



使用SpringMVC开发REST风格的服务

怀民

魚香烘蛋

议程

REST在实际环境中的应用
SpringMVC对REST的支持
编写符合工程质量的REST服务



REST在实际环境中的应用

REST简单介绍

REST是一种基于HTTP/1.1协议基础之上的“古老”的协议。

REST是HTTP的，因此它遵循HTTP本身的特征。

在工程实践中，REST更接近一种“风格”，而非一种“标准”。

REST是非常轻量级的协议。

REST简单介绍--GET

GET /users/26

GET 主要用来获取在一定时间范围内相对稳定的信息。

GET 调用多次时不应产生副作用。（而非每次的结果一定相同）

GET /news/latest

GET 可以设置条件或范围。

通过HTTP HEADER，实现缓存控制的主要手段

If-Modified-Since、If-None-Match等

REST简单介绍--GET

ETag

```
$ curl -i example.com/api/todo/4
```

The servers response:

```
HTTP/1.1 200 OK
Date: Sat, 09 Feb 2013 16:09:50 GMT
Server: Apache/2.2.22 (Ubuntu)
Last-Modified: Sat, 02 Feb 2013 12:02:47 GMT
ETag: "c0947-b1-4d0258df1f625"
Content-Type: application/json

{
  id: 4,
  item: "take out the trash",
  created: "Sat, 02 Feb 2013 08:29:53 GMT",
  updated: "Sat, 02 Feb 2013 12:02:47 GMT",
}
```

更复杂的使用场景--乐观锁

```
$ curl -i -H "If-None-Match: c0947-b1-4d0258df1f625" example.com/api/todo/4
```

If we suppose that the To Do items has **NOT** changed, we may see this result:

```
HTTP/1.1 304 Not Modified
Date: Sat, 09 Feb 2013 16:09:50 GMT
Server: Apache/2.2.22 (Ubuntu)
Last-Modified: Sat, 02 Feb 2013 12:02:47 GMT
ETag: "c0947-b1-4d0258df1f625"
```

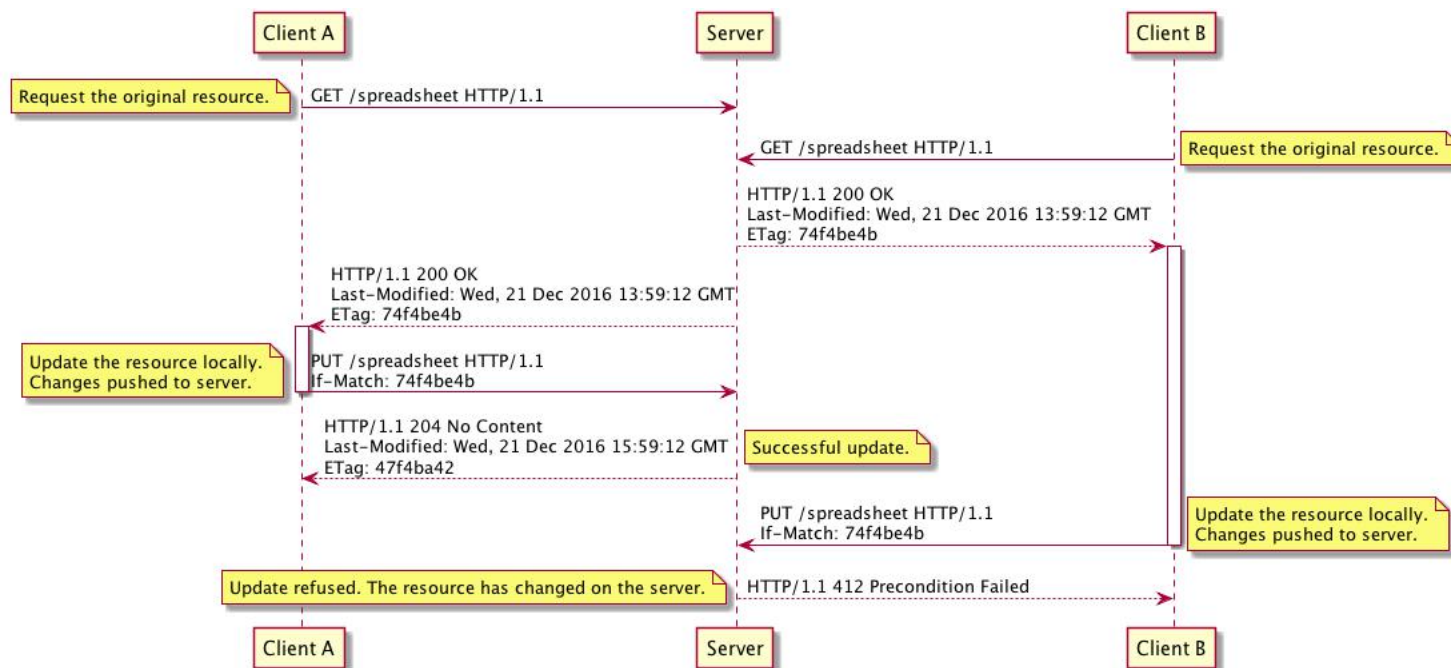
If, however, the To Do items **HAS** changed, we may see this result:

```
HTTP/1.1 200 OK
Date: Sat, 09 Feb 2013 16:29:24 GMT
Server: Apache/2.2.22 (Ubuntu)
Last-Modified: Sat, 02 Feb 2013 14:33:21 GMT
ETag: "c7493-d7-a6b64d37f6cc3" # New ETag!
Content-Type: application/json

{
  id: 4,
  item: "Take out the trash, TODAY!",
  created: "Sat, 02 Feb 2013 08:29:53 GMT",
  updated: "Sat, 02 Feb 2013 14:33:21 GMT",
}
```

REST简单介绍--GET

ETag更复杂的使用场景--乐观锁



REST简单介绍-- POST

POST /users

```
{"name":"Juergen","age":26,"gender","M"}
```

POST 主要用来针对具体的业务对象传递相关的动作请求。

POST 同一请求调用多次时应产生完全相同的副作用。（幂等性）

在创建业务对象的操作中容易产生问题。

在实际应用中，POST 操作一般会对业务对象产生创建或者更新的操作。

REST简单介绍-- PUT DELETE

没有在大范围内使用

很多遗留系统和中间件对GET、POST之外的HTTP协议支持有限

REST风格的API大规模使用，REST逐渐不仅仅是面向资源

DELETE 用于删除业务对象，包括立即删除和非立即删除

DELETE /users/26/disable

PUT 保存或者更新一个业务对象

REST简单介绍-- 状态码

Response返回时为什么要修改状态码？

REST是应用层协议，遵循HTTP协议本身能带来更好的适用性
中间件的支持（如Nginx）
调试、日志、监控、统计等

1xx：消息返回。表面请求已经被接受，请求处理尚未完成。（实际上并没有得到广泛的支持，不建议使用）

2xx：操作成功。

3xx：重定向到另一个URI。（如API版本迁移）

4xx：客户端请求错误。（如业务校验失败）

5xx：服务器端错误。

REST简单介绍-- REST风格接口设计三要素

安全

“安全”的REST接口，在语义上更接近“只读功能”。

安全的客户端请求，不应导致服务器端任何业务状态的改变。

反例：在获取车辆信息时冻结车辆状态。

幂等

一个接口的多次相同请求对服务器的预期影响与单个此类请求的效果相同，则该接口被认为是“幂等”的。

幂等在无状态的REST服务中有着非常重要的意义。

可缓存

如果一个接口被定义为“可缓存”，那么说明接口的响应数据允许被存储起来以便重用

通常不依赖于当前上下文或不需要认证的安全的接口可定义为是可缓存的。

REST简单介绍-- REST风格接口设计三要素

协议方法	安全	幂等	可缓存
GET	Y	Y	Y
POST	No	No	No

对于大多数Get方法，需要遵循协议本身，保证接口安全、幂等。并通过协议控制体现该接口是否可缓存。

对于大多数Post方法，应设计为不安全且不可缓存，并在服务的具体实现层面保证幂等。

REST简单介绍-- 实用主义风格

URI前缀: `http(s)://{域名}/{应用}/{版本号}/{模块}/...`

域名: 大的业务域划分。和团队的组织结构相关

版本: 同一应用下的所有接口应维护同一版本

版本信息要么维护在代码分支 (POM) 上, 要么维护在某个基础服务上

REST简单介绍-- 实用主义风格

使用复数形式的名词风格

接口	POST	GET	PUT	DELETE
/groups	创建一个group	返回Group列表	如果允许批量更新，则执行更新操作；否则返回405错误	如果允许批量删除，则执行操作；否则返回405错误
/groups/{group id}	不允许，返回405错误	返回具体group的明细数据	更新具体的group	删除指定的group

REST简单介绍-- 实用主义风格

复数形式的名词风格+限定词扩展

只使用GET和POST两种方法

GET /groups/{group_id}/{名词限定词}

POST /groups/{group_id}/{动词限定词}

常用的名词限定词：detail、desc、summary、list

常用的动词限定词：create、modify、apply、withdraw

根据实际的业务场景来定义

REST简单介绍-- 实用主义风格

参数

查询条件

分页 `offset=xx&limit=xx`

SpringMVC对REST的支持

注解

@RestController

@RequestMapping

对方法进行注释，处理HTTP 方法、URI 或 HTTP 头。

@Param

@PathVariable

将 URI 中的路径变量作为参数

@RequestHeader

将Header中的变量作为参数

@RequestBody

将整个Request Body作为参数

SpringMVC对REST的支持

序列化/反序列化

MessageConverter

StringHttpMessageConverter

text/*

FormHttpMessageConverter

application/x-www-form-urlencoded

MappingJackson2MessageConverter

application/json

Jaxb2RootElementHttpMessageConverter

application/xml

SpringMVC对REST的支持

序列化/反序列化

ContentNegotiation

需要同时支持视图模式和REST模式的场景

需要支持多种REST数据格式的场景

当一个接口需要Controller方法返回相同数据的多种表示（或视图）时，确定要返回的数据格式的实现机制称为内容协商。

SpringMVC对REST的支持

配置MessageConverter和ContentNegotiation

```
@EnableWebMvc
@Configuration
class WebConfig extends WebMvcConfigurerAdapter {

    @Override
    public void configureContentNegotiation(ContentNegotiationConfigurer c) {
        configurator.favorPathExtension(true). //支持uri后缀
            favorParameter(true). //在request的参数中指定具体
            parameterName("mediaType"). //request参数的名称
            ignoreAcceptHeader(false). //是否忽略Header中的Accept属性
            useJaf(false). //不使用JAF
            defaultContentType(MediaType.TEXT_PLAIN).
            mediaType("xml", MediaType.APPLICATION_XML).
            mediaType("json", MediaType.APPLICATION_JSON);
    }
}
```

```
@Override
public void configureMessageConverters(List<HttpMessageConverter<?>> converters) {
    converters.add(jsonConverter());
    converters.add(stringConverter());
    converters.add(xmlConverter());
    super.configureMessageConverters(converters);
}
}
```

```
@Primary
@Bean
public JsonMessageConverter jsonConverter() {
    JsonMessageConverter converter = new JsonMessageConverter(objectMapper());
    converter.setSupportedMediaTypes(
        Arrays.asList(
            MediaType.APPLICATION_JSON_UTF8
        )
    );
    return converter;
}
```

```
@Primary
@Bean
public Jaxb2RootElementHttpMessageConverter xmlConverter() {
    Jaxb2RootElementHttpMessageConverter converter = new Jaxb2RootElementHttpMessageConverter();
    converter.setSupportedMediaTypes(
        Arrays.asList(
            MediaType.TEXT_XML,
            MediaType.APPLICATION_XML
        )
    );
    return converter;
}
```

```
@Primary
@Bean
public HttpMessageConverter<String> stringConverter() {
    StringHttpMessageConverter converter = new StringHttpMessageConverter();
    converter.setSupportedMediaTypes(
        Arrays.asList(
            MediaType.TEXT_PLAIN
        )
    );
    return converter;
}
```

SpringMVC对REST的支持

ETag支持

```
<!-- etag处理 -->
<filter>
  <filter-name>etagFilter</filter-name>
  <filter-class>org.springframework.web.filter.ShallowEtagHeaderFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>etagFilter</filter-name>
  <url-pattern>/api/*</url-pattern>
</filter-mapping>
```

```
> curl -i http://127.0.0.1:8080/api/users/1111
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
ETag: "09d1664c7af31f4223d33e3b1c9acf38f"
Content-Length: 50
Server: Jetty(9.2.19.v20160908)

{"id":"1111","name":"Jerry","age":26,"gender":"M"}
```

```
Floating@X1 C:\Users\floating
> curl -H "If-None-Match: \"09d1664c7af31f4223d33e3b1c9acf38f\"" -i http://127.0.0.1:8080/api/users/1111
HTTP/1.1 304 Not Modified
Content-Type: application/json;charset=UTF-8
ETag: "09d1664c7af31f4223d33e3b1c9acf38f"
Server: Jetty(9.2.19.v20160908)
```

符合工程质量的REST服务

可阅读的服务

数据字典

数据传输对象（DTO）

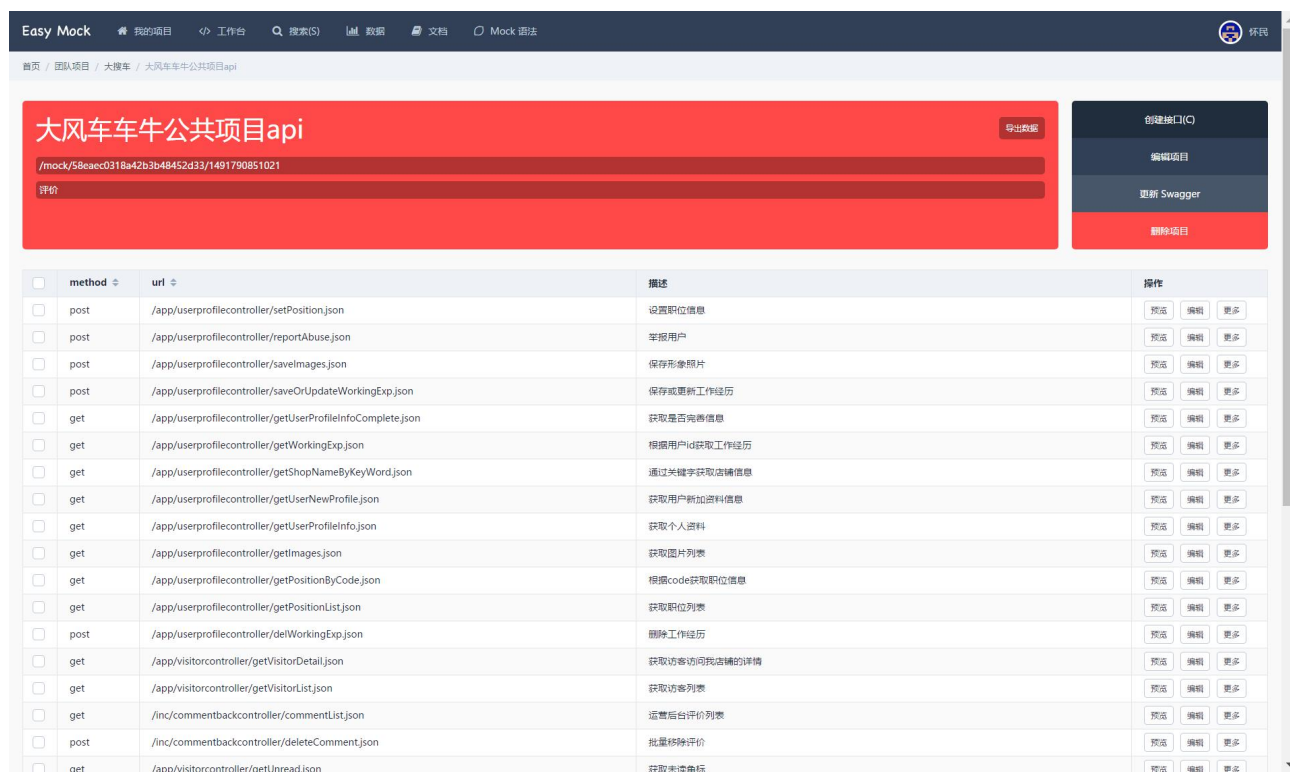
单值校验

异常包装

日志和监控

符合工程质量的REST服务--可阅读的服务

Swagger & EasyMock



大风车车牛公共项目api

/mock/58eac0318a42b3b48452d33/1491790851021

评价

method	url	描述	操作
post	/app/userprofilecontroller/setPosition.json	设置职位信息	预览 编辑 更多
post	/app/userprofilecontroller/reportAbuse.json	举报用户	预览 编辑 更多
post	/app/userprofilecontroller/saveImages.json	保存图片照片	预览 编辑 更多
post	/app/userprofilecontroller/saveOrUpdateWorkingExp.json	保存或更新工作经历	预览 编辑 更多
get	/app/userprofilecontroller/getUserProfileInfoComplete.json	获取是否完善信息	预览 编辑 更多
get	/app/userprofilecontroller/getWorkingExp.json	根据userId获取工作经历	预览 编辑 更多
get	/app/userprofilecontroller/getShopNameByKeyWord.json	通过关键字获取店铺信息	预览 编辑 更多
get	/app/userprofilecontroller/getUserNewProfile.json	获取用户新加资料信息	预览 编辑 更多
get	/app/userprofilecontroller/getUserProfileInfo.json	获取个人资料	预览 编辑 更多
get	/app/userprofilecontroller/getImages.json	获取图片列表	预览 编辑 更多
get	/app/userprofilecontroller/getPositionByCode.json	根据code获取职位信息	预览 编辑 更多
get	/app/userprofilecontroller/getPositionList.json	获取职位列表	预览 编辑 更多
post	/app/userprofilecontroller/deleteWorkingExp.json	删除工作经历	预览 编辑 更多
get	/app/visitorcontroller/getVisitorDetail.json	获取访客访问我店铺的详情	预览 编辑 更多
get	/app/visitorcontroller/getVisitorList.json	获取访客列表	预览 编辑 更多
get	/inc/commentbackcontroller/commentList.json	运营后台评价列表	预览 编辑 更多
post	/inc/commentbackcontroller/deleteComment.json	批量删除评价	预览 编辑 更多
get	/app/visitorcontroller/getUnread.json	获取未读角标	预览 编辑 更多

符合工程质量的REST服务-- 数据字典

定义

ID	CODE	NAME	LONG_DESC
1	APP_MARKET	应用市场名称	(Null)
2	APP_PLATFORM_TYP	用户访问渠道	处理accesslog的任务中根
3	APP_VERSION_TYPE	APP应用版本类型	定义该版本是最新版本还是
4	BAT_LOG_LEVEL	批处理日志级别	(Null)
5	BAT_TASK_STATE	批处理任务状态	(Null)

ID	MODULE_CODE	CODE	VALUE	EXTEND	IMG
205	WITHDRAW_TYPIW		withlocals	(Null)	(Null)
206	APP_MARKET	bd	百度助手	(Null)	(Null)
207	APP_MARKET	yb	应用宝	(Null)	(Null)
208	APP_MARKET	hw	华为	(Null)	(Null)
209	APP_MARKET	xm	小米	(Null)	(Null)
210	APP_MARKET	sl	360	(Null)	(Null)
211	APP_MARKET	ios	IOS	(Null)	(Null)

符合工程质量的REST服务-- 数据字典

定义

ID	CODE	NAME	LONG_DESC
1	APP_MARKET	应用市场名称	(Null)
2	APP_PLATFORM_TYP	用户访问渠道	处理accesslog的任务中根
3	APP_VERSION_TYPE	APP应用版本类型	定义该版本是最新版本还是
4	BAT_LOG_LEVEL	批处理日志级别	(Null)
5	BAT_TASK_STATE	批处理任务状态	(Null)

ID	MODULE_CODE	CODE	VALUE	EXTEND	IMG
205	WITHDRAW_TYPIW		withlocals	(Null)	(Null)
206	APP_MARKET	bd	百度助手	(Null)	(Null)
207	APP_MARKET	yb	应用宝	(Null)	(Null)
208	APP_MARKET	hw	华为	(Null)	(Null)
209	APP_MARKET	xm	小米	(Null)	(Null)
210	APP_MARKET	sl	360	(Null)	(Null)
211	APP_MARKET	ios	IOS	(Null)	(Null)

符合工程质量的REST服务-- 数据字典

代码生成和使用

```
task buildDictionary(type:JavaExec,dependsOn:[classes]) {
    classpath sourceSets.main.runtimeClasspath,sourceSets.test.runtimeClasspath
    jvmArgs '-Dfile.encoding=UTF-8'
    main 'cn.datek.earth.dictionary.tools.GenerateDictionaryContants'
    args 'cn.datek.earth.batch.constants','APP_MARKET|PRD_TYPE',false
}
```

```
public final class Modules {
    /**
     *
     */
    /**/
    public static final String PRD_TYPE = "PRD_TYPE";
    /**
     * 应用市场名称
     */
    /**/
    public static final String APP_MARKET = "APP_MARKET";
}
```

```
@RequestMapping("/genders")
public List<Dictionary> genders() {
    return dict.list(Modules.GENDER); //列出某个字典大类的所有字典项
}

@RequestMapping("/")
public User findUserByGender(@RequestParam("gender") String gender) {

    GENDER.check(gender); //字典类型的单值检查
    User user = getUser("1111");
    user.setGender(gender);
    user.setGender(GENDER.M); //字典类型赋值

    return user;
}
```

```
public final class APP_MARKET {

    /** 百度助手 */
    public static final String bd = "bd";

    /** 应用宝 */
    public static final String yb = "yb";

    /** 华为 */
    public static final String hw = "hw";

    /** 小米 */
    public static final String xm = "xm";

    /** 360 */
    public static final String sl = "sl";

    /** IOS */
    public static final String ios = "ios";

    /** 蓝色行星 */
    public static final String bp = "bp";

    /** 微信 */
    public static final String bpw = "bpw";

    /** m站 */
    public static final String bpm = "bpm";

    /** 豌豆荚 */
    public static final String wd = "wd";

    /** 字典code列表 */
    private static final String[] codes = {"bd","yb","hw","xm","sl","ios","bp","bpw","bpm","wd"};

    /** 校验APP_MARKET中是否存在指定的字典项 */
    public static void check(String code,ErrorCode ex, Object...args) {
```

符合工程质量的REST服务-- 数据字典

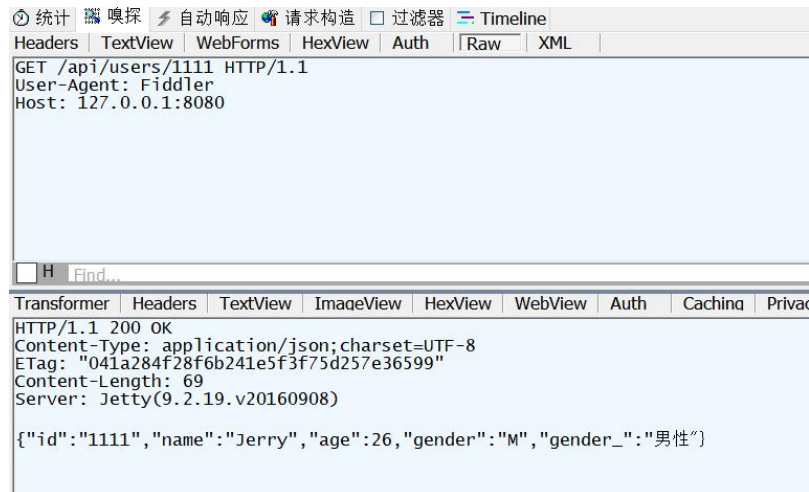
数据渲染：静态渲染 vs 动态渲染

```
@Data
public class User {
    private String id;
    private String name;
    private int age;
    private String gender;
    @DictionaryView(Modules.GENDER)
    private String gender_;
}
```

```
private User getUser(String id) {
    User user = new User();
    user.setId(id);
    user.setAge(26);
    user.setName("Jerry");
    user.setGender("M");

    DictionaryRender.render(user);

    return user;
}
```



```
@RequestMapping("/{userId}")
public User get(@PathVariable("userId") String userId) {

    final User user = getUser(userId);

    return json.use(
        JSon.with(user, helper).match(
            Match.on(User.class)
                .code("gender", Modules.GENDER) //声明User对象中的gender属性的字典类型
        )
    ).val();

    //return user;
}
```

符合工程质量的REST服务 -- 数据传输对象

封闭领域模型风格 Dozer vs MapStruct

手工编码的局限性

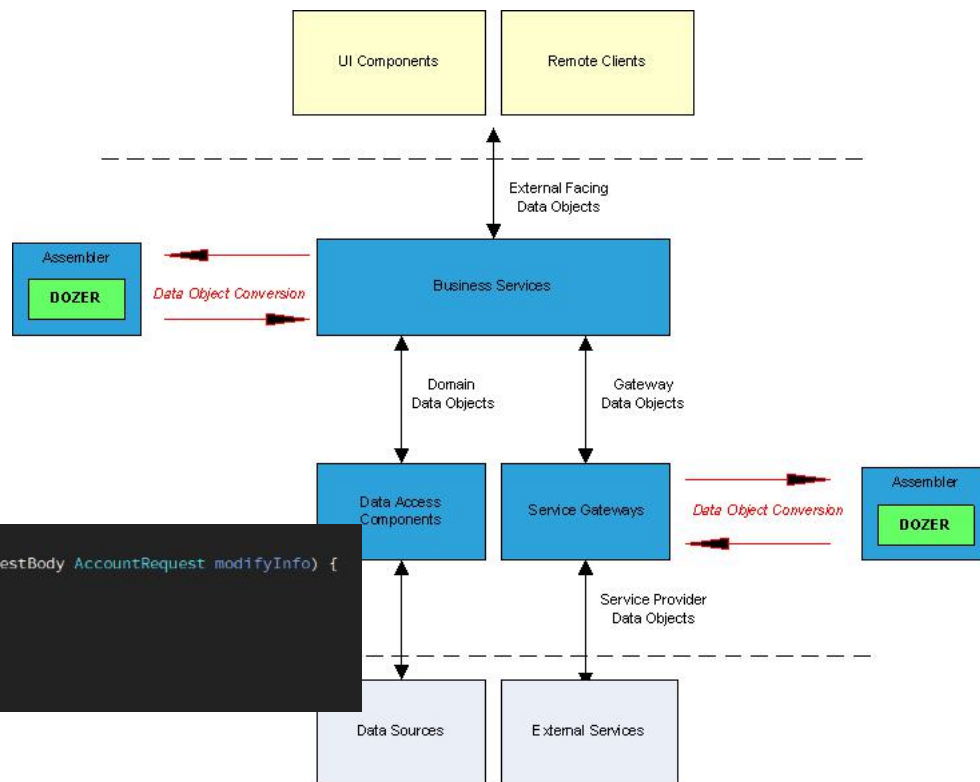
```
@Mapper
public interface UsersMapper {

    UsersMapper INST = Mappers.getMapper(UsersMapper.class);

    @Mappings({
        @Mapping(source="userName",target="name"),
        @Mapping(source="userAge",target="age"),
        @Mapping(source="userGender",target="gender")
    })
    public User Req2MainAccount(AccountRequest request);
}
```

```
@RequestMapping(value="/{userId}/modify",method=RequestMethod.POST)
public ResponseEntity<User> modify(@PathVariable("userId") String userId, @RequestBody AccountRequest modifyInfo) {

    User user = UsersMapper.INST.Req2MainAccount(modifyInfo);
    user.setId(userId);
    return new ResponseEntity<>(user,HttpStatus.OK);
}
```



符合工程质量的REST服务 -- 数据传输对象

开放领域模型风格

```
@RequestMapping("/create")
public User create(@RequestBody User user) {
    Checks.notNull(user.getName());
    user.setId("111111111111");

    return json.use(
        JSon.with(user, helper).match(
            Match.on(User.class)
                .exclude("*")
                .include("id", "name")
        )
    ).val();
}
//return user;
}
```

统计 嗅探 自动响应 请求构造 过滤器 Timeline

Headers | TextView | WebForms | HexView | Auth | Raw | XML

POST /api/users/create HTTP/1.1
Content-Type: application/json
Host: 127.0.0.1:8080
Content-Length: 37

{"name": "Gary", "age": 22, "gender": "M"}

H Find...

Transformer | Headers | TextView | ImageView | HexView | WebView | Auth |

HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: 36
Server: Jetty(9.2.19.v20160908)

{"id": "111111111111", "name": "Gary"}

符合工程质量的REST服务 -- 单值校验

单值校验 vs 业务逻辑校验

Checks风格

```
@ApiOperation("快捷开户")
@RequestMapping(value = "/fastOpenAccount", method = RequestMethod.POST)
public Response<QuickOpenAccountResponse> quickOpenAccount(@RequestBody QuickOpenAccountRequest quickOpenAccountRequest,
    @PathVariable("source") String source) {

    Track.request("快捷开户请求:{},{}", quickOpenAccountRequest, source);

    //校验
    CERTIFICATE_TYPE.check(quickOpenAccountRequest.getCertificateType()); // 证件类型
    Checks.idCard(quickOpenAccountRequest.getCertificateNo()); // 证件卡号
    Checks.notNull(quickOpenAccountRequest.getInvestorName()); // 投资人姓名
    BANK_CODE.check(quickOpenAccountRequest.getBankCode()); // bankName不验, 以bankCode为准
    Checks.phone(quickOpenAccountRequest.getInvestorMobileNo()); // 投资人手机号
    Checks.phone(quickOpenAccountRequest.getMobileTelNo()); // 银行预留手机号
    Checks.notNull(quickOpenAccountRequest.getAccountId()); // 银行卡号
    Checks.notNull(quickOpenAccountRequest.getOpenBank()); // 开户行号, 联行号
    Checks.notNull(quickOpenAccountRequest.getBankName()); // 开户行名称
    PROVINCE_CODE.check(quickOpenAccountRequest.getProvinceCode()); // 开户行省
    // customerName不验, 用investorName
    Checks.notNull(quickOpenAccountRequest.getCityNo()); // TODO 无city字典源
    // cityName不验, 以cityNo为准
    // Checks.notNull(quickOpenAccountRequest.getCustomRiskLevel()); // TODO 风险等级, 目前只使用分数, 不使用合作机构传入的等级
    Checks.digit(quickOpenAccountRequest.getTestScore()); // 风险评测总分
    int testScore = Integer.valueOf(quickOpenAccountRequest.getTestScore());
    if (testScore < 0 || testScore > 100) {
        throw Exceptions.fail(ErrorCodes.APPLICATION_NOT_AVAILABLE);
    }

    // 判断参数是否带有流水号和验证码, 如果不带则判定为第一次请求, 发送验卡申请
    if (StringUtils.isBlank(quickOpenAccountRequest.getOriginalSeriousNo())) {
        // 开立业务系统主账户
        MainAccount mainAccount = new MainAccount();
        mainAccount.setOrgId(source);
        mainAccount.setMobilePhone(quickOpenAccountRequest.getMobileTelNo());
        mainAccount.setInvestorName(quickOpenAccountRequest.getInvestorName());
        mainAccount.setCertificateType(quickOpenAccountRequest.getCertificateType());
        mainAccount.setCertificateNo(quickOpenAccountRequest.getCertificateNo());
        mainAccount.setInvestorType(INVESTOR_TYPE_1); // 目前仅支持个人
        mainAccount.setTradePassword(quickOpenAccountRequest.getPassword());

        //-----MapStruct风格-----

        mainAccount = QuickOpenAccountMapper.INST.Reg2MainAccount(quickOpenAccountRequest);
        mainAccount.setOrgId(source);
        mainAccount.setInvestorType(INVESTOR_TYPE_1);

        //-----
```

符合工程质量的REST服务 -- 异常包装

CodedException

便于统一管理

便于服务的消费方和调用方沟通

```
@Data
public final class ErrorCode implements Serializable {

    private static final long serialVersionUID = 7191462619159223378L;

    public static String ILLEGAL_ARGUMENT_MODULE = "ILLEGAL_ARGUMENT";

    /**
     * 模块
     */
    private String moduleCode;
    /**
     * 错误码
     */
    private String errorCode;
    /**
     * 错误信息
     */
    private String message;
```

```
@Data
public class CodedException extends RuntimeException {

    private static final long serialVersionUID = -5704728092736131237L;

    /**
     * 该异常的错误码
     */
    private ErrorCode errorCode;

    /**
     * 异常发生时的参数信息
     */
    private Object[] args;

    protected CodedException(ErrorCode errorCode, Throwable throwable, Object... arguments) {
        super(throwable);
        this.errorCode = errorCode;
        this.args = arguments;
    }

    protected CodedException(ErrorCode errorCode, Object... arguments) {
        super(errorCode.getMessage());
        this.errorCode = errorCode;
        this.args = arguments;
    }

}
```


符合工程质量的REST服务 -- 异常包装

业务异常和系统故障的区别

Http Status Code : 4xx vs 5xx

Exceptions.fail vs Exceptions.fault

异常信息的包装

CodedMappingExceptionHandler

@ControllerAdvice

符合工程质量的REST服务 -- 异常包装

```
@ControllerAdvice
public class RestResponseEntityExceptionHandler {

    @ExceptionHandler(value = { CodedException.class })
    @ResponseBody
    protected ResponseEntity<Response_<?>> handleConflict(CodedException ex, WebRequest request) {
        Response_<Object> response = new Response_<>();
        //如果是响应的错误信息，则用该信息包装response对象，否则返回未知异常
        final ErrorCode code = ex.getErrorCode();
        if(!ErrorCodes.isResponseFault(code)) {
            Track.fault(ex, ErrorCodes.UNKOWN_ERROR.getMessage());
            response.setMsg(ErrorCodes.UNKOWN_ERROR.getMessage());
            response.setFlag(Integer.valueOf(ErrorCodes.UNKOWN_ERROR.getErrorCode()));
        } else {
            Track.fail(code.getMessage());
            response.setMsg(code.getMessage());
            response.setFlag(Integer.valueOf(code.getErrorCode()));
        }
        //记录日志
        Track.response(response.toString());
        return new ResponseEntity<Response_<?>>(response, HttpStatus.BAD_REQUEST);
    }
}
```

符合工程质量的REST服务 -- 异常包装

```
1 package cn.datek.web;
2
3 import java.io.IOException;
4
27
28 /**
29  * CodedException统一处理
30  * 对所有未处理的CodedException类型的异常进行统一的包装处理
31  *     1. 对于jsp形式的调用
32  *         在Model中放入一个error的对象
33  *     2. 对于ajax方式的请求
34  *         以请求希望的格式 (JSON或者XML) 方式返回error数据
35  *
36  * @author 怀民
37  *
38  */
39 @EnableWebMvc
40 @Component("exceptionResolver")
41 public class CodedMappingExceptionHandler extends SimpleMappingExceptionHandler {
42
43     private static final Logger log = LoggerFactory.getLogger(CodedMappingExceptionHandler.class);
44
45     /**
46      * JSP模式下对于抛出CodedException并且应用层未处理时的统一错误页面
47      */
48     private static final String CODED_EXCEPTION_ERROR_PAGE = "/error/coded_error";
49
50     @Resource
51     /**
52      * @see WebConfig.objectMapper
53      */
54     ObjectMapper objectMapper;
55
56     @Resource
57     CodedMessageSource codedMessageSource;
58
59     @PostConstruct
60     public void init() {
61         Properties exceptionMappings = new Properties();
62         //在ajax方式的情况下, 将所有异常转向指定的错误页面
63         exceptionMappings.put(Exception.class.getName(), CODED_EXCEPTION_ERROR_PAGE);
64         this.setExceptionMappings(exceptionMappings);
65     }
66 }
```

符合工程质量的REST服务 -- 日志和监控

生产环境日志的监控的重要性

Track工具

```
@ApiOperation("快捷开户")
@RequestMapping(value = "/fastOpenAccount", method = RequestMethod.POST)
public Response<QuickOpenAccountResponse> quickOpenAccount(@RequestBody QuickOpenAccountRequest quickOpenAccountRequest,
    @PathVariable("source") String source) {

    Track.request("快捷开户请求:{},{}", quickOpenAccountRequest, source);
```

```
// 返回申请号等数据
QuickOpenAccountResponse quickOpenAccountResponse = new QuickOpenAccountResponse();
quickOpenAccountResponse.setClientId(mainAccount.getContractNo());
quickOpenAccountResponse.setSeriousNo(applyNo);
quickOpenAccountResponse.setSignFlag(QuickOpenAccountResponse.SIGN_FLAG_APPLIED);
Response<QuickOpenAccountResponse> response = new Response<QuickOpenAccountResponse>(quickOpenAccountResponse);

Track.response("快捷开户返回:{}", response);
return response;
}
```

符合工程质量的REST服务 -- 日志和监控

Track做了些什么事

创建不同的Logger

针对不同的场景（Request/Response、Service、DB、Gateway、Batch）

创建不同模式的度量模型（Metrics）

缓存并定期输出度量数据（ZabbixSender）

符合工程质量的REST服务 -- 日志和监控

Track做了些啥事

2017年4月12日 16:44

