 注解

方法注解 @MessageMapping 标记具体的消息处理方法，类似于 http 控制器里的 action。

注解类：Swoft\WebSocket\Server\Annotation\Mapping\MessageMapping；

作用范围：METHOD；

拥有属性：command string 消息命令名称；

完整的消息路由 path 是 上面的 preifx 和 command 由点拼接而成 PREFIX.COMMAND。

--

9、消息解析器

不同的使用者或者使用场景，用于 ws 通信的数据格式可能是不一样的。因此，在编写 ws 模块时，需要你绑定消息解析器。

(1)、内置解析器

Swoft\WebSocket\Server\MessageParser\RawTextParser 简单的字符串；

Swoft\WebSocket\Server\MessageParser\TokenTextParser 简单的 token 字符串协议(方便测试使用的)；

Swoft\WebSocket\Server\MessageParser\JsonParser 简单的 JSON 数据协议；

//

//// JSON 协议通信数据结构

{

"cmd": "message route path. eg: home.index", // type: string

"data": "message data", // type: mixed(array|string|int)

"ext": {"ip": "xx", "os": "mac"}, // optional, type: array

}

(2)、获取数据

有多种方式可以获取消息请求的数据信息。

Message 对象是一个通用的 websocket 数据对象，里面保存了解析后的数据，包含 cmd data ext 三个字段。我们可以通过 参数注入 或 上下文方法 来获取 Message 对象。

//

///1/ 示例：通过参数注入获取

use Swoft\WebSocket\Server\Message\Message;

use Swoft\WebSocket\Server\Message\Request;

...

// inject raw frame data string

public function autoReply(string $data): string

{

return $data;

}

// inject Message object

public function autoReply(Message $msg): string

{

return $msg->toString();

}

// inject Request object

public function autoReply(Request $req): string

{

return $req->getMessage()->toString();

}

//

///2/ 示例：通过上下文获取

use Swoft\WebSocket\Server\Message\Message;

...

public function autoReply(): string

{

$msg = context()->getMessage();

}

//

///3/ 示例：更多获取方式

use Swoft\WebSocket\Server\Message\Request;

$req = context()->getRequest();

/\*\* @var \Swoft\WebSocket\Server\Message\Message $msg \*/

$msg = $req->getMessage();

/\*\* @var \Swoole\WebSocket\Frame $frame \*/

$frame = $req->getFrame();

注意这里的 Request 是指消息阶段的请求对象，与打开连接时的请求对象是不同的。

--

10、使用示例

(1)、定义 ws 模块

要绑定消息处理控制器，通常也需要绑定你的消息解析器，可以使用内置的几个简单的解析器，也可以根据需要自定义。

<?php declare(strict\_types=1);

namespace App\WebSocket;

use App\WebSocket\Chat\HomeController;

use Swoft\Http\Message\Request;

use Swoft\WebSocket\Server\Annotation\Mapping\OnOpen;

use Swoft\WebSocket\Server\Annotation\Mapping\WsModule;

use Swoft\WebSocket\Server\MessageParser\TokenTextParser;

use function server;

/\*\*

\* Class ChatModule

\*

\* @WsModule(

\* "/chat",

\* messageParser=TokenTextParser::class,

\* controllers={HomeController::class}

\* )

\*/

class ChatModule

{

/\*\*

\* @OnOpen()

\* @param Request $request

\* @param int $fd

\*/

public function onOpen(Request $request, int $fd): void

{

server()->push($request->getFd(), "Opened, welcome!(FD: $fd)");

}

}

?>

定义的 ws 模块路径为 /chat；绑定了的控制器有：HomeController::class，你可以绑定多个控制器；绑定了一个内置的消息解析器。

这里定义 Ws 模块时，绑定了一个框架自带的消息解析器，TokenTextParser::class 内置了一个 decode() 的方法用来解析数据。

// 默认为字符串解析，消息路由格式 `控制器.方法:数据`

public function decode(string $data): Message

{

// use default message command

$cmd = '';

if (strpos($data, ':')) {

[$cmd, $body] = explode(':', $data, 2);

$cmd = trim($cmd);

} else {

$body = $data;

}

return Message::new($cmd, $body);

}

(2)、定义消息控制器

必须使用注解 @WsController 以及 @MessageMapping。

<?php declare(strict\_types=1);

namespace App\WebSocket\Chat;

use Swoft\Session\Session;

use Swoft\WebSocket\Server\Annotation\Mapping\MessageMapping;

use Swoft\WebSocket\Server\Annotation\Mapping\WsController;

/\*\*

\* Class HomeController

\*

\* @WsController()

\*/

class HomeController

{

/\*\*

\* Message command is: 'home.index'

\*

\* @return void

\* @MessageMapping()

\*/

public function index(): void

{

Session::mustGet()->push('hi, this is home.index');

}

/\*\*

\* Message command is: 'home.echo'

\*

\* @param string $data

\* @MessageMapping()

\*/

public function echo(string $data): void

{

Session::mustGet()->push('(home.echo)Recv: ' . $data);

}

/\*\*

\* Message command is: 'home.ar'

\*

\* @param string $data

\* @MessageMapping("ar")

\*

\* @return string

\*/

public function autoReply(string $data): string

{

return '(home.ar)Recv: ' . $data;

}

}

?>

自 v2.0.6 版本起，通过参数注入接收 websocket 原始数据时，需要加上类型 string。例如： public function echo(string $data)。

(3)、访问服务

根据以上定义好的 Ws模块、消息解析器、消息控制器 等内容后启动我们的服务。然后打开 webscoket 调试工具，链接Ws的地址：ws://localhost:port/chat 然后测试发送一个内容。

Send: testWS

Recv: hi, this is home.index

Send: home.echo:这是数据

Recv: (home.echo)Recv: 这是数据

--

11、消息发送

上一节我们知道了如何创建 ws 模块，并通过客户端连接到 server。可以从示例代码里看到有简单的消息发送使用了。

...

/\*\* @var \Swoole\WebSocket\Server $server \*/

$server->push($fd, 'hello, welcome! :)');

...

这里的 server 是 swoole 的 \Swoole\WebSocket\Server 对象，$fd 是与客户端的连接 ID，它表明了不同的客户端。

//

除了使用 $server 来发送消息外，我们还可以使用 swoft 封装好的 \server() 或者 \Swoft::server() 来发送消息。

\server()->sendTo($fd, 'hi, 你好啊！');

\Swoft::server()->sendTo($fd, 'hi, 你好啊！');

Swoft\WebSocket\Server\WebSocketServer 的实例对象，内部已经封装了各种发送消息的方法 API，前台运行时，通过它发送消息能从控制台看到消息发送 log。

(1)、消息发送 API

注意下面的方法都在类：Swoft\WebSocket\Server\WebSocketServer。

//

///1/ 发送给某个客户端

public function sendTo(int $receiver, string $data, int $sender = 0): int

$receiver int 接收者的 fd；

$data string 要发送的消息数据；

$sender int 发送者的 fd，可选的。

\server()->sendTo($fd, 'hi, 你好啊！');

//

///2/ 发送给指定的一些客户端

public function sendToSome(string $data, array $receivers = [], array $excluded = [], int $sender = 0, int $pageSize = 50): int

$data string 要发送的消息数据；

$receivers int[] 指定的接收者 fd 列表；

$excluded int[] 排除的接收者 fd 列表；

$sender int 发送者的 fd，可选的。

方法说明，当 $receivers 有数据时，将会忽略 $excluded，此时就是将消息指定的发给这些接收者；当 $receivers 为空时，若 $excluded 有值，将会给除了这些人之外的发送消息，若 $excluded 为空，相当于给所有人发消息。

\server()->sendToSome('hi, 你们好啊！', [$fd0, $fd1, ...]);

//

///3/ 广播消息

发送消息给除了 sender 外的所有人。使用分页方式发送，每 50 个一页，直到全部发送完毕。

broadcast(string $data, array $receivers = [], array $excluded = [], int $sender = 0): int

//

///4/ 发送给所有客户端

发送消息给所有客户端，相当于进行全员广播。使用分页方式发送，每 50 个一页，直到全部发送完毕。

public function sendToAll(string $data, int $sender = 0, int $pageSize = 50): int

$data string 要发送的消息数据；

$sender int 发送者的 fd，可选的。

\server()->sendToAll('hi, 大家好啊！');

//

///5/ send

参数跟 sendToSome 一样。会自动根据参数判断调用上面的（sendTo, sendToAll, sendToSome）中的一个方法。

//

///6/ 断开连接

服务端可以主动断开连接，断开后会触发 close 事件。

bean('wsServer')->disconnect($fd);

// OR

server()->disconnect($fd);

--

12、异常处理

前面我们了解了系统如何处理异常（详情请看 错误处理章节 https://swoft.org/documents/v2/basic-components/error-dispose/ ），以及 http server 里如何处理异常的。在 websocket server 也是类似的，我们只需定义 websocket 相关场景的异常处理器就行。

与 http server 里只有一个 request 场景不同, websocket 里有四个场景：handshake 握手环节，open 握手后连接打开，message 消息通信阶段，close 连接关闭。

下面我们编写 websocket 几个环节中最重要的 握手 和 消息通信 环节的异常处理。其他环节的可以参考和继承相关类来编写。

(1)、握手异常

因为 websocket 握手环节就是 http 请求处理，所以此环节的异常跟 http 里处理是一样的，当然你还是得 继承 为这个场景设计的 基础类 才行。必须继承 AbstractHandshakeErrorHandler 类，我们才能知道你要处理哪个 场景 里的异常。

<?php declare(strict\_types=1);

namespace App\Exception\Handler;

use ReflectionException;

use Swoft\Bean\Exception\ContainerException;

use Swoft\Error\Annotation\Mapping\ExceptionHandler;

use Swoft\Http\Message\Response;

use Swoft\WebSocket\Server\Exception\Handler\AbstractHandshakeErrorHandler;

use Throwable;

use function get\_class;

use function sprintf;

use const APP\_DEBUG;

/\*\*

\* Class HttpExceptionHandler

\*

\* @ExceptionHandler(\Throwable::class)

\*/

class WsHandshakeExceptionHandler extends AbstractHandshakeErrorHandler

{

/\*\*

\* @param Throwable $e

\* @param Response $response

\*

\* @return Response

\* @throws ReflectionException

\* @throws ContainerException

\*/

public function handle(Throwable $e, Response $response): Response

{

// Debug is false

if (!APP\_DEBUG) {

return $response->withStatus(500)->withContent(sprintf(

'%s At %s line %d', $e->getMessage(), $e->getFile(), $e->getLine()

));

}

$data = [

'code' => $e->getCode(),

'error' => sprintf('(%s) %s', get\_class($e), $e->getMessage()),

'file' => sprintf('At %s line %d', $e->getFile(), $e->getLine()),

'trace' => $e->getTraceAsString(),

];

// Debug is true

return $response->withData($data);

}

}

?>

(2)、消息通信异常

在握手成功后的消息通信阶段出现异常，也可以方便地捕获处理。你仍然需要继承专有场景的异常处理抽象类 AbstractMessageErrorHandler。

<?php declare(strict\_types=1);

namespace App\Exception\Handler;

use ReflectionException;

use Swoft\Bean\Exception\ContainerException;

use Swoft\Error\Annotation\Mapping\ExceptionHandler;

use Swoft\Log\Helper\Log;

use Swoft\WebSocket\Server\Exception\Handler\AbstractMessageErrorHandler;

use Swoole\WebSocket\Frame;

use Throwable;

use function server;

use const APP\_DEBUG;

/\*\*

\* Class WsMessageExceptionHandler

\*

\* @since 2.0

\*

\* @ExceptionHandler(\Throwable::class)

\*/

class WsMessageExceptionHandler extends AbstractMessageErrorHandler

{

/\*\*

\* @param Throwable $e

\* @param Frame $frame

\*

\* @throws ContainerException

\* @throws ReflectionException

\*/

public function handle(Throwable $e, Frame $frame): void

{

$message = sprintf('%s At %s line %d', $e->getMessage(), $e->getFile(), $e->getLine());

Log::error('Ws server error(%s)', $message);

// Debug is false

if (!APP\_DEBUG) {

server()->push($frame->fd, $e->getMessage());

return;

}

server()->push($frame->fd, $message);

}

}

?>

--

--

四、RPC Server

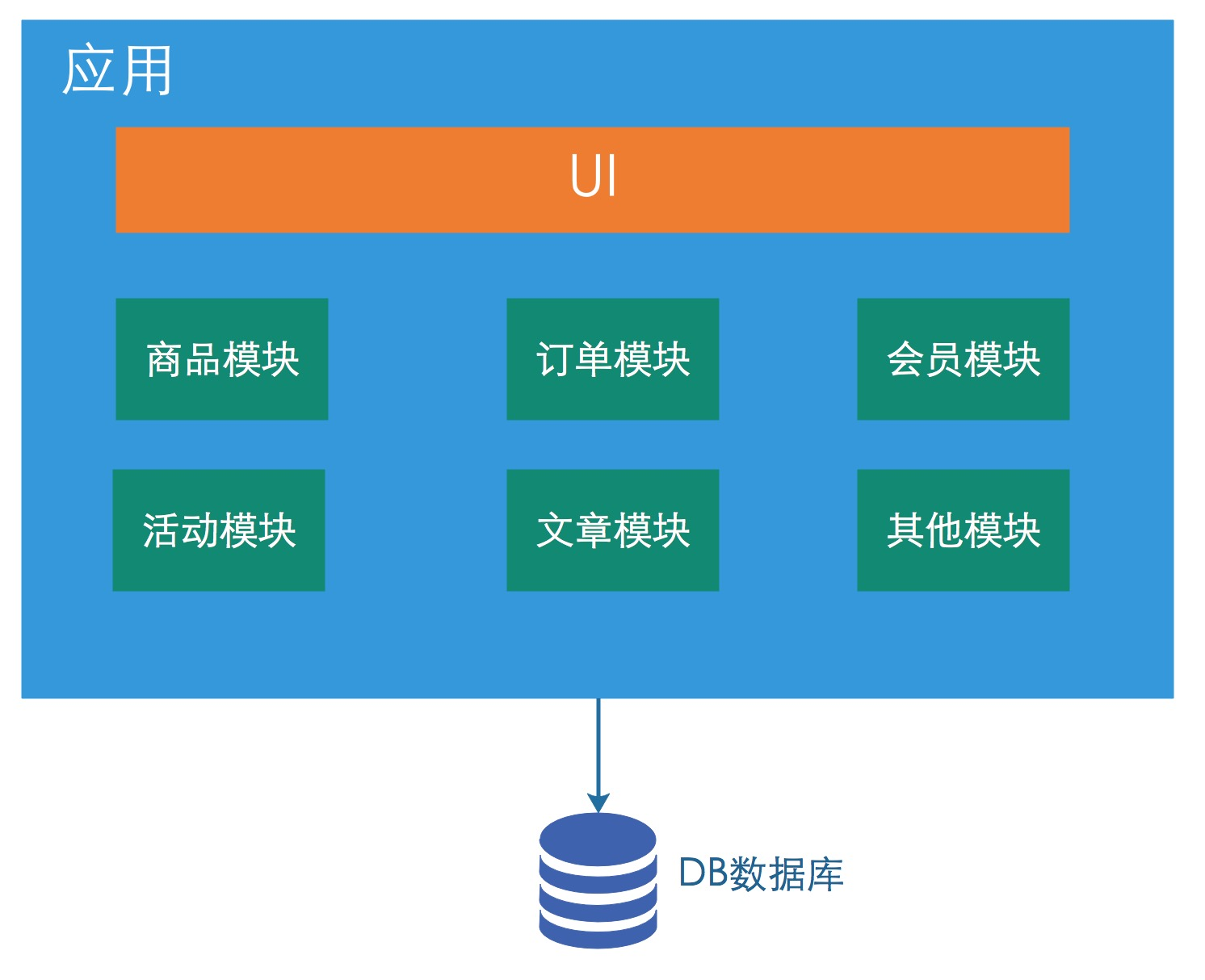
https://swoft.org/documents/v2/core-components/rpc-server/

RPC，是一种远程调用方式（Remote Procedure Call），通过 RPC 我们可以像调用本地方法一样调用别的机器上的方法，用户将无感服务器与服务器之间的通讯。RPC 在微服务当中起到相当大的作用，当然 RPC 不是微服务必须的一种方式，有别的方式也可以实现这种远程调用例如 RESTful API 就可以实现远程调用。如果有用过 SOAP 那么你使用 RPC 将会觉得很类似，都是可以直接调用别的机器上的方法。

随着业务的发展我们的项目从简单的单体结构逐渐的演化成微服务结构，我们为什么要拆分成微服务呢？那我们来说说微服务和单体架构的优缺点。

--

1、单体架构



( 图 https://swoft.org/img/rpc-1.png )

(1)、单体架构优点

部署容易，如 php 写的项目，只要一个文件夹复制到支持 php 的环境就可以了，java 只需要一个 jar 包；

测试容易，我们整体项目只要改了一个地方马上就可以测试得出结果；

负载均衡就可以解决，快速部署多个一模一样的项目在不同的机器运行分流；

(2)、单体架构缺点

部署的问题，对于 php 来说这点还好，但是对于 java 的项目来说，我们需要重新打包整个项目耗费的时间是很长的；

代码维护，由于所有的代码都写在一个项目里面，要想要修改某一个功能点那么需要对项目的整体逻辑和设计有较深的理解，否则代码耦合严重，导致维护难，特别对于新入职的员工来说这将是最容易出现问题的地方；

开发效率低，随着项目需求的不断改变和新的功能新增，老旧的代码又不敢随便删除，导致整个项目变得笨重，这将会增加你阅读代码的时间；

扩展性，在高并发的情况下，我们往往不是整个项目的每一个功能都处于高流量高请求的情况下的，很多时候都是某一个功能模块使用的人数比较多，在单体结构下我们没有办法针对单个功能实现分布式扩展，必须整个项目一起部署；

--

2、微服务架构

在2014年被提出，现在国内很多公司已经使用，微服务是一种架构设计，并不是说什么框架或者代替什么。微服务做的事情是按照项目颗粒度进行服务的拆分，把模块单独拿出来做成每一个单独的小项目。微服务的主要特点有：每一个功能模块是一个小项目、独立运行在不同进程或者机器上、不同功能可以由不同的人员开发、独立开发不松耦合、独立部署不需要依赖整体项目就可以启动单个服务、分布式管理。每一个服务只要做好自己的事情就好了。在设计微服务的时候还需要考虑到数据库的问题，是所有微服务使用共同一个数据库还是每一个服务单个数据库。

(1)、微服务架构优点

拆分业务，把整体大项目分割成不同小项目运行在不同进程或者机器上实现数据隔离。

技术栈，每个服务可以由不同的团队或者开发者进行开发，外部调用人员不需要操心具体怎么实现的，只需要类似调用自己方法一样或者接口一样按照服务提供者给出来的参数传递即可。

独立部署，每一个服务独立部署，部署一个服务不会影响整体项目，如果部署失败最多是这个服务的功能缺失，并不影响其他功能的使用。

按需部署，针对不同的需求可以给不同的服务自由扩展服务器，根据服务的规模部署满足需求的实例。

局部修改，当一个服务有新需求或者其他修改，不需要修改整体项目只要管好自己的服务就好了。

(2)、微服务架构缺点

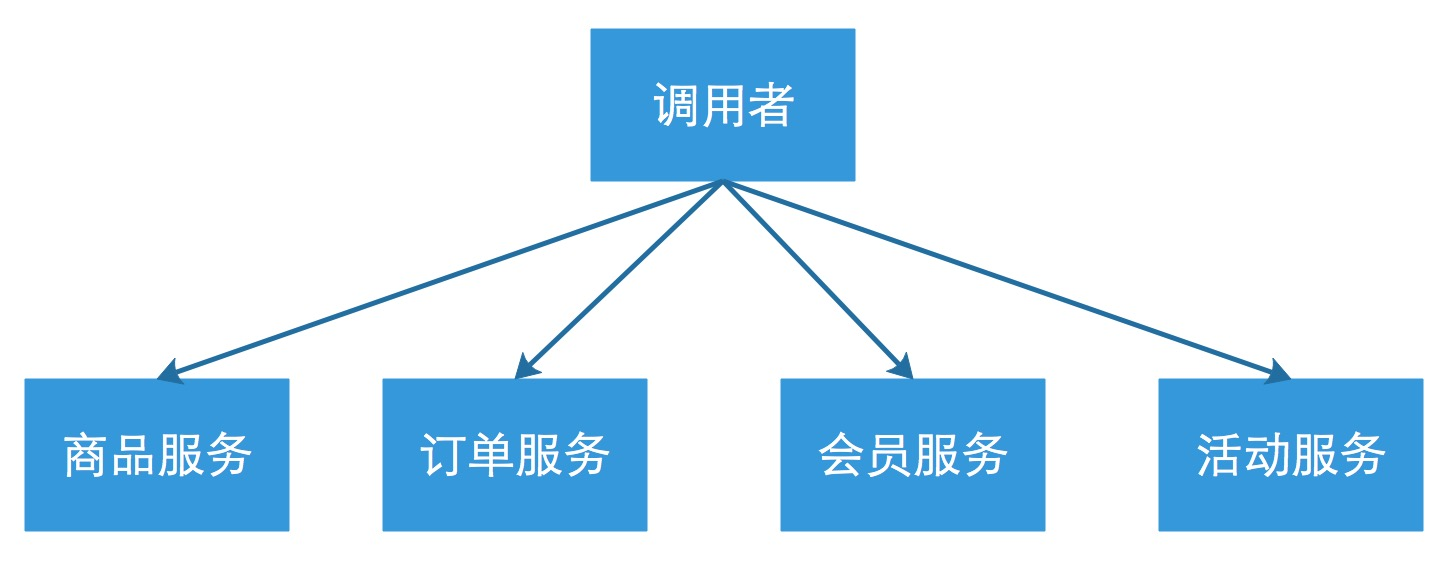
运维，微服务由于把业务拆分得细，有可能部署在不同机器上，因此对于运维人员的管理来说，这部分的成本会加大。

接口调整，微服务之间通过接口进行通信，如果修改某个微服务的API，可能所有使用了该接口的微服务都需要做调整。

重复劳动，很多服务可能都会使用到相同的功能。而这个功能并没有达到分解为一个微服务的程度，这个时候，可能各个服务都会开发这一功能，导致代码重复。

分布式，由于会把不同服务部署在不同机器上，那么对于这些服务的调用、容错、网络延迟、分布式事务等等都是一个很大的挑战，当然微服务不一定全部都是部署在不同服务器上。

(3)、服务调用



( 图 https://swoft.org/img/rpc-2.png )

如上图所示，RPC 就用于调用者与服务之间的通讯，RPC 协议可基于 TCP、UDP 或者 HTTP 实现，但是更推荐选择 TCP。

例如调用者需要调用商品的服务就可以通过 RPC 或者 RESTful API 来调用，那么 RPC 调用和 RESTful API 两者之间的区别在哪呢？

//

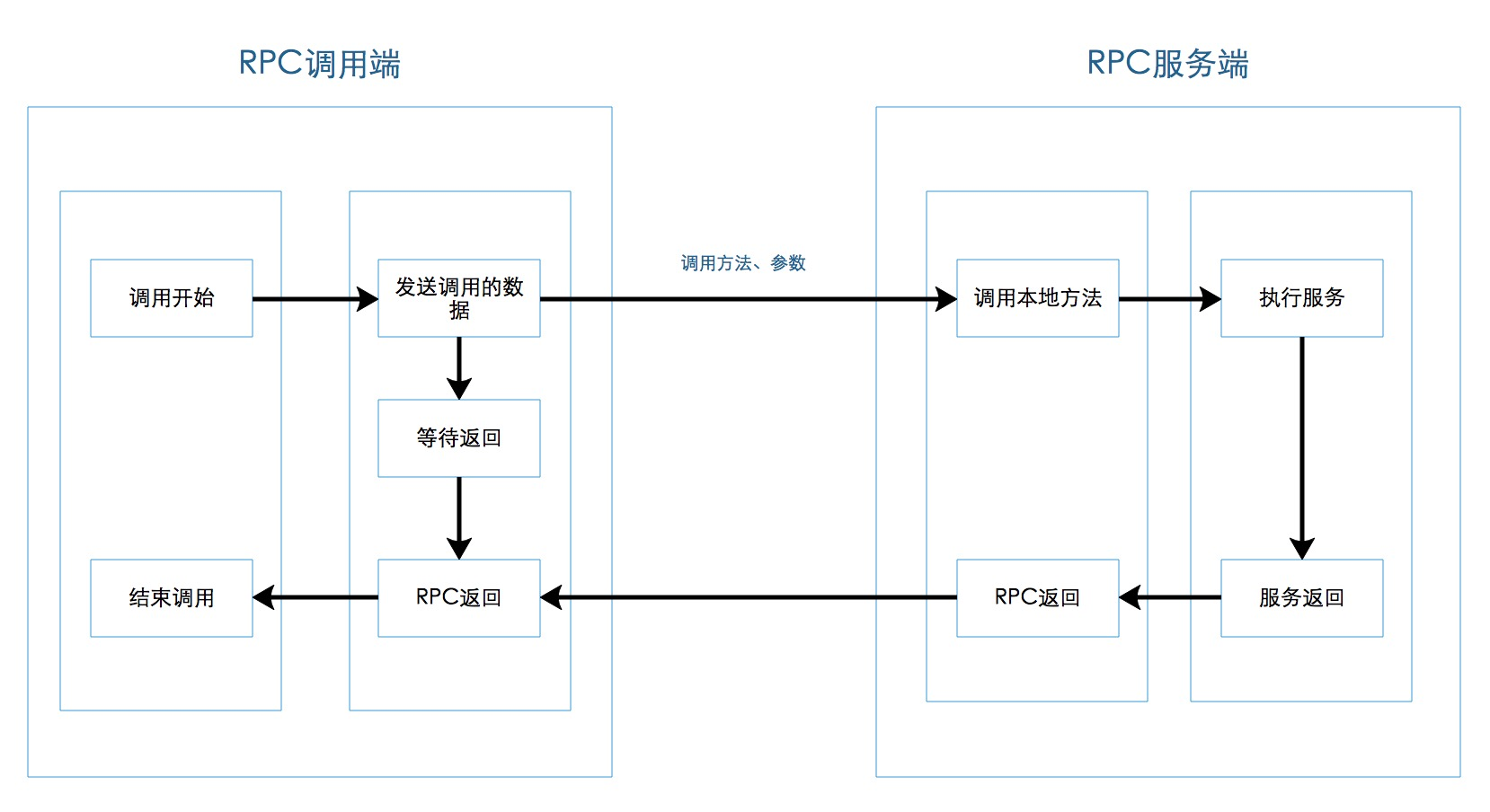
//// RPC调用 和 RESTful API 两者之间的区别

TCP 支持长连接，当调用服务的时候不需要每次都进行三次握手才实现，从性能和网络消耗来说 RPC 都具备了很好的优势。

RESTful API 基于 HTTP 的，也就是说每次调用服务都需要进行三次握手建立起通信才可以实现调用，当我们的并发量高的时候这就会浪费很多带宽资源。

服务对外的话采用 RESTful API 会比 RPC 更具备优势，因此看自己团队的服务是对内还是对外。

(4)、RPC 调用过程



( 图 https://swoft.org/img/rpc-3.png )

RPC 最主要的作用就是用于服务调用。我们从 RPC 的使用场景开山篇，对于单体架构和微服务的进行了一个描述。这个就是 RPC 的一个使用场景，也是最常用的一个使用场景。大家只有了解好 RPC 是什么、使用在什么场景才能更好的去使用。

Swoft 给我们提供了 RPC 的底层服务，我们并不需要去关心底层通讯细节和调用的过程。Swoft 通过定义接口，实现接口，启动 RPC Server 提供接口服务。我们只需要简单地写好几个类就可以实现一个简单RPC模块。

--

3、RPC server 服务命令

//// 在项目根目录执行如下命令

$ php bin/swoft rpc

Usage:

bin/swoft rpc:COMMAND [--opt ...] [arg ...]

//

Global Options:

--debug Setting the application runtime debug level(0 - 4)

--no-color Disable color/ANSI for message output

-h, --help Display this help message

-V, --version Show application version information

//

Commands:

reload Reload worker processes

restart Restart the rpc server

start Start the rpc server

stop Stop the currently running server

//

Example:

bin/swoft rpc:start Start the rpc server

bin/swoft rpc:stop Stop the rpc server

//

Rpc 的命令都在 Commands 中。reload 重新加载 worker 进程，restart 重启 RPC 服务器，start 启动 RPC 服务器，stop 停止 RPC 服务器。

(1)、前台运行

$ php bin/swoft rpc:start

Information Panel

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* RPC | Listen: 0.0.0.0:18307, type: TCP, mode: Process, worker: 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RPC server start success !

(2)、后台运行

$ php bin/swoft rpc:start -d

Information Panel

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* RPC | Listen: 0.0.0.0:18307, type: TCP, mode: Process, worker: 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RPC server start success !

--

4、配置参数

RPC 服务启动有单独启动和集成其它服务 (Http/Websocket) 两种方式，无论哪种方式都首先要在 bean.php 配置 RPC。

return [

'rpcServer' => [

'class' => ServiceServer::class,

'port' => 18308,

],

]

port 配置启动端口号；

setting 启动配置参数，对应 swooleServer->setting；

(1)、Http server 启动中集成 RPC 服务

return [

'httpServer' => [

'class' => HttpServer::class,

'port' => 18306,

'listener' => [

'rpc' => bean('rpcServer')

],

// ...

],

]

listener 单独监听一个 RPC 服务，且同时可以监听多个 RPC 服务。如果单独启动，无需其他配置直接可以启动。

--

5、声明服务

(1)、接口服务

定义接口并实现接口，才能提供 RPC 服务。

//

//// 官方应用中给出的 目录结构 如下

app/

Rpc/

- Lib/ // 服务的公共接口定义目录，里面通常只有php接口类

- Services/ // 具体的服务接口实现类，里面的类通常实现了 Lib 中定义的接口

当在多个服务中使用时， 要将lib库 app/Rpc/Lib 移到一个公共的 git 仓库里，然后各个服务通过 composer 来获取使用。

//

///1/ 定义接口

服务提供方定义好接口格式，存放到公共的 lib 库里面，服务调用方，加载 lib 库，就能使用接口服务，接口定义和普通接口完全一致。

\*\*

\* Class UserInterface

\*

\* @since 2.0

\*/

interface UserInterface

{

/\*\*

\* @param int $id

\* @param mixed $type

\* @param int $count

\*

\* @return array

\*/

public function getList(int $id, $type, int $count = 10): array;

/\*\*

\* @param int $id

\*

\* @return bool

\*/

public function delete(int $id): bool;

/\*\*

\* @return string

\*/

public function getBigContent(): string;

}

//

///2/ 接口实现

一个接口，会存在多种不同的实现，通过一个版本号来标识是哪个逻辑实现。

// @Service 注解

定义 RPC 服务类 @Service，version 定义接口版本，默认是 1.0。

//

// 示例：接口实现版本1。

/\*\*

\* Class UserService

\*

\* @since 2.0

\*

\* @Service()

\*/

class UserService implements UserInterface

{

/\*\*

\* @param int $id

\* @param mixed $type

\* @param int $count

\*

\* @return array

\*/

public function getList(int $id, $type, int $count = 10): array

{

return ['name' => ['list']];

}

/\*\*

\* @param int $id

\*

\* @return bool

\*/

public function delete(int $id): bool

{

return false;

}

/\*\*

\* @return string

\*/

public function getBigContent(): string

{

$content = Co::readFile(\_\_DIR\_\_ . '/big.data');

return $content;

}

}

//

// 示例：接口实现版本2。

/\*\*

\* Class UserServiceV2

\*

\* @since 2.0

\*

\* @Service(version="1.2")

\*/

class UserServiceV2 implements UserInterface

{

/\*\*

\* @param int $id

\* @param mixed $type

\* @param int $count

\*

\* @return array

\*/

public function getList(int $id, $type, int $count = 10): array

{

return [

'name' => ['list'],

'v' => '1.2'

];

}

/\*\*

\* @param int $id

\*

\* @return bool

\*/

public function delete(int $id): bool

{

return false;

}

/\*\*

\* @return string

\*/

public function getBigContent(): string

{

$content = Co::readFile(\_\_DIR\_\_ . '/big.data');

return $content;

}

}

不同的实现，需要定义不同的唯一版本号，如果存在相同，加载之后的服务会覆盖之前的服务。

--

6、RPC Client

服务调用方法，通过使用服务提供方法，提供的lib接口，调用接口实现服务，不需要了解实现细节。

(1)、RPC Client 配置参数

return [

'user' => [

'class' => ServiceClient::class,

'host' => '127.0.0.1',

'port' => 18307,

'setting' => [

'timeout' => 0.5,

'connect\_timeout' => 1.0,

'write\_timeout' => 10.0,

'read\_timeout' => 0.5,

],

'packet' => bean('rpcClientPacket')

],

'user.pool' => [

'class' => ServicePool::class,

'client' => bean('user')

],

];

如上定义了一个 user 服务，连接池配置参数和其它一样。

(2)、使用 RPC Client

/\*\*

\* Class RpcController

\*

\* @since 2.0

\*

\* @Controller()

\*/

class RpcController

{

/\*\*

\* rpc服务1.0

\* @Reference(pool="user.pool")

\*

\* @var UserInterface

\*/

private $userService;

/\*\*

\* rpc服务1.2

\* @Reference(pool="user.pool", version="1.2")

\*

\* @var UserInterface

\*/

private $userService2;

/\*\*

\* @RequestMapping("getList")

\*

\* @return array

\*/

public function getList(): array

{

$result = $this->userService->getList(12, 'type');

$result2 = $this->userService2->getList(12, 'type');

return [$result, $result2];

}

/\*\*

\* @RequestMapping("returnBool")

\*

\* @return array

\*/

public function returnBool(): array

{

$result = $this->userService->delete(12);

if (is\_bool($result)) {

return ['bool'];

}

return ['notBool'];

}

/\*\*

\* @RequestMapping()

\*

\* @return array

\*/

public function bigString(): array

{

$string = $this->userService->getBigContent();

return ['string'];

}

}

(3)、@Reference 注解

pool 指定使用哪个服务的连接池(使用哪个服务)；version 指定服务的版本。

(4)、非 Swoft 框架调用

默认消息协议是 json-rpc， 所以我们按照这个格式就可以了，需要注意的是，默认消息协议是以 \r\n\r\n 结尾的。

这里 method 的格式为 "{version}::{class\_name}::{method\_name}"。

{

"jsonrpc": "2.0",

"method": "{version}::{class\_name}::{method\_name}",

"params": [],

"id": "",

"ext": []

}

//

//// 示例：如果使用默认消息协议，可以按照如下方式进行封装

<?php

const RPC\_EOL = "\r\n\r\n";

function request($host, $class, $method, $param, $version = '1.0', $ext = []) {

$fp = stream\_socket\_client($host, $errno, $errstr);

if (!$fp) {

throw new Exception("stream\_socket\_client fail errno={$errno} errstr={$errstr}");

}

$req = [

"jsonrpc" => '2.0',

"method" => sprintf("%s::%s::%s", $version, $class, $method),

'params' => $param,

'id' => '',

'ext' => $ext,

];

$data = json\_encode($req) . RPC\_EOL;

fwrite($fp, $data);

$result = '';

while (!feof($fp)) {

$tmp = stream\_socket\_recvfrom($fp, 1024);

if ($pos = strpos($tmp, RPC\_EOL)) {

$result .= substr($tmp, 0, $pos);

break;

} else {

$result .= $tmp;

}

}

fclose($fp);

return json\_decode($result, true);

}

$ret = request('tcp://127.0.0.1:18307', \App\Rpc\Lib\UserInterface::class, 'getList', [1, 2], "1.0");

var\_dump($ret);

?>

--

7、1.0 RPC

如果系统之前使用的是 Swoft 1.0 RPC server，Swoft 2.0 定义了一种兼容 1.0 RPC 协议，使用很简单。

Available: >= v2.0.3

(1)、配置

使用 2.0 框架中调用 1.0 RPC server 提供的服务，首先必须配置 (app/bean.php) 1.0 RPC 协议。

return [

// ...

'user' => [

'class' => ServiceClient::class,

'host' => '127.0.0.1',

'port' => 8099,

'setting' => [

'timeout' => 0.5,

'connect\_timeout' => 1.0,

'write\_timeout' => 10.0,

'read\_timeout' => 0.5,

'package\_eof' => "\r\n",

],

'packet' => bean('rpcClientSwoftPacketV1')

],

// ...

];

host/port 配置 1.0 地址和端口即可；package\_eof 必须配置数据包结尾符，1.0 包结尾符是 \r\n；packet 必须配置使用 bean('rpcClientSwoftPacketV1') 1.0 打包器。

(2)、使用

以上配置完成后，就可以直接使用了。这里直接以调用 Swoft 1.x 的 App\Lib\DemoInterface 为例。

/\*\*

\* Class RpcController

\*

\* @since 2.0

\*

\* @Controller()

\*/

class RpcController

{

/\*\*

\* @Reference(pool="user.pool", version="0")

\*

\* @var DemoInterface

\*/

private $demoServcie;

/\*\*

\* @RequestMapping(route="swoftV1")

\*

\* @return array

\*/

public function swoftV1():array {

return [$this->demoServcie->getUser('1')];

}

}

调用 1.x RPC version 必须指定，因为 2.x 与 1.0 默认值不一样；不能调用 1.x 的 deferXxxx 方法，2.0 已经丢弃；2.x 里面调用的接口必须和 1.x 接口命名空间、类名以及方法名称参数完全一样。

--

--

五、TCP Server

https://swoft.org/documents/v2/core-components/tcp-server/

//

从 swoft 2.0.4 版本开始，提供经过 swoft 封装的 tcp 服务器实现。在原有 swoole server 的基础上，封装并细化功能使用。Available: >= v2.0.4。

--

1、安装

(1)、Composer 安装

composer require swoft/tcp-server

(2)、Git 仓库

Github tcp 数据协议 https://github.com/swoft-cloud/swoft-tcp

Github tcp-server https://github.com/swoft-cloud/swoft-tcp-server

--

2、参与贡献

欢迎参与贡献，您可以 fork 我们的开发仓库 swoft/component ( https://github.com/swoft-cloud/swoft-component )；修改代码然后发起 PR；阅读 提交代码( https://swoft.org/documents/v2/contribute/sub-codes/ ) 的注意事项。

--

3、功能特色

基于 swoft 的注解系统，使用方便快速；

提供统一的协议设置，同时支持 EOF 和 length 两种切包方式；

完善的数据收发解析，统一的上下文/请求与响应对象封装；

内置请求调度处理，可以像 http 一样细致地分发请求数据到不同的方法处理；

内置支持多种打包方式(json php token)，同时可以自由扩展。

--

4、配置服务

tcp server 的 host, port 等配置是都是完全可以自定义的。 配置需要编辑 app/bean.php 文件，下面列举了一些简单的配置，你也可以自由组合同时提供多种服务。

tcp server 的默认端口是 18309。

// ...

'tcpServer' => [

'class' => TcpServer::class,

'port' => 18307,

'on' => [

// 启用任务必须添加 task, finish 事件处理

SwooleEvent::TASK => bean(TaskListener::class),

SwooleEvent::FINISH => bean(FinishListener::class)

],

'listener' => [

// 引入 rpcServer

'rpc' => \bean('rpcServer')

],

'debug' => env('SWOFT\_DEBUG', 0),

/\* @see TcpServer::$setting \*/

'setting' => [

'log\_file' => alias('@runtime/swoole.log'),

// 任务需要配置 task worker

'task\_worker\_num' => 2,

'task\_enable\_coroutine' => true

],

],

可配置项用于 tcpServer bean 配置，除了 class 其他都是 TcpServer 的属性。

class 指定 tcp server 的 bean 类，默认即是 Swoft\Tcp\Server\TcpServer::class。

port 指定 tcp server 的端口。

listener 指定其他一同启动的服务，添加端口服务监听，可以多个，rpc 启动 RPC 服务。

on 配置监听的事件，注册 swoole 事件、设置对应事件的处理监听。

setting 这里是参考 Swoole Server配置选项( https://wiki.swoole.com/wiki/page/274.html )。

pidFile 设置进程 pid 文件 位置，默认值 @runtime/swoft-tcp.pid。

mode 运行的模式，参考 Swoole Server 构造函数 第三个参数( https://wiki.swoole.com/wiki/page/14.html )。

type 指定 Socket 的类型，支持 TCP、UDP、TCP6、UDP6、UnixSocket Stream/Dgram 等 Swoole Server 构造函数 第四个参数( https://wiki.swoole.com/wiki/page/14.html )。

--

5、协议配置

通常你只需配置好协议的分包方式，内部的细节配置会自动同步设置到 TcpServer。

/\*\* @see \Swoft\Tcp\Protocol \*/

'tcpServerProtocol' => [

'type' => \Swoft\Tcp\Packer\SimpleTokenPacker::TYPE,

// 'openEofCheck' => true, // Defalut use EOF check

// 'openLengthCheck' => true,

],

type string 默认的数据打包器的类型，默认是 token-text。

packers array 可用的数据打包器的列表，内置了 json php token-text 三种。

packageMaxLength int 同 swoole 的 package\_max\_length 默认 81920。

openEofCheck bool 同 swoole 的 open\_eof\_check 默认 true。

openLengthCheck bool 同 swoole 的 open\_length\_check，总是与 openEofCheck 相反，默认 false。

--

6、添加 RPC 服务

如果你想运行 tcp server 时，同时启动 RPC Server 服务。

// ...

'tcpServer' => [

'listener' => [

'rpc' => \bean('rpcServer') // 引入 rpcServer

],

],

'rpcServer' => [

'class' => ServiceServer::class,

'port' => 18308,

],

--

7、启动与管理

在项目目录下执行如下命令可以看到 tcp server 的管理命令. 跟 http server 的管理命令一致。

$ php bin/swoft tcp

Description:

There some commands for manage the tcp server

//

Usage:

tcp:{command} [arguments] [options]

//

Commands:

start Start the tcp server

stop Stop the running server

restart Restart the running server

//

Options:

-h, --help Show help of the command group or specified command action

//

(1)、前台启动

php bin/swoft tcp:start

注意：tcp服务器的默认端口是 18309。

(2)、后台启动

php bin/swoft tcp:start -d

(3)、重新启动

php bin/swoft tcp:restart

--

8、TCP 事件通知

通常情况下，我们无需关心 tcp server 相关的 connect close 事件。但 swoft 内部都是监听并触发了框架内部定义的相关事件，你同样可以监听并处理一些逻辑。

(1)、事件列表

<?php declare(strict\_types=1);

namespace Swoft\Tcp\Server;

/\*\*

\* Class TcpServerEvent

\*

\* @since 2.0

\*/

final class TcpServerEvent

{

/\*\*

\* On connect

\*/

public const CONNECT = 'swoft.tcp.server.connect';

/\*\*

\* On connect error

\*/

public const CONNECT\_ERROR = 'swoft.tcp.server.connect.error';

/\*\*

\* On receive

\*/

public const RECEIVE = 'swoft.tcp.server.receive';

/\*\*

\* On receive error

\*/

public const RECEIVE\_ERROR = 'swoft.tcp.server.receive.error';

/\*\*

\* On close

\*/

public const CLOSE = 'swoft.tcp.server.close';

/\*\*

\* On close error

\*/

public const CLOSE\_ERROR = 'swoft.tcp.server.close.error';

}

?>

(2)、监听事件

跟其他事件一样，直接通过 @Lisenter 监听对应事件名，就可以处理相关逻辑了。

<?php declare(strict\_types=1);

namespace App\Listener;

use Swoft\Event\Annotation\Mapping\Listener;

use Swoft\Event\EventHandlerInterface;

use Swoft\Event\EventInterface;

use Swoft\Tcp\Server\TcpServerEvent;

/\*\*

\* Class UserSavingListener

\*

\* @since 2.0

\*

\* @Listener(TcpServerEvent::CONNECT)

\*/

class TcpConnectListener implements EventHandlerInterface

{

/\*\*

\* @param EventInterface $event

\*/

public function handle(EventInterface $event): void

{

/\* @var int $fd \*/

$fd = $event->getTarget();

var\_dump(

$event->getParam(0), // Swoole server

$event->getParam(1), // reactorId

);

}

}

?>

--

9、TCP 控制器

与 http server 类似，tcp server 中也使用对应的控制器来处理系统分发的数据请求。tcp server 新增两个注解 @TcpController 和 @TcpMapping，由他们定义 tcp 控制器和处理方法。自 v2.0.7 开始支持添加全局中间件和指定控制器或方法的中间件。如果你不想使用内置的路由处理，请跳过这一节，查看下面的 自定义接收处理。

(1)、@TcpController 注解

类注解 @TcpControler 标记当前类是一个 Tcp 控制器。

注解类：Swoft\Tcp\Server\Annotation\Mapping\TcpController；

作用范围：CLASS；

拥有属性：prefix string 数据路由前缀，为空自动解析类名称为前缀；middlewares string[] 中间件，自 v2.0.7 开始支持中间件。

(2)、@TcpMapping

方法注解 @TcpMapping 标记具体的数据处理方法，类似于 http 控制器里的 action。

注解类：Swoft\Tcp\Server\Annotation\Mapping\TcpMapping；

作用范围：METHOD；

拥有属性：route string 命令名称，为空自动使用方法名称；root bool 命令名称是否是顶级命令，是则不会再添加控制器的 prefix，默认 false；middlewares string[] 中间件，自 v2.0.7 开始支持中间件。

自动解析 TcpControler 的前缀时，会自动尝试去除 Controler 部分，eg: DemoController 得到 demo；通常，完整的 tcp 命令是 上面的 preifx 和 route 由点拼接而成 PREFIX.ROUTE，eg: demo.index；当 TcpMapping.root 为 true 时，完整命令直接是 TcpMapping.route。

(3)、编写控制器

<?php declare(strict\_types=1);

namespace App\Tcp\Controller;

use App\Tcp\Middleware\DemoMiddleware;

use Swoft\Tcp\Server\Annotation\Mapping\TcpController;

use Swoft\Tcp\Server\Annotation\Mapping\TcpMapping;

use Swoft\Tcp\Server\Request;

use Swoft\Tcp\Server\Response;

/\*\*

\* Class DemoController

\*

\* @TcpController(middlewares={DemoMiddleware::class})

\*/

class DemoController

{

/\*\*

\* @TcpMapping("list", root=true)

\* @param Response $response

\*/

public function list(Response $response): void

{

$response->setData('[list]allow command: list, echo, demo.echo');

}

/\*\*

\* @TcpMapping("echo")

\* @param Request $request

\* @param Response $response

\*/

public function index(Request $request, Response $response): void

{

$str = $request->getPackage()->getDataString();

$response->setData('[demo.echo]hi, we received your message: ' . $str);

}

/\*\*

\* @TcpMapping("strrev", root=true)

\* @param Request $request

\* @param Response $response

\*/

public function strRev(Request $request, Response $response): void

{

$str = $request->getPackage()->getDataString();

$response->setData(\strrev($str));

}

/\*\*

\* @TcpMapping("echo", root=true)

\* @param Request $request

\* @param Response $response

\*/

public function echo(Request $request, Response $response): void

{

$str = $request->getPackage()->getDataString();

$response->setData('[echo]hi, we received your message: ' . $str);

}

}

?>

服务端代码已经编写好了。这里我们使用默认的配置 EOF 分包方式，数据协议格式也使用默认的 SimpleTokenPacker::TYPE。重新启动我们的 tcp server php bin/swoft tcp:start，接下来讲述如何与我们的 tcp server进行通信交互。

--

10、客户端通信

你可以直接使用 swoole 提供的 Swoole\Coroutine\Client 作为 tcp 客户端，快速的对接 swoft 的 tcp sever。

// 为了分包和数据解析与 tcp server 保持一致，你需要依赖 tcp 协议包

composer require swoft/tcp

开始之前，首先你得确认你已经启动了 tcp server 端，并且保持客户端与服务端的 协议设置是一致的。

//

///1/ 示例：Swoft 使用

<?php declare(strict\_types=1);

namespace App\Command;

use Swoft\Tcp\Protocol;

use Swoole\Coroutine\Client;

use Swoft\Console\Helper\Show;

use Swoft\Console\Input\Input;

use Swoft\Console\Output\Output;

use const SWOOLE\_SOCK\_TCP;

// ...

public function tcpTest(Input $input, Output $output): void

{

$proto = new Protocol();

// If your tcp server use length check.

// $proto->setOpenLengthCheck(true);

var\_dump($proto->getConfig());

$host = '127.0.0.1';

$port = 18309;

$client = new Client(SWOOLE\_SOCK\_TCP);

// Notice: config client

$client->set($proto->getConfig());

if (!$client->connect((string)$host, (int)$port, 5.0)) {

$code = $client->errCode;

/\*\* @noinspection PhpComposerExtensionStubsInspection \*/

$msg = socket\_strerror($code);

$output->error("Connect server failed. Error($code): $msg");

return;

}

// Send message $msg . $proto->getPackageEOf()

if (false === $client->send($proto->packBody($msg))) {

/\*\* @noinspection PhpComposerExtensionStubsInspection \*/

$output->error('Send error - ' . socket\_strerror($client->errCode));

return;

}

// Recv response

$res = $client->recv(2.0);

if ($res === false) {

/\*\* @noinspection PhpComposerExtensionStubsInspection \*/

$output->error('Recv error - ' . socket\_strerror($client->errCode));

return;

}

if ($res === '') {

$output->info('Server closed connection');

return;

}

// unpack response data

[$head, $body] = $proto->unpackData($res);

$output->prettyJSON($head);

$output->writef('<yellow>server</yellow>> %s', $body);

}

?>

//

///2/ 示例：非Swoft 使用

这里使用的json数据，因此你需要将服务端 tcpServerProtocol 的 type 配置为 json。

<?php

const PKG\_EOF = "\r\n\r\n";

function request(string $host, string $cmd, $data, $ext = []) {

$fp = stream\_socket\_client($host, $errno, $errstr);

if (!$fp) {

throw new Exception("stream\_socket\_client fail errno={$errno} errstr={$errstr}");

}

$req = [

'cmd' => $cmd,

'data' => $data,

'ext' => $ext,

];

$data = json\_encode($req) . PKG\_EOF;

fwrite($fp, $data);

$result = '';

while (!feof($fp)) {

$tmp = stream\_socket\_recvfrom($fp, 1024);

if ($pos = strpos($tmp, PKG\_EOF)) {

$result .= substr($tmp, 0, $pos);

break;

} else {

$result .= $tmp;

}

}

fclose($fp);

return json\_decode($result, true);

}

$ret = request('tcp://127.0.0.1:18309', 'echo', 'i an client');

var\_dump($ret);

?>

测试通信，你可以复制上面的示例代码，新建一个 php 文件来运行测试。当然，最方便直接的就是使用我们 devtool 包里提供的 dclient:tcp 工具命令。

运行：php bin/swoft dclient:tcp -h 查看命令帮助。

--

11、自定义接收处理

如果你不想使用swoft自带的路由处理，swoft 也可以支持自定义数据接收后的处理逻辑。自定义接收处理不支持中间件，处理流程都由用户自己控制。

// 首先，你需要关闭内置的路由解析处理

'tcpDispatcher' => [

'enable' => false,

]

//

// 然后，监听 TcpServerEvent::RECEIVE\_BEFORE 事件

<?php declare(strict\_types=1);

namespace App\Listener;

use Swoft\Event\Annotation\Mapping\Listener;

use Swoft\Event\EventHandlerInterface;

use Swoft\Event\EventInterface;

use Swoft\Tcp\Server\TcpServerEvent;

/\*\*

\* Class TcpReceiveListener

\*

\* @since 2.0

\*

\* @Listener(TcpServerEvent::RECEIVE\_BEFORE)

\*/

class TcpReceiveListener implements EventHandlerInterface

{

/\*\*

\* @param EventInterface $event

\*/

public function handle(EventInterface $event): void

{

/\* @var int $fd \*/

$fd = $event->getTarget();

var\_dump(

$event->getParam(0), // Swoole server

$event->getParam(1), // reactorId

);

// 接收到的数据

$content = context()->getRequest()->getRawData();

// 自定义解析数据和处理

// do something ....

}

}

?>

--

--

六、进程组件

https://swoft.org/documents/v2/core-components/process/

//

Swoft 框架中封装了一套经常操作方式，替换 PHP 的 pcntl，PHP 自带的 pcntl，存在很多不足，如：pcntl 没有提供进程间通信的功能；pcntl 不支持重定向标准输入和输出；pcntl 只提供了 fork 这样原始的接口，容易使用错误。

--

1、安装

Swoft 基于 Swoole 进程操作封装，功能与 Swoole 完全一样，建议开发者使用 Swoft 的封装操作，方便框架一起迭代升级。2.0.4+ 支持且需要安装 swoft-process 组件。

composer require swoft/process

--

2、方法列表

// 所有操作方法，全部在 Swoft\Process\Process 里面

new

\_\_construct

start

name

exec

write

read

setTimeout

setBlocking

useQueue

statQueue

freeQueue

exportSocket

push

pop

close

exit

kill

wait

daemon

signal

alarm

setAffinity

--

3、用户进程

Http/RPC/Websocket/TCP 等服务有些业务场景，需要一个后台运行进程去监控、上报或者其它特殊操作，此时可以在相应服务启动的时候，添加一个用户自定义工作进程，来实现。 自定义用户进程与服务一起启动，服务关闭一起退出，如果自定义用户进程被意外关闭，服务会重新启动一个新的自定义用户进程，保证自定义用户进程一直存在。2.0.4+ 支持且需要安装 swoft-process 组件。

(1)、声明用户进程

使用自定义用户进程之前，必须定义用户进程，如下定义一个监控上报信息的用户进程为例。

///1/ 示例：自定义用户进程入口。

<?php declare(strict\_types=1);

namespace App\Process;

use App\Model\Logic\MonitorLogic;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Bean\Annotation\Mapping\Inject;

use Swoft\Db\Exception\DbException;

use Swoft\Process\Process;

use Swoft\Process\UserProcess;

/\*\*

\* Class MonitorProcess

\*

\* @since 2.0

\*

\* @Bean()

\*/

class MonitorProcess extends UserProcess

{

/\*\*

\* @Inject()

\*

\* @var MonitorLogic

\*/

private $logic;

/\*\*

\* @param Process $process

\*

\* @throws DbException

\*/

public function run(Process $process): void

{

$this->logic->monitor($process);

}

}

?>

自定义用户进程必须实现 Swoft\Process\UserProcess 接口，自定义用户进程必须使用 @Bean 标记为一个 bean 对象。

//

///2/ 示例：业务处理。

<?php declare(strict\_types=1);

namespace App\Model\Logic;

use App\Model\Entity\User;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Db\Exception\DbException;

use Swoft\Log\Helper\CLog;

use Swoft\Process\Process;

use Swoft\Redis\Redis;

use Swoole\Coroutine;

/\*\*

\* Class MonitorProcessLogic

\*

\* @since 2.0

\*

\* @Bean()

\*/

class MonitorLogic

{

/\*\*

\* @param Process $process

\*

\* @throws DbException

\*/

public function monitor(Process $process): void

{

$process->name('swoft-monitor');

while (true) {

$connections = context()->getServer()->getSwooleServer()->connections;

CLog::info('monitor = ' . json\_encode($connections));

// Database

$user = User::find(1)->toArray();

CLog::info('user='.json\_encode($user));

// Redis

Redis::set('test', 'ok');

CLog::info('test='.Redis::get('test'));

Coroutine::sleep(3);

}

}

}

?>

自定义用户进程里面，开发者必须实现类似 while(true) 的业务，且里面可以直接使用 Swoft 封装好的所有IO操作，比如数据库、缓存、RPC，以及其它非 Swoft 封装的协程操作。

(2)、配置

定义好了用户进程，必须配置才会有效，Http/RPC/Websocket/TCP 服务配置自定义进程都一样，这里以如上自定义的用户进程配置为例。

return [

'httpServer' => [

'class' => HttpServer::class,

'port' => 18306,

'listener' => [

'rpc' => bean('rpcServer')

],

'process' => [

'monitor' => bean(MonitorProcess::class)

],

// ...

],

];

这里以注入的方式配置了一个自定义用户进程，名称为 monitor。如果配置成功，服务启动后，用户进程里面的业务会自动执行，无需其它操作。

--

4、进程池

进程池一般用于需要程序一直运行的场景，比如队列消费，数据计算。Swoft 框架中，基于 Swoole 进程池模型再次封装，便于开发者快速简单的使用进程池。2.0.4+ 支持且需要安装 swoft-process 组件。

(1)、进程池配置

组件安装成功后，默认不需要配置也可以使用，配置如下。

//// app/bean.php

return [

'processPool' => [

'class' => \Swoft\Process\ProcessPool::class,

'workerNum' => 3

]

];

workerNum worker 进程数；ipcType IPC类型；coroutine 是否开启协程，默认是开启。

Swoft 框架中必须是协程模式运行，协程模式运行下，可以使用 Swoft 封装的所有 IO 操作，以及其它非 Swoft 协程操作。

(2)、@Process 注解

@Process 标记类是一个进程池处理流程。

属性列表：workerId int|array 绑定的进程ID，可以是单个或者数组。默认情况，是绑定到其它未绑定的进程。

(3)、声明工作进程

配置好之后，就是声明工作进程。如下以 workerNum = 3，定义三个 worker 进程为例。

//

///1/ worker 进程1

<?php declare(strict\_types=1);

namespace App\Process;

use Swoft\Log\Helper\CLog;

use Swoft\Process\Annotation\Mapping\Process;

use Swoft\Process\Contract\ProcessInterface;

use Swoole\Coroutine;

use Swoole\Process\Pool;

/\*\*

\* Class Worker1Process

\*

\* @since 2.0

\*

\* @Process(workerId=0)

\*/

class Worker1Process implements ProcessInterface

{

/\*\*

\* @param Pool $pool

\* @param int $workerId

\*/

public function run(Pool $pool, int $workerId): void

{

while (true) {

CLog::info('worker-' . $workerId);

Coroutine::sleep(3);

}

}

}

?>

worker 进程必须实现 Swoft\Process\Contract\ProcessInterface 接口；开发者业务必须自己实现类似 while(true) 逻辑；@Process 注解 workerId=0 表示第1个进程绑定这个处理逻辑流程。

//

///2/ worker 进程2和进程3

<?php declare(strict\_types=1);

namespace App\Process;

use App\Model\Entity\User;

use Swoft\Db\Exception\DbException;

use Swoft\Log\Helper\CLog;

use Swoft\Process\Annotation\Mapping\Process;

use Swoft\Process\Contract\ProcessInterface;

use Swoft\Redis\Redis;

use Swoole\Coroutine;

use Swoole\Process\Pool;

/\*\*

\* Class Worker2Process

\*

\* @since 2.0

\*

\* @Process(workerId={1,2})

\*/

class Worker2Process implements ProcessInterface

{

/\*\*

\* @param Pool $pool

\* @param int $workerId

\*

\* @throws DbException

\*/

public function run(Pool $pool, int $workerId): void

{

while (true) {

// Database

$user = User::find(1)->toArray();

CLog::info('user='.json\_encode($user));

// Redis

Redis::set('test', 'ok');

CLog::info('test='.Redis::get('test'));

CLog::info('worker-' . $workerId.' context='.context()->getWorkerId());

Coroutine::sleep(3);

}

}

}

?>

worker 进程必须实现 Swoft\Process\Contract\ProcessInterface 接口；开发者业务必须自己实现类似 while(true) 逻辑；@Process 注解 workerId={1,2} 表示第2个进程和第3个进程，同时绑定这个处理逻辑流程。

workerId 绑定ID 是从 0 开始算起, workerId 如果不写默认情况，当前程序流程绑定到其它未绑定的进程。

(4)、启动进程池

配置和声明工作进程完成后，就是启动进程池，启动进程池和其它服务启动很类似。

///1/ 前台启动

php bin/swoft process:start

//

///2/ 后台启动

php bin/swoft process:start -d

2019/07/16-09:44:34 [INFO] Swoft\SwoftApplication:setSystemAlias(496) Set alias @base=/data/www/swoft

... ...

2019/07/16-09:44:44 [INFO] Swoft\Processor\ConsoleProcessor:handle(39) Console command route registered (group 13, command 40)

//

///3/ 重启所有 worker 进程

php bin/swoft process:reload

2019/07/16-09:45:52 [INFO] Swoft\SwoftApplication:setSystemAlias(496) Set alias @base=/data/www/swoft

... ...

2019/07/16-09:45:59 [INFO] Swoft\Processor\ConsoleProcessor:handle(39) Console command route registered (group 13, command 40)

Server bin/swoft is reloading

Process pool bin/swoft reload success

//

///4/ 停止服务

php bin/swoft process:stop

2019/07/16-09:46:35 [INFO] Swoft\SwoftApplication:setSystemAlias(496) Set alias @base=/data/www/swoft

... ...

2019/07/16-09:46:45 [INFO] Swoft\Processor\ConsoleProcessor:handle(39) Console command route registered (group 13, command 40)

Stopping .. Successful!

--

--

七、任务组件

https://swoft.org/documents/v2/core-components/task/

//

某些场景对主流程没有依赖，可以直接使用任务来实现类似这些功能。框架为开发者提供了 协程 和 异步 两种任务。切记无论是 协程任务 还是 异步任务，任务里面操作都 只支持协程，且只能使用框架封装的所有 IO 操作(数据库、缓存…)。

协程任务投递任务的时候不会阻塞主进程相当于一次协程调用，一般用于需要等待任务结果返回的场景。

有些场景主流程并不关心的任务执行的结果，此时就可以使用异步任务。

--

1、配置

任务配置参数，可以直接在对应的 Server->setting 配置即可，如果要启用任务更简单，Server 新增一个 on 事件。

(1)、协程任务

//// Http Server 配置开启任务为例

return [

'httpServer' => [

// ...

'on' => [

SwooleEvent::TASK => \bean(TaskListener::class), // Enable task must task and finish event

SwooleEvent::FINISH => \bean(FinishListener::class)

],

/\* @see HttpServer::$setting \*/

'setting' => [

'task\_worker\_num' => 12,

'task\_enable\_coroutine' => true

]

],

]

task\_enable\_coroutine 必须为 true；task 事件和 finish 事件必须配置，且为 TaskListener::class 和 FinishListener::class。

任务配置与启用，在 Http Server / Rpc Server / Websocket Server 都完全一样，启用任务需要监听 task finish 两个事件。

(2)、同步阻塞任务

Swoft 不仅提供 协程任务，并且支持 同步任务，同步任务 和 协程任务 只能选择一种运行，两种不能同时存在。同步任务只需配置 task 事件，不支持异步 finish 事件。官方建议使用 协程任务 实现业务，如果需要通过任务实现 MongoDB、PostgreSQL 类似这种场景才使用同步任务。2.0.4+ 支持。

//

//// 如下以 Http-server 为例

return [

'httpServer' => [

// ...

'on' => [

SwooleEvent::TASK => bean(SyncTaskListener::class), // Enable sync task

],

/\* @see HttpServer::$setting \*/

'setting' => [

'task\_worker\_num' => 6,

'task\_enable\_coroutine' => false

]

],

]

task\_enable\_coroutine 必须设置为 false，task 事件必须是 SyncTaskListener::class。

同步阻塞任务，不能直接使用框架提供的所有 IO 操作(数据库、缓存、RPC等等)以及应用日志，控制器日志可以使用。 同步阻塞任务的定义和使用与协程任务一样，但是没有上下文。

--

2、声明任务

使用任务前，必须定义任务，定义任务很简单。如下定一个任务。

/\*\*

\* Class TestTask

\*

\* @since 2.0

\*

\* @Task(name="testTask")

\*/

class TestTask

{

/\*\*

\* @TaskMapping(name="list")

\*

\* @param int $id

\* @param string $default

\*

\* @return array

\*/

public function getList(int $id, string $default = 'def'): array

{

return [

'list' => [1, 3, 3],

'id' => $id,

'default' => $default

];

}

/\*\*

\* @TaskMapping()

\*

\* @param int $id

\*

\* @return bool

\*/

public function delete(int $id): bool

{

if ($id > 10) {

return true;

}

return false;

}

}

?>

(1)、@Task 注解

标记类是一个任务，name 指定任务名称，默认全路径类名。

(2)、@TaskMapping 注解

映射名称，name 名称映射，默认就类的方法名称。

//

被 @Task 标记类的每个方法就是一个任务，如果方法没有使用 @TaskMapping 注解，不会解析成任务。

Task投递前，会经过 Swoft\Task\Packet::pack() 方法被json\_encode，因此，投递entity时，task获得的是entity的数组。同时，投递无法被json\_encode的参数会导致报错 （如果是异步任务且没有开启额外的日志，可能效果是task直接结束而没有报错信息）。

--

3、任务投递

(1)、协程任务投递

协程任务投递提供了两种方式，单个投递和批量投递，单个投递是在批量投递的基础之上封装的。如下协程任务投递(备注：co/async 方法第三个参数传递为数组，当任务声明的参数为数组时，co/async 第三个参数需要 [[xxx,xxx,xxx]]使用)。

//

///1/ 单个协程任务投递

Task::co(string $name, string $method, array $params = [], float $timeout = 3, array $ext = []);

单个任务投递，返回数据和任务方法返回的数据完全一致类型也一样。name 投递任务任务名称；method 投递任务的方法名称；params 任务传递的参数即是任务方法的参数，数组格式传递；timeout 超时时间，默认 3s 超时；ext 任务扩展信息，会传递给任务进程里面。

//

///2/ 多个协程任务投递

$tasks = [

[

'taskName',

'method',

['params']

]

];

Task::cos(array $tasks, float $timeout = 3, array $ext = []);

多个任务投递，tasks 多个任务集合，格式如上；timeout 超时时间，默认 3s 超时；ext 任务扩展信息，会传递给任务进程里面。

//

///3/ 示例：协程任务投递

use Swoft\Task\Task;

// 协程投递

$data = Task::co('testTask', 'list', [12]);

// 协程投递

$result = Task::co('testTask', 'delete', [12]);

//

///4/ 任务 Request

namespace Swoft\Task;

class Request implements RequestInterface

{

// ...

}

getServer 获取任务 Server 信息；

getTaskId 获取任务 ID，对应 Swoole 任务 ID；

getSrcWorkerId 任务来自的 workerId；

getData 投递任务的原始数据；

getName 任务名称；

getMethod 任务方法；

getParams 任务参数；

getExt 任务扩展信息；

getExtKey 根据 key 快速获取用户信息；

getType 任务类型；

getTaskUniqid 任务全局唯一ID；

(2)、异步任务投递

Task::async(string $name, string $method, array $params = [], array $ext = [], int $dstWorkerId = -1, callable $fallback = null);

异步任务投递，返回一个全局唯一的任务ID。name 投递任务任务名称；method 投递任务的方法名称；params 任务传递的参数即是任务方法的参数，数组格式传递；ext 任务扩展信息，会传递给任务进程里面；dstWorkerId 投递的进程 workerId，默认底层按需选择进程 workerId。

异步任务一般用于不需要结果的场景且异步区执行，不影响主流程。如下异步任务投递。

//

///1/ 异步任务投递

use Swoft\Task\Task;

$data = Task::async('testTask', 'list', [12]);

$data = Task::async('testTask', 'delete', [12]);

//

///2/ 异步任务结果

有很多情况不需要关注异步任务处理结果，但是也有部分场景需要关注异步任务处理结果，框架为开发者提供了一种事件监听的方式来处理异步任务结果。此事件和普通事件完全一样。

// 如下定义是事件监听

use Swoft\Event\Annotation\Mapping\Listener;

use Swoft\Event\EventHandlerInterface;

use Swoft\Event\EventInterface;

use Swoft\Log\Helper\CLog;

use Swoft\Task\TaskEvent;

/\*\*

\* Class FinishListener

\*

\* @since 2.0

\*

\* @Listener(event=TaskEvent::FINISH)

\*/

class FinishListener implements EventHandlerInterface

{

/\*\*

\* @param EventInterface $event

\*/

public function handle(EventInterface $event): void

{

CLog::info(\context()->getTaskUniqid());

}

}

?>

事件必须监听 TaskEvent::FINISH；如果需要获取数据可以从上下文中获取，注意此时获取的是任务完成的上下文对象与任务上下对象不一样。

--

4、任务上下文

有些场景需要在任务里面拿到任务的详细信息，这些信息全部在上下文里面。此时可以使用全局函数 context() 获取 Swoft\Task\TaskContext 上下文对象。上下文提供两个方法，分别获取 Swoft\Task\Request 与 Swoft\Task\Response 对象，里面包含投递任务的所有信息。

$request = context()->getRequest();

$response = context()->getRespone();

提示：一定要在任务里面获取上下文，否则获取的是其它环境的上下文。

(1)、异步任务完成上下文

在异步任务完成监听器里面可以通过 content() 全局函数获取上下文 Swoft\Task\FinishContext 对象。

$taskData = context()->getTaskData();

$taskId = context()->getTaskId();

$taskUniqid = context()->getTaskUniqid();

$server = context()->getServer();

getTaskData 任务处理的结果内容；getTaskId 任务 ID，对应 Swoole 任务 ID；getTaskUniqid 全局任务唯一ID，框架生成，与任务投递时的全局任务ID一样；getServer 获取任务 Server 相关信息。

--

5、定时任务

在某些情况下需要定时的去执行某些任务，通常我们会使用 Linux 系统自带的 Crontab 去定时地执行我们编写好的脚本，但是这样及其不方便，首先 Linux 系统默认的 Crontab 最小单位只能支持到分钟，无法支持秒级任务，其次，如果我们重新编写脚本，则不能很方便友好地复用框架内的资源，如 Mysql 连接资源，框架中的各种类库。针对以上问题，框架为我们内置了一个 Crontab 组件，可以支持秒级任务。可用自 >= v2.0.5。

(1)、安装

//// 使用定时任务前，必须安装，安装如下

composer require swoft/crontab

(2)、注解

在 Swoft 中，定时任务的使用非常的简单，只需要使用相关注解定义你的任务类即可。

//

///1/ @Scheduled 注解

用于声明定时任务，如果是声明定时任务类，则必须使用此注解。参数name 任务类的名称，为空则为此类的完整 namespace 路径。使用示例：@Scheduled()、@Scheduled("taskName")、@Scheduled(name="taskName")。

//

///2/ @Cron 注解

声明需要运行的方法，如果没有使用此注解，则该方法不会被运行。参数value 任务的 Crontab 表达式，支持到秒。使用示例：@Cron("\* \* \* \* \* \*")、@Cron(value="\* \* \* \* \* \*")，表达式可简写，例如一个每秒都要执行的任务则可定义为 @Cron("\*")。

//

///3/ Cron格式说明

\* \* \* \* \* \*

- - - - - -

| | | | | |

| | | | | +----- day of week (0 - 6) (Sunday=0)

| | | | +----- month (1 - 12)

| | | +------- day of month (1 - 31)

| | +--------- hour (0 - 23)

| +----------- min (0 - 59)

+------------- sec (0-59)

\* \* \* \* \* \* 表示每秒执行一次。

0 \* \* \* \* \* 表示每分钟的第0秒执行一次，即每分钟执行一次。

0 0 \* \* \* \* 表示每小时的0分0秒执行一次，即每小时执行一次。

0/10 \* \* \* \* \* 表示每分钟的第0秒开始每10秒执行一次。

10-20 \* \* \* \* \* 表示每分钟的第10-20秒执行一次。

10,20,30 \* \* \* \* \* 表示每分钟的第10,20,30秒各执行一次。

(3)、声明定时任务

在 Swoft 中使用定时任务相当的简单，只需两步操作，声明定时任务 和 配置启用，这两部操作都相当的简单，我们先来看声明任务。

<?php declare(strict\_types=1);

namespace App\Crontab;

use Swoft\Crontab\Annotaion\Mapping\Cron;

use Swoft\Crontab\Annotaion\Mapping\Scheduled;

/\*\*

\* Class CronTask

\*

\* @since 2.0

\*

\* @Scheduled()

\*/

class CronTask

{

/\*\*

\* 每秒执行

\* @Cron("\* \* \* \* \* \*")

\*/

public function secondTask()

{

printf("second task run: %s ", date('Y-m-d H:i:s', time()));

}

/\*\*

\* 每分之行

\* @Cron("0 \* \* \* \* \*")

\*/

public function minuteTask()

{

printf("minute task run: %s ", date('Y-m-d H:i:s', time()));

}

}

?>

(4)、配置启用

定时任务的执行是基于 Swoft 的 进程,所以我们需要和使用 用户进程 的方式一样在配置中启用 Crontab 组件的自定义进程即可。

return [

'httpServer' => [

// ...

'process' => [

'crontab' => bean(Swoft\Crontab\Process\CrontabProcess::class)

],

// ...

],

];

如上我们就配置成功了服务启动后，我们的定时任务进程也会随之启动。

(5)、手动执行

除了定时执行我们设置好的任务外，我们还可以在业务代码中直接手动执行我们的定时任务，方法如下。

$crontab = BeanFactory::getBean("crontab");

$crontab->execute("testCrontab", "method");

通过 Bean 容器拿到 crontab 管理器，然后直接使用 execute($beanName,$methodName) 方法，此方法有两个参数，$beanName 就是传入在 @Scheduled() 注解中设置的名字，$methodName 则是传入 @Scheduled() 标注的类中，@Cron() 所标注的方法。

--

--

八、日志

https://swoft.org/documents/v2/core-components/log/

//

框架日志由控制台日志和应用日志组成，控制台日志一般用于调试打印，应用日志用于记录开发者的业务日志和框架运行日志。无论是控制台日志还是应用日志都提供了很多灵活的参数，方便开发者。

swoft/log 基于著名的 php 日志库 monolog 扩展而来。你可以很方便地添加自定义 handler 或 processor。

--

1、控制台日志

Swoft 提供简便的控制台日志使用，便于在开发时打印调试信息。

(1)、控制台日志配置

启动应用里面 (app\Application.php) 重写父类方法，可以覆盖配置控制台日志参数。

namespace App;

use Swoft\SwoftApplication;

/\*\*

\* Class Application

\*

\* @since 2.0

\*/

class Application extends SwoftApplication

{

public function getCLoggerConfig(): array

{

return [

'name' => 'swoft',

'enable' => true,

'output' => true,

'levels' => 'info,error',

'logFile' => ''

];

}

}

name 名称；enable 是否开启；output 是否打印到控制台；levels 输入日志的级别，为空全部输出，具体日志级别配置值，可以引用 Logger::NOTICE/...；logFile 控制台日志默认打印到控制台，也可以配置路径，同时写到指定文件。

swoft 2.0.3 levels 修改成字符串，方便开发者覆盖框架默认配置。

(2)、控制台日志使用

控制台日志可以直接使用框架提供的 CLog 类里面的静态方法操作。

每个日志级别方法都可以传递参数，底层是一个 sprintf() 函数封装；特殊的 debug 日志级别，需要开启 SWOFT\_DEBUG 才会显示；框架内置不同级别不同颜色。

// debug

CLog::debug('debug');

// info

CLog::info('info');

// warning

CLog::warning('warning');

// error

CLog::error('error');

// 2019/05/12-07:02:57 [DEBUG] Swoft\Processor\ConsoleProcessor:handle(33) debug

// 2019/05/12-07:02:57 [INFO] Swoft\Processor\ConsoleProcessor:handle(33) info

// 2019/05/12-07:02:57 [WARNING] Swoft\Processor\ConsoleProcessor:handle(33) warning

// 2019/05/12-07:02:57 [ERROR] Swoft\Processor\ConsoleProcessor:handle(33) error

(3)、关闭信息

默认情况下，启动时会打印一些启动信息到控制台。如果你的 .env 开启了 SWOFT\_DEBUG=1 将会看到更多详细的启动与加载信息。

// 如果你想关闭这些信息，可以在 app/Application 添加

public function getCLoggerConfig(): array

{

$config = parent::getCLoggerConfig();

// Disable print console log

$config['enable'] = false;

return $config;

}

修改保存后，重启 swoft，可以看到不会有任何信息输出了。

--

2、应用日志

(1)、应用日志配置

// 在 bean.php 里面配置应用日志的参数

return [

'lineFormatter' => [

'format' => '%datetime% [%level\_name%] [%channel%] [%event%] [tid:%tid%] [cid:%cid%] [traceid:%traceid%] [spanid:%spanid%] [parentid:%parentid%] %messages%',

'dateFormat' => 'Y-m-d H:i:s',

],

'noticeHandler' => [

'class' => FileHandler::class,

'logFile' => '@runtime/logs/notice-%d{Y-m-d}.log', // 2.0.6 支持日志按时间切割

'formatter' => \bean('lineFormatter'),

'levels' => 'notice,info,debug,trace',

],

'applicationHandler' => [

'class' => FileHandler::class,

'logFile' => '@runtime/logs/error.log',

'formatter' => \bean('lineFormatter'),

'levels' => 'error,warning',

],

'logger' => [

'flushRequest' => false,

'enable' => false,

'handlers' => [

'application' => \bean('applicationHandler'),

'notice' => \bean('noticeHandler'),

],

]

];

此配置也是框架默认的配置文件，把应用日志按日志级别分别写到两个不同的文件里面。swoft 2.0.3 levels 修改成字符串，方便开发者覆盖框架默认配置。

(2)、日志格式

lineFormatter 配置日志格式：format 日志输到文件格式；dateFormat 日志输出时间格式。

(3)、日志处理器

noticeHandler 和 applicationHandler 处理器，应用日志可以配置多个处理器，处理器可以把日志输出到文件、邮箱、第三方系统。

配置详细参数：class 配置采用哪种类型的，框架默认提供文件，用户可以自己扩展其它类型；logFile 输出日志文件路径，支持别名，2.0.6 支持日志按时间切割，格式： %d{Y-m-d}， { } 中的格式适用 PHP date format；formatter 日志输出使用日志格式，就是之前配置的日志格式；levels 支持日志输出的日志级别。

(4)、日志配置

logger 日志配置：name 名称；flushInterval 日志输出频率默认 1；flushRequest 是否每个请求结束输出日志开，默认 false；enable 是否开启日志，默认 false；json 是否 JSON 格式输出，默认 false；items 日志其它数据，配置是一个数组集合。

(5)、应用日志使用

开发者直接通过 Log 类静态方法，打印日志。应用日志根据不同的需求分为如下不同的日志记录：emergency，debug，alert，info，warning，error。

// Tag start

Log::profileStart('tagName');

//

Log::debug('this %s log', 'debug');

Log::info('this %s log', 'info');

Log::warning('this %s log', 'warning');

Log::error('this %s log', 'error');

Log::alert('this %s log', 'alert');

Log::emergency('this %s log', 'emergency');

//

// Pushlog

Log::pushLog('key', 'value');

Log::pushLog('key', ['value']);

Log::pushLog('key', 'value');

//

// Tag end

Log::profileEnd('tagName');

//

// Counting

Log::counting('mget', 1, 10);

Log::counting('mget', 2, 10);

//

// 2019-05-11 06:57:27 [info] [swoft] [request] [tid:4] [cid:4] [traceid:] [spanid:] [parentid:] trace[LogController.php:29,App\Http\Controller\LogController->test] info message

// 2019-05-11 06:57:27 [debug] [swoft] [request] [tid:4] [cid:4] [traceid:] [spanid:] [parentid:] trace[LogController.php:35,App\Http\Controller\LogController->test] this debug log

// 2019-05-11 06:57:27 [info] [swoft] [request] [tid:4] [cid:4] [traceid:] [spanid:] [parentid:] trace[LogController.php:36,App\Http\Controller\LogController->test] this info log

// 2019-05-11 06:57:27 [warning] [swoft] [request] [tid:4] [cid:4] [traceid:] [spanid:] [parentid:] trace[LogController.php:37,App\Http\Controller\LogController->test] this warning log

// 2019-05-11 06:57:27 [error] [swoft] [request] [tid:4] [cid:4] [traceid:] [spanid:] [parentid:] trace[LogController.php:38,App\Http\Controller\LogController->test] this error log

// 2019-05-11 06:57:27 [notice] [swoft] [request] [tid:4] [cid:5] [traceid:] [spanid:] [parentid:] [8.52(ms)] [27(MB)] [/log/test] [key=value key=["value"] key=value] profile[tagName=5.02(ms)/2] counting[mget=3/20]

日志级别可以根据自己的业务定义不同的含义，debug 日志级别只有开启 APP\_DEBUG 默认才会输出. notice 日志级别，每个请求结束框架自动追加上的一条日志，该条日志会记录这次请求，框架和业务运行的详细信息，比如数据库操作记录、缓存操作记录以及请求信息。日志方法可以像 sprintf() 函数一样传递参数，底层自动封装。

除常用的日志级别，框架还提供 pushLog profile。pushLog 用于记录请求执行过程中的参数，notice 日志里面可以查看。profile 使用的时候，需要定义一个标记，标记开始和结束为止，该方法可以记录标记之间运行时间，notice 日志里面可以查看。

--

--

九、验证器

https://swoft.org/documents/v2/core-components/validator/

//

正常情况都需要对用户输入参数进行校验，此时就会用到验证器。验证器可以验证控制器中参数，也支持验证 Websocket 参数以及 RPC 参数验证，提供默认和自定义两种类型的验证器，还可添加自定义验证规则。

--

1、安装

//// 使用验证器前，需要安装验证器组件，安装如下

composer require swoft/validator

--

2、启用

成功安装好验证组件后，接下来需要启用验证器，这里以 Http-server 启用为例，其它一样(app/bean.php)。

return [

// ......

'httpDispatcher' => [

// ......

'afterMiddlewares' => [

\Swoft\Http\Server\Middleware\ValidatorMiddleware::class

]

// ......

],

// ......

];

2.0.5+ 验证器默认没有启动，需要开发者自己开启。

--

3、声明验证器

一个验证器由多个验证条件组合，建议验证器按数据库表进行组合，这样可以充分的重复利用验证器里面的组合条件。

(1)、验证器

如下定义一个 TestValidator 验证器，由多个验证项（验证器条件）组成。

<?php declare(strict\_types=1);

namespace App\Validator;

use App\Annotation\Mapping\AlphaDash;

use Swoft\Validator\Annotation\Mapping\IsInt;

use Swoft\Validator\Annotation\Mapping\IsString;

use Swoft\Validator\Annotation\Mapping\Validator;

/\*\*

\* Class TestValidator

\*

\* @since 2.0

\*

\* @Validator(name="TestValidator")

\*/

class TestValidator

{

/\*\*

\* @IsString() //类型注解

\* @Email() //条件注解

\* @var string

\*/

protected $name = 'defualtName';

/\*\*

\* @IsInt(message="type must Integer")

\*

\* @var int

\*/

protected $type;

}

?>

(2)、@Validator 注解

声明一个验证器，name 定义验证器的名称，方便引用，如果不定义默认就是类名全路径。

(3)、验证项

验证项是组成验证器的唯一条件，标记有类型注解的属性就是一个验证项，一个验证器可以有多个验证项。

|> 属性的默认值就是参数的默认值，如果属性没有定义默认值，代表参数没有定义默认值且必须传递。

|> 一个属性必须定义一个类型注解，否则不是一个验证项且对参数验证无效。

|> 一个属性可以多个条件注解，按照定义顺序验证数据。

|> 默认属性名称就是需要验证的参数名称，也可以通过类型注解的 name 参数映射需要验证的字段名称。

|> 若验证不通过时，将会抛出 Swoft\Validator\Exception\ValidatorException 异常。

(4)、类型注解

一个属性必须定义一个类型注解，否则不是一个验证项且对参数验证无效。属性的默认值就是参数的默认值，如果属性没有定义默认值，代表参数没有定义默认值且必须传递。

//// @IsArray

验证规则：验证参数值必须是数组，使用 is\_array()函数进行校验。

参数说明：name，映射需要验证的字段名称，默认属性名称；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @IsArray(name="field", message="error message")。

//

//// @IsBool

验证规则：验证参数值必须是 bool 类型，注意字符串 true false，会验证成bool 类型，其余数据将会使用is\_bool()函数进行验证。

参数说明：name，映射需要验证的字段名称，默认属性名称；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @IsBool(name="field", message="error message")。

//

//// @IsFloat

验证规则：验证参数值必须是浮点数，使用filter\_var()函数进行验证。

参数说明：name，映射需要验证的字段名称，默认属性名称；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @IsFloat(name="field", message="error message")。

//

//// @IsInt

验证规则：验证参数值必须是整数，使用filter\_var()函数进行验证。

参数说明：name，映射需要验证的字段名称，默认属性名称；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @IsInt(name="field", message="error message")。

//

//// @IsString

验证规则：验证参数值必须是字符串，使用is\_string()函数进行验证。

参数说明：name，映射需要验证的字段名称，默认属性名称；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @IsString(name="field", message="error message")。

(5)、条件注解

一个属性可以多个条件注解，按照定义顺序验证数据。

//

//// @AfterDate

验证规则：验证参数值必须在某个日期之后，参数支持 字符串时间戳、格式化日期字符串（只支持 Y-m-d H:i:s）、整型时间戳，可在@IsString 或 @IsInt 类型注解中使用。

参数说明：date，要对比的日期值，只能是 Y-m-d H:i:s 格式；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @AfterDate(date="2019-01-01 00:00:00", message="error message")。

//

//// @Alpha

验证规则：验证参数值必须是 大写字母 或 小写字母。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Alpha(message="error message")。

//

//// @AlphaDash

验证规则：验证参数值必须是 大写字母、小写字母、数字、短横 -、下划线 \_。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @AlphaDash(message="error message")。

//

//// @AlphaNum

验证规则：验证参数值必须是 大写字母、小写字母、数字。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @AlphaNum(message="error message")。

//

//// @BeforeDate

验证规则：验证参数值必须在某个日期之前，支持 字符串时间戳、格式化日期字符串（只支持 Y-m-d H:i:s）、整型时间戳，可在@IsString 或 @IsInt 类型注解中使用。

参数说明：date，要对比的日期值，只能是 Y-m-d H:i:s 格式；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @BeforeDate(date="2019-01-01 00:00:00", message="error message")。

//

//// @Chs

验证规则：验证参数值只能是 中文。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Chs(message="error message")。

//

//// @ChsAlpha

验证规则：验证参数值必须是 中文、大写字母、小写字母。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @ChsAlpha(message="error message")。

//

//// @ChsAlphaDash

验证规则：验证参数值必须是 中文、大写字母、小写字母、数字、短横 -、下划线 \_。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @ChsAlphaDash(message="error message")。

//

//// @ChsAlphaNum

验证规则：验证参数值必须是 中文、大写字母、小写字母、数字。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @ChsAlphaNum(message="error message")。

//

//// @Confirm

验证规则：验证参数值必须和另外一个字段参数值相同。

参数说明：name，需要确认对比的字段名，在类型注解中设置过的 name 或者是默认的 属性名；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Confirm(name="field", message="error message")。

//

//// @Different

验证规则：验证参数值必须和另外一个字段参数值不同。

参数说明：name，需要确认对比的字段名，在类型注解中设置过的 name 或者是默认的 属性名；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Different(name="field", message="error message")。

//

//// @GreaterThan

验证规则：验证参数值必须比另外一个字段参数值大，只支持 int 或 float, 字符串会被转化为 float 后进行对比。

参数说明：name，需要确认对比的字段名，在类型注解中设置过的 name 或者是默认的 属性名；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @GreaterThan(name="field", message="error message")。

//

//// @LessThan

验证规则：验证参数值必须比另外一个字段参数值小，只支持 int 或 float, 字符串会被转化为 float 后进行对比。

参数说明：name，需要确认对比的字段名，在类型注解中设置过的 name 或者是默认的 属性名；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @LessThan(name="field", message="error message")。

//

//// @Date

验证规则：验证参数值必须是日期格式，支持 字符串时间戳、格式化日期字符串（只支持 Y-m-d H:i:s）、整型时间戳，可在@IsString 或 @IsInt 类型注解中使用。注意由于时间戳的特殊性默认为一个整型 大于 PHP\_INT\_MIN , 小于 PHP\_INT\_MAX 常量的数值均为有效时间戳。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Date(message="error message")。

//

//// @DateRange

验证规则：验证参数值必须在某个日期范围之内（可以等于临界日期），支持 字符串时间戳、格式化日期字符串（只支持 Y-m-d H:i:s）、整型时间戳，可在@IsString 或 @IsInt 类型注解中使用。

参数说明：start，要对比的开始日期值，只能是 Y-m-d H:i:s 格式；end，要对比的结束日期值，只能是 Y-m-d H:i:s 格式；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @DateRange(start="2019-01-01 00:00:00",end="2019-01-01 00:00:00", message="error message")。

//

//// @Dns

验证规则：验证参数值必须是一个具有有效 DNS 记录域名或者ip，使用 checkdnsrr() 函数校验。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Dns(message="error message")。

//

//// @Email

验证规则：验证参数值格式必须为邮箱。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Email(message="error message")。

//

//// @Enum

验证规则：验证参数值必须在枚举数组里面。

参数说明：values，枚举数组集合；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Enum(values={1,2,3},message="error message")。

//

//// @File

验证规则：验证此参数的值必须是文件，可以是单个文件，也可以是表单数组上传的多个文件。注意文件上传后文件域的获取需要通过 Swoft\Http\Message\Request 对象去获取。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @File(message="error message")。

//

//// @FileMediaType

验证规则：使用此条件前必须使用 @File 规则为基础，验证每个上传的文件 mediaType 类型，支持表单数组，批量文件。

参数说明：mediaType，mediaType类型数组；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @FileMediaType(mediaType={"image/gif","image/png"},message="error message")。

//

//// @FileSize

验证规则：使用此条件前必须使用 @File 规则为基础。验证每个上传的文件尺寸大小（单位 byte 字节）,支持表单数组，批量文件。

参数说明：size，文件尺寸大小，单位 byte；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @FileSize(size=1024,message="error message")。

//

//// @FileSuffix

验证规则：使用此条件前必须使用 @File 规则为基础。验证每个上传的文件后缀名，支持表单数组，批量文件。

参数说明：suffix，后缀名数组；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @FileSuffix(suffix={"png","jpg"},message="error message")。

//

//// @Ip

验证规则：验证参数值必须是个IP类型。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例：@Ip(message="error message")。

//

//// @Length

验证规则：验证参数值长度限制。

参数说明：min，最小值(包含当前值)；max，最大值(包含当前值)。message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Length(min=1,max=4,message="error message")。

//

//// @Low

验证规则：验证参数值必须是小写字母。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Low(message="error message")。

//

//// @Max

验证规则：最大值验证，必须是整数。

参数说明：value 最大值(包含当前值)；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Max(value=5,message="error message")。

//

//// @Min

验证规则：最小值验证。

参数说明：value 最小值(包含当前值)；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Min(value=5,message="error message")。

//

//// @Mobile

验证规则：手机号验证。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Mobile(message="error message")。

//

//// @NotEmpty

验证规则：参数值不能为空验证。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例：@NotEmpty(message="error message")。

//

//// @NotInEnum

验证规则：验证参数值必须不在枚举数组中。

参数说明：values 枚举数组集合；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @NotInEnum(values={1,2,3},message="error message")。

//

//// @NotInRange

验证规则：验证参数值必须不在范围内。

参数说明：min，最小值(包含当前值)；max，最大值(包含当前值)；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @NotInRange(min=1,max=5,message="error message")。

//

//// @Pattern

验证规则：正则表达式验证。

参数说明：regex，正则表达式；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Pattern(regex="/^1\d{10}$/", message="error message")。

//

//// @Range

验证规则：参数值范围验证。

参数说明：min，最小值(包含当前值)；max，最大值(包含当前值)；message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Range(min=1,max=5,message="error message")。

//

//// @Upper

验证规则：验证参数值必须是大写字母。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Upper(message="error message")。

//

//// @Url

验证规则：验证参数值必须是有效的URL格式，使用 filter\_var()函数验证。

参数说明：message，验证失败时的错误提示，若不设置则默认使用框架内置的。

使用示例: @Url(message="error message")。

//

暂时官方提供了这些条件验证，如果有其它需求可以根据 自定义验证器规则( https://swoft.org/documents/v2/core-components/validator/#%E8%87%AA%E5%AE%9A%E4%B9%89%E9%AA%8C%E8%AF%81%E5%99%A8%E8%A7%84%E5%88%99 ) 此章节内容自行添加验证规则。

--

4、自定义验证器

常见的业务默认验证器就能解决，但是有些业务默认验证器是没法验证，此时就需要用户根据自己业务需求，定义满足自己业务的验证器。

(1)、声明验证器很简单

定一个类实现 Swoft\Validator\Contract\ValidatorInterface 接口；

实现 validate 方法，里面可以定义自己的业务验证器逻辑；

使用 @Validator 注解标记这是一个验证器，此注解使用和功能和之前介绍完全一样；

//

//// 如下定义了一个验证器，验证 start 开始时间不能大于 end 结束时间

<?php declare(strict\_types=1);

namespace App\Validator;

use Swoft\Validator\Annotation\Mapping\Validator;

use Swoft\Validator\Contract\ValidatorInterface;

use Swoft\Validator\Exception\ValidatorException;

/\*\*

\* Class CustomerValidator

\*

\* @since 2.0

\*

\* @Validator(name="userValidator")

\*/

class CustomerValidator implements ValidatorInterface

{

/\*\*

\* @param array $data

\* @param array $params

\*

\* @return array

\* @throws ValidatorException

\*/

public function validate(array $data, array $params): array

{

$start = $data['start'] ?? null;

$end = $data['end'] ?? null;

if ($start === null && $end === null) {

throw new ValidatorException('Start time and end time cannot be empty');

}

if ($start > $end) {

throw new ValidatorException('Start cannot be greater than the end time');

}

return $data;

}

}

?>

(2)、验证方法详细介绍

public function validate(array $data, array $params): array

$data 用户输入参数，通过对应的解析器，已解析成数组。

$params 传递给验证器的参数，后续章节详细介绍。

//

验证成功返回验证处理后的数据，如果验证失败，抛出 Swoft\Validator\Exception\ValidatorException 异常，其它由框架处理。如果验证器里面需要修改参数值，可以直接修改，修改后获取值的地方会得到新的值。

--

5、自定义验证器规则

之前我们介绍过了如何使用自定义验证器，但是有些时候我们想要扩展我们的验证器规则，而非重新自定义一个验证器，对于这种场景的应用，我们也能够比较方便的添加我们自己的验证器规则，请阅读下方详细步骤。Available: >= v2.0.3。

我们在 声明验证器( https://swoft.org/documents/v2/core-components/validator/#%E5%A3%B0%E6%98%8E%E9%AA%8C%E8%AF%81%E5%99%A8 ) 这一节中声明的验证器基础上，添加一个 password 字段，使用一个我们自定义的验证规则来验证该字段。

/\*\*

\* @AlphaDash(message="Passwords can only be alphabet, numbers, dashes, underscores")

\*

\* @var string

\*/

protected $password;

如上我们就添加好了一个 password 字段，并且使用了一个 @AlphaDash() 的验证规则，该规则就是我们接下来要自定义的规则，它的功能是校验该字段的格式，使其只能是 字母`,数字,`-`,`\_。

思考下由于我们的验证规则是以注解的方式工作的，所以定义验证规则，其实也就是相当于定义一个我们自己的自定义注解命令，这一点清楚了之后我们继续。

(1)、声明规则注解

注意 我们强烈建议你按照 应用结构( https://swoft.org/documents/v2/quick-start/directory-struct/ ) 中的建议，将自定义的注解定义到 App/Annotation 路径中。

<?php declare(strict\_types=1);

namespace App\Annotation\Mapping;

use Doctrine\Common\Annotations\Annotation\Attribute;

use Doctrine\Common\Annotations\Annotation\Attributes;

/\*\*

\* Class AlphaDash

\*

\* @since 2.0

\*

\* @Annotation

\* @Attributes({

\* @Attribute("message",type="string")

\* })

\*/

class AlphaDash

{

/\*\*

\* @var string

\*/

private $message = '';

/\*\*

\* @var string

\*/

private $name = '';

/\*\*

\* StringType constructor.

\*

\* @param array $values

\*/

public function \_\_construct(array $values)

{

if (isset($values['value'])) {

$this->message = $values['value'];

}

if (isset($values['message'])) {

$this->message = $values['message'];

}

if (isset($values['name'])) {

$this->name = $values['name'];

}

}

/\*\*

\* @return string

\*/

public function getMessage(): string

{

return $this->message;

}

/\*\*

\* @return string

\*/

public function getName(): string

{

return $this->name;

}

}

?>

@Annotation 声明这个类是一个注解命令；

@Attributes 声明注解参数集合；

@Attribute 声明注解具体的参数，name 参数的名字；type 参数值的类型。

(2)、类属性方法说明

$message 就是我们使用该注解时传入的提示，例如 @IsString(message="该字段必须是字符串")。$name 就是字段的名字，为空的话则默认就是属性名，例如 @IsString(name="user\_name")。$value 则是我们需要传递给验证规则的一些数据，若无需传参则可以不用定义。例如 @Enum(values=[1,2,3]) getMessage(), getName() 方法必须存在，用来获取$message和$name。

(3)、声明注解解析

至此，我们定义好了注解命令，但是注解命令要想能够执行，则还需要定义一个注解命令的解析器，下方就是一个注解解析器，需要继承 Swoft\Annotation\Annotation\Parser\Parser 类。

<?php declare(strict\_types=1);

namespace App\Annotation\Parser;

use ReflectionException;

use Swoft\Annotation\Annotation\Mapping\AnnotationParser;

use Swoft\Annotation\Annotation\Parser\Parser;

use App\Annotation\Mapping\AlphaDash;

use Swoft\Validator\Exception\ValidatorException;

use Swoft\Validator\ValidatorRegister;

/\*\*

\* Class AlphaDashParser

\*

\* @AnnotationParser(annotation=AlphaDash::class)

\*/

class AlphaDashParser extends Parser

{

/\*\*

\* @param int $type

\* @param object $annotationObject

\*

\* @return array

\* @throws ReflectionException

\* @throws ValidatorException

\*/

public function parse(int $type, $annotationObject): array

{

if ($type != self::TYPE\_PROPERTY) {

return [];

}

//向验证器注册一个验证规则

ValidatorRegister::registerValidatorItem($this->className, $this->propertyName, $annotationObject);

return [];

}

}

?>

@AnnotationParser 声明要解析的注解命令。

parse() 由于我们继承了 Swoft\Annotation\Annotation\Parser\Parser，而它有声明了一个 Swoft\Annotation\Annotation\Parser\ParserInterface 接口，而这个方法正是 ParserInterface 这个接口所定义的一个必须由我们来实现的一个接口。其实这里就是我们要处理的业务逻辑了，Swoft 解析到一个注解命令后，就会执行这个注解所对应的解析器中的 parse() 这个方法。

(4)、声明一个验证规则

经过之前的步骤我们已经定义好了验证规则的注解以及它的解析器，但是我们还没有定义我们的具体的验证规则，所以接下来，我们将声明我们具体的验证规则，其实很简单，我们只需要实现一个 Swoft\Validator\Contract\RuleInterface 接口就可以了。

<?php declare(strict\_types=1);

namespace App\Validator\Rule;

use App\Annotation\Mapping\AlphaDash;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Validator\Contract\RuleInterface;

use Swoft\Validator\Exception\ValidatorException;

/\*\*

\* Class AlphaDashRule

\*

\* @Bean(AlphaDash::class)

\*/

class AlphaDashRule implements RuleInterface

{

/\*\*

\* @param array $data

\* @param string $propertyName

\* @param object $item

\* @param null $default

\*

\* @return array

\* @throws ValidatorException

\*/

public function validate(array $data, string $propertyName, $item, $default = null): array

{

$message = $item->getMessage();

if (!isset($data[$propertyName]) && $default === null) {

$message = (empty($message)) ? sprintf('%s must exist!', $propertyName) : $message;

throw new ValidatorException($message);

}

$rule = '/^[A-Za-z0-9\-\\_]+$/';

if (preg\_match($rule, $data[$propertyName])) {

return [$data];

}

$message = (empty($message)) ? sprintf('%s must be a AlphaDash!', $propertyName) : $message;

throw new ValidatorException($message);

}

}

?>

@Bean 由于验证器内部是通过 Bean 容器来获得到我们的验证规则的，代码如下。

$rule = BeanFactory::getBean($itemClass);//这里通过容器拿到了我们的验证规则

$data = $rule->validate($data, $propName, $item, $default);

所以这里我们就要使用 @Bean 来注册我们的验证规则，名字就是和我们的注解命令相同。

validate() 这是 RuleInterface 接口中规定要实现的方法，到了这里其实就是写我们具体的验证规则了。array $data 待验证的所有数据；string $propertyName 需要验证的字段名；$item 注解类的对象；$default 字段的默认值。array $data 待验证的所有数据；string $propertyName 需要验证的字段名；$item 注解类的对象；$default 字段的默认值。至此我们已经定义好了一个验证器规则。

--

6、控制器中使用

如果想在控制器中使用验证器很简单，只需要是一个注解 @Validate 就行。一个 action 可以定义多个 @Validate 使用多个验证器；多个验证器按照配置顺序验证。

//

//// 如下定义一个 ValidatorController, 同时使用默认验证器和自定义验证器以及我们自定义的验证规则。

<?php declare(strict\_types=1);

namespace App\Http\Controller;

use Swoft\Http\Message\Request;

use Swoft\Http\Server\Annotation\Mapping\Controller;

use Swoft\Http\Server\Annotation\Mapping\RequestMapping;

use Swoft\Validator\Annotation\Mapping\Validate;

/\*\*

\* Class ValidatorController

\*

\* @Controller()

\*/

class ValidatorController

{

/\*\*

\* 验证TestValidator验证器中的所有已定义字段

\*

\* @RequestMapping()

\* @Validate(validator="TestValidator")

\* @param Request $request

\*

\* @return array

\*/

public function validateAll(Request $request): array

{

return $request->getParsedBody();

}

/\*\*

\* 仅验证TestValidator验证器中的 type 字段

\*

\* @RequestMapping()

\* @Validate(validator="TestValidator",fields={"type"})

\* @param Request $request

\*

\* @return array

\*/

public function validateType(Request $request): array

{

return $request->getParsedBody();

}

/\*\*

\* 仅验证TestValidator验证器中的 password 字段 password字段使用的是自定义的验证规则。

\*

\* @RequestMapping()

\* @Validate(validator="TestValidator",fields={"password"})

\* @param Request $request

\*

\* @return array

\*/

public function validatePassword(Request $request): array

{

return $request->getParsedBody();

}

/\*\*

\* 使用userValidator自定义验证器

\*

\* @RequestMapping()

\* @Validate(validator="userValidator")

\* @param Request $request

\*

\* @return array

\*/

public function validateCustomer(Request $request): array

{

return $request->getParsedBody();

}

}

?>

$request->getParsedBody() 所有解析数据；

$request->parsedBody('key', 'default') 指定 KEY 解析数据；

$request->getParsedQuery() 所有解析的 Query 参数；

$request->parsedQuery('key', 'default') 指定 KEY 解析数据(>=2.0.2)；

(1)、@Validate

validator 指定验证器名称；

fields 指定验证器里面验证的字段，这样可以高效地重复使用验证器；

type 默认 body，ValidateType::GET 验证 GET 请求 query 参数；

params 自定义验证器使用，传递给自定义验证器的参数。

$request->getParsedBody() 获取的请求数据，是已经通过验证器修改的数据。验证器可以支持表单、请求 body 数据验证，但是 body 验证需要定义对应的数据解析器，框架默认提供 JSON/XML 类型数据解析器，详细介绍，请参考 Http Server 章节( https://swoft.org/documents/v2/core-components/http-server/ )。

//

//// 验证 GET 请求 query 参数

use Swoft\Validator\Annotation\Mapping\ValidateType;

/\*\*

\* @RequestMapping()

\* @Validate(validator="TestValidator", type=ValidateType::GET)

\* @param Request $request

\*

\* @return array

\*/

public function validateType(Request $request): array

{

return $request->getParsedBody();

}

--

7、非注解验证器

注解方式引用和使用验证器是有限制，只支持在 Http server/ Rpc server /Websocket server 等特定位置使用，在实际业务开发中，其它地方也会涉及参数的验证。

(1)、全局方法

function validate(array $data, string $validatorName, array $fields = [], array $userValidators = []): array

全局函数使用，当验证器失败会抛出 Swoft\Validator\Exception\ValidatorException 异常。

$data 需要验证的数据，必须是数组 KV 格式；$validatorName 使用的验证器( @Validator() 注解标记的 )；$fields 需要验证的字段，为空验证器所有字段；$userValidators 同时使用的自定义验证器，支持两种格式。

//

///1/ 示例：所有参数验证

use Swoft\Validator\Annotation\Mapping\Validator;

$data = [

'email' => 'swoft@xx'

]

$result = validate($data, Validator::class);

//

///2/ 示例：指定字段验证

use Swoft\Validator\Annotation\Mapping\Validator;

$data = [

'email' => 'swoft@xx'

]

$result = validate($data, Validator::class, ['email']);

//

///3/ 示例：同时使用自定义验证器

use Swoft\Validator\Annotation\Mapping\Validator;

$data = [

'start' => 12,

'end' => 16,

];

$result = validate($data, Validator::class, [], ['testUserValidtor']);

//

///4/ 示例：同时使用自定义验证器且传递参数

use Swoft\Validator\Annotation\Mapping\Validator;

$data = [

'start' => 12,

'end' => 16,

'params' => [1, 2]

];

$users = [

'testUserValidtor' => [1, 2]

];

$result = validate($data, Validator::class, [], $users);

--

--

十、国际化

https://swoft.org/documents/v2/core-components/i18n/

//

一般用于根据用户语言，需要输出不同的文案。如果没有国际化的封装，业务里面会有大量的判断，并且业务也不好维护，没法统一管理所有文案。

--

1、安装

composer require swoft/i18n

--

2、Git 仓库

Github https://github.com/swoft-cloud/swoft-event

--

3、参与贡献

欢迎参与贡献，您可以 fork 我们的开发仓库 swoft/component( https://github.com/swoft-cloud/swoft-component )；修改代码然后发起 PR；阅读 提交代码( https://swoft.org/documents/v2/contribute/sub-codes/ ) 的注意事项。

--

4、配置参数

使用 i18n 功能,一共需要三步 资源文件夹中导入语言文本；配置文件中配置相关参数；调用相关方法完成文本转换。

(1)、导入文本

我们在 resource/language/ 文件夹下可创建语言组文件夹与文本文件，一个语言组中可以有多个模板文件，如下所示。

# resource/language

|-- zh

|-- default.php

|-- msg.php

`-- en

|-- default.php

|-- msg.php

而文本格式是由关联数组组成，键值为文本，我们可以在其中插入参数，格式大致如下。

// ../en/default.php

return [

// 文本中可用大括号注入参数

'sayhello' => 'Hey {name}!',

'saygoodbye' => 'Bye!',

];

//

// ../en/msg.php

return [

'sayhello' => "What's up! {name}",

'saygoodbye' => 'See you tomorrow!',

];

//

// ../zh/default.php

return [

'sayhello' => "早上好,{name}",

'saygoodbye' => '再见',

];

// ../zh/msg.php

return [

'sayhello' => "晚上好,{name}",

'saygoodbye' => '明天见',

];

(2)、相关配置

i18n 相关功能配置非常简单, 只需要在 app/bean.php 配置文件中，配置以下参数，即可开启国际化功能，参数描述见注释。

return [

// .... 其他配置

'i18n' => [

// 设置到文本资源目录

'resoucePath' => '@resource/language/', // 结尾斜线必须

// 设置默认文本文件夹名称

// 未填写则默认 en 文件夹

'defaultLanguage' => 'en',

// 设置默认文本文件名称

// 未填写则默认 default.php

'defualtCategory' => 'default',

],

// .... 其他配置

];

至此相关参数已配置完毕，下面就可以使用该功能了。

(3)、使用案例

//// 语言转换仅仅至于要调用一个简单的方法, 下面是方法描述

/\*\*

\* @param string $key 文本数组中,对应文本键值

\* @param array $params 注入到文本中的参数,以关联数组的形式

[

'param1' => 'str1',

'param2' => 'str2'

]

\* @param string $locale 资源文件夹下, 分组文件名称.

\*/

\Swoft::t(string $key, array $params, string $locale): string

//

//// 实例演示

<?php declare(strict\_types=1);

namespace App\Http\Controller;

use Swoft\Http\Server\Annotation\Mapping\Controller;

use Swoft\Http\Message\Request;

use Swoft\Http\Server\Annotation\Mapping\RequestMapping;

use Swoft\Http\Server\Annotation\Mapping\RequestMethod;

/\*\*

\* Test用演示实例

\*

\* @since 1.0.0

\*

\* @Controller(perfix="test")

\*/

class TestController

{

/\*\*

\* i18n 示例

\*

\* @RequestMapping(route="i18n", method={RequestMethod::GET})

\*

\* @return array

\*/

public function i18n(Request $request):array

{

// 显示 language/en/default.php 文本模板:

// 'sayhello' => 'Hey {name}!',

$res['en'] = \Swoft::t('sayhello', ['name' => 'man']);

// 更换模板可以使用 '.'

// 显示 language/en/msg.php 模板

$res['en-msg'] = \Swoft::t('msg.sayhello', ['name' => 'man'], 'en');

// 显示 language/zh/default.php 模板

$res['zh'] = \Swoft::t('sayhello', ['name' => '李华'], 'zh');

// 显示 language/zh/msg.php 模板

$res['zh-msg'] = \Swoft::t('msg.sayhello', ['name' => '李华'], 'zh');

return $res;

}

// ...

}

?>

结果输出:

{

"en": "Hey man!",

"en-msg": "What's up! man",

"zh": "早上好,李华",

"zh-msg": "晚上好,李华"

}

--

--

第七卷 MYSQL

--

--

一、配置

https://swoft.org/documents/v2/mysql/config/

GitHub: https://github.com/swoft-cloud/swoft-db

--

1、简介

Swoft 数据库组件高度兼容 Laravel，支持原生 SQL、查询构造器和 Eloquent ORM，去掉了复杂的对象模型关联。数据库组件采用原生 PDO 的方式连接，IO 操作会被自动转换为类似 Swoole 的 MySQL 协程客户端，使开发变得简单，更贴近传统框架。

为什么使用原生 PDO：由于 MySQLnd 模式的 PDO、MySQLi 扩展会加入 Hook 监听，如果未启用 MySQLnd 将不支持协程化。

--

2、安装

//// 数据库组件需独立安装

composer require swoft/db

--

3、基础配置

数据库的配置位于 app/bean.php 文件，未使用繁琐的 .env 环境文件配置。配置的 db 是一个 bean 对象。

return [

'db' => [

'class' => Swoft\Db\Database::class,

'dsn' => 'mysql:dbname=dbname;host=127.0.0.1:3306',

'username' => 'test',

'password' => 'test',

'charset' => 'utf8mb4',

'prefix' => 't\_',

'options' => [

PDO::ATTR\_CASE => PDO::CASE\_NATURAL

],

'config' => [

'collation' => 'utf8mb4\_unicode\_ci',

'strict' => true,

'timezone' => '+8:00',

'modes' => 'NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES',

'fetchMode' => PDO::FETCH\_ASSOC

]

]

];

配置方式类似 Yii2 的对象属性注入。可通过 bean('db') 获取当前配置的 Database 对象。

//

//// 配置项说明

class：数据库类，自定义时需指定，默认为 Swoft 数据库类；

dsn：PDO 连接信息，指定数据库名称、地址及端口；

username：数据库用户名；

password：数据库密码；

charset：数据库字符集；

prefix：数据表前缀；

options：PDO 选项，参阅：PHP: PDO::setAttribute - Manual( https://www.php.net/manual/zh/pdo.setattribute.php )；

config：额外配置。

> collation：字符集排序规则；

> strict：是否启用严格模式；

> timezone：时区设置，国内请设置 +8:00；

> modes：连接模式，完整列表参考：MySQL :: MySQL 5.6 Reference Manual - Full List of SQL Modes ( https://dev.mysql.com/doc/refman/5.6/en/sql-mode.html#sql-mode-full )；

> fetchMode：PDO 获取模式，默认为 PDO::FETCH\_ASSOC，即以关联数组返回，参考：PHP: PDO 预定义常量 - Manual( https://www.php.net/manual/zh/pdo.constants.php )；

--

4、读写分离

'db\_rw' => [

'charset' => 'utf8mb4',

'prefix' => 't\_',

'config' => [

'collation' => 'utf8mb4\_unicode\_ci',

'strict' => true,

'timezone' => '+8:00',

'modes' => 'NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES'

],

'writes' => [

[

'dsn' => 'mysql:dbname=db-write;host=127.0.0.1:3306',

'username' => 'write',

'password' => 'write'

]

],

'reads' => [

[

'dsn' => 'mysql:dbname=db-read;host=127.0.0.1:3306',

'username' => 'read',

'password' => 'read'

]

]

]

writes：写 配置，执行 增、删、改 操作时会从 连接池 随机选择 写 节点；

reads：读 配置，执行 读 操作时会从 连接池 随机选择 读 节点；

读写配置中公共部分请参考 基础配置( https://swoft.org/documents/v2/mysql/config/#%E5%9F%BA%E7%A1%80%E9%85%8D%E7%BD%AE )，其中 dsn、username、password、charset、prefix、options、config 支持配置公共应用，之后在 writes/reads 中配置差异部分即可。

--

5、连接池

使用过 Swoft 1.0 的小伙伴对连接池并不陌生，2.x 相比更加简洁。

DB 的连接从 连接池 创建和释放，通过 ConnectionManager 类进行管理。创建的连接为短连接，操作执行失败后会重试 一次。每当调用 toSql() 方法或执行完毕后会将连接归还至连接池中。连接池的默认名称为 db.pool，使用的数据库配置来自 bean('db')，由 基础配置( https://swoft.org/documents/v2/mysql/config/#%E5%9F%BA%E7%A1%80%E9%85%8D%E7%BD%AE ) 提供。连接池配置同样位于 app/bean.php 文件中。

每一个 Worker 都会创建一个同样的连接池，连接池并不是越多越好，参数配置需根据服务器配置和 Worker 的数量权衡。

--

--

二、模型

https://swoft.org/documents/v2/mysql/model/

--

1、简介

无论是基础查询还是高级查询，实际都会依赖表实体，一个表字段和一个类属性的关系通过映射实现，而对类的操作也就相当于在对数据表操作。Swoft 2.x 中实体类对比 1.x 使用起来更简单，它兼容 Builder 查询构造器所有的方法，使用实体类和查询构造器的方法一致。

--

2、实体定义

一个实体类对应一张数据库表，一个实体对象代表了数据表中一行数据记录。注意：实体不能作为属性被注入到任何类，因为每个实体对象来自不同的数据记录行，实体对象应当在需要使用的地方创建。

//

//// 下方为一个实体定义示例，文件参考：User.php ( https://github.com/swoft-cloud/swoft-db/blob/master/test/testing/Entity/User.php )

<?php declare(strict\_types=1);

namespace SwoftTest\Db\Testing\Entity;

use Swoft\Db\Annotation\Mapping\Column;

use Swoft\Db\Annotation\Mapping\Entity;

use Swoft\Db\Annotation\Mapping\Id;

use Swoft\Db\Eloquent\Model;

/\*\*

\* Class User

\*

\* @since 2.0

\*

\* @Entity(table="user", pool="db.pool")

\*/

class User extends Model

{

/\*\*

\* @Id(incrementing=true)

\*

\* @Column(name="id", prop="id")

\* @var int|null

\*/

private $id;

/\*\*

\* @Column()

\* @var string|null

\*/

private $name;

/\*\*

\* @Column()

\* @var int|null

\*/

private $hahh;

/\*\*

\* @return null|int

\*/

public function getHahh(): ?int

{

return $this->hahh;

}

/\*\*

\* @param null|int $hahh

\*/

public function setHahh(?int $hahh): void

{

$this->hahh = $hahh;

}

/\*\*

\* @Column(name="password", hidden=true)

\* @var string|null

\*/

private $pwd;

/\*\*

\* @Column()

\*

\* @var int|null

\*/

private $age;

/\*\*

\* @Column(name="user\_desc", prop="udesc")

\*

\* @var string|null

\*/

private $userDesc;

/\*\*

\* this key is hump

\*

\* @Column()

\*

\* @var string|null

\*/

private $testHump;

/\*\*

\* @Column(name="test\_json", prop="testJson")

\* @var array|null

\*/

private $testJson;

/\*\*

\* @Column()

\* @var float|null

\*/

private $amount;

/\*\*

\* @return float|null

\*/

public function getAmount(): ?float

{

return $this->amount;

}

/\*\*

\* @param float|null $amount

\*/

public function setAmount(?float $amount): void

{

$this->amount = $amount;

}

/\*\*

\* @return null|array

\*/

public function getTestJson(): ?array

{

return $this->testJson;

}

/\*\*

\* @param null|array $testJson

\*/

public function setTestJson(?array $testJson): void

{

$this->testJson = $testJson;

}

/\*\*

\* @return null|string

\*/

public function getTestHump(): ?string

{

return $this->testHump;

}

/\*\*

\* @param null|string $testHump

\*/

public function setTestHump(?string $testHump): void

{

$this->testHump = $testHump;

}

/\*\*

\* @return int|null

\*/

public function getId(): ?int

{

return $this->id;

}

/\*\*

\* @param int|null $id

\*/

public function setId(?int $id): void

{

$this->id = $id;

}

/\*\*

\* @return int|null

\*/

public function getAge(): int

{

return $this->age;

}

/\*\*

\* @param int|null $age

\*/

public function setAge(?int $age): void

{

$this->age = $age;

}

/\*\*

\* @return string|null

\*/

public function getName(): ?string

{

return $this->name;

}

/\*\*

\* @param string|null $name

\*/

public function setName(?string $name): void

{

$this->name = $name;

}

/\*\*

\* @return string|null

\*/

public function getPwd(): string

{

return $this->pwd;

}

/\*\*

\* @param string|null $pwd

\*/

public function setPwd(?string $pwd): void

{

$this->pwd = $pwd;

}

/\*\*

\* @return string|null

\*/

public function getUserDesc(): string

{

return $this->userDesc;

}

/\*\*

\* @param string|null $userDesc

\*/

public function setUserDesc(?string $userDesc): void

{

$this->userDesc = $userDesc;

}

}

?>

//

//// 在完成数据库基本配置后，可通过 Swoft Devtool 快速生成，通过下方命令查看帮助信息

php ./bin/swoft entity:create -h

--

3、注解

(1)、Entity

通过注解标签 @Entity 指定类为实体类。

参数说明：table，需映射的数据库表名（必填）；pool，连接池，默认为 db.pool。

(2)、Column

通过注解标签 @Column 指定成员属性为字段名。如果字段未添加 @Column 标签，那么在查询时该列（字段）不会展示。即使新增字段也不会影响生产环境。

参数说明：name，需映射的数据表字段名，默认值为成员属性；prop，字段别名，仅在调用 toArray 方法时被转换，使用 where 等子方法时仍需使用数据库字段名；hidden，字段是否隐藏，仅在调用 toArray 方法时会被隐藏，但并不影响通过 getter 方法获取，可以通过调用实体 addVisible 方法取消隐藏。

说明：所有字段属性，必须要有 getter 和 setter 方法，你可以使用 PhpStorm 快捷键 Alt + Insert （macOS 为 command + N 或 control + enter）根据属性快速生成 getter 和 setter 方法。若表字段名存在下划线，类属性需以 小驼峰 方式定义。例：字段名为 user\_name，则属性应当写为 $userName。

(3)、Id

该注解标签指定成员属性为主键。一个实体类仅能设置一个 @Id 注解标签。

参数说明：incrementing是否为递增主键，默认为 true。

(4)、Prop 操作

Swoft 版本需 >= 2.0.6。模型支持使用 prop 直接操作，在上方示例实体类中，数据表字段 user\_desc 的 prop 为 udesc，Swoft 底层会自动转换，所以并不影响我们使用。

//

//// 新增数据示例

User::new([

'udesc' => $descString,

])->save();

//

//// 查询条件示例

$where = [

'pwd' => md5(uniqid()),

['udesc', 'LIKE', 'swoft%'],

['whereIn', 'id', [1, 2, 3]]

];

// SELECT \* FROM `user` WHERE (`password` = ? AND `user\_desc` LIKE ? AND `id` IN (?))';

$sql = User::whereProp($where)->toSql();

在条件中使用需通过 whereProp 方法连接，whereProp 与 where 用法相同。

--

4、新增数据

(1)、对象方式

$user = User::new();

//

$user->setName('Swoft');

$user->setPwd('123456');

$user->setAge(2);

$user->setUserDesc('Great Framework');

//

$user->save();

// 保存之后获取 ID

$userId = $user->getId();

(2)、数组方式

$attributes = [

'name' => 'Swoft',

'pwd' => '123456',

'age' => 2,

'user\_desc' => 'Great Framework'

];

$user = User::new($attributes);

$user->save();

$userId = $user->getId();

(3)、批量新增

批量新增数据可以直接使用 User::insert($array) 方法，该方法与查询构造器方法一致。

(4)、其它方式

你还可以使用 firstOrCreate 和 firstOrNew 方法来新增数据。firstOrCreate 方法会使用给定的字段及其值在数据库中查找记录，如果在数据库中找不到该模型，则会使用第一个参数中的属性以及可选的第二个参数中的属性新增数据。

firstOrNew 方法类似 firstOrCreate 方法。它会在数据库中查找匹配给定属性的记录，如果模型未被找到则会返回一个新的模型实例。请注意，在这里面 firstOrnew 返回的模型还尚未保存到数据库，必须调用 save 方法才能写入到数据库中。

//

//// 以下为示例，通过 name 属性检索航班

// 当结果不存在时创建

$flight = Flight::firstOrCreate(['name' => 'Flight 10']);

// 当结果不存在的时候用 name 属性和 delayed 属性创建

$flight = Flight::firstOrCreate(

['name' => 'Flight 10'],

['delayed' => 1]

);

// 当结果不存在时实例化...

$flight = Flight::firstOrNew(['name' => 'Flight 10']);

// 当结果不存在的时候用 name 属性和 delayed 属性实例化

$flight = Flight::firstOrNew(

['name' => 'Flight 10'],

['delayed' => 1]

);

--

5、删除数据

(1)、ID 方式

$user = User::find($id);

$user->delete();

(2)、条件删除

User::where('id', 1)->delete();

--

6、数据更新

(1)、setter 方式

$user = User::find($id);

$user->setAge(2);

$user->save();

(2)、填充方式

$attributes = [

'name' => 'Swoft',

'pwd' => '123456',

'age' => 2,

'user\_desc' => 'Come on'

];

// 方式一

User::new($attributes)->save();

// 方式二

User::new()->fill($attributes)->save();

注意：如果该字段 没有匹配 到 @Column 标签时将会被忽略，这样能保证安全的更新和插入。

(3)、update 方式

User::find($id)->update(['age' => 2]);

(4)、批量更新

User::where([

'name' => 'Swoft',

['age', '>=', 1]

])->limit(2)->update(['user\_desc' => 'Very nice']);

(5)、主键批量更新

$values = [

['id' => 1, 'age' => 18],

['id' => 2, 'age' => 19],

];

User::batchUpdateByIds($values);

使用批量更新必须指定主键值，框架会根据主键值进行批量更新。在此例中， id 是 User 实体的 @Id() 主键。

(6)、快速更新

除 [update 方式](#update 方式) 示例代码中通过 find($id) 获取对象实体后 update 的方式外，还可以使用 modifyById 的方式进行快速更新。

$row = User::modifyById($id, ['age' => 2]);

除 批量更新( https://swoft.org/documents/v2/mysql/model/#%E6%89%B9%E9%87%8F%E6%9B%B4%E6%96%B0 ) 示例代码中通过 where 条件查询后 update 的方式外，还可以使用 modify 的方式进行快速更新。

$where = ['name' => 'Swoft'];

$values = ['user\_desc' => 'I love Swoft'];

User::modify($where, $values);

(7)、更新/新增

可使用 updateOrCreate 方法实现目标数据不存在时新增数据，方法返回数据实体。

$user = User::updateOrCreate(['id' => 1], ['age' => 18, 'name' => 'Swoft Framework']);

echo $user->getName();

也可以使用 updateOrInsert 方法，该方法返回值为 bool 类型。

User::updateOrInsert(['id' => 1], ['age' => 18, 'name' => 'Swoft Framework']);

(8)、自增/自减

//// 单字段 自增/自减

User::find($id)->increment('age', 1);

User::find($id)->increment('age', 1, ['name' => 'Swoft 2.0']);

User::where('id', 1)->decrement('age', 1);

提示：参数 $extra 为指定同步更新的数据。

//

//// 多字段 自增/自减

User::updateAllCounters(['name' => 'Swoft 2.0'], ['age' => -1]);

User::updateAllCountersById((array)$id, ['age' => 1], ['name' => 'Swoft 2.0']);

User::find($id)->updateCounters(['age' => -1, 'name' => 'Swoft 2.0']);

请谨慎使用条件更新，尽量使用主键更新以免造成锁表。

--

7、查询数据

模型的查询方法与查询构造器完全兼容。使用 join 进行查询操作时，不会返回对象实体。

(1)、单行数据查询

User::find(1, ['id', 'name']);

User::where('id', 1)->first(['id', 'name']);

(2)、多行数据查询

User::findMany([1, 2, 3, 4], ['id','name']);

User::whereIn('id', [1, 2, 3, 4])->get(['id', 'name']);

(3)、对象实体查询

$users = User::where('age', '>=', 18)->getModels(['id', 'age']);

/\* @var User $user \*/

foreach ($users as $user) {

$age = $user->getAge();

}

(4)、映射关系

假设我们需要以某个字段作为 key 映射逻辑关系，我们可以通过 keyBy 方法实现。

$users = User::forPage(1, 10)->get(['id', 'age'])->keyBy('id');

/\* @var User $user \*/

foreach ($users as $id => $user) {

$age = $user->getAge();

}

(5)、结果分块

如需处理数千个 Eloquent 记录，可以使用 chunk 方法，chunk 方法会检索 Eloquent 模型的「分块」，然后将它们提供给指定的 Closure 进行处理。在处理大型结果集时，使用 chunk 方法可节省内存开销。

Flight::chunk(200, function ($flights) {

foreach ($flights as $flight) {

// TODO:

}

});

传递到方法的第一个参数都是希望每个「分块」接收的数据量。闭包作为第二个参数传递，它会在查询每个块时被调用。

(6)、Cursor（游标）

cursor 用来遍历数据，游标只执行一次查询。在处理大量数据时，使用游标可以大幅度减少内存开销。

foreach (Flight::where('foo', 'bar')->cursor() as $flight) {

// TODO:

}

(7)、聚合函数

模型支持 查询构造器( https://www.swoft.org/docs/2.x/zh-CN/db/builder.html ) 提供的 count、sum、min、max 等聚合函数。

$count = Flight::where('active', 1)->count();

$max = Flight::where('active', 1)->max('price');

(8)、NotFound 异常

如需在模型不存在时抛出异常，可以使用 findOrFail 或 firstOrFail 方法。该方法会检索查询的第一个结果，如果目标结果不存在，则会抛出一个 DbException。

$model = Flight::findOrFail(1);

$model = Flight::where('legs', '>', 100)->firstOrFail();

--

8、自动写时间戳

默认情况下 Eloquent 会对数据表中 created\_at 和 updated\_at 两个字段进行自动写入。如果你不需要依赖 Swoft 自动更新这两个字段，只需在模型内将 $modelTimestamps 属性设置为 false 即可。

class User

{

/\*\*

\* @var bool

\*/

protected $modelTimestamps = false;

}

(1)、自定义时间格式

该属性决定了存储在数据库中的日期格式以及模型被序列化成数组或 JSON 时的格式。

/\*\* @var string \*/

protected $modelDateFormat = 'Y-m-d H:i:s';

(2)、自定义字段

如果数据库中使用了其它名称的时间字段，通过设置常量 CREATED\_AT 以及 UPDATED\_AT 变更。

protected const CREATED\_AT = 'add\_time';

protected const UPDATED\_AT = 'update\_time';

框架自动维护 CREATED\_AT 和 UPDATED\_AT 字段必须要有与之对应的 getter 和 setter。

--

9、事件

Eloquent 模型会触发事件，可以在模型的生命周期的以下几点进行监控：creating、created、updating、updated、saving、saved、deleting、deleted。

事件能在每次在数据库中保存或更新特定模型类时轻松地执行代码，当然你完全可以通过 AOP 来实现它。

当新模型第一次被保存时，creating 以及 created 事件会被触发，而如果模型已经存在于数据库中并且调用了 save 方法时，则会触发 updating 和 updated 事件。在这两种情况下，saving/saved 事件都会触发。

//

//// 事件名称

事件名称由 swoft.model + 模型名 + 动作名 组成。

模型名：首字母默认为小写，例如实体名称 SendMessage 在监听 saving 动作时，格式为 swoft.model.sendMessage.saving。

(1)、单模型单动作监听

<?php declare(strict\_types=1);

namespace App\Listener;

use App\Model\Entity\User;

use Swoft\Event\Annotation\Mapping\Listener;

use Swoft\Event\EventHandlerInterface;

use Swoft\Event\EventInterface;

/\*\*

\* Class UserSavingListener

\*

\* @since 2.0

\*

\* @Listener("swoft.model.user.saving")

\*/

class UserSavingListener implements EventHandlerInterface

{

/\*\*

\* @param EventInterface $event

\*/

public function handle(EventInterface $event): void

{

/\* @var User $user \*/

$user = $event->getTarget();

if ($user->getAge() > 100) {

// stopping saving

$event->stopPropagation(true);

$user->setAdd(100);

}

}

}

?>

(2)、多模型单动作监听

<?php declare(strict\_types=1);

namespace App\Listener;

use App\Model\Entity\User;

use Swoft\Db\DbEvent;

use Swoft\Db\Eloquent\Model;

use Swoft\Event\Annotation\Mapping\Listener;

use Swoft\Event\EventHandlerInterface;

use Swoft\Event\EventInterface;

/\*\*

\* Class RanListener

\*

\* @since 2.0

\*

\* @Listener(DbEvent::MODEL\_SAVED)

\*/

class ModelSavedListener implements EventHandlerInterface

{

/\*\*

\* @param EventInterface $event

\*/

public function handle(EventInterface $event): void

{

/\* @var Model $modelStatic \*/

$modelStatic = $event->getTarget();

if ($modelStatic instanceof User) {

// to do something....

}

// ....

}

}

?>

(3)、公共事件

完整列表参考 DbEvent.php( https://github.com/swoft-cloud/swoft-db/blob/master/src/DbEvent.php ) 文件。

事件/参数/描述

swoft.db.transaction.begin 无 事务启动；

swoft.db.transaction.commit 无 事务提交；

swoft.db.transaction.rollback 无 事务回滚；

swoft.model.saving target，具体操作实体类 实体保存中事件；

swoft.model.saved target，具体操作实体类 实体已保存事件；

swoft.model.updating target，具体操作实体类 实体更新中事件；

swoft.model.updated target，具体操作实体类 实体已更新事件；

swoft.model.creating target，具体操作实体类 实体创建中事件；

swoft.model.created target，具体操作实体类 实体已创建事件；

swoft.model.deleting target，具体操作实体类 实体删除中事件；

swoft.model.deleted target，具体操作实体类 实体已删除事件；

swoft.db.ran target，连接对象；参数1，预处理SQL；参数2，绑定参数 所有SQL已执行事件；

swoft.db.affectingStatementing target，连接对象；参数1，正在处理的PDO statement；参数2，绑定参数 update和delete正在执行事件；

swoft.db.selecting target，连接对象；参数1，正在处理的PDO statement；参数2，绑定参数 查询中事件；

对 \*\*正在进行时（ing）\*\*监听事件中调用了 “ $event->stopPropagation(true); ” 时，后续操作会被终止并直接返回结果。无法对 已完成（ed） 事件执行停止操作。

--

10、FAQ 常见问题（Frequently Asked Question）

使用模型时建议使用 select 方法，尽量不要使用 AS，可能会导致查询结果与实体映射问题。使用模型方法执行 更新/插入 时会进行过滤处理，没有定义 @Column 标签的值将会被过滤。

--

--

三、查询构造器

https://swoft.org/documents/v2/mysql/query/

--

1、简介

在 Swoft 中，数据库查询构造器为创建和执行数据库查询提供了一个方便的接口，它可用于执行应用程序中大部分数据库操作，且可在所有支持的数据库系统上运行。

Swoft 的查询构造器使用 PDO 参数绑定来保护您的应用程序免受 SQL 注入攻击。因此没有必要清理作为绑定传递的字符串。

你可以使用 DB::table('table') 得到一个 Builder 对象，也可以使用 Builder::new()->from('table')，两种方式会返回相同的结果。Builder 对象不会分配连接，只有执行 SQL 语句的时候才会从连接池中获取。

--

2、新增数据

(1)、insertGetId

通过 insertGetId 方法我们可以轻松的获取新增数据的结果自增 ID。

$id = DB::table('user')->insertGetId([

'name' => 'Swoft',

'age' => 18

]);

(2)、insert

在无需获取自增 ID 或需新增多条数据时，将数组传递给 insert 方法即可。

// 单数据

/\*\* @var bool \*/

$ret = DB::table('users')->insert(

['email' => 'john@example.com', 'votes' => 0]

);

// 多数据

/\*\* @var bool \*/

$ret = DB::table('users')->insert([

['email' => 'taylor@example.com', 'votes' => 0],

['email' => 'dayle@example.com', 'votes' => 0]

]);

--

3、删除数据

和模型一样，查询构造器也可以使用 delete 方法将数据从表中删除。

DB::table('users')->where('votes', '>', 100)->delete();

(1)、截断表

截断（Truncate）方法会清空数据表并重置自增 ID 为 0，请慎用该方法。

DB::table('users')->truncate();

--

4、更新数据

在查询构造器中，和模型一样可以通过 update 方法更新已有的数据。update 方法和 insert 方法相同，接受需更新字段和值的数组。同时也可以使用 where 链式调用进行条件过滤。

(1)、更新 JSON 字段

更新类型为 JSON 的字段时，你可以使用 -> 语法访问 JSON 对象，该操作 只能 用于支持 JSON 字段类型的数据库(比如mysql5.7)。

DB::table('users')

->where('id', 1)

->update(['options->enabled' => true]);

(2)、自增/自减

自增/自减操作默认对字段进行 +1 或 -1 更新。第三个参数为指定需同时更新的数据。

DB::table('users')->increment('votes');

DB::table('users')->increment('votes', 5, ['name' => 'Swoft']);

DB::table('users')->decrement('votes');

DB::table('users')->decrement('votes', 5, ['name' => 'Swoft']);

//

//// 如需进行自定义更新操作，参考示例

$res = DB::table('user')->where('id', $id)->update([

'posts' => DB::raw('`posts` + 1'),

'views' => Expression::new('`views` + 1'),

'name' => 'Swoft',

]);

DB::raw($sql) 等同 Expression::new($sql)。使用这两个方法时注意防范 SQL 注入。

--

5、查询数据

查询构造器中查询方法与模型完全兼容。

(1)、单行/单列查询

//// 通过 first 方法从数据表中获取一行数据。

$user = DB::table('user')->where('name', 'Swoft')->first();

if ($user) {

echo $user->name;

}

//

//// 如果只需获取一行数据中的某个字段值，则可以通过 value 方法获取。

$name = DB::table('users')->where('name', 'Swoft')->value('name');

//

//// 单列数据获取通过 pluck 方法完成。

$titles = DB::table('articles')->pluck('title');

foreach ($titles as $title) {

echo $title;

}

//

//// 还可以在返回的结果集中指定字段的自定义键值，指定 pluck 方法的第二个参数即可。

$roles = DB::table('users')->pluck('email', 'name');

foreach ($roles as $name => $email) {

echo $email;

}

(2)、多行查询

与模型中的查询几乎一致，不同的是，使用构造器时需通过 table 方法指定数据表，然后通过 get 方法获取。当然，也可以在之前通过链式调用 where 方法进行条件过滤。

$users = DB::table('user')->get();

foreach ($users as $user) {

echo $user->name;

}

//

如需更快的查询全部的数据，可以用 cursor 方法。其底层采用 yield 实现。

$users = DB::table('user')->cursor();

foreach ($users as $user){

echo $user->name;

}

(3)、结果分块

如果需要处理上千条数据时，你可以考虑使用 chunk 方法进行数据分块。该方法一次获取结果集的一小块，并将其传递给闭包函数进行处理。例如，我们可以将全部 user 表数据切割成一小块，一次处理 100 条记录。

DB::table('users')->orderBy('id')->chunk(100, function (Swoft\Stdlib\Collection $users) {

foreach ($users as $user) {

echo $user->name;

}

});

//

在闭包函数中返回 false 来终止继续获取分块结果

DB::table('users')->orderBy('id')->chunk(100, function (Swoft\Stdlib\Collection $users) {

return false;

});

闭包传递的 $users 是一个 Collection 对象，each 方法也是通过 chunk 实现，只是参数与位置不同。

(4)、聚合函数

查询构造器支持各种聚合函数，比如：count、avg、max、min 、sum 等。你可以在构造任何查询语句后调用。

$userNum = DB::table('users')->count();

$price = DB::table('orders')->max('price');

$price = DB::table('orders')->where('status', 1)->avg('price');

没有查询到任何数据时将返回一个 int 类型的 0。除 count 方法固定返回 int 类型外，其它聚合函数可能涉及到浮点型计算，Swoft 底层未对数据类型进行强转，所以返回类型为 float|int。

(5)、原生语句

注意：原生语句将会被当做字符串注入到查询中，因此应当小心使用防止 SQL 注入。

//// selectRaw

$users = DB::table('user')

->selectRaw('count(\*) AS `user\_count`, avg(age) AS `avg\_age`'))

->get();

//

//// whereRaw/orWhereRaw

whereRaw 和 orWhereRaw 方法将原生的 where 注入到你的查询中。第二个参数为可选项，值为需绑定参数的数组。

$users = DB::table('user')

->whereRaw('age > :age', ['age' => 18])

->select('name', 'age as user\_age')

->get();

//

//// havingRaw/orHavingRaw

havingRaw 和 orHavingRaw 方法用于将原生字符串设置为 having 语句的值。

$orders = DB::table('user')

->selectRaw('sum(age) as age')

->groupBy('user\_desc')

->havingRaw('age > ?', [17])

->get();

//

//// orderByRaw

orderByRaw 方法用于将原生字符串设置为 ORDER BY 子句的值。

$time = time();

$orderBy = 'if(`dead\_time` > ' . $time . ', update\_time, 0) DESC, create\_time DESC';

$orders = DB::table('ticket')

->orderByRaw($orderBy)

->get();

//

//// fromRaw

fromRaw 方法用于自定义 FROM 关键字参数，比如使用 强制索引：

$sql = DB::table('')

->select('id', 'name')

->fromRaw('`user` FORCE INDEX(`idx\_user`)')

->get();

(6)、其它方法

//// 记录是否存在

除 count 方法可以判断结果是否存在外，还可以使用 exists 及 doesntExist 方法。

return DB::table('orders')->where('id', 1)->exists();

return DB::table('orders')->where('id', 1)->doesntExist();

//

//// 指定字段

指定字段可以通过 find、first、get 指定外，也可以通过 select 指定。还可以通过 AS 设置别名。

$users = DB::table('users')->get(['name', 'user\_desc AS udesc']);

$users = DB::table('users')->select('name', 'user\_desc AS udesc')->get();

如果已存在一个查询构造器实例，可以通过 addSelect 方法添加字段。

$query = DB::table('users')->select('name');

$users = $query->addSelect(['age'])->get();

//

//// 去重

distinct 方法会强制查询结果不重复。

$users = DB::table('users')->distinct()->get();

--

6、Join 关联

Swoft 查询构造器支持完整的 Join 操作，基本语法为 join(右表名, 左表字段, 操作符, 右表字段)。

(1)、Inner Join

$users = DB::table('users')

->join('contacts', 'users.id', '=', 'contacts.user\_id')

->join('orders', 'users.id', '=', 'orders.user\_id')

->select('users.\*', 'contacts.phone', 'orders.price')

->get();

(2)、Left Join

「左连接」 关联与 Join 使用方法相同。

$users = DB::table('users')

->leftJoin('posts', 'users.id', '=', 'posts.user\_id')

->get();

(3)、Right Join

「右连接」 关联与 Join 使用方法相同。

$users = DB::table('users')

->rightJoin('posts', 'users.id', '=', 'posts.user\_id')

->get();

(4)、Cross Join

crossJoin 为「交叉连接」，交叉连接会生成两表的笛卡尔积。

$users = DB::table('users')

->crossJoin('orders', 'orders.user\_id', '=', 'users.id')

->get();

(5)、JoinSub 关联子查询

你可以使用 joinSub、leftJoinSub、rightJoinSub 方法关联子查询。

$latestPosts = DB::table('posts')

->select('MAX(created\_at) as last\_created\_at')

->where('is\_published', true)

->groupBy('user\_id');

//

// $latestPosts 为一个 Query 对象

$users = DB::table('users')

->joinSub($latestPosts, 'latest\_posts', function($join) {

$join->on('users.id', '=', 'latest\_posts.user\_id');

})->get();

(6)、高级用法

join 方法的第一个参数支持闭包函数，可通过该方式实现高级用法。

DB::table('users')

->join('contacts', function (Swoft\Db\Query\JoinClause $join) {

$join->on('users.id', '=', 'contacts.user\_id')->orOn(...);

})

->get();

//

如果你想要在 Join 上使用「where」风格的语句，你可以在连接上使用 where 和 orWhere 方法。这些方法会将列和值进行比较，而不是列和列进行比较。

DB::table('users')

->join('contacts', function ($join) {

$join->on('users.id', '=', 'contacts.user\_id')->where('contacts.user\_id', '>', 5);

})

->get();

(7)、Union 关联

查询构造器还支持 union 方法「联合」两个查询。比如，在创建一个查询后使用 union 方法和第二个查询进行联合。

Builder::new()

->from('users')

->unionAll(function (Builder $builder) {

$builder->from('users');

})

->union(Builder::new()->from('user'))

->get();

unionAll 方法与 union 方法用法相同。

--

7、条件语句

(1)、基础运用

where 方法中的三个参数分别为：字段名、操作符、比较值。

$users = DB::table('users')->where('money', '>=', 100)->get(); // 查询 money 大于等于 100 的用户;

//

如果条件中的操作符为 =，可将比较值简化至第二个参数。

$users = DB::table('users')->where('age', 18)->get();

//

其它运用示例。

// <> 与 != 皆为不等于

$users = DB::table('users')->where('age', '<>', 18)->get();

$users = DB::table('users')->where('age', '!=', 18)->get();

// 模糊查询

$users = DB::table('users')->where('name', 'LIKE', '%Swoft%')->get();

// 混合条件数组

$users = DB::table('users')->where([

'name' => 'Swoft',

['status', '=', 1],

['age'], '>=', 18

])->get();

// “或”语句，在 where 中第四个参数传入 or 或直接使用 orWhere 方法

$users = DB::table('users')

->where('money', '>=', 100, 'or')

->orWhere('name', 'John')

->get();

仅 where 方法存在第四个参数，默认值为 and。

(2)、高级运用

//// 闭包函数

where 方法中第一个参数支持闭包函数，通过该方式可以实现复杂的逻辑操作。

$isLogin = true;

DB::table('users')->where(function (Builder $query) use ($isLogin) {

if ($isLogin) {

$query->select('name', 'age', 'user\_desc AS udesc');

} else {

$query->select("'未登录' AS name");

}

})->get();

//

whereExists 与 where 方法使用类似。

DB::table('users')->whereExists(function ($query) {

$query->from('orders')->whereRaw('orders.user\_id = users.id');

})->get();

(3)、JsonWhere

同 [更新 JSON 字段](#更新 JSON 字段)，我们也能对 JSON 类型的字段应用条件。通过 -> 获取 JSON 对象属性。

$users = DB::table('users')

->where('options->language', 'en')

->get();

$users = DB::table('users')

->where('preferences->dining->meal', 'cookie')

->get();

//

使用 whereJsonContainers 查询 JSON 数组。

$users = DB::table('users')

->whereJsonContains('options->languages', 'zh')

->get();

$users = DB::table('users')

->whereJsonContains('options->languages', ['zh', 'en'])

->get();

(4)、其它 Where 语句

whereBetween：用于验证字段值是否在指定两值之间。

whereNotBetween：与 whereBetween 作用相反。

whereIn：用于验证字段值是否在指定数组中。

whereNotIn：与 whereIn 作用相反。

whereNull：用于验证字段值是否为 NULL。

whereNotNull：与 whereNull 作用相反。

whereDate：用于比较字段值与给定的日期。

whereYear：用于比较字段值与给定的年份。

whereMonth：用于比较字段值与给定的月份。

whereDay：用于比较字段值与给定的日期。

whereTime：用于比较字段值与给定的时间，格式为 H:i:s。

whereColumn：用于比较两个字段值，支持运算符 =、>、>=、<、<=。

在使用日期相关函数时需注意 MySQL 时区 设置。

--

8、分组、分页及排序

(1)、分组

groupBy 和 having 方法可以对结果进行分组。

$users = DB::table('users')

->selectRaw("count(\*) count")

->groupBy('type')

->having('count', '>', "100")

->get();

(2)、分页

limit/take（take方法为limit方法的别名）用于限制返回结果数量，而offset/skip（skip方法为offset方法的别名）用于指定跳过的数量，从而实现分页。

$users = DB::table('users')

->offset(10)

->limit(5)

->get();

//

我们也可以使用更简单的 forPage 方法实现分页。实际等同 offset 结合 limit 运用。

DB::table('users')

->forPage($page, $pageSize)

->get();

(3)、排序

orderBy 方法用于对结果集按需排序。

DB::table('users')

->orderBy('age', 'asc')

->get();

DB::table('users')

->orderByDesc('age')

->get();

//

latest/oldest方法可以非常轻松地通过日期排序，它默认使用created\_at字段作为排序依据，当然你也可以指定字段。

$users = DB::table('users')

->latest()

->get();

$users = DB::table('users')

->oldest('created\_at')

->get();

//

使用 inRandomOrder 方法进行随机排序。随机获取一条数据。

$randomUser = DB::table('users')

->inRandomOrder()

->first();

--

9、锁机制

(1)、共享锁

查询构造器支持在 SELECT 语法上实现「悲观锁定」函数。若想在查询中实现一个「共享锁」，你可以使用sharedLock 方法。共享锁可防止选中的数据列被篡改，直到事务被提交为止。

DB::table('orders')->where('user\_id', $userId)->sharedLock()->get();

(2)、排它锁

使用 lockForUpdate 方法实现排它锁可避免行被其它共享锁修改或选取。

DB::table('users')->find($id)->lockForUpdate()->first();

--

10、连接池

当应用存在多个连接池时，默认使用的连接池为 db.pool，可通过 query 方法指定连接池。

$pool = 'db.pool2';

$user = DB::query($pool)->from('users')->find($id);

--

11、连接释放

只有在 执行 SQL 的时候数据库底层才会从连接池中获取连接，执行完毕后会自动释放。Builder 对象不再依赖 Connection。

连接释放即将连接返还到连接池。

--

12、SQL 打印

通过 toSql 方法获取需执行的 SQL。

echo DB::table('users')->where('id', $id)->toSql();

--

13、FAQ 常见问题（Frequently Asked Question）

// where 方法闭包函数错误运用示例。

$res = DB::table('user')

->where(function (Builder $query) {

$query->forPage(1, 10)

->orderBy('age', 'ase')

->where('id', 1);

})

->orderBy('id', 'desc')

->get();

上述示例中，在闭包函数使用的 orderBy 和 forPage 方法不会生效。

--

--

四、事务

https://swoft.org/documents/v2/mysql/transaction/

--

1、开启事务

注意：开启事务后，事务之间的所有操作都同属一个连接，因此事务中 不能 使用并发操作。

(1)、快速使用

开启一个事务的最简单方式是使用 DB 下的 transaction 方法。

DB::transaction(function () {

DB::table('users')->update(['name' => 'Swoft']);

DB::table('posts')->delete();

});

// 需指定连接池时

DB::connection('db.pool')->transaction(function () {

DB::table('users')->update(['name' => 'Swoft']);

DB::table('posts')->delete();

});

// 参数说明

$callback：传入一个闭包函数用于数据库操作

$attempts：重试次数，默认为 1

如果闭包函数中没有发生任何异常，该事务会 自动提交。如果闭包函数中出现异常时，事务会 自动回滚。

(2)、处理死锁

transaction 方法的第二个参数用来指定事务发生死锁时重复执行的次数。一旦定义的次数尝试完毕，就会抛出一个异常。

DB::transaction(function () {

DB::table('users')->update(['name' => 'Swoft']);

DB::table('posts')->delete();

}, 3);

(3)、手动开启

如需自行控制事务提交及回滚操作，可以使用 DB 下的 beginTransaction 方法开启事务。

DB::beginTransaction();

// 需指定连接池时

DB::connection('db.pool')->beginTransaction();

事务一旦开启，当前连接会绑定到当前的协程环境中，使提交、回滚、查询同属一个连接以保证数据的安全性和完整性，只有在事务被提交或回滚后才会解除绑定。不同连接池的事务相互独立，不存在关联关系。

在复杂逻辑中，为避免死锁情况，请一定记得在每个流程分支中执行事务提交或回滚操作。

(4)、事务回滚

操作发生异常或逻辑判断错误时应对事务进行回滚。

DB::rollBack();

// 需指定连接池时

DB::connection('db.pool')->rollBack();

(5)、事务提交

操作未发生异常且逻辑判断正常时应对事务进行提交，以确保数据正常写入到数据库中。

DB::commit();

// 需指定连接池时

DB::connection('db.pool')->commit();

--

2、常见问题

(1)、事务嵌套

MySQL 本身不支持事务嵌套，如果在事务中进行嵌套操作时，上一个事务 会被 隐式提交，然后开启一个新事务。而在框架中，利用数据库的 Savepoint（保存点）功能，可以实现事务嵌套操作。在 MySQL 中，利用 Savepoint 可以回滚指定部分的事务，从而使得事务处理更加精细灵活。所以在 Swoft 中对事务进行了嵌套时，嵌套的事务会保存在 Savepoint。

// 事务一

DB::beginTransaction();

$user = User::find($id);

$user->update(['name' => 'Swoft Framework']);

// 事务二

DB::beginTransaction();

User::find($id)->update(['name' => 'Swoft']);

// 事务二回滚

DB::rollBack();

// 事务一提交

DB::commit();

上述示例代码中，事务二 进行的回滚操作并不会影响外层 事务一 的数据更新操作。Savepoint 仅部分数据库支持，而非所有数据库都支持该功能。

(2)、忘记提交

DB::connection('db.pool')->beginTransaction();

User::find($id)->update(['name' => 'Swoft']);

类似上述代码我们并没有进行事务提交或回滚操作，此时 Swoft 会在 SwoftEvent::COROUTINE\_DEFER 事件中检查当前是否处于事务开启状态，如果是则会自动回滚，然后连接归还至连接池。

(3)、错误运用示例

DB::beginTransaction();

$user = User::find($id);

sgo(function () use ($id) {

$user1 = User::find($id);

});

$user->update(['name' => 'Swoft' . mt\_rand(110, 10000)]);

DB::commit();

类似这样的代码虽然能够正常执行，但这是 错误的写法。前文已强调：请不要在事务中使用协程。使用协程会造成上下文不一致导致数据错乱。事务与 当前协程 是绑定关系，切换协程后会使用另一个新的连接。

--

--

五、原生操作

https://swoft.org/documents/v2/mysql/origin/

--

1、简介

在 DB 中 Swoft 提供了 select、selectOne、update、insert、delete、cursor、statement、affectingStatement、unprepared 方法进行原生操作。

--

2、新增数据

$bool = DB::insert('INSERT INTO users (`id`, `name`) VALUES (?, ?)', [1, 'Swoft']);

// 参数说明

$query：SQL 字符串。

$bindings：绑定参数。

如示例代码中，id 及 name 的值通过 $bindings 参数传入，该操作返回 bool 值。使用参数绑定可有效避免 SQL 注入。

--

3、删除数据

$deletedLines = DB::delete('DELETE FROM `users`');

// 参数说明

$query：SQL 字符串。

$bindings：绑定参数。

该方法返回受影响的行数。

--

4、更新数据

$updatedLines = DB::update('UPDATE `users` SET `status` = ? WHERE `name` = ?', [1, 'Swoft']);

// 参数说明

$query：SQL 字符串。

$bindings：绑定参数。

该方法返回受影响的行数。

--

5、查询数据

以下三种查询方式拥有相同的参数。

// 参数说明

$query：SQL 字符串。

$bindings：绑定参数。

$useReadPdo：是否使用读库，默认为 true。

(1)、Select

$users = DB::select('SELECT \* FROM `users` WHERE `status` = ?', [1]);

select 方法返回一个数组，数组中的每个结果都是 array。

(2)、SelectOne

如需查询一条数据可使用 selectOne 方法。

$sql= 'SELECT \* FROM `users` WHERE `id` = ?';

$user = DB::selectOne($sql, [1]);

if ($user) {

echo $user['name'];

}

selectOne 方法执行成功后返回一个数组。

(3)、Cursor 游标

游标用于快速遍历所有数据。

$sql= 'SELECT \* FROM `users`';

$users = DB::cursor($sql);

foreach($users as $user){

echo $user['name'];

}

底层采用 yield 机制获取数据，比 chunk 方法更快。

(4)、命名绑定

除使用 ? 进行参数绑定外，还可以使用如下方式进行命名绑定。

$results = DB::select('SELECT \* FROM `user` WHERE `id` = :id', ['id' => 1]);

--

6、其它语句

(1)、Statement

数据库语句不存在返回值时，可以使用 statement 方法来执行。

$bool = DB::statement('DROP TABLE `users`');

该方法支持参数绑定，返回执行结果的 bool 值。

(2)、Unprepared

执行未预处理的 SQL 语句，也是真正意义上的原生语句。使用 unprepared 方法执行。

$unprepared = DB::unprepared('DROP TRIGGER IF EXISTS `sync\_to\_item\_table`;');

if ($unprepared) {

// TODO:

}

仅该方法支持多条 SQL，同时该方法不支持参数绑定，返回执行结果的 bool 值。

执行多条 SQL 时注意“;”分号隔开。

--

--

六、数据库切换

https://swoft.org/documents/v2/mysql/select-db/

--

1、简介

在 Swoft 2.0.2 版本之前，连接池中没有数据库切换功能，这导致了不同数据库需要配置多个连接池，大大增加了维护成本。所以在 2.0.2 开始新增了切换数据库功能。你可以在链式操作中使用 db() 方法进行指定，这显然不够灵活难以维护，下面推荐一个根据上下文切换数据的操作。Swoft 版本需 >= 2.0.2。

--

2、DbSelector

(1)、实现

使用 DbSelector 需实现 Swoft\Db\Contract\DbSelectorInterface 接口。

<?php declare(strict\_types=1);

namespace App\Common;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Db\Connection\Connection;

use Swoft\Db\Contract\DbSelectorInterface;

/\*\*

\* Class DbSelector

\*

\* @since 2.0

\*

\* @Bean()

\*/

class DbSelector implements DbSelectorInterface

{

/\*\*

\* @param Connection $connection

\*/

public function select(Connection $connection): void

{

// 在请求中获取 ID

$selectIndex = (int)context()->getRequest()->query('id', 0);

$createDbName = $connection->getDb();

if ($selectIndex == 0) {

$selectIndex = '';

}

// 数据库名 + ID。例如：order\_database\_1，好处是会根据上下文自动切库

$dbName = sprintf('%s%s', $createDbName, (string)$selectIndex);

$connection->db($dbName);

}

}

?>

(2)、配置

实现 DbSelector 后，还需在 bean.php 中定义 dbSelector 属性，例如。

'db2' => [

'dsn' => 'mysql:dbname=test;host=127.0.0.1:3306',

'username' => 'root',

'password' => '123456',

'dbSelector' => bean(App\Common\DbSelector::class) //自定义数据库选择;

],

'db2.pool' => [

'database' => bean('db2')

]

通过在 db 中添加 dbSelector 属性以指定实现类。之后每次操作均会调用该类自动切换数据库。使用这种方式后大大降低了维护成本，官方推荐。

(3)、其它说明

如果 DbSelector 的方式无法满足需求时，仅能通过 db 方法手动指定数据库。

// 模型类

User::db('test2')->insertGetId([

'name' => uniqid(),

'password' => md5(uniqid()),

'age' => mt\_rand(1, 100),

'user\_desc' => 'Swoft'

]);

// DB 方式

DB::table('user')->db('test2')->insertGetId([

'name' => uniqid(),

'password' => md5(uniqid()),

'age' => mt\_rand(1, 100),

'user\_desc' => 'Swoft Framework'

]);

不推荐该方式，使用 db 方法会加大维护成本，且容错率低。

--

--

第八卷 Redis

--

--

一、配置

https://swoft.org/documents/v2/redis/config/

//

GitHub: https://github.com/swoft-cloud/swoft-redis

--

1、安装

// Redis 组件需独立安装

composer require swoft/redis

--

2、基础配置

Redis 配置与数据库配置相同，位于 app/bean.php 文件，配置的 redis 也是一个 bean 对象。

return [

'redis' => [

'class' => Swoft\Redis\RedisDb::class,

'host' => '127.0.0.1',

'port' => 6379,

'password' => '',

'database' => 0,

'retryInterval' => 10,

'readTimeout' => 0,

'timeout' => 10,

'option' => [

'prefix' => 'Swoft',

'serializer' => Redis::SERIALIZER\_PHP

]

]

]

// 配置项说明

class：Redis 驱动类，默认为 Swoft Redis 驱动，自定义时需指定；

host：Redis 连接地址，默认为 127.0.0.1；

port：Redis 连接端口，默认为 6379；

password：Redis 连接密码，默认为空；

database：指定 Redis 库，默认为 0；

retryInterval：重试间隔，默认为 10 秒；

readTimeout：读取超时时间，默认为 0 秒，永不超时；

timeout：连接超时时间，默认为 0 秒，永不超时；

option：Redis 额外配置，prefix 存储前缀，默认为空；serializer 序列器，支持以下四种方式，不使用请设置为 Redis::SERIALIZER\_NONE 或 0，Redis::SERIALIZER\_PHP PHP 默认序列器、Redis::SERIALIZER\_IGBINARY 需安装 Igbinary 扩展并启用、Redis::SERIALIZER\_MSGPACK 需安装 MessagePack 扩展并启用、Redis::SERIALIZER\_JSON：需安装 Json 扩展（一般情况下默认已安装并启用）。

--

3、集群配置

集群配置与基础配置同样配置于 app/bean.php 文件。集群配置增加了 clusters，连接机制会优先使用 集群配置，不存在集群配置时才会使用的普通配置。

'redis' => [

'class' => Swoft\Redis\RedisDb::class,

'option' => [

'timeout' => 3,

'persistent' => true

],

'clusters' => [

[

'host' => '127.0.0.1',

'port' => 6379,

'password' => '123456',

'database' => 1,

'prefix' => 'Swoft',

'read\_timeout' => 10

]

]

]

// 配置项说明

class：Redis 驱动类，默认为 Swoft Redis 驱动，自定义时需指定；

option：集群配置项，timeout 连接超时时间，默认为 0 秒；persistent 是否启用持久化连接，默认为 false 不启用；

clusters：集群节点，host Redis 连接地址；port Redis 连接端口；password Redis 连接密码；database Redis 库；prefix 存储前缀；read\_timeout 读取超时时间；

集群不存在序列化配置。

--

4、连接池配置

连接池适合需使用不同的 Redis 库或不同节点等场景。连接池的配置同样在 app/bean.php 文件。

注意：每一个 Worker 都会创建一个同样的连接池，连接池并不是越多越好，需根据服务器配置和 Worker 数量权衡。

(1)、集群连接池配置

'redis-clusters' => [

'class' => Swoft\Redis\RedisDb::class,

'option' => [

'timeout' => 10,

'persistent' => true

],

'clusters' => [

[

'host' => '127.0.0.1',

'port' => 6379,

'password' => '123456',

'database' => 1,

'prefix' => 'Swoft-Clusters',

'read\_timeout' => 1

]

]

],

'redis.clusters-pool' => [

'class' => Swoft\Redis\Pool::class,

'redisDb' => bean('redis-clusters'),

'minActive' => 10,

'maxActive' => 20,

'maxWait' => 0,

'maxWaitTime' => 0,

'maxIdleTime' => 40

]

// 连接池配置项说明

class：连接池驱动类，仅自定义时需指定，默认为 Swoft 连接池驱动；

redisDb：指定 Redis 配置；

minActive：最小连接数；

maxActive：最大连接数；

maxWait：最大等待连接数，默认为 0 无限制；

maxWaitTime：连接最大等待时间，默认为 0 秒无限制；

maxIdleTime：连接最大空闲时间，单位秒；

//

// 集群连接池运用示例

Redis::connection('redis.clusters-pool')->get($key);

(2)、普通连接池配置

'redis-2' => [

'class' => Swoft\Redis\RedisDb::class,

'host' => '10.0.0.2',

'port' => 6379,

'database' => 1,

'retryInterval' => 10,

'readTimeout' => 0,

'timeout' => 2,

'option' => [

'prefix' => 'Swoft',

'serializer' => Redis::SERIALIZER\_PHP

]

],

'redis.pool-2' => [

'class' => Swoft\Redis\Pool::class,

'redisDb' => bean('redis-2'),

'minActive' => 10,

'maxActive' => 20,

'maxWait' => 0,

'maxWaitTime' => 0,

'maxIdleTime' => 60,

]

// 连接池配置项说明

class：连接池驱动类，仅自定义时需指定。默认为 Swoft 连接池驱动；

redisDb：指定 Redis 配置；

minActive：最少连接数；

maxActive：最大连接数；

maxWait：最大等待连接数，默认为 0 无限制；

maxWaitTime：连接最大等待时间，默认为 0 秒无限制；

maxIdleTime：连接最大空闲时间，单位秒；

// 更换默认连接池中的连接配置

'redis.pool' => [

'class' => Swoft\Redis\Pool::class,

'redisDb' => bean('redis-custom')

]

之后对 Redis 后的操作所使用的连接为 redis-custom。

Redis::set($key, ['name' => 'Swoft']);

--

--

二、如何使用

https://swoft.org/documents/v2/redis/usage/

//

//// Redis\_Redis命令参考

http://doc.redisfans.com/

--

1、基础运用

Redis::set('user:profile:' . $id, "Swoft");

$userDesc = Redis::get('user:profile:' . $id);

//

你可以通过 Redis:: 调用任何 Redis 命令。Swoft 使用魔术方法将命令传递给 Redis 服务端，因此只需传递 Redis 命令所需的参数即可。

Redis::set('name', 'Taylor');

$values = Redis::lrange('names', 5, 10);

所有方法和操作 phpredis 相同，本质是通过魔术方法代理访问的，让操作变得更简单。

//

当然普通的字符串内容不能满足我们的日常开发，你也可以使用数组。

$data = [

'Swoft' => [

'id' => 1,

'desc' => 'Great Framework'

]

];

Redis::set($key, $data);

通过 Redis::get() 方法调用时，底层会自动反序列化，保证数据的一致性。

//

有序集合 zAdd 使用示例。

$scores = [

'key1' => 11,

'key3' => 11,

'key4' => 11,

'key2' => 21,

];

$result1 = Redis::zAdd('keys', $scores);

Key 为成员，value 为分数。成员不可重复。

--

2、连接池注入

//// 示例

<?php declare(strict\_types=1);

namespace App\Http\Controller;

use Swoft\Bean\Annotation\Mapping\Inject;

use Swoft\Http\Server\Annotation\Mapping\Controller;

use Swoft\Http\Server\Annotation\Mapping\RequestMapping;

use Swoft\Redis\Pool;

use Throwable;

/\*\*

\*

\* @Controller("redis")

\*/

class RedisController

{

/\*\*

\* 例子一：如果 Inject 没有参数则会使用 var 定义的类型

\*

\* @Inject()

\*

\* @var Pool 默认连接使用的是 redis.pool

\*/

private $redis;

/\*\*

\* 例子二：如果 Inject 指定参数，则使用指定的 pool 注入到该属性。和 var 定义的类型没关系

\*

\* @Inject("redis.inc.pool")

\*

\* @var Pool

\*/

private $redisInc;

/\*\*

\* @RequestMapping(route="find")

\*

\* @return array

\*

\* @throws Throwable

\*/

public function find(): array

{

$this->redis->set('user', ['name' => 'Swoft', 'age' => 2]);

$this->redisInc->incrBy('user-count', 1);

return $this->redis->get('user');

}

}

?>

你可以在 Inject 中指定使用的连接池，如果在 Inject 中没有指定将默认使用 @var 指定的类型注入。

--

3、连接池指定

所有的连接都通过连接池分配。使用自定义连接示例。

$poolName = 'redis.pool-clusters';

$redis = Redis::connection($poolName);

$redis->get('a');

//

如何创建连接池请参考 连接池配置( https://swoft.org/documents/v2/redis/config/#%E8%BF%9E%E6%8E%A5%E6%B1%A0%E9%85%8D%E7%BD%AE )，默认是连接池为 redis.pool。

如果 Redis::connection() 没有指定连接池，则会从默认连接池中获取连接。该方法返回 连接 而不是连接池。可以共享连接池但不能共享连接。

--

4、事件监听

(1)、before

在 Redis 执行前 Swoft 底层会抛出 RedisEvent::BEFORE\_COMMAND 事件。

<?php declare(strict\_types=1);

namespace App\Listener;

use Swoft\Event\Annotation\Mapping\Listener;

use Swoft\Event\EventHandlerInterface;

use Swoft\Event\EventInterface;

use Swoft\Redis\RedisEvent;

/\*\*

\* Class FinishListener

\*

\* @since 2.0

\*

\* @Listener(event=RedisEvent::BEFORE\_COMMAND)

\*/

class RedisBeforeListener implements EventHandlerInterface

{

/\*\*

\* @param EventInterface $event

\*/

public function handle(EventInterface $event): void

{

[$method, $parameters] = $event->getParams();

}

}

?>

(2)、after

在 Redis 执行后会抛出 RedisEvent::AFTER\_COMMAND 事件。你可以监听命令的执行情况，如果在执行命令中出现异常则不会抛出 RedisEvent::AFTER\_COMMAND 事件。

<?php declare(strict\_types=1);

namespace App\Listener;

use Swoft\Event\Annotation\Mapping\Listener;

use Swoft\Event\EventHandlerInterface;

use Swoft\Event\EventInterface;

use Swoft\Redis\RedisEvent;

/\*\*

\* Class FinishListener

\*

\* @since 2.0

\*

\* @Listener(event=RedisEvent::AFTER\_COMMAND)

\*/

class RedisAfterListener implements EventHandlerInterface

{

/\*\*

\* @param EventInterface $event

\*/

public function handle(EventInterface $event): void

{

[$method, $parameters, $result] = $event->getParams();

}

}

?>

--

5、缓存命中率

如需查看某个 key 的获取命中情况，可以在你的 notice 日志中获取。前缀格式为 redis.hit/req....，有助于你分析缓存命中率。

--

--

三、事务

https://swoft.org/documents/v2/redis/transaction/

--

1、简介

Redis 事务不支持回滚，但能保证原子性。但通过 lua 脚本也能实现 Redis 事务效果。

--

2、示例

事务操作的返回数据比较特殊，返回偶数为是否成功，奇数为执行 key。下方为一个结果遍历的事务操作示例。

$count = 2;

$result = Redis::transaction(function (Redis $redis) use ($count) {

for ($i = 0; $i < $count; $i++) {

$key = "key:{$i}";

$redis->set($key, $i);

$redis->get($key);

}

});

/\*

$result = array(4) {

[0] => bool(true)

[1] => int(0)

[2] => bool(true)

[3] => int(1)

}\*/

foreach ($result as $index => $value) {

if ($index % 2 == 0) {

// 是否执行成功

if ($value) {

// TODO

}

}

}

--

--

四、管道

https://swoft.org/documents/v2/redis/pipeline/

--

1、简介

如果你需要在同一个操作中向服务端发送多个命令，推荐你使用管道命令（Pipeline）。pipeline 方法接收一个带有 Redis 实例的闭包函数。你可以将所有需要执行的命令发送给这个 Redis 实例，它们会一次性执行完毕。

--

2、示例

比如连续设置 10 个 Key，返回值为数组，你可以通过遍历判断是否全部执行成功。

public function demoPipeline()

{

$count = 10;

// 管道命令

$result = Redis::pipeline(function (Redis $redis) use ($count) {

for ($i = 0; $i < $count; $i++) {

$redis->set("key:{$i}", $i);

}

});

// count($result) == $count;

foreach ($result as $index => $value) {

// $index 命令索引

// $value == true 或 $value == false

}

}

--

--

五、发布订阅

https://swoft.org/documents/v2/redis/pub-sub/

--

1、简介

Redis 的列表类型键可以用来实现队列，并且支持阻塞式读取，所以 Redis 能够非常容易地实现一个高性能的优先队列。同时在更高层面上，Redis 还支持“发布/订阅”的消息模式，可以基于此构建一个聊天系统。

--

2、发布示例

发布（Publish）即将消息发布到频道中。示例代码。

// 发送消息

Redis::publish('chan-1', 'Hello, World!');

// 发送消息

Redis::publish('chan-2', 'Hello, Swoft!');

// 参数说明

$channel：目标频道；

$message：消息内容；

--

3、订阅示例

订阅（Subscribe）即从频道中接收消息。示例代码。

function f($redis, $chan, $msg) {

switch($chan) {

case 'chan-1':

// TODO:

break;

case 'chan-2':

// TODO:

break;

case 'chan-3':

// TODO:

break;

}

}

// 订阅 3 个频道

Redis::subscribe(['chan-1', 'chan-2', 'chan-3'], 'f');

// 参数说明

$channels：目标频道，字符串数组；

$callback：回调函数，接收 3 个参数，$redis Redis 实例、$chan 频道名称、$msg 接收消息内容；

--

4、匹配模式(订阅频道)

使用 psubscribe 方法支持以通配符的方式订阅频道，可以用来获取所有频道上的消息。

Redis::psubscribe(['\*'], function ($redis, $pattern, $chan, $msg) {

echo "Pattern: $pattern\n";

echo "Channel: $chan\n";

echo "Payload: $msg\n";

});

Redis::psubscribe(['users.\*'], function ($redis, $pattern, $chan, $msg) {

echo "Pattern: $pattern\n";

echo "Channel: $chan\n";

echo "Payload: $msg\n";

});

// 参数说明

$patterns：匹配规则数组；

$callback：回调函数，接收 4 个参数，$redis Redis 实例、$pattern 匹配规则、$chan 频道名称、$msg 接收消息内容；

--

5、Socket 断连问题

由于订阅功能源自原生 phredis，使用过程中可能会有 Socket 断连问题，所以在开启发布订阅前应加大 PHP 配置中 default\_socket\_timeout 的值或设置为 -1（永不超时）。该值默认为 60 秒。

ini\_set('default\_socket\_timeout', -1);

Redis::psubscribe(['users.\*'], function ($redis, $pattern, $chan, $msg) {

echo "Pattern: $pattern\n";

echo "Channel: $chan\n";

echo "Payload: $msg\n";

});

关于 default\_socket\_timeout，请参考：PHP运行时配置 - Manual( https://www.php.net/manual/zh/filesystem.configuration.php )。

--

--

第九卷 扩展组建

--

--

一、视图

https://swoft.org/documents/v2/ext-components/view/

//

Swoft 官方提供了一个简单的视图渲染组件。使用 php 原生语法，提供基本的布局，内部引入文件等功能。

--

1、安装

视图渲染作为一个额外的独立组件，需要手动安装。

(1)、Composer 安装

composer require swoft/view

(2)、Git 仓库

Github https://github.com/swoft-cloud/swoft-view

--

2、配置组件

当你安装了 view 组件后，swoft 将会自动的注册它。你可以配置视图文件存放目录。视图组件注册到容器里的名称为：view。

//

//// bean配置(file: app/beans.php)

'view' => [

// class 配置是可以省略的, 因为 view 组件里已经配置了它

// 'class' => \Swoft\View\Renderer::class,

'viewsPath' => dirname(\_\_DIR\_\_) . '/resource/views/',

],

现在在任何地方都可以通过 view() 或者 \Swoft::getBean('view') 来获取组件实例。

(1)、配置项说明

viewsPath 视图存放路径；

layout 默认的布局文件，调用 render() 方法时会默认的使用它；

suffix 默认的视图后缀(默认是 php)；

suffixes 允许的视图后缀列表，用于判断是否需要添加默认后缀；

placeholder 在布局文件里使用的内容占位符，默认 {\_CONTENT\_}；

(2)、如何查找视图

若你不添加后缀，会自动追加配置的默认后缀；

使用相对路径时，将会在我们配置的视图目录里找到对应的view文件；

使用绝对路径时，将直接使用它来渲染(支持使用路径别名 @resource/views/my-view.php)。

(3)、使用视图

通过方法：view() 渲染一个视图文件；

通过 \Swoft::getBean('view')->rander('view file') 渲染一个视图文件；

在控制器的action注释上还可以快捷地使用 @View() 来使用(2.0 暂未支持)；

(4)、使用示例

/\*\*

\* 控制器demo

\*

\* @Controller(prefix="/demo")

\*/

class DemoController

{

/\*\*

\* 视图渲染demo - 没有使用布局文件(请访问 /demo/view)

\* @RequestMapping()

\*/

public function view()

{

$data = [

'name' => 'Swoft',

'repo' => 'https://github.com/swoft-cloud/swoft',

'doc' => 'https://swoft.org/docs',

'method' => \_\_METHOD\_\_,

];

// 将会渲染 `resource/views/site/index.php` 文件

return view('site/index', $data);

}

/\*\*

\* 视图渲染demo - 使用布局文件(请访问 /demo/layout)

\*

\* @RequestMapping()

\*/

public function layout()

{

$layout = 'layouts/default.php';

$data = [

'name' => 'Swoft',

'repo' => 'https://github.com/swoft-cloud/swoft',

'doc' => 'https://swoft.org/docs',

'method' => \_\_METHOD\_\_,

'layoutFile' => $layout

];

return view('site/content', $data, $layout);

}

}

(5)、使用布局文件

使用布局文件，方式有两种：在配置中 配置默认的布局文件，那么即使不设置 layout，也会使用默认的(视图的可用配置请看上一节)；如示例一样，可以手动设置一个布局文件，它的优先级更高（即使有默认的布局文件，也会使用当前传入的替代）；你可以传入 layout=false 来禁用渲染布局文件。

--

3、加载静态文件

Swoft 可以提供静态资源访问的支持（由 swoole 提供），通常建议将静态文件放置于根目录下的 public 目录内。

(1)、配置Http Server

首先我们需要配置静态文件处理，在 app/bean.php 的 http server 加上如下配置。

'httpServer' => [

/\*\*

\* @see HttpServer::$setting

\* @link https://wiki.swoole.com/wiki/page/620.html

\*/

'setting' => [

// enable static handle

'enable\_static\_handler' => true,

// swoole v4.4.0以下版本, 此处必须为绝对路径

'document\_root' => dirname(\_\_DIR\_\_) . '/public',

]

],

//

下面是一个引用的示例（引用时无需再包含public）。

// 真实文件为： public/static/some.js

<script type="text/javascript" src="/static/some.js"></script>

(2)、引入其他视图文件

// 在视图文件里包含其他视图文件，可以使用

include(string $view, array $data, bool $outputIt = true)

fetch(string $view, array $data)

两个方法的区别是 fetch() 需要你手动调用 echo <?= $this->fetch('layouts/default/header') ?>。

变量数据有作用域限制。即是传入视图的变量，无法在包含的视图里直接使用，需要通过第二个参数 $data 传入到子级视图。

<body>

<?php $this->include('layouts/default/header', ['logo' => 'xx/yy/logo.jpg']) ?>

<div class="container">

<!-- Content here -->

<div id="page-content" style="padding: 15px 0;">{\_CONTENT\_}</div>

<?php $this->include('layouts/default/footer') ?>

</div>

</body>

--

--

二、PostgreSql

https://swoft.org/documents/v2/ext-components/postgresql/

//

curtis18/swoft-pgsql 是对 PostgreSQL 在 Swoft 中使用的简单封装，支持连接池配置和某些非连接池的原生配置函数。

--

1、安装

curtis18/swoft-pgsql 需要使用 Swoole 的 swoole/ext-postgresql 扩展。而作为 Swoft 的一个额外的扩展组件，需要手动安装。

(1)、Composer 安装

composer require curtis18/swoft-pgsql

(2)、Git 仓库

Github https://github.com/curtis18/swoft-pgsql.git

--

2、参与贡献

欢迎参与贡献，您可以 fork 开发仓库 curtis18/swoft-pgsql( https://github.com/curtis18/swoft-pgsql )；修改代码然后发起 PR；阅读 提交代码( https://swoft.org/documents/v2/contribute/sub-codes/ ) 的注意事项。

--

3、基础配置

Swoft 应用的 PostgreSQL 配置都在配置文件 app/bean.php 中。

use Swoft\Pgsql\PgsqlDb;

'pgsql' => [

'class' => PgsqlDb::class,

'host' => '127.0.0.1',

'port' => 5432,

'database' => 'dbname',

'schema' => ['postgis', 'public'],

'charset' => 'utf8',

'user' => 'username',

'password' => 'pass'

],

class 指定当前配置驱动类型；

host 连接地址 默认 127.0.0.1；

port 端口 默认 5432；

database 连接数据库，默认 postgres；

--

4、连接池配置

Swoft 所有连接池配置都差不多，配置都在配置文件 app/bean.php 中，默认的连接池名為 pgsql.pool。

'pgsql.pool' => [

'class' => \Swoft\Pgsql\Pool::class,

'pgsqlDb' => bean('pgsql'),

'minActive' => 2,

'maxActive' => 20,

'maxWait' => 0,

'maxWaitTime' => 0,

'maxIdleTime' => 60,

],

每一个 worker 都会创建一个同样的连接池。并不是越多越好，参数配置要根据，机器配置和 worker 个数衡量。

--

5、使用连接池

<?php declare(strict\_types=1);

namespace App\Model\Data;

use Swoft\Pgsql\Pool;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Bean\Annotation\Mapping\Inject;

use Swoft\Config\Annotation\Mapping\Config;

/\*\*

\* Class Test

\*

\* @since 2.0

\*

\* @Bean()

\*/

class Test

{

/\*\*

\* @Inject("pgsql.pool")

\* @var Pool

\*/

private $pgsql;

public function getTest(): array

{

$connection = $this->pgsql->createConnection();

return $connection->select("SELECT \* FROM test;");

}

public function fetchNumTest(): array

{

$connection = $this->pgsql->createConnection();

return $connection->selectFetchNum("SELECT testid, testname FROM test;");

}

public function bindingTest(int $id = 1, string name = "myname"): array

{

$connection = $this->pgsql->createConnection();

return $connection->select("SELECT \* FROM test WHERE testid = AND testname = ;", array($id, $name));

}

// 非连接池的原生配置函数

public function copyFromTest(): bool

{

$table = 'test';

$data = array(

array('1|C252525A|0|0|02921|02921|2|0|Welcome To Swoft|02921||benny|2019-09-03 14:40:55|1|'),

array('2|C252525B|0|0|02921|02921|2|0|Welcome To Swoole|02921||curtis|2019-07-03 14:40:55|1|'),

);

$connection = $this->pgsql->createConnection();

$connection->select("TRUNCATE ".$table.";");

return $connection->copyFrom($table, $data);

}

// 非连接池的原生配置函数

public function copyToTest(): array

{

$table = 'test';

$connection = $this->pgsql->createConnection();

return $connection->copyTo($table);

}

}

?>

--

--

三、Whoops

https://swoft.org/documents/v2/ext-components/whoops/

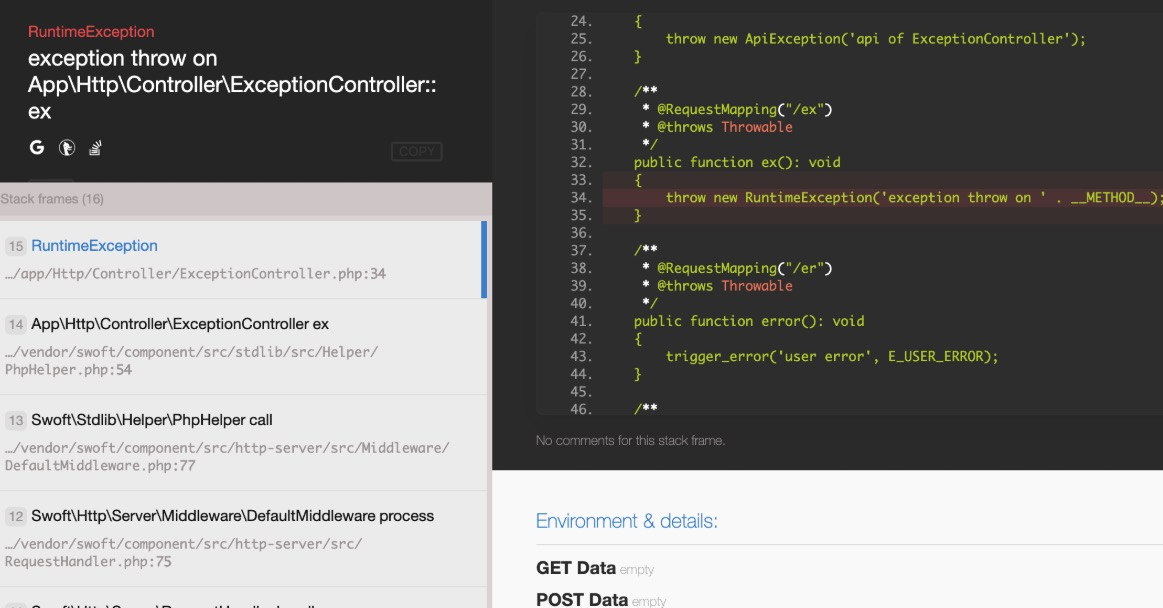
//

swoft/whoops 是对 filp/whoops 在swoft中使用的简单封装，用于渲染并显示更加利于阅读的异常错误信息。

1、安装

swoft/whoops 作为一个额外的扩展组件，需要手动安装。

( 图 https://swoft.org/img/swoft-whoops.jpg )



(1)、Composer 安装

composer require swoft/whoops

(2)、Git 仓库

Github https://github.com/swoft-cloud/swoft-whoops.git

--

2、参与贡献

欢迎参与贡献，您可以 fork 开发仓库 swoft/whoops( https://github.com/swoft-cloud/swoft-whoops )；修改代码然后发起 PR；阅读 提交代码( https://swoft.org/documents/v2/contribute/sub-codes/ ) 的注意事项。

--

3、作为中间件使用

使用 Swoft\Whoops\WhoopsMiddleware 作为一个全局的Http中间件(app/bean.php)。

'httpDispatcher' => [

// Add global http middleware

'middlewares' => [

// Notice: Please add to the first place

\Swoft\Whoops\WhoopsMiddleware::class,

],

],

注意: 请将 Swoft\Whoops\WhoopsMiddleware 加到第一个位置。

--

4、异常处理里使用

我们稍微调整一下默认的http异常处理类(App\Exception\Handler\HttpExceptionHandler)。

<?php declare(strict\_types=1);

namespace App\Exception\Handler;

use ReflectionException;

use Swoft\Bean\Exception\ContainerException;

use Swoft\Error\Annotation\Mapping\ExceptionHandler;

use Swoft\Http\Message\Response;

use Swoft\Http\Server\Exception\Handler\AbstractHttpErrorHandler;

use Swoft\Log\Helper\CLog;

use Swoft\Whoops\WhoopsHandler;

use Throwable;

use function bean;

use function context;

use const APP\_DEBUG;

/\*\*

\* Class HttpExceptionHandler

\*

\* @ExceptionHandler(\Throwable::class)

\*/

class HttpExceptionHandler extends AbstractHttpErrorHandler

{

/\*\*

\* @param Throwable $e

\* @param Response $response

\*

\* @return Response

\* @throws ReflectionException

\* @throws ContainerException

\*/

public function handle(Throwable $e, Response $response): Response

{

$request = context()->getRequest();

if ($request->getUriPath() === '/favicon.ico') {

return $response->withStatus(404);

}

// Log

CLog::error($e->getMessage());

// Debug is false

if (!APP\_DEBUG) {

return $response

->withStatus(500)

->withContent($e->getMessage());

}

// Debug is true

$whoops = bean(WhoopsHandler::class);

$content = $whoops->run($e, $request);

return $response->withContent($content);

}

}

现在我们重启 http server，当出现异常时就可以看到由 whoops 渲染的错误信息页面了。

--

--

四、Apollo

https://swoft.org/documents/v2/ext-components/apollo/

//

Apollo（阿波罗）是携程框架部门研发的分布式配置中心，能够集中化管理应用不同环境、不同集群的配置，配置修改后能够实时推送到应用端，并且具备规范的权限、流程治理等特性，适用于微服务配置管理场景。Swoft 基于 Apollo 提供的 API，在之上进行封装，使之能在 Swoft 中快速使用。

--

1、安装

swoft/whoops 作为一个额外的扩展组件，需要手动安装。

(1)、Composer 安装

composer require swoft/apollo

--

2、配置bean

下面以拉取 Apollo 命令空间为 application 配置为例。

//

///1/ 首先在 app/bean.php 配置启用 Apollo

return [

// ...

'apollo' => [

'host' => '192.168.2.102',

'timeout' => 6

]

// ...

];

(1)、参数详解

host 地址；

port 端口号；

appId Apollo 应用ID；

clusterName Apollo 集群名称；

timeout 超时时间单位秒；

如果使用 apollo 监听更新回调函数，超时时间必须大于 60 秒。Apollo 配置完成后，像一个普通的 Bean 一样注入 Apollo Swoft\Apollo\Config 即可使用。

(2)、使用

<?php declare(strict\_types=1);

namespace App\Model\Logic;

use Swoft\Apollo\Config;

use Swoft\Apollo\Exception\ApolloException;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Bean\Annotation\Mapping\Inject;

/\*\*

\* Class ApolloLogic

\*

\* @since 2.0

\*

\* @Bean()

\*/

class ApolloLogic

{

/\*\*

\* @Inject()

\*

\* @var Config

\*/

private $config;

/\*\*

\* @throws ApolloException

\*/

public function pull(): void

{

$data = $this->config->pull('application');

// Print data

var\_dump($data);

}

}

?>

以上就是一个简单的 Apollo 配置拉取，swoft-apollo 除此方法外，还提供了更多的使用方法。使用 Demo 请查看 app/Console/Command/AgentCommand.php。

--

3、方法列表

(1)、pullWithCache

public function pullWithCache(string $namespace, string $clientIp = ''): array

从 Apollo 缓存中拉取配置，大概有1秒的延迟。$namespace 命名空间名称；$clientIp 客户端 IP，为空底层自动获取当前机器 IP，用于灰度发布。

(2)、pull

public function pull(string $namespace, string $releaseKey = '', string $clientIp = ''): array

实时拉取配置，没有延迟时间。

$namespace 命名空间名称；

$releaseKey 上一次拉取返回的版本号，更多描述，参考 Apollo 官方文档；

$clientIp 客户端IP，为空底层自动获取当前机器IP，用于灰度发布。

(3)、batchPull

public function batchPull(array $namespaces, string $clientIp = ''): array

此方法是 pull 方法的一个批量封装，用于同时拉取多个 namespace 配置，但是不支持 $releaseKey 参数。

$namespaces 命名空间名称集合数组；

$clientIp 客户端IP，为空底层自动获取当前机器IP，用于灰度发布；

(4)、listen

public function listen(array $namespaces, $callback, array $notifications = [], string $clientIp = ''): void

如果一直轮询拉取配置，会有很大一部分资源浪费，此方法就是解决该问题，当配置有更新的时候，会回调监听改变的函数。

$namespaces 命名空间名称集合数组；

$callback 配置改变监听函数，可以使闭包、对象方法都可以；

$notifications 监听信息，详细描述参考 Apollo 官放文档；

$clientIp 客户端IP，为空底层自动获取当前机器IP，用于灰度发布；

如果配置改变，监听函数回调的时候会传递一个参数，此参数信息记录了改变后的最新配置信息。

$callback = function(array $data){

// ...

}

--

--

五、Consul

https://swoft.org/documents/v2/ext-components/consul/

//

Consul可以用来实现分布式系统的服务发现与配置，它是HashiCorp公司推出的一款实用开源工具，支持Linux等平台。Consul是分布式的、高可用的、可横向扩展的。swoft-consul 组件，整合了 consul 功能，开发者可以直接通过该组件使用 consul 功能。

--

1、安装

composer require swoft/consul

--

2、使用

本章以操作 consul KV 为例，首先必须配置(启动) swoft-consul 组件。

--

3、配置

//// app/bean.php 文件中配置

return [

// ...

'consul' => [

'host' => '192.168.4.11'

]

// ...

];

host consul 地址IP；

port consul 端口号；

timeout 请求超时时间；

--

4、实例

<?php declare(strict\_types=1);

namespace App\Model\Logic;

use ReflectionException;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Bean\Annotation\Mapping\Inject;

use Swoft\Bean\Exception\ContainerException;

use Swoft\Consul\Agent;

use Swoft\Consul\Catalog;

use Swoft\Consul\Exception\ClientException;

use Swoft\Consul\Exception\ServerException;

use Swoft\Consul\Health;

use Swoft\Consul\KV;

use Swoft\Consul\Session;

/\*\*

\* Class ConsulLogic

\*

\* @since 2.0

\*

\* @Bean()

\*/

class ConsulLogic

{

/\*\*

\* @Inject()

\*

\* @var Agent

\*/

private $agent;

/\*\*

\* @Inject()

\*

\* @var Health

\*/

private $health;

/\*\*

\* @Inject()

\*

\* @var Catalog

\*/

private $catalog;

/\*\*

\* @Inject()

\*

\* @var KV

\*/

private $kv;

/\*\*

\* @Inject()

\*

\* @var Session

\*/

private $session;

/\*\*

\* @throws ReflectionException

\* @throws ContainerException

\* @throws ClientException

\* @throws ServerException

\*/

public function kv(): void

{

$value = 'value content';

$this->kv->put('/test/my/key', $value);

$response = $this->kv->get('/test/my/key');

var\_dump($response->getBody(), $response->getResult());

}

}

?>

consul 所有操作返回都是 Swoft\Consul\Response 对象，该对象提供如下方法。

getHeaders() 请求返回的 headers；

getBody() 请求返回原始 body 信息；

getStatusCode() 请求返回 http code；

getResult() 请求数据通过 json\_decode 格式化的数据；

可以直接注入 agent / health / catalog / kv / session 对象简单灵活，实际业务按需要注入相应对象操作，注意命名空间要引用正确。

--

5、方法列表

(1)、agent

checks()；

services()；

members(array $options = [])；

self()；

join(string $address, array $options = [])；

forceLeave(string $node)；

registerCheck(array $check)；

deregisterCheck(string $checkId)；

passCheck(string $checkId, array $options = [])；

warnCheck(string $checkId, array $options = [])；

failCheck(string $checkId, array $options = [])；

registerService(array $service)；

deregisterService(string $serviceId)；

(2)、health

node(string $node, array $options = [])；

checks(string $service, array $options = [])；

service(string $service, array $options = [])；

state(string $state, array $options = [])；

(3)、catalog

register(array $node)；

deregister(array $node)；

datacenters()；

nodes(array $options = [])；

node(string $node, array $options = [])；

services(array $options = [])；

service(string $service, array $options = [])；

(4)、kv

get(string $key, array $options = [])；

put(string $key, string $value, array $options = [])；

delete(string $key, array $options = [])；

(5)、session

create(array $body = null, array $options = [])；

destroy(string $sessionId, array $options = [])；

info(string $sessionId, array $options = [])；

node(string $node, array $options = [])；

all(array $options = [])；

renew(string $sessionId, array $options = [])；

--

--

六、Swoole Tracker

https://swoft.org/documents/v2/ext-components/swoole-tracker/

//

Swoole Tracker( https://www.swoole-cloud.com/tracker.html ) 作为 Swoole 官方出品的一整套企业级 PHP 和 Swoole 分析调试工具，更专一、更专业。

--

1、介绍

///1/ 时刻掌握应用架构模型

自动发现应用依赖拓扑结构和展示，时刻掌握应用的架构模型。

//

///2/ 分布式跨应用链路追踪

支持无侵入的分布式跨应用链路追踪，让每个请求一目了然，全面支持协程/非协程环境，数据实时可视化。

//

///3/ 全面分析报告服务状况

各种维度统计服务上报的调用信息， 比如总流量、平均耗时、超时率等，并全面分析报告服务状况。

//

///4/ 拥有强大的调试工具链

本系统支持远程调试，可在系统后台远程开启检测内存泄漏、阻塞检测和代码性能分析。

//

///5/ 完善的系统监控

支持完善的系统监控，零成本部署，监控机器的CPU、内存、网络、磁盘等资源，可以很方便的集成到现有报警系统。

//

///6/ 零成本接入系统

本系统的客户端提供脚本可一键部署，服务端可在Docker环境中运行，简单快捷。

--

2、安装

(1)、安装依赖

注册完账户后，进入控制台( https://www.swoole-cloud.com/dashboard/catdemo/ )，并申请试用，下载对应客户端。相关文档，请移步 试用文档( https://www.kancloud.cn/swoole-inc/ee-base-wiki/1214079 ) 或 详细文档( https://www.kancloud.cn/swoole-inc/ee-help-wiki/1213080 )。具体文档地址，以从控制台下载的对应客户端中展示的为准。

//

///1/ 将客户端中的所有文件以及以下两个文件复制到项目目录 swoole-tacker 中。

// entrypoint.sh

#!/usr/bin/env bash

/opt/swoole/script/php/swoole\_php /opt/swoole/node-agent/src/node.php &

php /var/www/swoft/bin/swoft http:start

//

///2/ 然后将下面的 Dockerfile 复制到项目根目录中。

FROM swoft/swoft

LABEL maintainer="sakuraovq <sakuraovq@gmail.com>" version="2.6"

# This php version

ENV PHP\_VERSION=72

# This is tracker ini config

ENV SWOOLE\_TRACHER\_INI=/usr/local/etc/php/conf.d/swoole-tracker.ini

ADD . /var/www/swoft

# The address here is displayed on the client side.

RUN cd ./swoole-tracker && ./deploy\_env.sh www.swoole-cloud.com \

&& chmod 777 entrypoint.sh \

&& php\_dir=$(php -r "echo @ini\_get("extension\_dir").PHP\_EOL;") \

&& cp ./swoole\_tracker${PHP\_VERSION}.so $php\_dir/swoole\_tracker.so \

# Enable swoole\_tracker

&& echo "extension=swoole\_tracker.so" > ${SWOOLE\_TRACHER\_INI} \

# Open the main switch

&& echo "apm.enable=1" >> ${SWOOLE\_TRACHER\_INI} \

# Sampling Rate, eg: 10%

&& echo "apm.sampling\_rate=10" >> ${SWOOLE\_TRACHER\_INI} \

# Turn on memory leak detection Default 0 Off

&& echo "apm.enable\_memcheck=1" >> ${SWOOLE\_TRACHER\_INI}

CMD ["sh", "./swoole-tracker/entrypoint.sh"]

//

///3/ 构建镜像

docker build -t swoft/tracker .

//

///4/ 运行容器

docker run --rm --name swoft-tracker -v $(pwd):/var/www/swoft -p 18306:18306 swoft/tracker

(2)、采样率

采样率按照需求配置, 如果为 100%, 全部请求都会进行上报。

--

3、使用

(1)、不依赖组件

Swoole Tracker的支持自动生成应用名称并创建应用，无需修改任何代码，生成的应用名称格式为：Swoole的HttpServer：ip:prot；其他的Server：ip(hostname):prot。

即安装好swoole\_tracker扩展之后就可以正常使用Swoole Tracker的功能。

(2)、依赖组件

当你需要自定义应用名称时则需要安装组件，使用Composer安装。

//

///1/ 安装组件

composer require swoft/swoole-tracker

//

安装完成后在 app/bean.php 配置文件中注册 Swoft\Swoole\Tracker\Middleware\SwooleTrackerMiddleware 中间件即可。

<?php

return [

'httpDispatcher' => [

// Add global http middleware

'middlewares' => [

\Swoft\Swoole\Tracker\Middleware\SwooleTrackerMiddleware::class,

],

],

];

?>

配置了中间件后框架会自动开启链路追踪。

//

///2/ 单独使用

你也可以在需要链路追踪透传TraceId/SpanId、自定义应用名称时单独使用它。

/\*\* @var SwooleTracker $swooleTracker \*/

$swooleTracker = bean(SwooleTracker::class);

/\*\*

\* 参数说明

\* 1. $func eg. 'App\Login\Weibo::login'

\* 2. $serviceName 自定义的应用名称，后台会自动创建 eg. 'user'

\* 3. $serverIp eg. '192.1.1.1'

\* 4. $traceId

\* 5. $spanId

\*/

$tick = $swooleTracker->startRpcAnalysis('/get/user', 'demo', '53.65.77.11', $traceId, $spanId);

//

// todo ...

$handlerStatus = false;

$errno = 401;

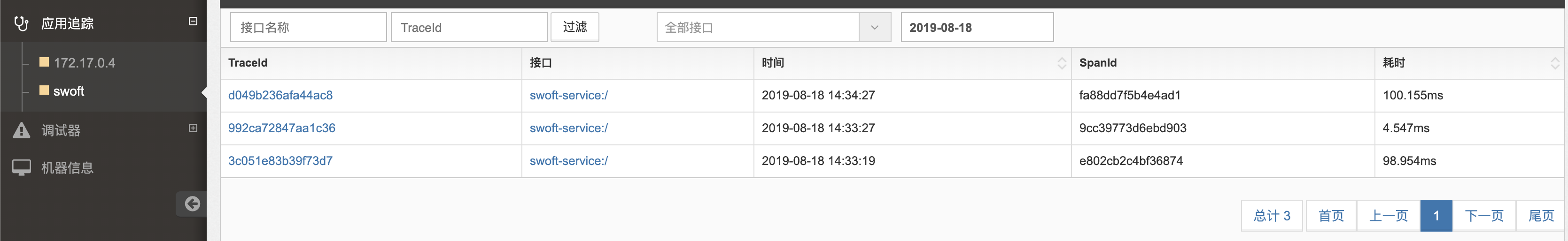
//

$swooleTracker->endRpcAnalysis($tick, $handlerStatus, $errno);

//

///3/ 登录管理后台看

链路追踪( https://swoft.org/img/trace.png )



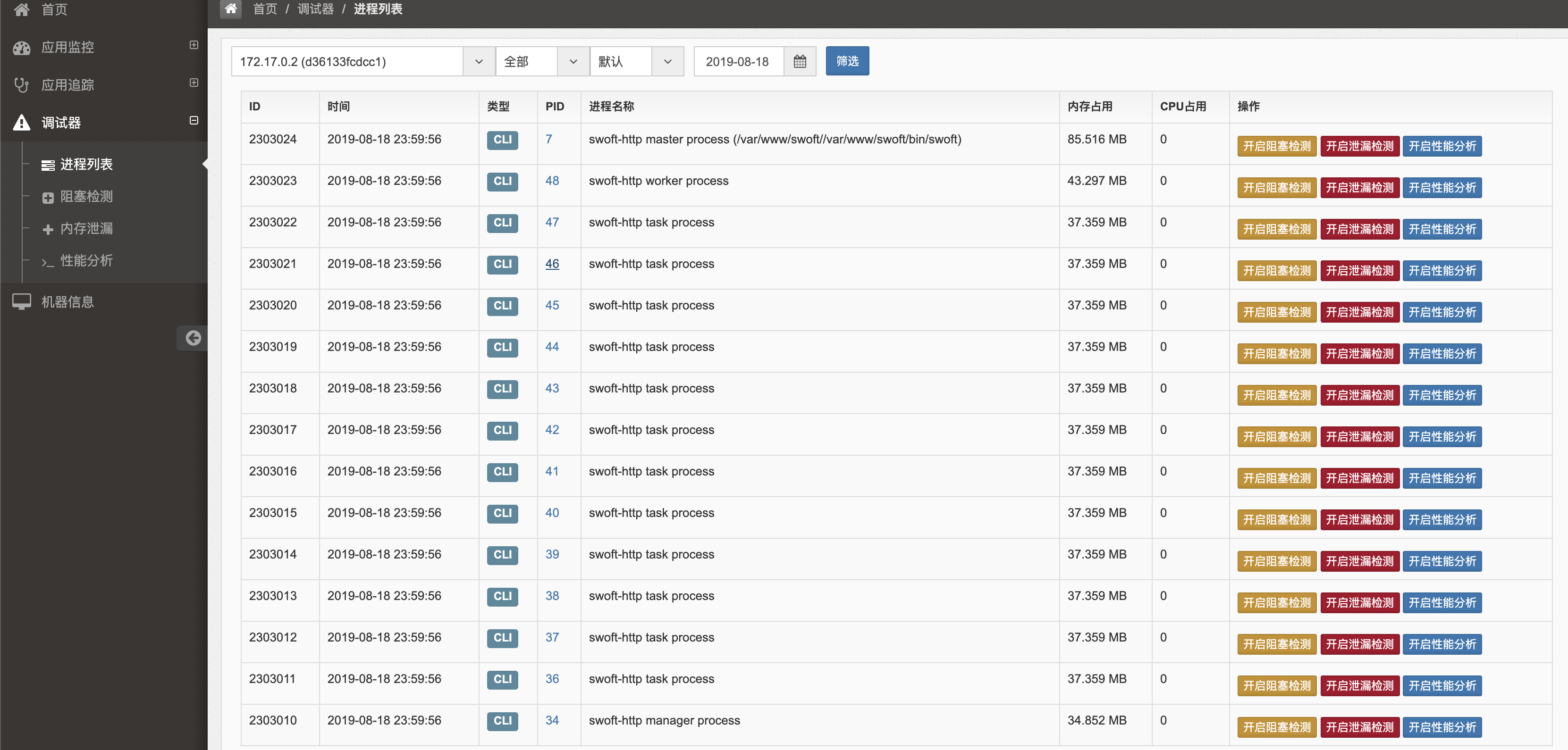
(3)、应用监控

( 图 https://swoft.org/img/monitor.png )



(4)、调试器

( 图 https://swoft.org/img/process.png )



(5)、内存泄露

( 图 https://swoft.org/img/memory.png )



(6)、阻塞检查

( 图 https://swoft.org/img/block.png )

如果需要开启 阻塞检查、泄露分析、性能分析 在 进程列表 中后面的按钮开启就是, 不需要客户端手动上报。

--

--

第十卷 微服务

--

--

一、服务注册与发现

https://swoft.org/documents/v2/microservice/register-find/

//

微服务治理过程中，经常会涉及注册启动的服务到第三方集群，比如 consul / etcd 等等，本章以 Swoft 框架中使用 swoft-consul 组件，实现服务注册与发现为例。

--

1、服务注册

无论是 http / rpc / ws 服务，启动的时候只需监听 SwooleEvent::START 事件，即可把启动的服务注册到第三方集群。

(1)、注册服务

本章这里以启动 http server 注册服务为例。

<?php declare(strict\_types=1);

namespace App\Listener;

use Swoft\Bean\Annotation\Mapping\Inject;

use Swoft\Consul\Agent;

use Swoft\Event\Annotation\Mapping\Listener;

use Swoft\Event\EventHandlerInterface;

use Swoft\Event\EventInterface;

use Swoft\Http\Server\HttpServer;

use Swoft\Log\Helper\CLog;

use Swoft\Server\Swoole\SwooleEvent;

/\*\*

\* Class RegisterServiceListener

\*

\* @since 2.0

\*

\* @Listener(event=SwooleEvent::START)

\*/

class RegisterServiceListener implements EventHandlerInterface

{

/\*\*

\* @Inject()

\*

\* @var Agent

\*/

private $agent;

/\*\*

\* @param EventInterface $event

\*/

public function handle(EventInterface $event): void

{

/\* @var HttpServer $httpServer \*/

$httpServer = $event->getTarget();

$service = [

'ID' => 'swoft',

'Name' => 'swoft',

'Tags' => [

'http'

],

'Address' => '127.0.0.1',

'Port' => $httpServer->getPort(),

'Meta' => [

'version' => '1.0'

],

'EnableTagOverride' => false,

'Weights' => [

'Passing' => 10,

'Warning' => 1

]

];

// Register

$this->agent->registerService($service);

CLog::info('Swoft http register service success by consul!');

}

}

?>

// 详细流程

$event->getTarget() 获取的对象是启动服务的对象，比如启动的 http 拿到的就是 Swoft\Http\Server\HttpServer 对象；

target 对象里面可以获取启动服务的所有信息包括配置信息；

根据服务配置信息和业务实际情况，注册服务信息到 consul；

必须使用 Swoole\Coroutine\Scheduler 否则无法注册服务，swoole 4.4+ 支持该功能。

(2)、取消服务

服务启动注册服务，服务关闭或者退出则需要取消服务注册，此时这里和注册一样监听一个 SwooleEvent::SHUTDOWN 事件即可，本章这里还是以 取消 Http server 服务为例。

<?php declare(strict\_types=1);

namespace App\Listener;

use Swoft\Bean\Annotation\Mapping\Inject;

use Swoft\Consul\Agent;

use Swoft\Event\Annotation\Mapping\Listener;

use Swoft\Event\EventHandlerInterface;

use Swoft\Event\EventInterface;

use Swoft\Http\Server\HttpServer;

use Swoft\Server\Swoole\SwooleEvent;

/\*\*

\* Class DeregisterServiceListener

\*

\* @since 2.0

\*

\* @Listener(SwooleEvent::SHUTDOWN)

\*/

class DeregisterServiceListener implements EventHandlerInterface

{

/\*\*

\* @Inject()

\*

\* @var Agent

\*/

private $agent;

/\*\*

\* @param EventInterface $event

\*/

public function handle(EventInterface $event): void

{

/\* @var HttpServer $httpServer \*/

$httpServer = $event->getTarget();

$this->agent->deregisterService('swoft');

}

}

?>

// 详细流程

$event->getTarget() 获取的对象是启动服务的对象，比如启动的 http 拿到的就是 Swoft\Http\Server\HttpServer 对象；

target 对象里面可以获取启动服务的所有信息包括配置信息；

根据服务配置信息和业务实际情况，取消已经支持的服务；

必须使用 Swoole\Coroutine\Scheduler 否则无法注册服务，swoole 4.4+ 支持该功能。

(3)、服务发现

本章这里以 Rpc client 为例，通过第三方集群 consul 下发可用的服务列表。

//

///1/ 首先定义服务提供者

<?php declare(strict\_types=1);

namespace App\Common;

use ReflectionException;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Bean\Annotation\Mapping\Inject;

use Swoft\Bean\Exception\ContainerException;

use Swoft\Consul\Agent;

use Swoft\Consul\Exception\ClientException;

use Swoft\Consul\Exception\ServerException;

use Swoft\Rpc\Client\Client;

use Swoft\Rpc\Client\Contract\ProviderInterface;

/\*\*

\* Class RpcProvider

\*

\* @since 2.0

\*

\* @Bean()

\*/

class RpcProvider implements ProviderInterface

{

/\*\*

\* @Inject()

\*

\* @var Agent

\*/

private $agent;

/\*\*

\* @param Client $client

\*

\* @return array

\* @throws ReflectionException

\* @throws ContainerException

\* @throws ClientException

\* @throws ServerException

\* @example

\* [

\* 'host:port',

\* 'host:port',

\* 'host:port',

\* ]

\*/

public function getList(Client $client): array

{

// Get health service from consul

$services = $this->agent->services();

$services = [

];

return $services;

}

}

?>

// 详细流程

实现 Swoft\Rpc\Client\Contract\ProviderInterface 接口；

根据 $client 参数获取当前 RPC 服务信息；

根据服务信息，到第三方集群(consul) 里面查询可用服务；

返回一个规定格式数组；

此类，必须用 @Bean 标记为一个 bean 对像；

db / redis 也是支持这种方式发现可用的服务，但是需要实现其相应的接口。

//

///2/ 现在我们有了服务提供者，接下来将其配置到对应的 RPC 服务上面。

// app/bean.php

return [

'user' => [

'class' => ServiceClient::class,

'host' => '127.0.0.1',

'port' => 18307,

'setting' => [

'timeout' => 0.5,

'connect\_timeout' => 1.0,

'write\_timeout' => 10.0,

'read\_timeout' => 0.5,

],

'packet' => bean('rpcClientPacket'),

'provider' => bean(RpcProvider::class)

],

'user.pool' => [

'class' => ServicePool::class,

'client' => bean('user')

],

];

这里在 user 服务上，通过 provider 参数注入了一个服务提供者 RpcProvider::class (bean 名称)。

--

--

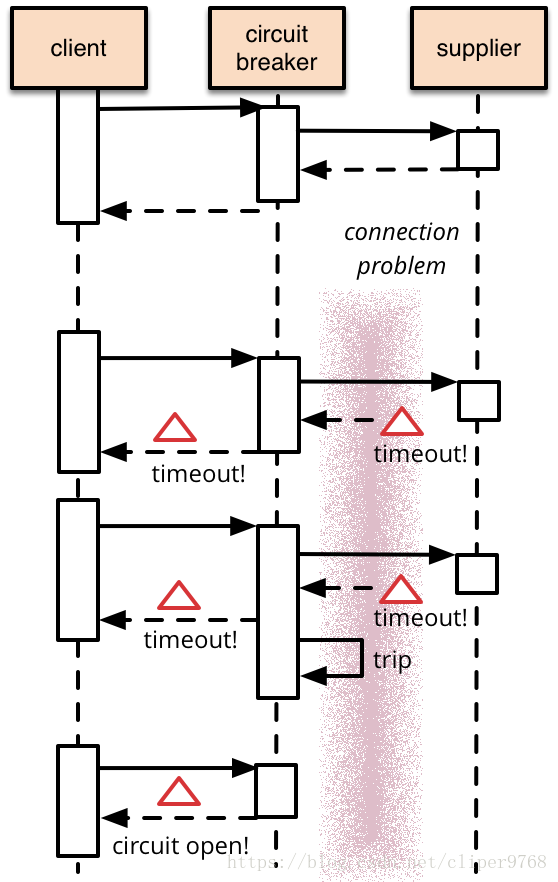
二、服务熔断与降级

https://swoft.org/documents/v2/microservice/blown-downgraded/

//

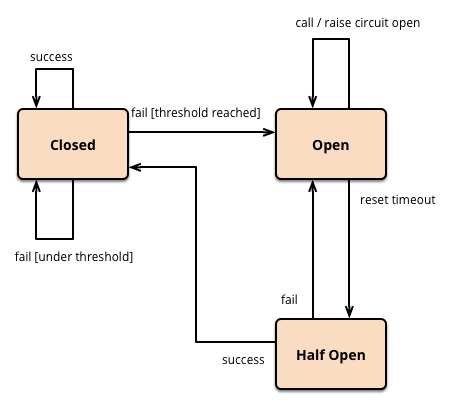
在分布式环境下，特别是微服务结构的分布式系统中，一个软件系统调用另外一个远程系统是非常普遍的。这种远程调用的被调用方可能是另外一个进程，或者是跨网路的另外一台主机, 这种远程的调用和进程的内部调用最大的区别是，远程调用可能会失败，或者挂起而没有任何回应，直到超时。更坏的情况是，如果有多个调用者对同一个挂起的服务进行调用，那么就很有可能的是一个服务的超时等待迅速蔓延到整个分布式系统，引起连锁反应，从而消耗掉整个分布式系统大量资源。最终可能导致系统瘫痪。断路器（Circuit Breaker）模式就是为了防止在分布式系统中出现这种瀑布似的连锁反应导致的灾难。

( 图 https://swoft.org/img/breaker\_base.png )



上图是断路器（Curcuit Breaker）的结构，它有两个基本状态（close和open）和一个基本trip动作：close 状态下， client 向 supplier 发起的服务请求，直接无阻碍通过断路器，supplier 的返回值直接由断路器交回给 client；open 状态下，client 向 supplier 发起的服务请求后，断路器不会将请求转到 supplier, 而是直接返回 client, client 和 supplier 之间的通路是断的；trip: 在 close 状态下，如果 supplier 持续超时报错，达到规定。

( 图 https://swoft.org/img/breaker\_ext.png )



上图是断路器的 扩展模式。

基本的断路器模式下，保证了断路器在 open 状态时，保护 supplier 不会被调用，但我们还需要额外的措施可以在 supplier 恢复服务后，可以重置断路器。一种可行的办法是断路器定期探测 supplier 的服务是否恢复，一但恢复，就将状态设置成 close。断路器进行重试时的状态为半开（half-open）状态。

Swoft 熔断器底层采用 扩展模式设计，开发者必须先熟悉 熔断器状态变化，这样有利于对熔断器的理解。

--

1、安装

使用熔断器前必须安装 swoft-breaker 组件，安装方式如下。

composer require swoft/breaker

--

2、配置

熔断器不使用配置也是可以使用的，但是有些业务场景需要全局配置熔断器，可以参考如下配置。

//// app/bean.php

return [

'breaker' => [

'timeout' => 3,

]

]

timeout 超时时间；

failThreshold 连续失败多少次状态切换阀门；

sucThreshold 连续成功多少次状态切换阀门；

retryTime 熔断器由开启状态到半开状态尝试切换时间；

全局配置会被 @Breaker 上面的局部注解覆盖。

--

3、使用

熔断器的使用相当简单且功能强大，使用一个 @Breaker 注解即可，Swoft 中的熔断是针对于类里面的方法熔断，只要方法里面没有抛出异常就说明是成功访问的，所以 @Breaker 注解可以在任何 bean 对象方法上面使用。

<?php declare(strict\_types=1);

namespace App\Model\Logic;

use Exception;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Breaker\Annotation\Mapping\Breaker;

/\*\*

\* Class BreakerLogic

\*

\* @since 2.0

\*

\* @Bean()

\*/

class BreakerLogic

{

/\*\*

\* @Breaker(fallback="funcFallback")

\*

\* @return string

\* @throws Exception

\*/

public function func(): string

{

// Do something

throw new Exception('Breaker exception');

}

/\*\*

\* @Breaker()

\*

\* @return string

\* @throws Exception

\*/

public function func2(): string

{

// Do something

return 'func2';

}

/\*\*

\* @return string

\*/

public function funcFallback(): string

{

return 'funcFallback';

}

/\*\*

\* @Breaker()

\*

\* @return string

\* @throws Exception

\*/

public function unFallback(): string

{

// Do something

throw new Exception('Breaker exception');

}

/\*\*

\* @Breaker(fallback="loopFallback")

\*

\* @return string

\* @throws Exception

\*/

public function loop(): string

{

// Do something

throw new Exception('Breaker exception');

}

/\*\*

\* @Breaker(fallback="loopFallback2")

\*

\* @return string

\* @throws Exception

\*/

public function loopFallback(): string

{

// Do something

throw new Exception('Breaker exception');

}

/\*\*

\* @Breaker(fallback="loopFallback3")

\*

\* @return string

\* @throws Exception

\*/

public function loopFallback2(): string

{

// Do something

throw new Exception('Breaker exception');

}

/\*\*

\* @return string

\*/

public function loopFallback3(): string

{

return 'loopFallback3';

}

}

?>

//

//// 这里示例定义了几种使用场景

函数正常没有一次抛出 (func2)；

函数抛出异常通过降级函数返回数据 (func)；

定义熔断器没有定义降级函数 (unFallback)；

循环执行熔断器 (loop)；

//

//// 循环执行熔断器这里详细讲解下流程

执行 loop 函数，抛出异常，执行 loopFallback 降级函数；

loopFallback 降级函数，抛出异常，但是定义了熔断器和降级函数，所以继续执行 loopFallback2 降级函数；

loopFallback3 降级函数，此时这里正常返回降级的数据；

fallback 函数必须和 @Breaker 标记的函数完全一样除了名称不一样且在同一个类里面。

--

4、注解

熔断器就用了一个 @Breaker 注解，该注解，必须定义在 bean 对象里面。

(1)、@Breaker

标记方法开启熔断器，如下参数详解。

fallback 降级函数，必须和 @Breaker 标记的函数完全一样除了名称不一样且在同一个类里面；

sucThreshold 连续成功多少次状态切换阀门；

failThreshold 连续失败多少次状态切换阀门；

timeout 超时时间；

retryTime 熔断器由开启状态到半开状态尝试切换时间；

注意：注解上面的配置会覆盖熔断器全局配置。

--

--

三、服务限流

https://swoft.org/documents/v2/microservice/limit/

//

限流是对稀缺资源访问时，比如秒杀、抢购的商品时，来限制并发和请求的数量，从而有效的进行削峰并使得流量曲线平滑。限流的目的是对并发访问和并发请求进行限速，或者一个时间窗口内请求进行限速从而来保护系统，一旦达到或超过限制速率就可以拒绝服务，或者进行排队等待等。

--

1、限流算法

(1)、计数器

采用计数器实现限流有点简单粗暴，一般我们会限制一秒钟的能够通过的请求数，比如限流 qps 为100，算法的实现思路就是从第一个请求进来开始计时，在接下去的1s内，每来一个请求，就把计数加1，如果累加的数字达到了100，那么后续的请求就会被全部拒绝。等到1s结束后，把计数恢复成0，重新开始计数。

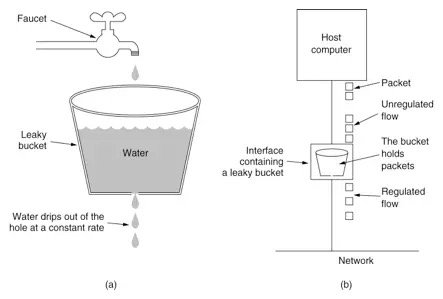
这种实现方式，相信大家都知道有一个弊端：如果我在单位时间1s内的前10ms，已经通过了100个请求，那后面的990ms，只能眼巴巴的把请求拒绝，我们把这种现象称为“突刺现象”。

(2)、漏桶

为了消除"突刺现象”，可以采用漏桶算法实现限流，漏桶算法这个名字就很形象，算法内部有一个容器，类似生活用到的漏斗，当请求进来时，相当于水倒入漏斗，然后从下端小口慢慢匀速地流出。不管上面流量多大，下面流出的速度始终保持不变。

不管服务调用方多么不稳定，通过漏桶算法进行限流，每10毫秒处理一次请求。因为处理的速度是固定的，请求进来的速度是未知的，可能突然进来很多请求，没来得及处理的请求就先放在桶里，既然是个桶，肯定是有容量上限，如果桶满了，那么新进来的请求就丢弃。

( 图 https://swoft.org/img/limiter\_leaky.png )



这种算法，在使用过后也存在弊端：无法应对短时间的突发流量。

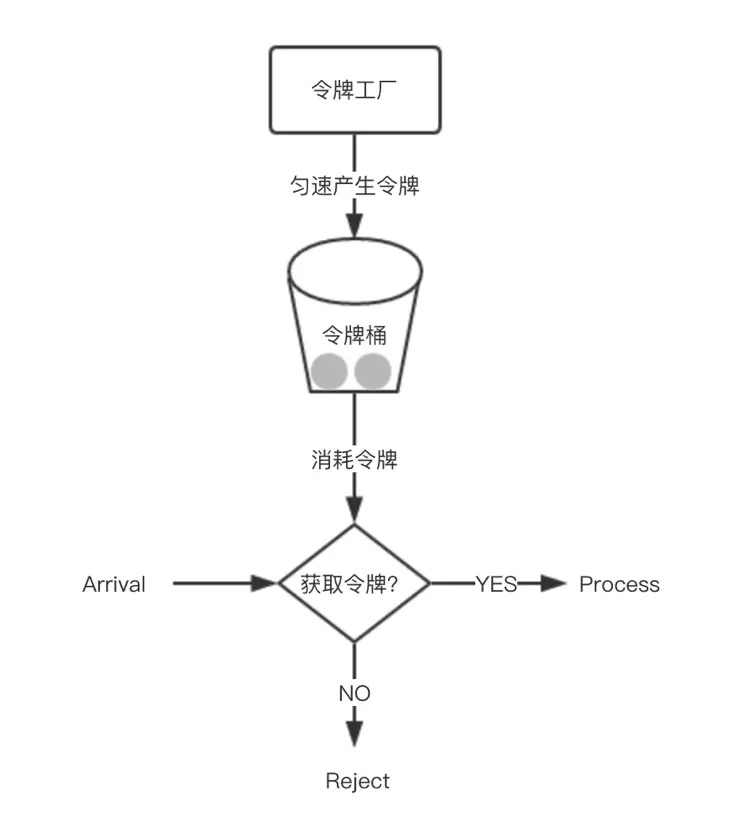
(3)、令牌桶

某种意义上讲，令牌桶算法是对漏桶算法的一种改进，漏桶算法能够限制请求调用的速率，而令牌桶算法能够在限制调用的平均速率的同时还允许一定程度的突发调用。

在令牌桶算法中，存在一个桶，用来存放固定数量的令牌。算法中存在一种机制，以一定的速率往桶中放令牌。每次请求调用需要先获取令牌，只有拿到令牌，才有机会继续执行，否则选择选择等待可用的令牌、或者直接拒绝。

放令牌这个动作是持续不断的进行，如果桶中令牌数达到上限，就丢弃令牌，所以就存在这种情况，桶中一直有大量的可用令牌，这时进来的请求就可以直接拿到令牌执行，比如设置 qps 为100，那么限流器初始化完成一秒后，桶中就已经有100个令牌了，这时服务还没完全启动好，等启动完成对外提供服务时，该限流器可以抵挡瞬时的100个请求。所以，只有桶中没有令牌时，请求才会进行等待，最后相当于以一定的速率执行。

( 图 https://swoft.org/img/limiter\_token.png )



Swoft 限流器底层采用的是令牌桶算法，底层依赖于 Redis 实现分布式限流。

(4)、令牌桶实现原理

//// 对于令牌桶中令牌的产生一般有两种做法

一种解法是，开启一个定时任务，由定时任务持续生成令牌。这样的问题在于会极大地消耗系统资源，如，某接口需要分别对每个用户做访问频率限制，假设系统中存在6W用户，则至多需要开启6W个定时任务来维持每个桶中的令牌数，这样的开销是巨大的。

第二种解法是延迟计算，定义一个 resync 函数。该函数会在每次获取令牌之前调用，其实现思路为，若当前时间晚于 nextFreeTicketMicros，则计算该段时间内可以生成多少令牌，将生成的令牌加入令牌桶中并更新数据。这样一来，只需要在获取令牌时计算一次即可。

//

Swoft 采用的是第二种，当每次获取令牌时，先执行 resync 来更新令牌桶中令牌的数量，从而达到异步产生令牌的目的。下面开始详解讲解，如何使用 swoft-limiter 组件，进行开发。

--

2、安装

// 如上已经详解讲解了限流器实现原理，使用限流器前，必须安装 swoft-limiter 组件，安装命令如下。

composer require swoft/limiter

--

3、配置

限流器不使用配置也是可以使用的，但是有些业务场景需要全局配置限流器，可以参考如下配置。

return [

'rateLimiter' => [

'class' => RateLimter::class,

'rateLimter' => bean('redisRateLimiter'),

]

];

//

//// 详细参数

name 限流器名称，默认 swoft:limiter；

rate 允许多大的请求访问，请求数/秒；

max 最大的请求数；

default 初始化请求数；

全局配置会被 @Limiter 注解的局部配置覆盖。

//

//// 缓存限速信息也是可以配置的，配置如下。

return [

'redisRateLimiter' => [

'pool' => 'redis.pool'

];

];

pool 指定使用的缓存连接池名称，默认使用框架默认的连接池。

使用限速器，一定要配置安装 redis 组件，且配置可用的 redis 缓存。

--

4、使用

(1)、控制器限速

//// 本节以常见的限速场景为例

<?php declare(strict\_types=1);

namespace App\Http\Controller;

use Swoft\Http\Message\Request;

use Swoft\Http\Server\Annotation\Mapping\Controller;

use Swoft\Http\Server\Annotation\Mapping\RequestMapping;

use Swoft\Limiter\Annotation\Mapping\RateLimiter;

/\*\*

\* Class LimiterController

\*

\* @since 2.0

\*

\* @Controller(prefix="limiter")

\*/

class LimiterController

{

/\*\*

\* @RequestMapping()

\* @RateLimiter(key="request.getUriPath()")

\*

\* @param Request $request

\*

\* @return array

\*/

public function requestLimiter(Request $request): array

{

$uri = $request->getUriPath();

return ['requestLimiter', $uri];

}

/\*\*

\* @RequestMapping()

\* @RateLimiter(rate=20, fallback="limiterFallback")

\*

\* @param Request $request

\*

\* @return array

\*/

public function requestLimiter2(Request $request): array

{

$uri = $request->getUriPath();

return ['requestLimiter2', $uri];

}

/\*\*

\* @RequestMapping()

\* @RateLimiter(key="request.getUriPath()~':'~request.query('id')")

\*

\* @param Request $request

\*

\* @return array

\*/

public function paramLimiter(Request $request): array

{

$id = $request->query('id');

return ['paramLimiter', $id];

}

/\*\*

\* @param Request $request

\*

\* @return array

\*/

public function limiterFallback(Request $request): array

{

$uri = $request->getUriPath();

return ['limiterFallback', $uri];

}

}

?>

//

requestLimiter 方法根据 URI 请求地址限流；

requestLimiter2 方法根据类名+方法名称限流，其实和 requestLimiter限流的功能是一样的， 但是定义了一个降级函数；

paramLimiter 方法根据URI地址+参数限流；

//

key 这里支持 symfony/expression-language 表达式，可以实现很多复杂的功能，详细文档( http://www.symfonychina.com/doc/current/components/expression\_language/syntax.html )，如果被限速会抛出 Swoft\Limiter\Exception\RateLImiterException 异常。key 表达式内置 CLASS(类名) 和 METHOD(方法名称) 两个变量，方便开发者使用。

(2)、方法限速

Swoft 限速器不仅可以限流控制器，也可以限制任何 bean 里面的方法，可以控制方法的访问速率。这里以下面使用示例详解。

<?php declare(strict\_types=1);

namespace App\Model\Logic;

use Swoft\Bean\Annotation\Mapping\Bean;

use Swoft\Limiter\Annotation\Mapping\RateLimiter;

/\*\*

\* Class LimiterLogic

\*

\* @since 2.0

\*

\* @Bean()

\*/

class LimiterLogic

{

/\*\*

\* @RateLimiter(fallback="limterFallback")

\*

\* @return array

\*/

public function limter(): array

{

// Do something

return [];

}

/\*\*

\* @RateLimiter(key="requestBean.getName('name')")

\*

\* @param RequestBean $requestBean

\*

\* @return array

\*/

public function limterParams(RequestBean $requestBean): array

{

// Do something

return [];

}

/\*\*

\* @return array

\*/

public function limterFallback(): array

{

return [];

}

}

?>

limtier 方法，根据类名+方法名称限速且定义了一个降级函数；

limtierParams 方法，根据调用 RequestBean 对象的方法，返回值作为限流 key；

//

fallback 指定的函数必须和 limiter 一样，包括名称和参数、返回值，并且必须在同一个类里面。如果被限速会抛出 Swoft\Limiter\Exception\RateLImiterException 异常。key 表达式内置 CLASS(类名) 和 METHOD(方法名称) 两个变量，方便开发者使用。

--

5、注解

限流器注解很简单，只涉及一个 @RateLimiter 注解。

(1)、@RateLimiter

标记方法，开启限流策略，详细参数如下。

name 缓存前缀；

rate 允许多大的请求访问，请求数/秒；

key 限流 key, 支持 symfony/expression-language 表达式，key 表达式内置 CLASS(类名) 和 METHOD(方法名称) 两个变量，方便开发者使用；

max 最大的请求数；

default 初始化请求数；

fallback 降级函数，和 breaker 一样；

--

--

四、配置中心

https://swoft.org/documents/v2/microservice/config-center/

//

配置中心主要就是把配置集中化管理，方便统一和维护。本章以 Apollo 为例，从远端配置中心拉取配置以及安全重启服务。如果对 Apollo 不熟悉，可以先看 Swoft 扩展 Apollo 组件以及阅读 Apollo 官方文档。

//

//// 配置中心使用流程

编写本地 agent 监听配置的变化，如果有变更，修改本地配置文件；

重启业务对应的服务；

提示：本地 agent 一定要比服务先启动，否则服务启动，没法获取最新的配置信息。

--

1、使用

本章以 Swoft 中使用 apollo 为例，当 apollo 配置变更后，重启服务(http-server / rpc-server/ ws-server)。如下是一个 agent 例子。

(1)、声明 agent

<?php declare(strict\_types=1);

namespace App\Console\Command;

use ReflectionException;

use Swoft\Apollo\Config;

use Swoft\Apollo\Exception\ApolloException;

use Swoft\Bean\Annotation\Mapping\Inject;

use Swoft\Bean\Exception\ContainerException;

use Swoft\Co;

use Swoft\Console\Annotation\Mapping\Command;

use Swoft\Console\Annotation\Mapping\CommandMapping;

use Swoft\Http\Server\HttpServer;

use Swoft\Log\Helper\CLog;

use Swoft\Rpc\Server\ServiceServer;

use Swoft\WebSocket\Server\WebSocketServer;

use Throwable;

/\*\*

\* Class AgentCommand

\*

\* @since 2.0

\*

\* @Command("agent")

\*/

class AgentCommand

{

/\*\*

\* @Inject()

\*

\* @var Config

\*/

private $config;

/\*\*

\* @CommandMapping(name="index")

\*/

public function index(): void

{

$namespaces = [

'application'

];

while (true) {

try {

$this->config->listen($namespaces, [$this, 'updateConfigFile']);

} catch (Throwable $e) {

CLog::error('Config agent fail(%s %s %d)!', $e->getMessage(), $e->getFile(), $e->getLine());

}

}

}

/\*\*

\* @param array $data

\*

\* @throws ContainerException

\* @throws ReflectionException

\*/

public function updateConfigFile(array $data): void

{

foreach ($data as $namespace => $namespaceData) {

$configFile = sprintf('@config/%s.php', $namespace);

$configKVs = $namespaceData['configurations'] ?? '';

$content = '<?php return ' . var\_export($configKVs, true) . ';';

Co::writeFile(alias($configFile), $content, FILE\_NO\_DEFAULT\_CONTEXT);

CLog::info('Apollo update success！');

/\*\* @var HttpServer $server \*/

$server = bean('httpServer');

$server->restart();

// /\*\* @var ServiceServer $server \*/

// $server = bean('rpcServer');

// $server->restart();

// /\* @var WebSocketServer $server \*/

// $server = bean('wsServer');

// $server->restart();

}

}

}

?>

这里声明了一个本地 agent，它监听 apollo 远程配置变更，如果有变更，回调函数，同时返回最新的配置信息，根据业务实际情况重新修改配置文件，再重启相应的服务。

(2)、启动 agent

php bin/swoft agent:index

agent 启动，可以后台守护进程启动，防止挂掉。

(3)、启动服务

// 以启动 Http server 为例

/usr/local/php/bin/php /data/www/swoft/bin/swoft http:start -d

//

启动命令的路径，必须是绝对路径 (/data/www/swoft/bin/swoft) 且后台运行模式 (-d)，服务的启动必须在 agent 之后，否则无法获取最新配置信息。本章只是一个简单的使用例子，开发者可以根据自己的实际业务情况，监听 apollo 配置变更，生成配置文件，重启服务。

--

--

第十一卷 贡献

--

--

一、提交问题

https://swoft.org/documents/v2/contribute/sub-questions/

1、概念介绍

2、如何使用

3、使用示例

--

--

二、提交代码

https://swoft.org/documents/v2/contribute/sub-codes/

//

开发组非常欢迎各位向我们提交PR(Pull Request)，但是为了保证代码质量和统一的风格，向官方的主仓库 swoft/swoft( https://github.com/swoft-cloud/swoft ) 和 开发仓库 贡献代码时需要注意代码和commit格式。

--

1、发起PR时的注意事项

请不要提交 PR 到各个组件仓库，它们都是 只读的；

核心组件的 开发仓库 是 swoft/swoft-component( https://github.com/swoft-cloud/swoft-component )；

扩展组件的 开发仓库 是 swoft/swoft-ext( https://github.com/swoft-cloud/swoft-ext )；

请 fork 对应的 开发仓库，修改后，请把你的PR提交到对应的开发仓库；

发布版本时官方会将代码同步到各个子仓库。

--

2、Commit Message

commit message 只能是英文信息；

请尽量保证 commit message 是有意义的说明；

最好以 add: update: fix: 等关键字开头；

--

3、代码风格

提交的 PHP 代码 必须 遵循 PSR-2 代码风格；

合理且有意义的类、方法、变量命名；

适当的注释，合理的使用空行保持代码的简洁，易于阅读；

不要包含一些无意义的信息，例如 @author 等(提交者是能够从 commit log 里看到的)；

--

--

三、开发组件

https://swoft.org/documents/v2/contribute/develop/

1、概念介绍

2、如何使用

3、使用示例

--

--

第十二卷 附录

--

--

一、使用案例 (无)

https://swoft.org/documents/v2/appendix/case/

--

--

二、开源项目 (无)

https://swoft.org/documents/v2/appendix/open-source/

1、概念介绍

2、如何使用

3、使用示例

--

--

三、认识团队 (无)

https://swoft.org/documents/v2/appendix/team/

1、概念介绍

2、如何使用

3、使用示例

--

--

四、编写规范

https://swoft.org/documents/v2/appendix/doc-standard/

//

这里我将描述文档编写规范格式，将从以下几个方面进行详细的介绍。标题、文本、段落、数字、标点符号、文档体系。

--

1、标题

定义文档中标题的使用规范。

(1)、层级

一级标题：文章的主标题；

二级标题：文章的主干章节标题；

三级标题：章节下的分支标题；

四级标题：属三级标题内的分支标题；

//

//// 示例

# 一级标题

## 二级标题

### 三级标题

#### 四级别标题

(2)、原则

|1>、一级标题，全文仅能有一个一级标题，在本站中每篇文章的一级标题不能改手动设置，一级标题的值为正文头部的YAML 或 TOML 格式的数据元中 title 字段的值，标题使用不能跨级。

示例：标题跨级，缺少二级标题

# 一级标题

### 三级标题

|2>、二级标题，为文档的主干章节的标题，二级标题会出现在文档右侧的章节导航中，因此每个二级标题应当是一个完整的内容单元，同时在使用二级标题时应当避免出现孤立标题。

示例：下方的文章结构中，二级标题 A 中只有包含一个三级标题，不推荐此写法，完全可以省略三级标题。

## 二级标题 A

### 三级标题

## 二级标题 B

|3>、下级标题不能与上级标题重复。

示例：下方结构中，三级标题与二级标题同名，应当避免。

## 概述

### 概述

|4>、谨慎使用4级标题，应尽量避免出现，保持层级简洁，若三级标题下有并列性的内容，建议使用项目列表。

示例：下方有两个文章结构，结构二比结构一好，后者更适合内容较长的篇幅。

结构一

### 三级标题

#### 四级标题A

#### 四级标题B

#### 四级标题C

结构二

### 三级标题

\*\*（1）A\*\*

\*\*（2）B\*\*

\*\*（3）C\*\*

--

2、文本

(1)、字间距

全角中文字符与半角英文字符之间，应当有一个半角空格。

错误：本手册将指导您如何使用Swoft框架。

正确：本手册将指导您如何使用 Swoft 框架。

//

全角中文字符与半角阿拉伯数字之间，有没有加空格都可以，但是要保证风格统一不能两种风格混杂。英文单位若不翻译，单位与前方的数字不留空格。

错误：这是一部 4 GB 运存的手机。

正确：这是一部 4GB 运存的手机。

半角字符与全角标点符号之间不留空格。

(2)、写作风格

|1>、尽量使用主动语态，避免被动语态。

错误：假如此软件尚未被安装。

正确：假如尚未安装这个软件。

|2>、不使用非正式语言风格

错误：这个框架简直爽翻天。

正确：使用这个框架使人感到愉悦。

|3>、不要使用生僻字、文言文的词语

错误：这是唯二的方法。

正确：这是仅有的两种方法。

|4>、请用对 的、地、得

形容词+的+名词

副词+地+动词

动词+得+副词

|5>、使用代词（如其、该、此、这等词）时必须明确代词的指向，避免歧义。

|6>、名词前不要使用过多的形容词。

|7>、不包含任何标点符号的句子，或以逗号分隔的句子，长度尽量保持在20字以内，30～39个字的句子语义必须明确，任何情况下都不接受多于40个字的句子。

|8>、只有一种意思的句子尽量使用肯定句表达，不使用否定句。

错误：请确保没有开启省电模式。

正确：请确保省电模式已关闭。

|9>、避免使用双重否定句

错误：没有删除权限的用户，不能删除此文件。

正确：用户必须拥有删除权限，才能删除此文件。

(3)、英文处理

英文原文如果使用了复数形式，翻译时应将其还原成单数形式。

英文：information stored in random access memory (RAMs)

中文：存储在随机存储器（RAM）里的信息

//

英文缩写可以使用半角圆点.表示缩写，例如 U.S.A、Apple,Inc.

英文书名或电影名该用中文表达时，双引号应改为书名号。

第一次出现的英文词汇，应在括号中给出中文标注，后续再次出现时，直接使用缩写。

--

3、段落

(1)、段落原则

一个段落只能有一个主题，或一个中心句子。

段落的中心句子方段首，对全段内容做概述，后面的陈述句为核心句服务。

段落长度避免过长，最佳长度应小于等于四行，一定不能超过七行。

段落的句子预期要使用陈述和肯定预期，避免使用感叹语气。

段落开头不要流出空白字符。

书写时每个段落应该用空行隔开。

(2)、引用

// 引用第三方内容时，应注明出处。

引用内容。–引用出处

//

// 如果是全篇转载，请在全文开头显著位置注明作者和出处，若原文有链接地址，应给出原文链接。

本文转自 百度( https://www.baidu.com/ )

--

4、数值

(1)、半角数字

所有数字一律使用英文半角形式，不得使用中文全角形式。

(2)、千分号

数值为千位以上时，应添加千分号“,”。

示例：实际到账金额为 RMB1,000,000。

对于4～6位的数值，千分号可用可不用，比如1000和1,000都可以接受。而对于6位以上的数值，则必须带千分号。

对于多位的小数点则应按照小数点后开始从左向右添加千分号例如5.235,345。

(3)、货币

货币应为阿拉伯数字，并且在数字前方写出货币符号或者在货币后方给出中文货币名。

¥998

$230

456美元

(4)、数值范围

表示数值范围时，用~连接，对于带有单位或百分号时，~符号两边的数值都要加上单位或百分号，不能只加一个。

错误：100～130KG

正确：100KG~130KG

(5)、变化程度

数字的增加使用“增加了”、“增加到”。“了”表示增量，“到”表示定量。

内存增加到之前的两倍（指的是过去为一，现在为二）。

内存比之前增加了两倍（指的是过去为一，现在为三）。

数字的减少与增加规则同理，应使用“降低了”、“降低到”。值得注意的是不能使用“降低N倍“或”减少N倍“的表示法，要用”降低百分之几“或”减少百分之几“，因为减少或降低一倍表示的意思是数值原来为一百，现在等于零。

--

5、标点符号

(1)、标点符号原则

中文语句的标点符号应全都采用全角符号，保证视觉统一。

如果证据为英文，则使用英文半角符号。

句号、问号、叹号、逗号、顿号、分号和冒号都不得出现在行首。

(2)、句号

中文语句中的结尾处应该使用全角句号“。”。

句子末尾有括号时，句号应加在括号之外。

(3)、逗号

逗号，表示句子内部的停顿。

注意避免出现“一逗到底”的情况，即整个段落除了结尾句号，全部使用逗号停顿。

(4)、顿号

中文句子内部的并列词语之间使用全角顿号分割“、”，即使并列词是英文也是如此。

英文句子中的并词之间应使用半角逗号 “,” 分隔。

(5)、分号

分号；表示复句内并列分句之间的停顿。

(6)、引号

引用时，应该使用全角引号“”，注意前后引号不同。

引号内部还需使用引号时，外层使用双引号，内层使用单引号‘’，注意都是全角的，前后单引号不同。

(7)、圆括号

补充性说明的内容，应使用圆括号（）括起来，括号前后不加空格。

(8)、冒号

全角冒号：常用在需要解释的词语后面，引出解释说明。

表示时间时，应使用半角冒号 :。

(9)、省略号

省略号“……”表示语句未完成、或语气不连续。它占用两个汉子空间，包含六个省略点，不要使用“。。。”或 “...” 等非标准形式。同时省略号不应该与“等”这个词一起使用。

(10)、感叹号

编写文档时应当使用平静的语气叙述，尽量避免使用感叹号“！”，不得多个感叹号连用，例如“！！！”。

(11)、破折号

破折号“————”一般用于进一步解释说明。

破折号应占用两个汉字位。如果破折号本身只占用一个汉字位置时，前后应该空出一个半角空格。

(12)、连接号

连接号-主要用于两个词语的连接。以下场景中连接号只占用一个半角字符位，两个名词的连接；图表的编号。

以下场景中连接号应该使用波浪号～，占一个全角字符位，数值范围。

注意，波浪号前后两个值都应加上单位。波浪号也可以用汉字“至”代替。

--

6、文档体系

(1)、文档文件名

文档的文件名不得有空格，不能使用全角字符。因此不能使用中文做文件名。

文件名只建议使用小写字母，不建议有大写。

文件名包含多个单词时，单词之间使用连词线-连接。

(2)、内容深度

此文档的核心作用是作为操作手册来使用，不需要过深的介绍概念原理。

主要写作方向应当是以框架本身为核心，介绍框架如何使用，应当避免涉及过多的框架以外的内容，保持文档内容简洁不混乱。

其他的知识点内容应当以引用的方式向读者说明。关于更深层次的文档，应当分离出来独立进行编写。

举例子来说，操作手册就等同于一台设备的说明书，主要是介绍设备如何操作、运行、维护等内容，而不会涉及到介绍这台设备是如何制造的，用了什么材料，如何加工的，使用的什么电路板和芯片。

//

//// 文档片段示例

在使用本框架时，我们会假设您已经对 Swoole，有了一定的基础，我们不会解释关于 Swoole 的部分。

(3)、内容广度

在每一个篇章中，应当是一个独立的知识点介绍，不应当重复描述其他章节的内容，如有必要可给出所需章节的链接指向。

每个篇章中，应当尽可能详细的对本篇文章内容做解释说明，对于文档中所提到的相关操作必须解释清楚。

我们的目标是编写一份通俗易懂，且能详细的展现框架的各项功能使用方式、注意事项等内容。

//

//// 通常情况下我们每篇文章的内容都应当符合以下核心

解释一个功能具体的使用方式；

把功能描述清楚，而不是假设大家对这个功能很熟悉；

示例应尽可能的简单易懂；

链接到其它文档，而不是在这里重新解释概念；

不要解释过程，是只告诉你最终结果；

介绍此功能的使用场景，包括不适用的场景；

(4)、内容结构示例

## 基本功能介绍

1. 用清晰简短的语句描述功能作用

2. 描述清楚使用此功能的相关注意事项

## 功能的相关配置选项

与本功能相关的所有操作选项等尽可能完整的介绍清楚

## 使用场景

1. 介绍适合或不适合使用本功能的场景

2. 若本功能的不适用场景有替代方案时应当介绍

## 简单的示例

提供使用本功能的简单示例

## 额外的上下文

可以写一些额外的补充性说明内容

具体示例请查看 文档示例( https://swoft.org/documents/v2/appendix/doc-sample )。

--

7、文档编写指导说明

(1)、目录结构

网站的所有文档内容都在 content/documents/ 目录下，其中 v1 对应的是文档1.0，v2 对应的是文档2.0，每个文档内的 \_index.md 就是文档的默认首页。

文档2.0的一级目录大纲在 config/\_default/menus.toml 文件内以 [[v2]] 标记标注的，name 是一级目录的标题；weight 是一级目录的排序数值，数值越大越靠后，若一级目录是单独可点击的没有子目录则使用 url 字段，若有子目录则使用 identifier 字段标记一级目录的 ID，此字段不可重复。

# v2文档一级菜单

[[v2]]

name = "框架概述" #目录的名字

weight = 1 #目录的排序，序号越小越靠前

url = "/documents/v2/" #无子目录时，点击直接跳转的地址

[[v2]]

name = "开发基础" #目录名字

weight = 10 #目录的排序

identifier = "dev-basis" #本级目录的 ID 标志，全局不可重复。

文档的二级菜单是通过具体的文章内的标记元确定的。

注意 子目录的存放位置与一级导航的 ID 标记无任何关联，但是为了方便日后维护，因此将以一级目录的 ID 值为文件夹，在文件夹内存放子目录的内容。

(2)、文档标记元

每篇文档的头部都会有一些标记元，可以是 YAML 格式或 TOML 格式的配置项，下面会对配置项做一个解释说明。下面会以 TOML 格式为准解释相关标记元素。

+++ #TOML标记符号，标记符号必须在第一行，前面不能有空格、空行。

title = "使用案例" #一级标题，若没有使用 linktitle 字段则目录上的名字和此字段名一致。

linktitle = "案例" #目录标题

toc = true #是否显示文档右侧的小目录

type = "docs" #不许更改

date = "2018-09-19" #文档创建日期

lastmod = "2018-09-20" #最后修改日期

[menu.v2] #目录标记

parent = "dev-basis" #所属上级目录，值为一级目录的 identifier 字段

weight = 1 #一级目录下的排序

weight = 1 #文档的导航器，此顺序建议和 menu 下的 weight 顺序保持一致，此选项将影响页面上的上一页和下一页的导航。

+++

特别说明，若是在文档中的 \_index.md 进行设置，则以下字段有特别的含义。

+++

linktitle = "v2文档" #字段值将会在文档列表中显示为文档的标题

summary = "文档描述" #字段值是文档的描述字段，将会显示在文档列表中

weight = 1 #如果有多份文档，则此字段影响本文档的排序顺序

+++

关于更多的标记元素请查阅 hugo文档( https://gohugo.io/ ) 和 academic模板( https://sourcethemes.com/academic/ ) 文档。

--

--

五、文档示例

https://swoft.org/documents/v2/appendix/doc-sample/

//

组件功能介绍和使用场景。

--

1、准备

了解组件，需要额外的知识。

(1)、知识点1

(2)、知识点2

--

2、安装

通过 Composer 安装

composer require swoft/xxx

--

3、注解

@XxxAnnotation

注解功能描述；

arg 注解功能描述；

arg2 注解功能描述；

--

4、用法

以不同的使用场景，概括用法点，所有介绍切入点，都为使用场景。

(1)、实体

重要函数介绍格式;

// 全局函数格式

xxxx(int $a, int $b) : array

//

// 静态函数格式，且必须加上完整类名

Xxx\Xxx::xx(int $a, int $b) : object

//

// 对象方法格式

public function xxx(string $c) : int

函数功能介绍；

$a 参数介绍；

$b 参数介绍；

返回值介绍；

重要说明。

//

//// 函数xx用法

......

//

//// 函数xx用法

......

//

//// 普通用法格式

$headers = $request->getHeaders();

foreach ($headers as $name => $values) {

echo $name . ": " . implode(", ", $values).PHP\_EOL;

}

重要说明。

//

//// 命令介绍格式

命令功能介绍，贴出帮助信息。

$ php bin/swoft http

Provide some commands to manage the swoft HTTP Server

Group: http (alias: httpserver,httpServer,http-server)

Usage:

bin/swoft http:COMMAND [--opt ...] [arg ...]

Global Options:

--debug Setting the application runtime debug level(0 - 4)

--no-color Disable color/ANSI for message output

-h, --help Display this help message

-V, --version Show application version information

Commands:

reload Reload worker processes

restart Restart the http server

start Start the http server

stop Stop the currently running server

Example:

bin/swoft http:start Start the http server

bin/swoft http:stop Stop the http server

//

选项列表：

--debug 选项描述

-V 选项描述

//

命令列表：

reload 命令描述

restart 命令描述

//

命令xx用法

php bin/swoft http:start

//

命令xx用法

php bin/swoft http:stop

重要说明。

//

//// Bean配置格式

描述配置功能

return [

// ...

'apollo' => [

'host' => '192.168.2.102',

'timeout' => 6

]

// ...

];

参数列表：

host 参数功能描述

port 参数功能描述

重要说明。

//

//// 表结构格式

表结构用于一些一系列的列表集合。

名称/描述

ServerEvent::BEFORE\_SETTING setting() 方法之前

ServerEvent::BEFORE\_SETTING setting() 方法之前

ServerEvent::BEFORE\_SETTING setting() 方法之前

ServerEvent::BEFORE\_SETTING setting() 方法之前

--

--

--