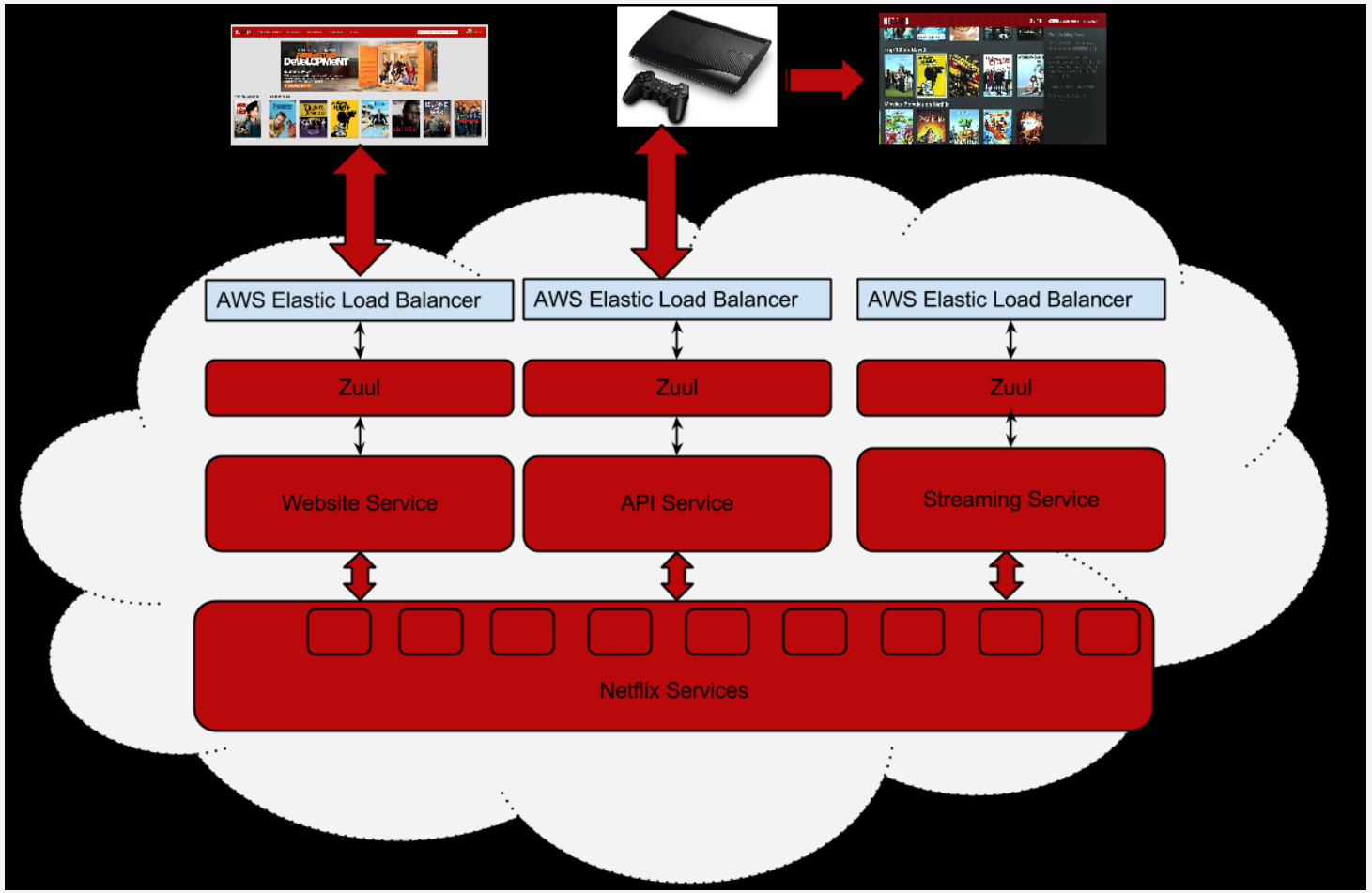
关于Zuul的经验分享和扩展思路

翟永超（程序猿DD）

Spring Cloud Netflix Zuul

Netflix Zuul



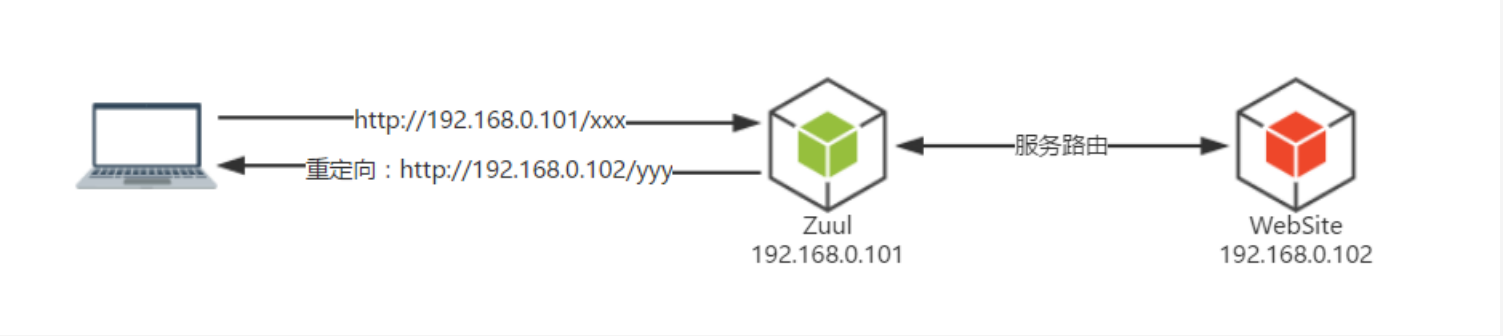
- Zuul简介
- 对接Website时的问题
- 过滤器的统一异常处理
- 自定义错误格式
- 域名路由与contextPath

对接Website时的问题

- 会话保持问题
  - 原因：头信息中的Cookie和Authorization信息没有被正确传递
  - 设置sensitiveHeaders属性
    - 全局设置：
      - zuul.sensitive-headers=
    - 指定路由设置：
      - zuul.routes.<routeName>.sensitive-headers=
      - zuul.routes.<routeName>.custom-sensitive-headers=true

- 相关过滤器：org.springframework.cloud.netflix.zuul.filters.pre.PreDecorationFilter

对接Website时的问题——重定向问题

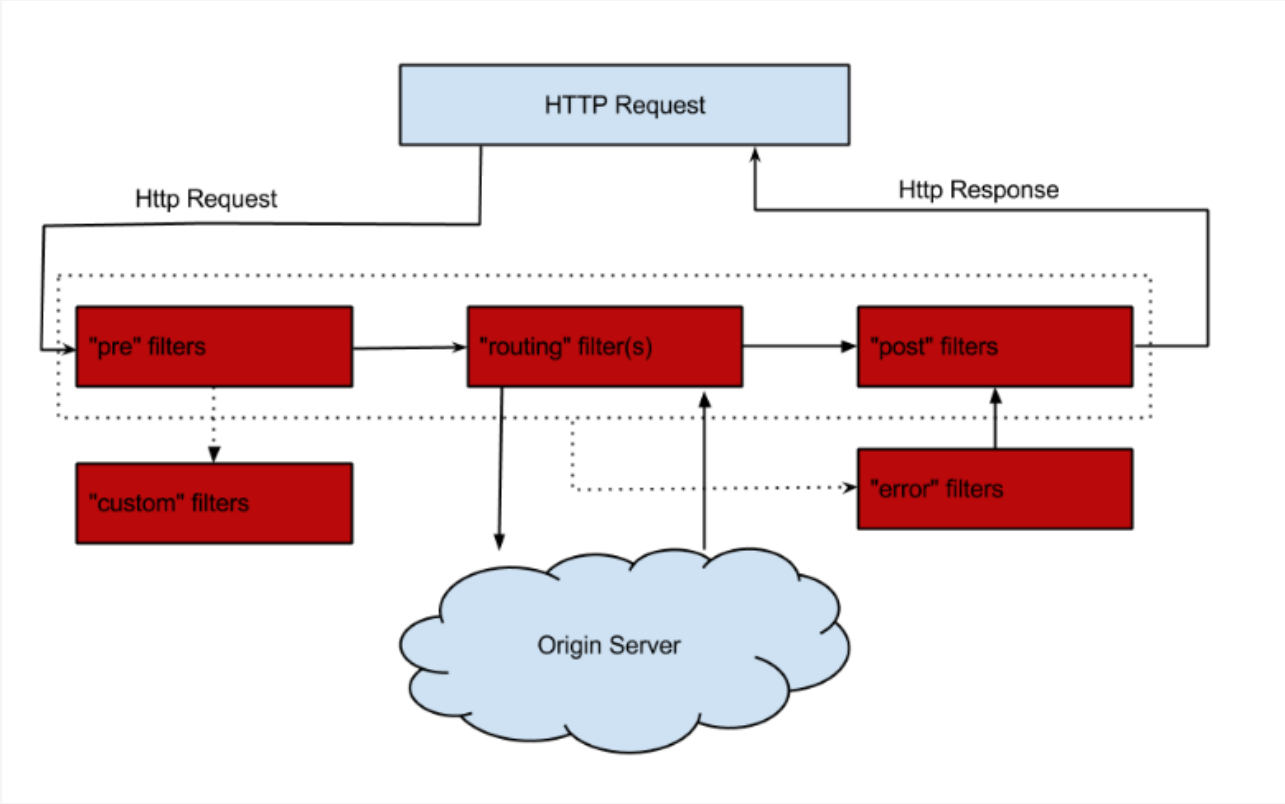


- 重定向问题
  - 原因：头信息中的host没有被正确处理
  - Brixton版本：需扩展Filter来支持
  - Camden版本：设置属性zuul.add-host-header: true

- 相关过滤器：org.springframework.cloud.netflix.zuul.filters.pre.PreDecorationFilter

过滤器的统一异常处理

- 为什么要扩展统一异常处理？
  - 当自定义过滤器中抛出异常，客户端没有任何错误信息！
- 为什么没有返回错误信息呢？

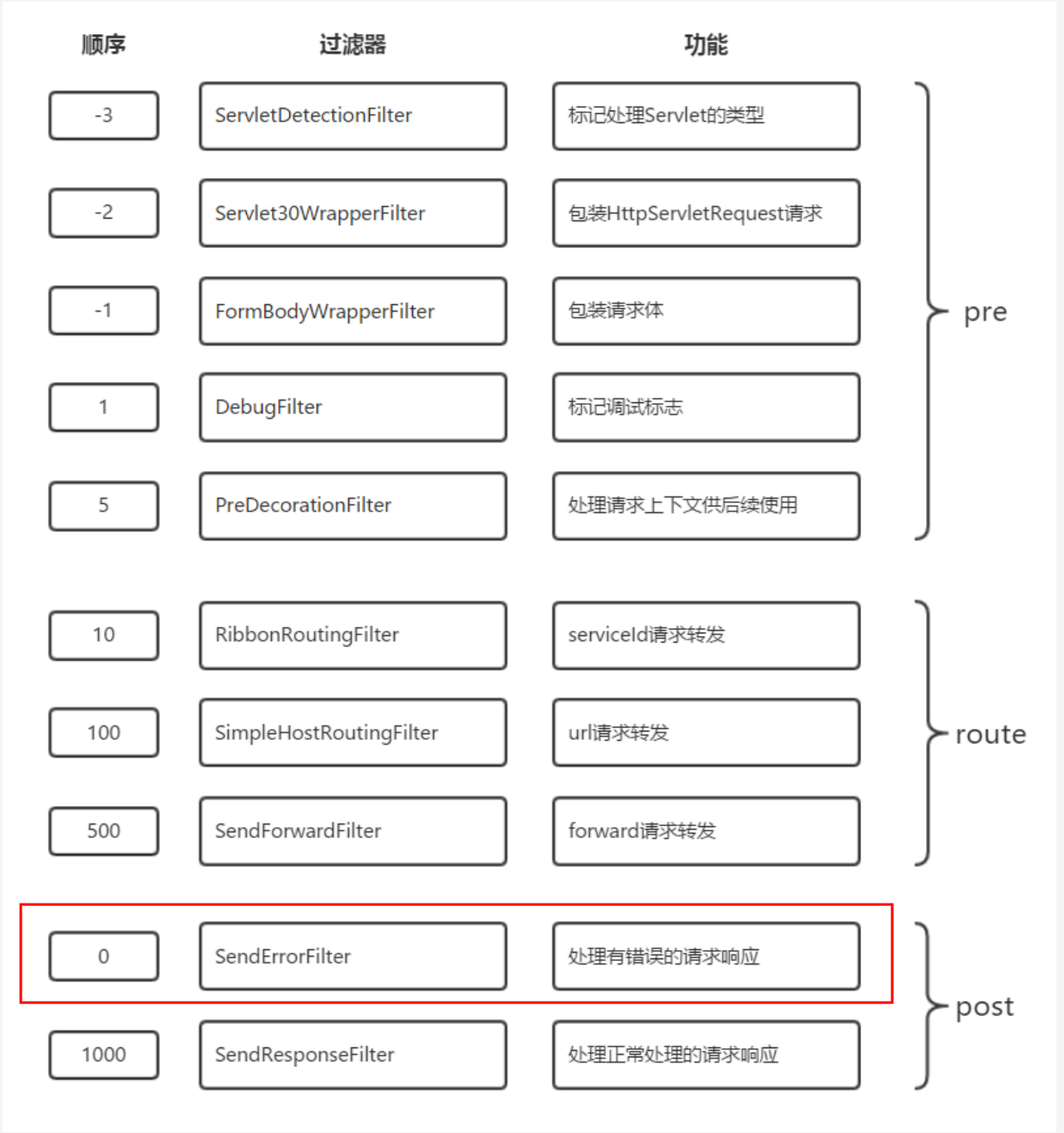


过滤器的统一异常处理

- 异常信息是如何返回？
  - SendErrorFilter

```
public boolean shouldFilter() {
    RequestContext ctx = RequestContext.getCurrentContext();
    return ctx.containsKey("error.status_code") &&
      !ctx.getBoolean(SEND_ERROR_FILTER_RAN, false);
}
```

- 异常信息的格式与传递？
  - error.status_code：错误编码
  - error.exception：Exception异常对象
  - error.message：错误信息



过滤器的统一异常处理

- 解决方法一：
  - 严格的try-catch处理
  - 在异常中按异常传递规则来组织请求上下文

```
public Object run() {
    RequestContext context = RequestContext.getCurrentContext();
    this.helper.addIgnoredHeaders();
    try {
        RibbonCommandContext commandContext = buildCommandContext(context);
        ClientHttpResponse response = forward(commandContext);
        setResponse(response);
        return response;
    }
    catch (ZuulException ex) {
        context.set(ERROR_STATUS_CODE, ex.nStatusCode);
        context.set("error.message", ex.errorCause);
        context.set("error.exception", ex);
    }
    catch (Exception ex) {
        context.set("error.status_code", HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        context.set("error.exception", ex);
    }
    return null;
}
```

过滤器的统一异常处理

- 解决方法二：
  - 由于任何阶段的异常都会被error阶段的过滤器处理，所以我们可以通过它来统一处理异常信息

```
public class ErrorFilter extends ZuulFilter {

    ...

    @Override
    public Object run() {
        RequestContext ctx = RequestContext.getCurrentContext();
        Throwable throwable = ctx.getThrowable();
        log.error("this is a ErrorFilter : {}", throwable.getCause().getMessage());
        ctx.set("error.status_code", HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        ctx.set("error.exception", throwable.getCause());
        return null;
    }

}
```

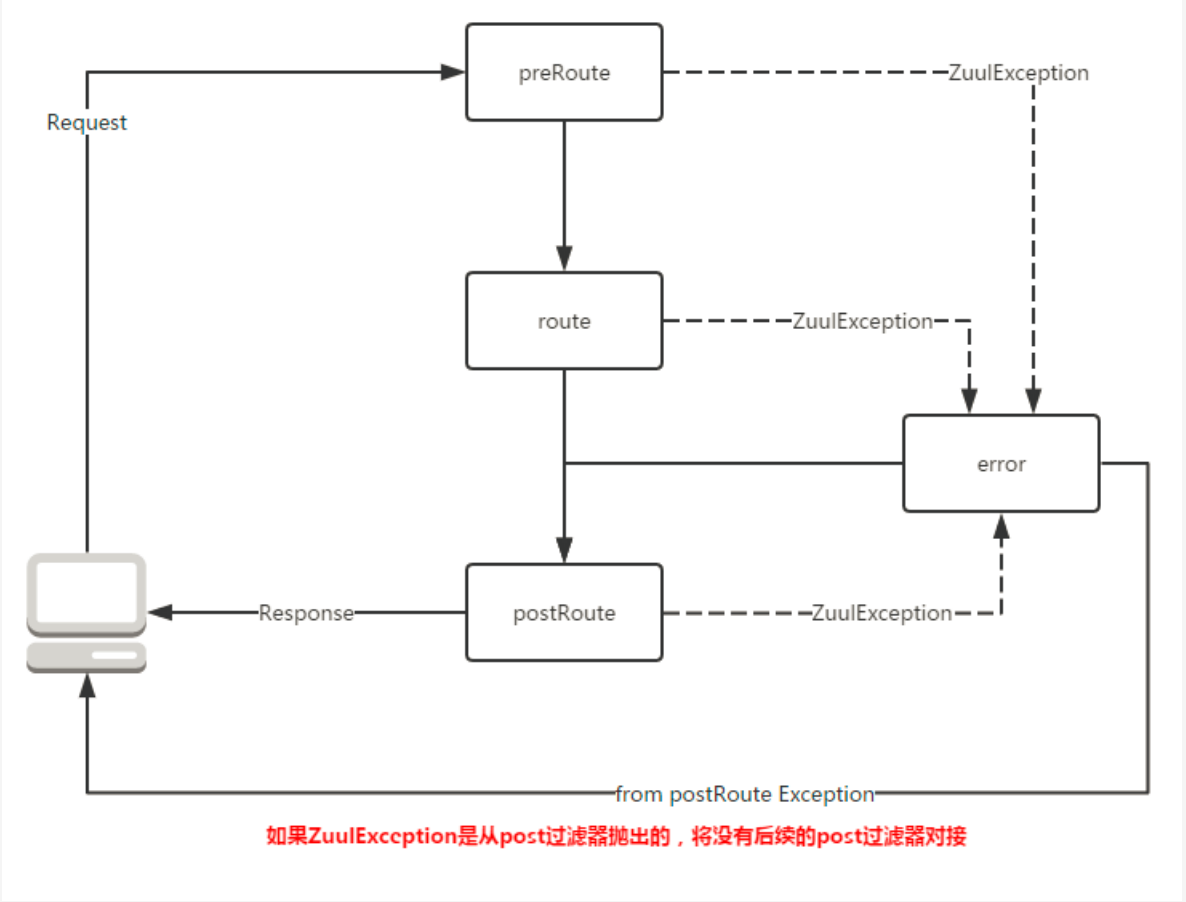
过滤器的统一异常处理

- 新的问题！
  - post类型的过滤器抛出异常时，任何没有返回错误信息！

```
@Override
public void service(javax.servlet.ServletRequest servlet
    try {
        init((HttpServletRequest) servletRequest, (HttpS

        // Marks this request as having passed through t
        // explicitly bound in web.xml, for which reques
        RequestContext context = RequestContext.getCurre
        context.setZuulEngineRan();

        try {
            preRoute();
        } catch (ZuulException e) {
            error(e);
            postRoute();
            return;
        }
        try {
            route();
        } catch (ZuulException e) {
            error(e);
            postRoute();
            return;
        }
        try {
            postRoute();
        } catch (ZuulException e) {
            error(e);
            return;
        }

    } catch (Throwable e) {
        error(new ZuulException(e, 500, "UNHANDLED_EXCEP
    } finally {
        RequestContext.getCurrentContext().unset();
    }
}
```

过滤器的统一异常处理

- 原因：
  - pre、route阶段的error处理之后都继续被post过滤器处理，SendErrorFilter可以将组织好的错误信息进行输出，但是post阶段的异常被error过滤器处理后，就不会再被post过滤器处理



过滤器的统一异常处理

- 进一步优化
  - 增加error过滤器对错误信息的返回
  - 通过继承SendErrorFilter来复用输出逻辑
- 新的问题！
  - 如何判断是来自post过滤器的异常呢？

```java
@Component
public class ErrorExtFilter extends SendErrorFilter {

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

    @Override
    public int filterOrder() {
        return 30;  // 大于ErrorFilter的值
    }

    @Override
    public boolean shouldFilter() {
        // TODO 判断：仅处理来自post过滤器引起的异常
        return true;
    }

}
```

过滤器的统一异常处理

- 扩展FilterProcessor
  - 当过滤器执行发生异常时，在请求上下文中传递出错的过滤器，让Error过滤器可以获取到出现异常的过滤器。

```
public class DidiFilterProcessor extends FilterProcessor {

    @Override
    public Object processZuulFilter(ZuulFilter filter) throws ZuulException {
        try {
            return super.processZuulFilter(filter);
        } catch (ZuulException e) {
            RequestContext ctx = RequestContext.getCurrentContext();
            ctx.set("failed.filter", filter);
            throw e;
        }
    }

}
```

```
@Override
public boolean shouldFilter() {
    // 判断：仅处理来自post过滤器引起的异常
    RequestContext ctx = RequestContext.getCurrentContext();
    ZuulFilter failedFilter = (ZuulFilter) ctx.get("failed.filter");
    if(failedFilter != null && failedFilter.filterType().equals("post")) {
        return true;
    }
    return false;
}
```

自定义错误格式

```
{
  "timestamp": 1481674980376,
  "status": 500,
  "error": "Internal Server Error",
  "exception": "java.lang.RuntimeException",
  "message": "Exist some errors..."
}
```

- 方法一：重写post阶段的过滤器，返回不同的内容（不推荐）
- 方法二：扩展DefaultErrorAttributes的实现（/error端点）

```
public class DidiErrorAttributes extends DefaultErrorAttributes {


    @Override
    public Map<String, Object> getErrorAttributes (
            RequestAttributes requestAttributes, boolean includeStackTrace){
        Map<String, Object> result = super.getErrorAttributes(requestAttributes, includeStackTrace);
        result.remove("exception");
        return result;
    }


}
```

域名路由与ContextPath支持

- 推进Zuul时的困难
  - 前提：小范围试错，灰度上线，兼容上面这个各种花样的路由规则
  - 困难：各种各样不可预测的路由规则
    - http://service-a.xxx.com/ ==> http://service-a/
    - http://www.xxx.com/service-b ==> http://service-b/
    - http://www.xxx.com/service-c ==> http://service-c/rest/
    - http://www.yyy.com/fuck-pm ==> http://service-d/

域名路由与ContextPath支持

- 域名路由改造思路：
  - 前提：沿用Zuul的路由匹配原则（根据请求路径的前缀/<prefix>/**来匹配服务）
  - 改造：
    - 在路由匹配之前，根据域名在请求路径前增加/<prefix>前缀
    - 增加根路径的路由配置以及其他重要配置

域名路由与ContextPath支持

- 域名路由改造实现：在路由匹配之前，根据域名在请求路径前增加/<prefix>前缀

```
class HostRouteFilter extends ZuulFilter {
    ...
    // 域名与路由前缀的映射关系
    def hostRoutePrefixMap = [
        "www.xxx.com" : "service-a"
    ]

    @Override
    Object run() {
        RequestContext ctx = RequestContext.getCurrentContext()
        HttpServletRequest request = ctx.getRequest()
        String host = request.getHeader("host")
        String routePrefix = hostRoutePrefixMap.get(host)
        if(routePrefix != null) {
            ctx.put("host_route_prefix",routePrefix)
        }
        return null
    }
}
```

```
public class DidiRouteLocator extends DiscoveryClientRouteLocator  {
    ...
    @Override
    public Route getMatchingRoute(String path) {
        RequestContext ctx = RequestContext.getCurrentContext();
        String routePrefix = (String) ctx.get("host_route_prefix");
        if(routePrefix != null) {
            // 需要根据域名来添加前缀，该前缀将用户后续的路由匹配
            path = "/" + routePrefix + path;
        }
        return super.getMatchingRoute(path);
    }
}
```

域名路由与ContextPath支持

- 域名路由改造实现：增加根路径的路由配置以及其他重要配置

```
zuul:
  routes:
    service-a:
      path: /service-a
      serviceId: service-a
    default:
      path: /**
      url: forward:/
  ignoredPatterns: /health,/resume,/heapdump,/mappings,/autoconfig,/info,
                   /metrics,/metrics/**,/hystrix.stream,/hystrix.stream/**,
                   /configprops,/beans,/trace,/error,/features,/routes,
                   /dump,/refresh,/pause,/consul,/logfile,/archaius,
                   /env,/env/**
```

域名路由与ContextPath支持

- ContextPath支持的实现：

```
class ContextPathFilter extends ZuulFilter {


    def serviceContextPathMap = [
            "b2c-platform" : "/b2c"
    ]


    @Override
    Object run() {
        RequestContext ctx = RequestContext.getCurrentContext()
        String proxy = ctx.get("proxy")
        String contextPath = serviceContextPathMap.get(proxy)
        if(contextPath != null) {
            ctx.put("requestURI", contextPath + ctx.get("requestURI"))
        }
        return null
    }
}
```

域名路由与ContextPath支持

- 改造后的过滤器处理链：

域名路由与ContextPath支持

- 更友好的实现：

```
zuul:
  routes:
    service-a:
      host: www.xxx.com
      path: /service-a
      serviceId: service-a
      contextPath: /rest
```

这里太小了，具体实现...待续！
欢迎关注！感谢支持！