极客时间

构建微服务云原生应用

架构师杨波

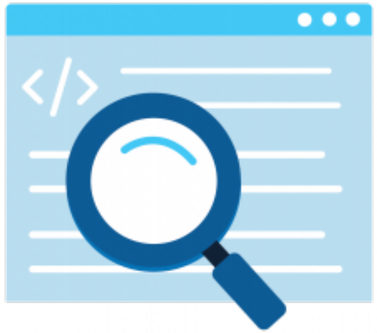
极客时间

扫码试看/订阅

《Spring Boot & Kubernetes 云原生微服务实践》

CHAPTER 03

服务开发框架设计和实践

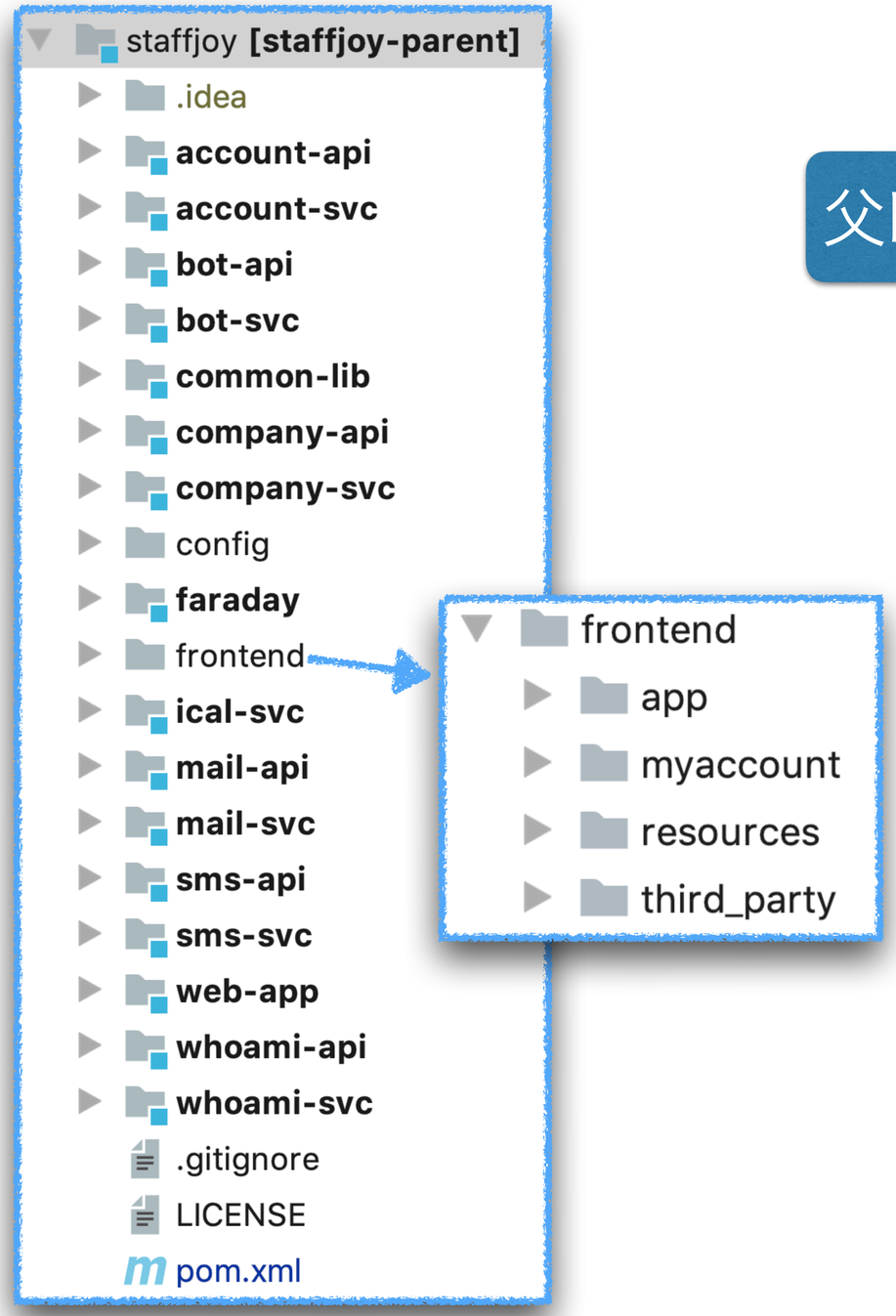


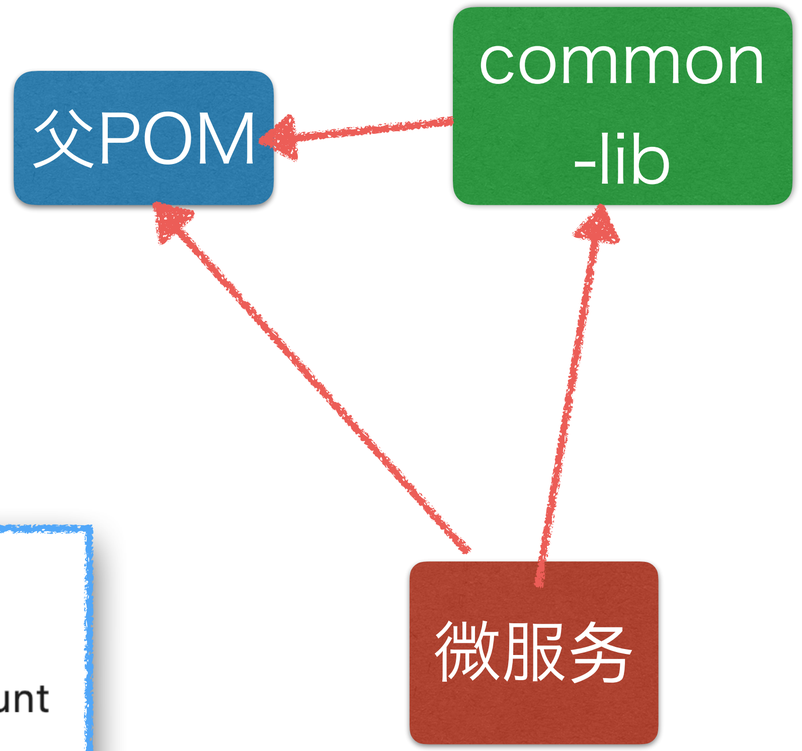
第 1 部分

Staffjoy 项目代码组织

项目代码组织

```
<modules>
    <module>common-lib</module>
    <module>account-svc</module>
    <module>account-api</module>
    <module>company-api</module>
    <module>company-svc</module>
    <module>mail-api</module>
    <module>mail-svc</module>
    <module>sms-svc</module>
    <module>sms-api</module>
    <module>bot-api</module>
    <module>bot-svc</module>
    <module>ical-svc</module>
    <module>whoami-api</module>
    <module>whoami-svc</module>
    <module>web-app</module>
    <module>faraday</module>
</modules>
```





依赖管理

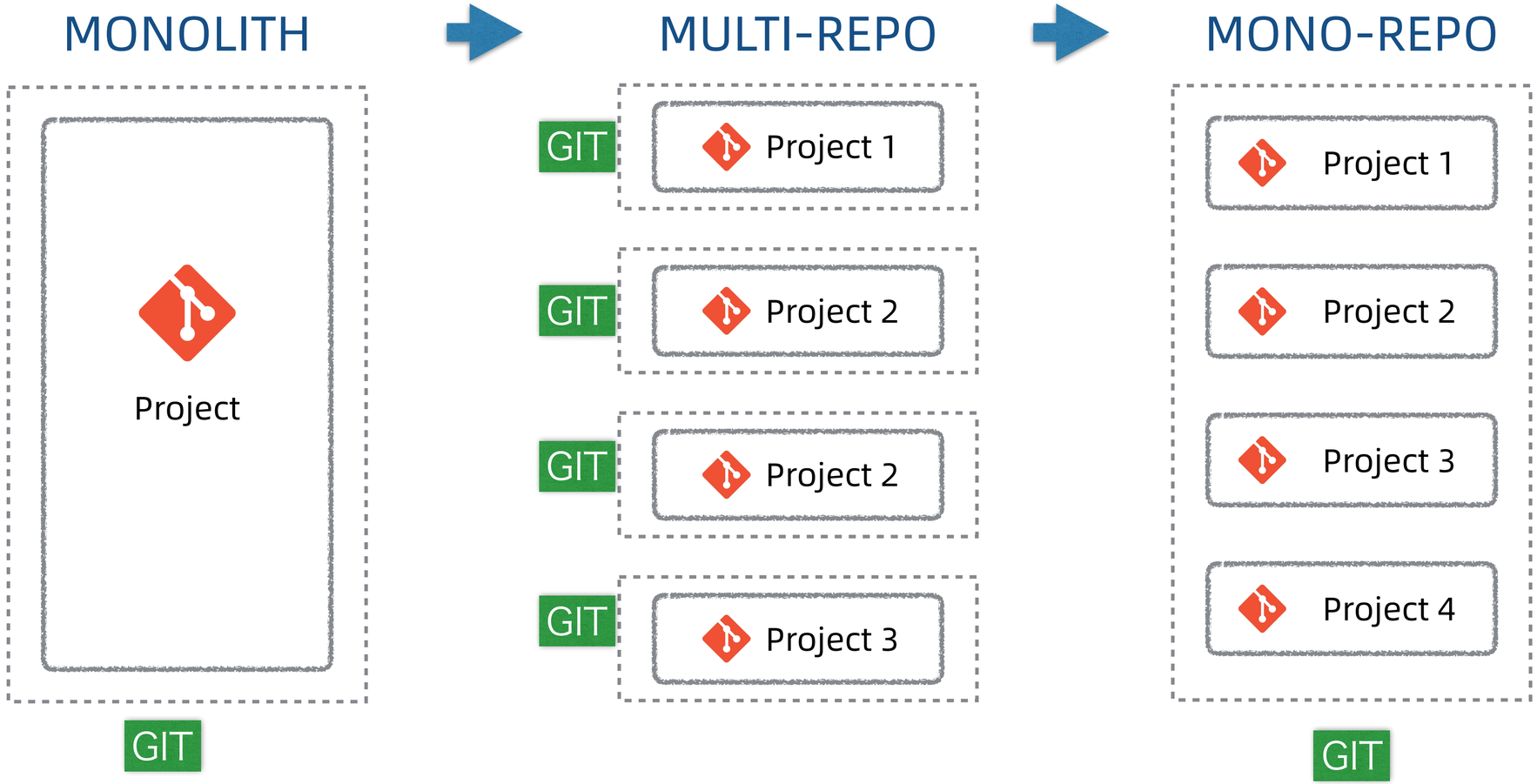
```xml
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>1.8</java.version>
    <spring.boot.version>2.1.2.RELEASE</spring.boot.version>
    <spring.cloud.version>Greenwich.RELEASE</spring.cloud.version>
</properties>
```

```xml
<dependencyManagement>
    <dependencies>
        <!-- Spring Boot -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>${spring.boot.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <!-- Spring Cloud -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring.cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
```

第 2 部分

谷歌为什么采用单体仓库 (Mono-Repo)

Multi-Repo vs Mono-Repo



多仓库

单体仓库

【业界的最佳实践】

谁在用 MonoRepo?

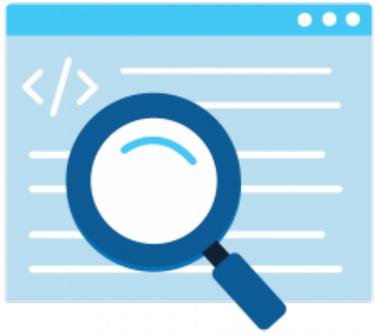
https://bazel.build/

单体仓库构建的工具

https://buck.build/

Shippable的微服务之道：mono repo vs multiple repositories

http://blog.shippable.com/our-journey-to-microservices-and-a-mono-repository



第 3 部分

微服务接口参数校验为何重要？

控制器接口参数校验

```
@{...}
public GenericAccountResponse getAccountByPhonenumber(@RequestParam @PhoneNumber String phoneNumber)

@{...}
public ListAccountResponse listAccounts(@RequestParam int offset, @RequestParam @Min(0) int limit) {

@{...}
public GenericAccountResponse getAccount(@RequestParam @NotBlank String userId) {...}

@{...}
public GenericAccountResponse updateAccount(@RequestBody @Valid AccountDto newAccountDto) {...}

@{...}
public BaseResponse updatePassword(@RequestBody @Valid UpdatePasswordRequest request) {...}
```

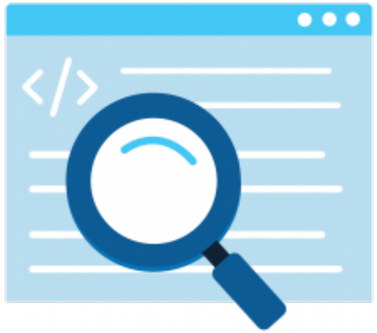
DTO 参数校验

```
public class AccountDto {
    @NotBlank
    private String id;
    private String name;
    @Email(message = "Invalid email")
    private String email;
    private boolean confirmedAndActive;
    @NotNull
    private Instant memberSince;
    private boolean support;
    @PhoneNumber
    private String phoneNumber;
    @NotEmpty
    private String photoUrl;
}
```

自定义标注

```java
package xyz.staffjoy.common.validation;

import ...

@Documented
@Constraint(validatedBy = PhoneNumberValidator.class)
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
public @interface PhoneNumber {
    String message() default "Invalid phone number";
    Class[] groups() default {};
    Class[] payload() default {};
}
```

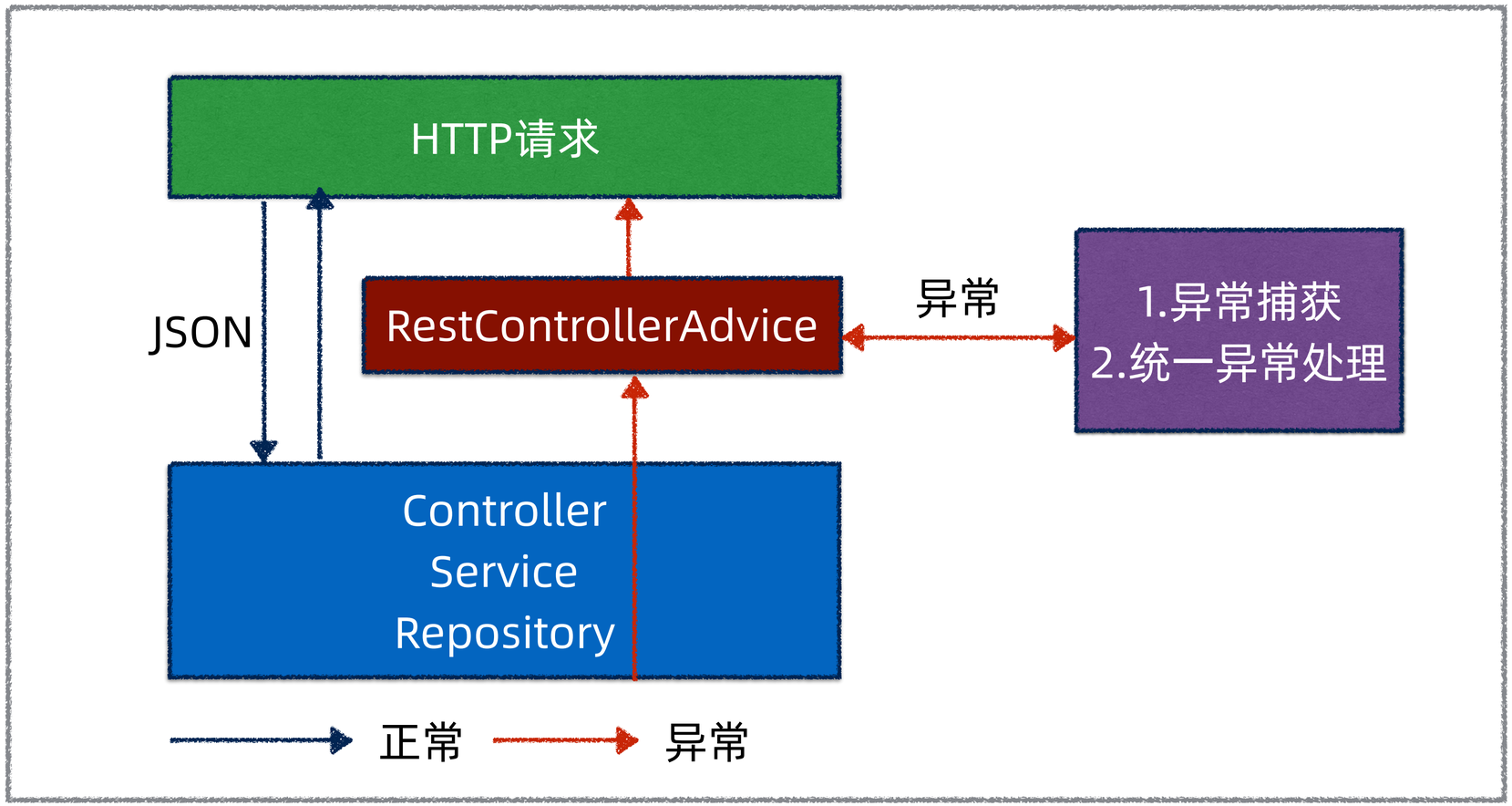
```java
package xyz.staffjoy.common.validation;

import ...

public class PhoneNumberValidator implements ConstraintValidator<PhoneNumber, String> {

    @Override
    public boolean isValid(String phoneField, ConstraintValidatorContext context) {
        if (phoneField == null) return true; // can be null
        return phoneField != null && phoneField.matches( regex: "[0-9]+")
                && (phoneField.length() > 8) && (phoneField.length() < 14);
    }
}
```



第 4 部分

如何实现统一异常处理？

统一异常处理


HTTP请求
JSON
RestControllerAdvice
异常
1.异常捕获
2.统一异常处理
Controller
Service
Repository
正常
异常

RestControllerAdvice

GlobalExceptionTranslator

```
package xyz.staffjoy.common.error;

import ...

@RestControllerAdvice
public class GlobalExceptionTranslator {

    static final ILogger logger = SLoggerFactory.getLogger(GlobalExceptionTranslator.class);

    @ExceptionHandler(MissingServletRequestParameterException.class)
    public BaseResponse handleError(MissingServletRequestParameterException e) {...}

    @ExceptionHandler(MethodArgumentTypeMismatchException.class)
    public BaseResponse handleError(MethodArgumentTypeMismatchException e) {...}

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public BaseResponse handleError(MethodArgumentNotValidException e) {...}

    @ExceptionHandler(BindException.class)
    public BaseResponse handleError(BindException e) {...}

    @ExceptionHandler(ConstraintViolationException.class)
    public BaseResponse handleError(ConstraintViolationException e) {...}

    @ExceptionHandler(NoHandlerFoundException.class)
    public BaseResponse handleError(NoHandlerFoundException e) {...}
```

统一异常捕获

```
@ExceptionHandler(ServiceException.class)
public BaseResponse handleError(ServiceException e) {
    logger.error( message: "Service Exception", e);
    return BaseResponse
            .builder()
            .code(e.getResultCode())
            .message(e.getMessage())
            .build();
}

@ExceptionHandler(PermissionDeniedException.class)
public BaseResponse handleError(PermissionDeniedException e) {...}

@ExceptionHandler(Throwable.class)
public BaseResponse handleError(Throwable e) {
    logger.error( message: "Internal Server Error", e);
    return BaseResponse
            .builder()
            .code(ResultCode.INTERNAL_SERVER_ERROR)
            .message(e.getMessage())
            .build();
}
```

BaseResponse

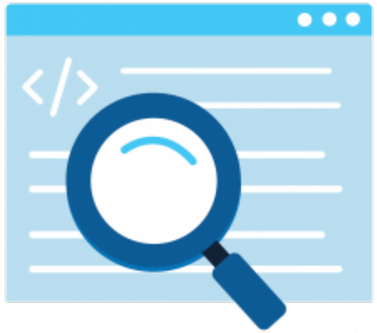
BaseResponse

```java
package xyz.staffjoy.common.api;

import ...

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class BaseResponse {
    private String message;
    @Builder.Default
    private ResultCode code = ResultCode.SUCCESS;

    public boolean isSuccess() { return code == ResultCode.SUCCESS; }
}
```

Web MVC ErrorController

GlobalErrorController

```
package xyz.staffjoy.web.controller;

import ...

@Controller
/_uplicates/
public class GlobalErrorController implements ErrorController {

    static final ILogger logger = SLoggerFactory.getLogger(GlobalErrorController.class);

    @Override
    public String getErrorPath() { return "/error"; }

    @RequestMapping("/error")
    public String handleError(HttpServletRequest request, Model model) {

        Object statusCode = request.getAttribute(RequestDispatcher.ERROR_STATUS_CODE);
        Object exception = request.getAttribute(RequestDispatcher.ERROR_EXCEPTION);

        ErrorPage errorPage = null;
        if (statusCode != null && (Integer)statusCode == HttpStatus.NOT_FOUND.value()) {
            errorPage = errorPageFactory.buildNotFoundPage();
        } else {
            errorPage = errorPageFactory.buildInternalServerErrorPage();
        }
```



第 5 部分

DTO 和 DMO为什么要互转？

DTO 和 DMO

AccountDto

```java
package xyz.staffjoy.account.dto;

import ...

@{...}
public class AccountDto {
    @NotBlank
    private String id;

    private String name;

    @Email(message = "Invalid email")
    private String email;

    private boolean confirmedAndActive;

    @NotNull
    private Instant memberSince;

    private boolean support;

    @PhoneNumber
    private String phoneNumber;

    @NotEmpty
    private String photoUrl;
}
```

Account

```java
package xyz.staffjoy.account.model;

import ...

@{...}
public class Account {
    @Id
    @GenericGenerator(name = "system-uuid", strategy = "uuid")
    @GeneratedValue(generator = "system-uuid")
    private String id;
    private String name;
    private String email;
    private boolean confirmedAndActive;
    private Instant memberSince;
    private boolean support;
    private String phoneNumber;
    private String photoUrl;
}
```

DTO:数据传输对象

DMO:数据模型对象，别称 EntityObject,DomainObject

DTO 和 DMO互转

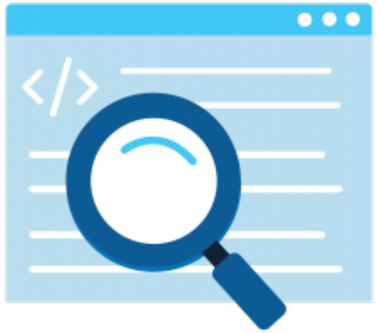
使用【DataMapper工具】处理大多数的字段映射

```
public AccountDto update(AccountDto newAccountDto) {
    Account newAccount = this.convertToModel(newAccountDto);
```

```
    AccountDto accountDto = this.convertToDto(newAccount);
    return accountDto;
}
```

```
private AccountDto convertToDto(Account account) {
    return modelMapper.map(account, AccountDto.class);
}

private Account convertToModel(AccountDto accountDto) {
    return modelMapper.map(accountDto, Account.class);
}
```

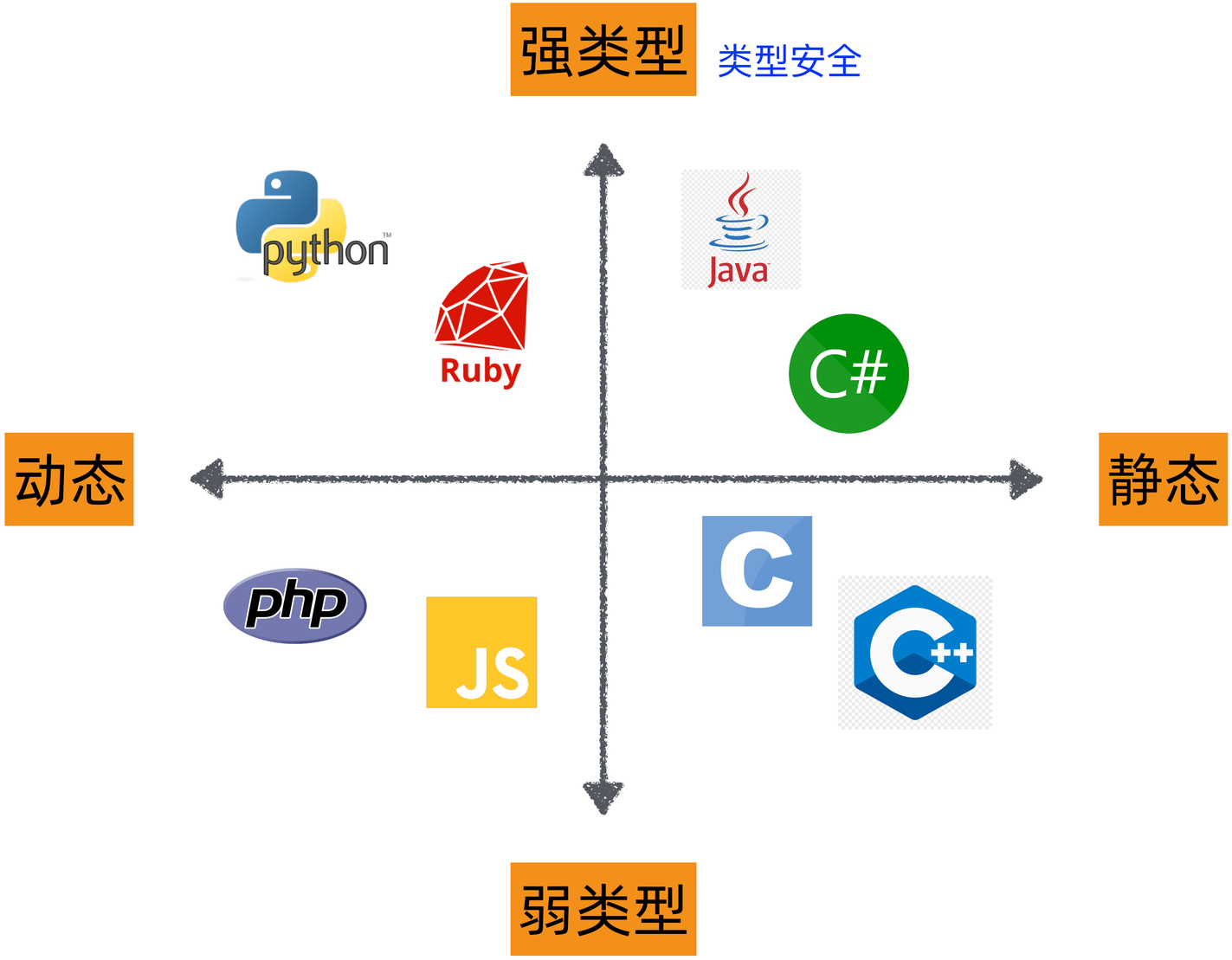
https://github.com/modelmapper/modelmapper



第 6 部分

如何实现强类型接口设计？

强类型 vs 弱类型



Apache Thrift™

GRPC

强类型

一般是二进制

如grpc、thrift

弱类型

spring boot

如json、xml

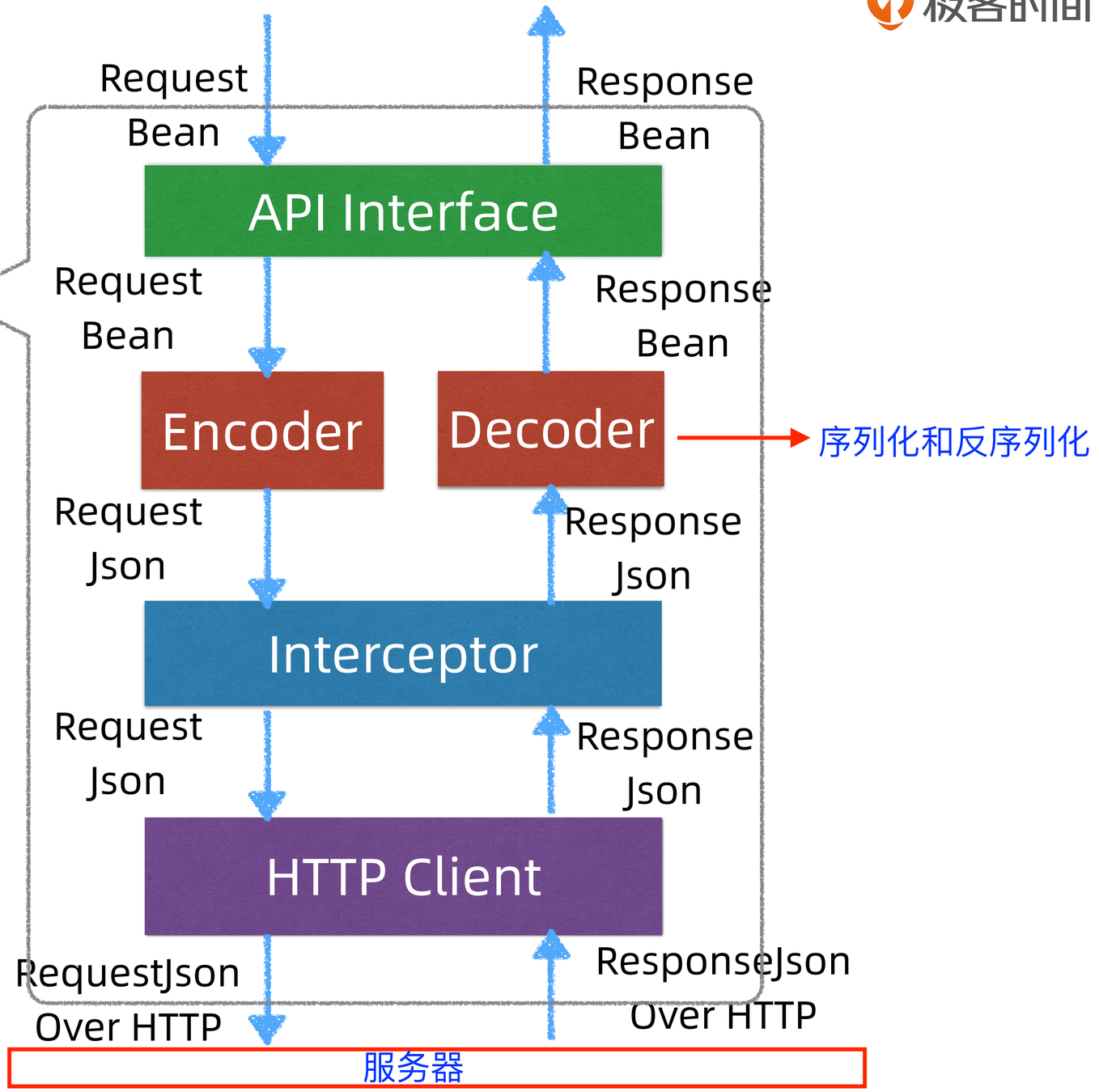
Spring Feign

Dynamic Proxy

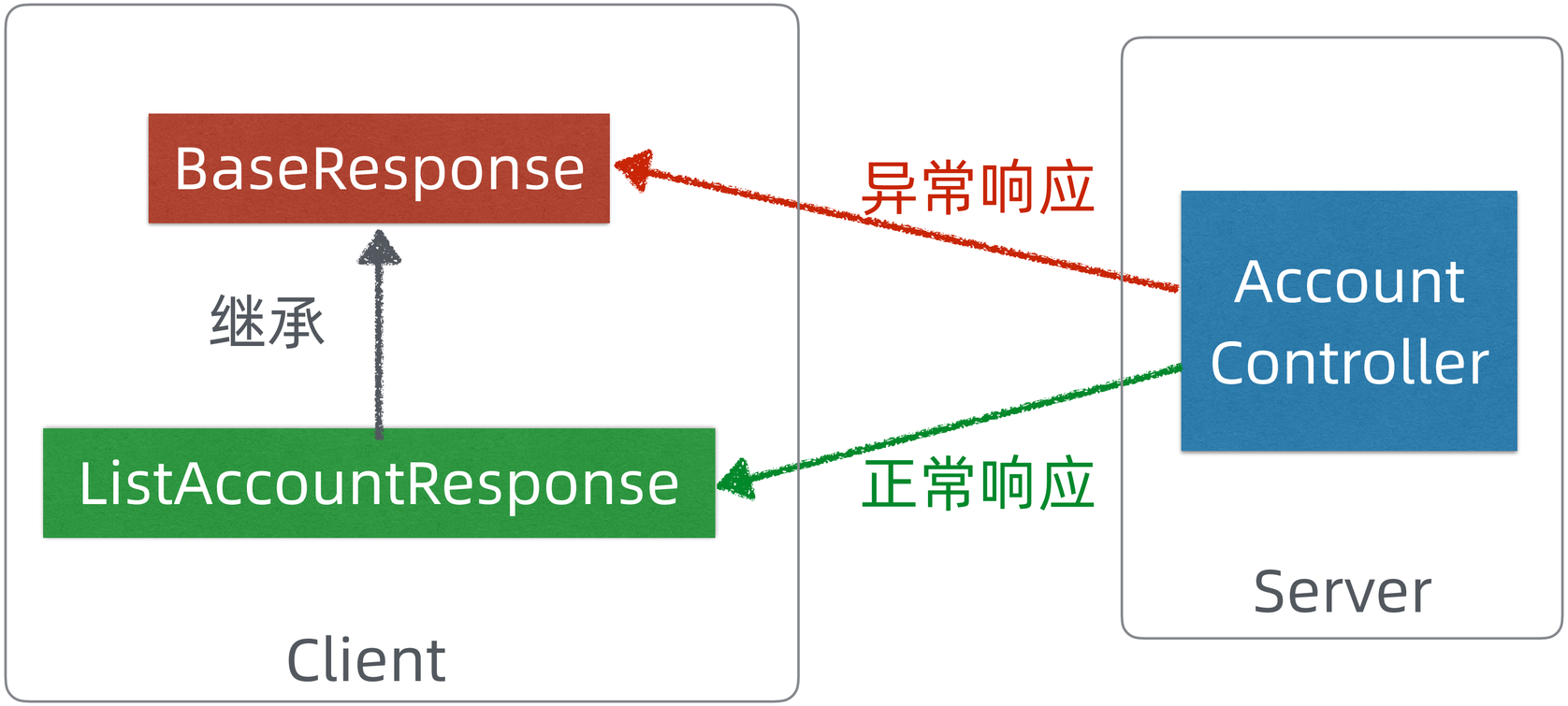
动态代理

能够在运行时拼装出对应类型的强类型客户端

可以同时利用强弱类型的好处



强类型接口设计


BaseResponse
异常响应
Account Controller
继承
ListAccountResponse
正常响应
Client
Server

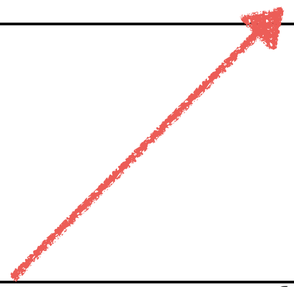
Account Client

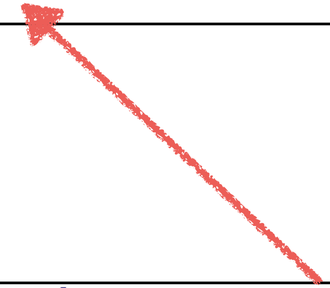
AccountClient › listAccounts()

```java
package xyz.staffjoy.account.client;

import ...

@FeignClient(name = AccountConstant.SERVICE_NAME, path = "/v1/account", url = "${staffjoy.account-serv
// TODO Client side validation can be enabled as needed
// @Validated
public interface AccountClient {

    @PostMapping(path = "/create")
    GenericAccountResponse createAccount(@RequestHeader(AuthConstant.AUTHORIZATION_HEADER) String auth

    @PostMapping(path = "/track_event")
    BaseResponse trackEvent(@RequestBody @Valid TrackEventRequest request);

    @PostMapping(path = "/sync_user")
    BaseResponse syncUser(@RequestBody @Valid SyncUserRequest request);

    @GetMapping(path = "/list")
    ListAccountResponse listAccounts(@RequestHeader(AuthConstant.AUTHORIZATION_HEADER) String authz, (

    // GetOrCreate is for internal use by other APIs to match a user based on their phonenumber or ema
    @PostMapping(path= "/get_or_create")
    GenericAccountResponse getOrCreateAccount(@RequestHeader(AuthConstant.AUTHORIZATION_HEADER) String

    @GetMapping(path = "/get")
    GenericAccountResponse getAccount(@RequestHeader(AuthConstant.AUTHORIZATION_HEADER) String authz,
```

继承关系

```java
public class BaseResponse {
    private String message;
    @Builder.Default
    private ResultCode code = ResultCode.SUCCESS;

    public boolean isSuccess() { return code == ResultCode.SUCCESS; }
}
```





```java
public class ListAccountResponse extends BaseResponse {
    private AccountList accountList;
}
```

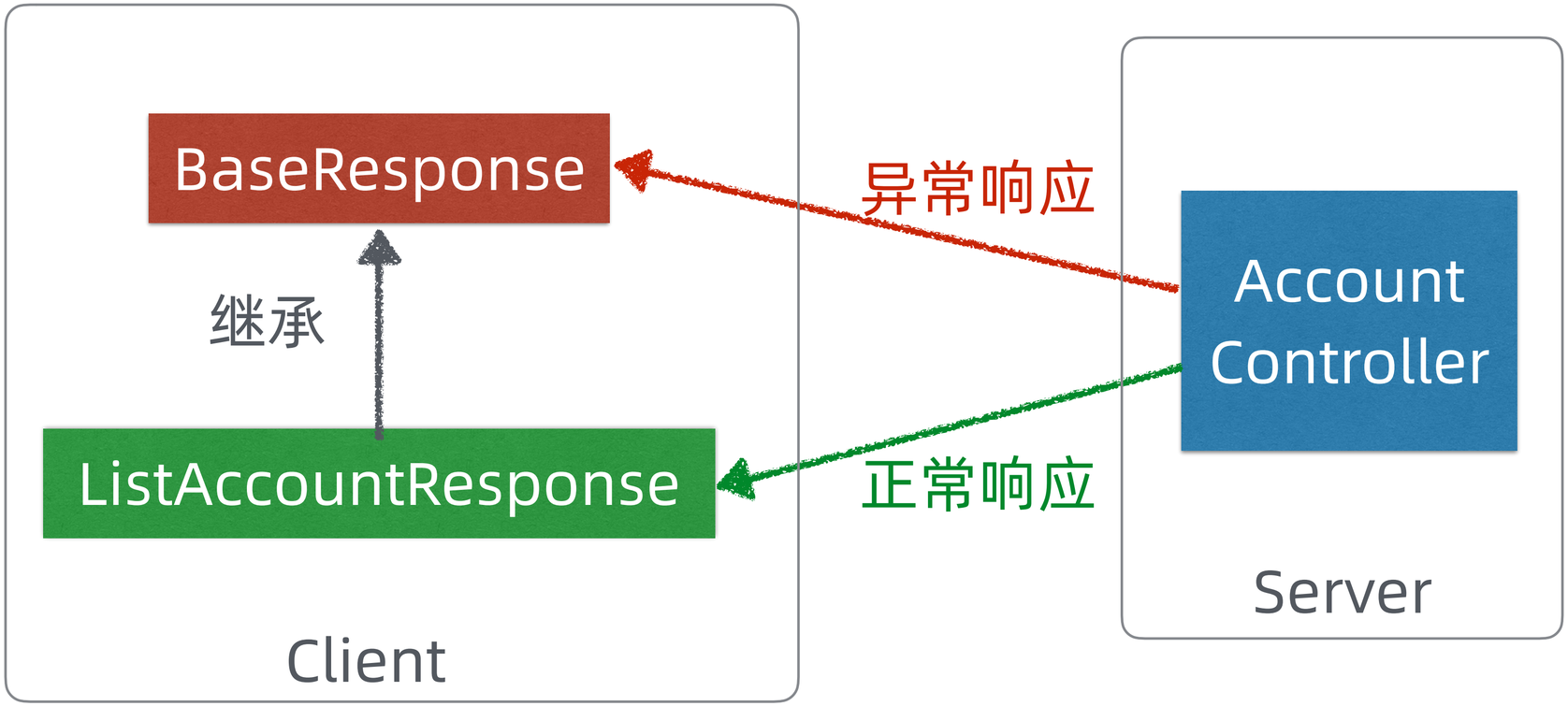
```java
public class GenericAccountResponse extends BaseResponse {
    private AccountDto account;
}
```

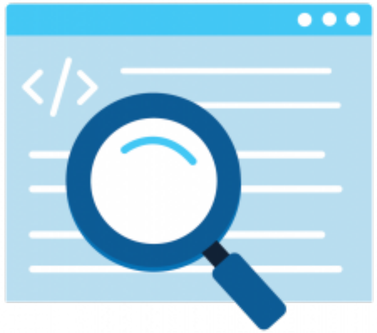
客户端调用范例

```
GenericAccountResponse genericAccountResponse = null;
try {
    genericAccountResponse = this.accountClient.getAccount(AuthConstant.AU
} catch (Exception ex) {
    String errMsg = "unable to get account";
    handleErrorAndThrowException(ex, errMsg);
}
if (!genericAccountResponse.isSuccess()) {
    handleErrorAndThrowException(genericAccountResponse.getMessage());
}
AccountDto account = genericAccountResponse.getAccount();
```

封装消息+捎带

即把状态码封装成一个枚举对象，不用每个地方自己处理不同的状态码



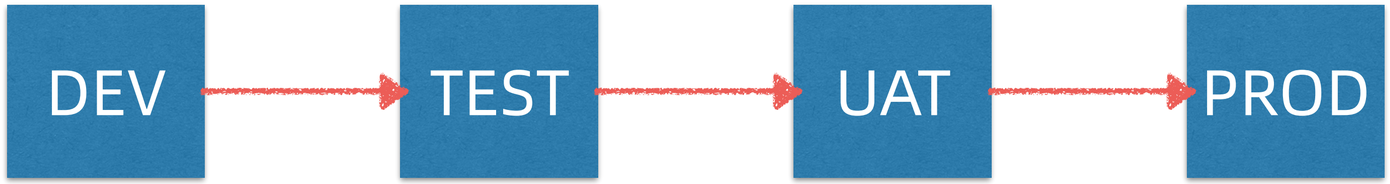


第 7 部分

为什么框架层要考虑分环境配置？

环境定义

```java
package xyz.staffjoy.common.env;

public class EnvConstant {
    public static final String ENV_DEV = "dev";
    public static final String ENV_TEST = "test";
    public static final String ENV_UAT = "uat"; // similar to staging
    public static final String ENV_PROD = "prod";
}
```

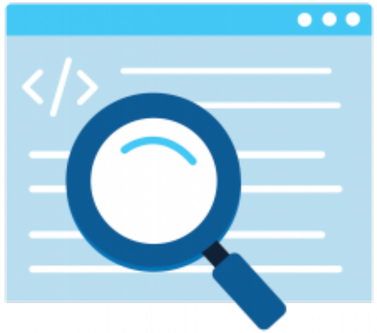


环境配置

```
EnvConfig envConfig = EnvConfig.builder().name(EnvConstant.ENV_DEV)
        .debug(true)
        .externalApex("staffjoy-v2.local")
        .internalApex(EnvConstant.ENV_DEV)
        .scheme("http")
        .build();
map.put(EnvConstant.ENV_DEV, envConfig);

envConfig = EnvConfig.builder().name(EnvConstant.ENV_TEST)
        .debug(true)
        .externalApex("staffjoy-v2.local")
        .internalApex(EnvConstant.ENV_DEV)
        .scheme("http")
        .build();
map.put(EnvConstant.ENV_TEST, envConfig);

envConfig = EnvConfig.builder().name(EnvConstant.ENV_UAT)
        .debug(false)
        .externalApex("staffjoy-uat.xyz")
        .internalApex(EnvConstant.ENV_UAT)
        .scheme("https")
        .build();
map.put(EnvConstant.ENV_UAT, envConfig);

envConfig = EnvConfig.builder().name(EnvConstant.ENV_PROD)
        .debug(false)
        .externalApex("staffjoy.xyz")
        .internalApex(EnvConstant.ENV_PROD)
        .scheme("https")
```

开发测试环境禁用Sentry异常日志

```java
@Aspect
@Slf4j
public class SentryClientAspect {

    @Autowired
    EnvConfig envConfig;

    @Around("execution(* io.sentry.SentryClient.send*(..))")
    public void around(ProceedingJoinPoint joinPoint) throws Throwable {
        // no sentry logging in debug mode
        if (envConfig.isDebug()) {
            log.debug("no sentry logging in debug mode");
            return;
        }
        joinPoint.proceed();
    }
}
```

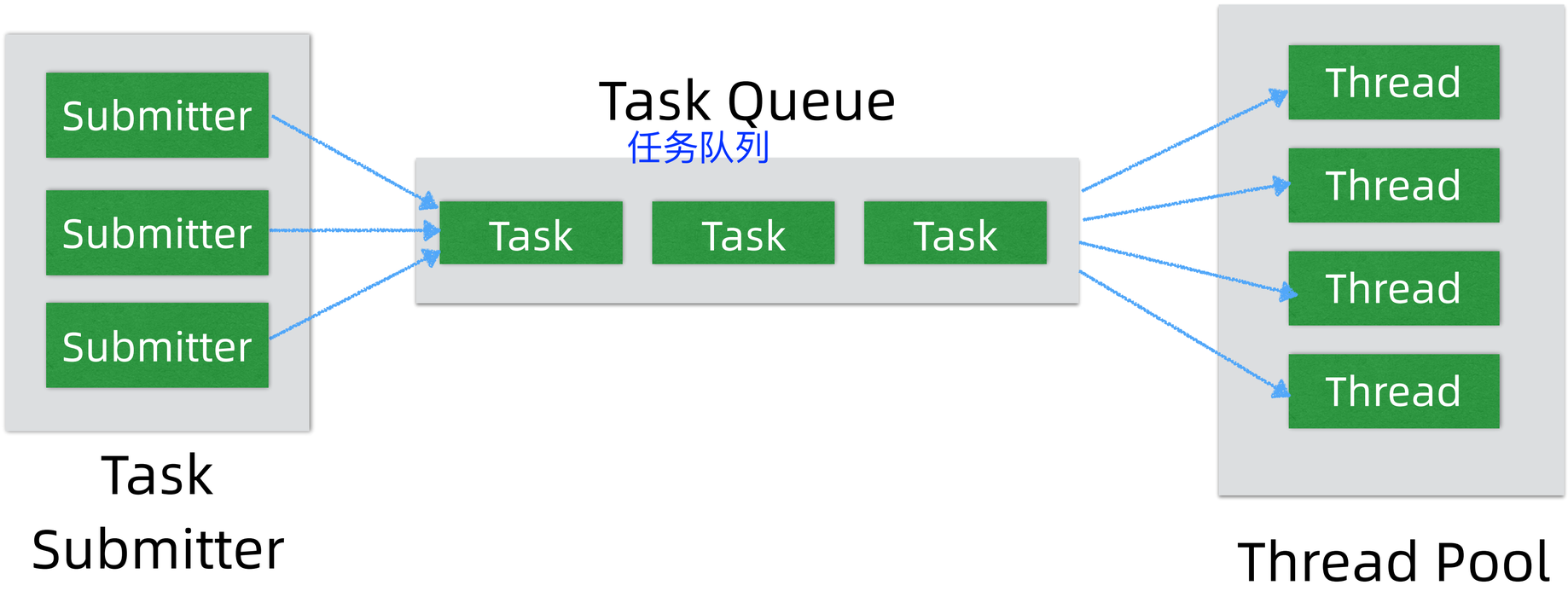


第 8 部分

异步调用处理

ThreadPoolTaskExecutor

Java 线程池支持异步



AsyncExecutor 配置

```
public class AppConfig {

    public static final String ASYNC_EXECUTOR_NAME = "asyncExecutor";

    @Bean(name=ASYNC_EXECUTOR_NAME)
    public Executor asyncExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        // for passing in request scope context
        executor.setTaskDecorator(new ContextCopyingDecorator());
        executor.setCorePoolSize(3);
        executor.setMaxPoolSize(5);
        executor.setQueueCapacity(100);
        executor.setWaitForTasksToCompleteOnShutdown(true);
        executor.setThreadNamePrefix("AsyncThread-");
        executor.initialize();
        return executor;
    }
```

Async 标注

⚠️：异步调用的方法不能放在调用的bean中，需要隔离

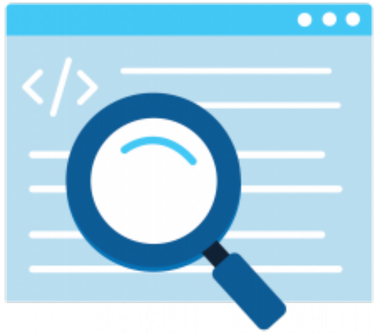
```
@Async(AppConfig.ASYNC_EXECUTOR_NAME)
public void trackEventAsync(String userId, String eventName) {
    if (envConfig.isDebug()) {
        logger.debug( message: "intercom disabled in dev & test environment");
        return;
    }

    Event event = new Event()
            .setUserID(userId)
            .setEventName("v2_" + eventName)
            .setCreatedAt(Instant.now().toEpochMilli());

    try {
        Event.create(event);
    } catch (Exception ex) {
        String errMsg = "fail to create event on Intercom";
        handleException(logger, ex, errMsg);
        throw new ServiceException(errMsg, ex);
    }

    logger.debug( message: "updated intercom");
}
```

线程上下文拷贝

⚠️：如果使用的是异步调用，在上下文切换时，上下文中的信息就会丢失，所以需要拷贝，如用户信息

```java
public class ContextCopyingDecorator implements TaskDecorator {
    @Override
    public Runnable decorate(Runnable runnable) {
        RequestAttributes context = RequestContextHolder.currentRequestAttributes();
        return () -> {
            try {
                RequestContextHolder.setRequestAttributes(context);
                runnable.run();
            } finally {
                RequestContextHolder.resetRequestAttributes();
            }
        };
    }
}
```

```java
public class AppConfig {

    public static final String ASYNC_EXECUTOR_NAME = "asyncExecutor";

    @Bean(name=ASYNC_EXECUTOR_NAME)
    public Executor asyncExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        // for passing in request scope context
        executor.setTaskDecorator(new ContextCopyingDecorator());
        executor.setCorePoolSize(3);
        executor.setMaxPoolSize(5);
        executor.setQueueCapacity(100);
        executor.setWaitForTasksToCompleteOnShutdown(true);
        executor.setThreadNamePrefix("AsyncThread-");
        executor.initialize();
        return executor;
    }
```



第 9 部分

Swagger 接口文档

swagger文档是基于 openapi的规范开发

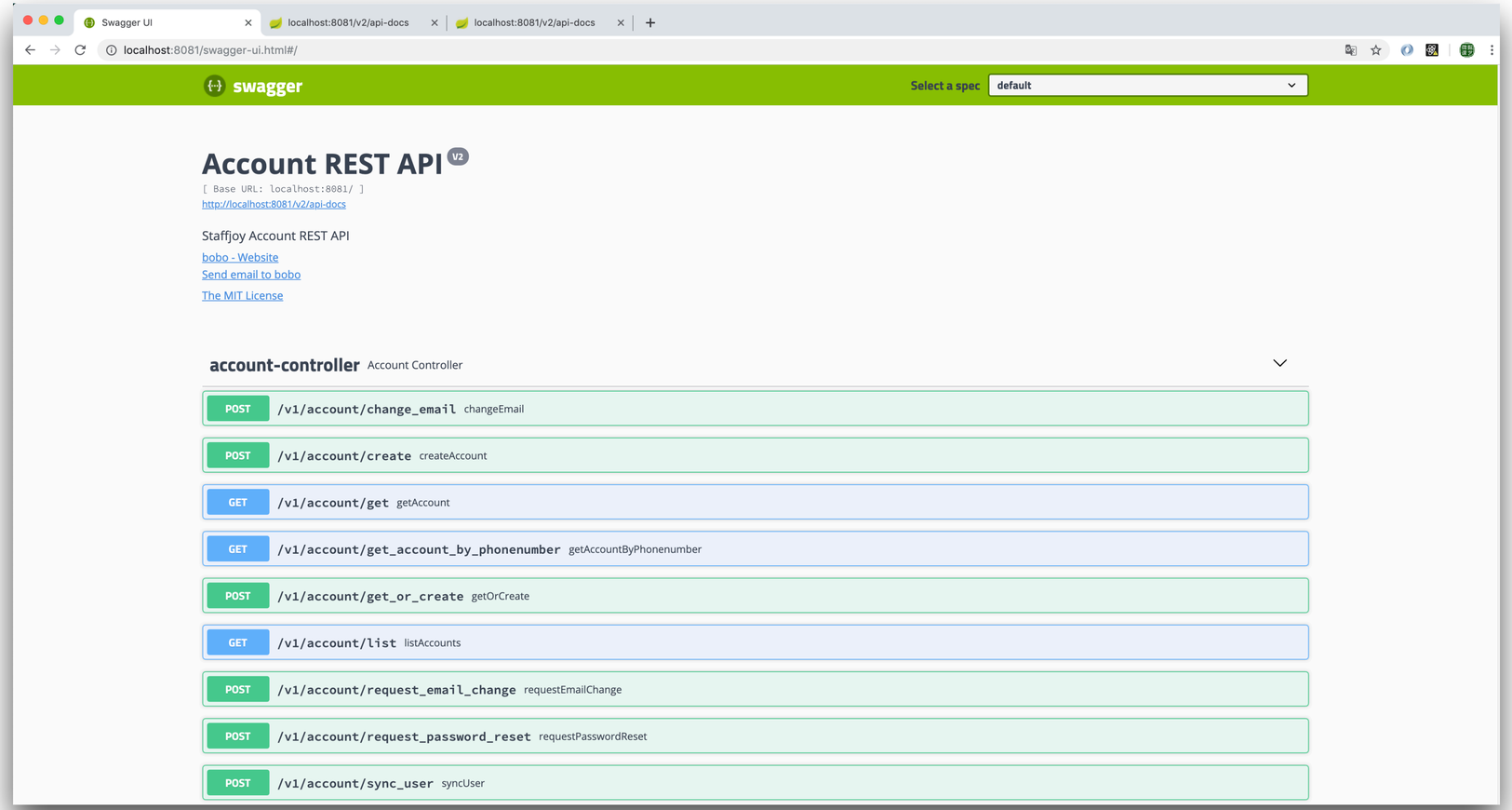
https://swagger.io/docs/specification/about/

Swagger 配置

```xml
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
</dependency>
```

```java
@Configuration
@EnableSwagger2
public class SwaggerConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select() ApiSelectorBuilder
                .apis(RequestHandlerSelectors.basePackage("xyz.staffjoy.account.controller"))
                .paths(PathSelectors.any()) ApiSelectorBuilder
                .build() Docket
                .apiInfo(apiEndPointsInfo()) Docket
                .useDefaultResponseMessages(false);
    }

    private ApiInfo apiEndPointsInfo() {
        return new ApiInfoBuilder().title("Account REST API")
                .description("Staffjoy Account REST API")
                .contact(new Contact( name: "bobo",  url: "https://github.com/jskillcloud",  email:
                .license("The MIT License")
                .licenseUrl("https://opensource.org/licenses/MIT")
                .version("V2")
                .build();
    }
}
```

Swagger UI



https://dzone.com/articles/spring-boot-2-restful-api-documentation-with-swagg

Swagger JSON Doc

可以生成文档，也可以生成代码

https://editor.swagger.io

第 10 部分

主流服务框架概览

主流服务框架概览

	支持公司	编程风格	编程模型	支持语言	亮点
Spring(Boot)	Pivotal	REST	代码优先	Java	社区生态好
Dubbo	阿里	RPC/REST	代码优先	Java	阿里背书+ 服务治理
Motan	新浪	RPC	代码优先	Java为主	轻量版Dubbo
gRpc	谷歌	RPC	契约优先	跨语言	谷歌背书+多语言支持+HTTP2支持

需要引入 rpc api gateway

参考链接

1. **Shippable的微服务之旅：mono repo vs multiple repositories** http://blog.shippable.com/our-journey-to-microservices-and-a-mono-repository
2. **Why startups need to use monorepo** https://medium.com/@hoangbkit/why-monorepo-in-2018-89221acd4bfb
3. **Google Build System Bazel** https://bazel.build/
4. **Facebook Build System Buck** https://buck.build/
5. **Model Mapper** https://github.com/modelmapper/modelmapper
6. **gRPC** https://grpc.io/
7. **Thrift** https://github.com/apache/thrift
8. **Swagger Codegen** https://github.com/swagger-api/swagger-codegen
9. **Spring Boot2 Restful API Document with Swagger** https://dzone.com/articles/spring-boot-2-restful-api-documentation-with-swagg
10. **Swagger和Open API** https://swagger.io/docs/specification/about/

极客时间

扫码试看/订阅

《Spring Boot & Kubernetes 云原生微服务实践》