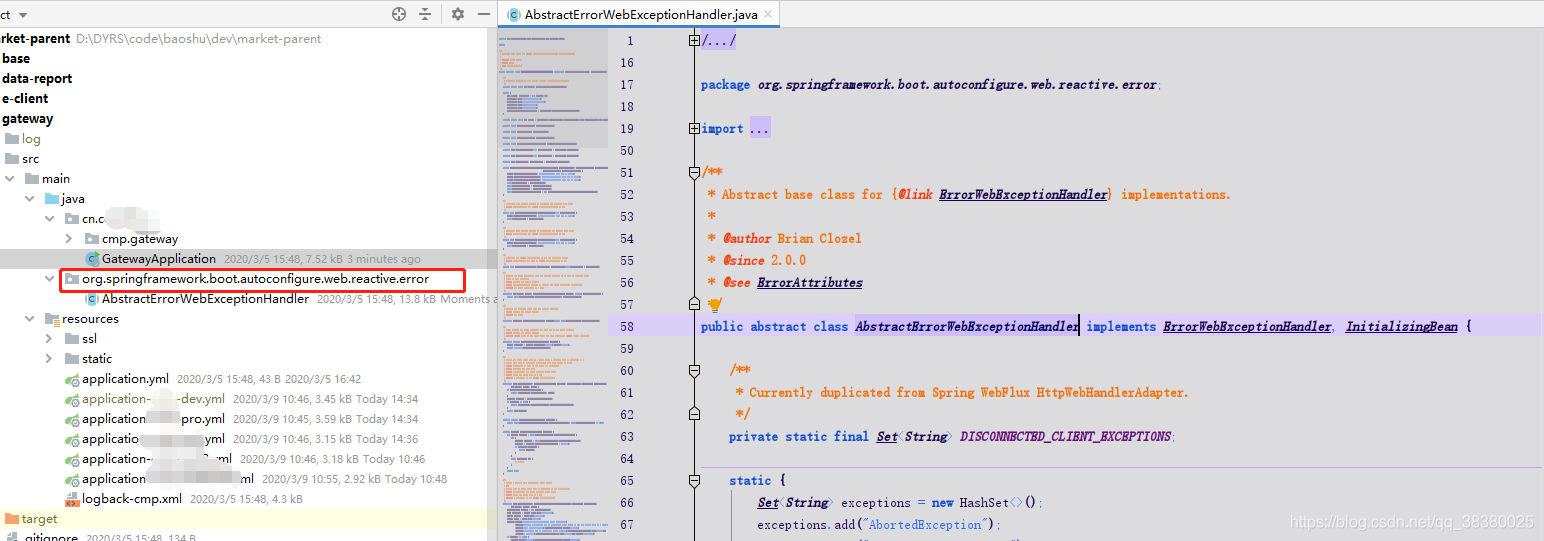
AbstractErrorWebExceptionHandler 找到这个class 拿出来

并创建一个 一模一样的包放到里面， 因为是就近原则所有会走后提出来的文件



org.springframework.boot.autoconfigure.web.reactive.error

在handle 方法中进行添加状态的拦截校验，朋友们可以根据自己需求进行添加修改，



完整类

package org.springframework.boot.autoconfigure.web.reactive.error;/\*

\* Copyright 2012-2019 the original author or authors.

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* https://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

import com.alibaba.fastjson.JSONObject;

import lombok.extern.slf4j.Slf4j;

import org.apache.commons.logging.Log;

import org.springframework.beans.factory.InitializingBean;

import org.springframework.boot.autoconfigure.template.TemplateAvailabilityProviders;

import org.springframework.boot.autoconfigure.web.ResourceProperties;

import org.springframework.boot.web.reactive.error.ErrorAttributes;

import org.springframework.boot.web.reactive.error.ErrorWebExceptionHandler;

import org.springframework.context.ApplicationContext;

import org.springframework.core.NestedExceptionUtils;

import org.springframework.core.io.Resource;

import org.springframework.core.io.buffer.DataBuffer;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpLogging;

import org.springframework.http.HttpStatus;

import org.springframework.http.codec.HttpMessageReader;

import org.springframework.http.codec.HttpMessageWriter;

import org.springframework.http.server.reactive.ServerHttpResponse;

import org.springframework.util.Assert;

import org.springframework.util.CollectionUtils;

import org.springframework.util.StringUtils;

import org.springframework.web.reactive.function.BodyInserters;

import org.springframework.web.reactive.function.server.RouterFunction;

import org.springframework.web.reactive.function.server.ServerRequest;

import org.springframework.web.reactive.function.server.ServerResponse;

import org.springframework.web.reactive.result.view.ViewResolver;

import org.springframework.web.server.ResponseStatusException;

import org.springframework.web.server.ServerWebExchange;

import org.springframework.web.util.HtmlUtils;

import reactor.core.publisher.Mono;

import java.nio.charset.StandardCharsets;

import java.util.\*;

/\*\*

\* Abstract base class for {@link ErrorWebExceptionHandler} implementations.

\*

\* @author Brian Clozel

\* @since 2.0.0

\* @see ErrorAttributes

\*/

@Slf4j

public abstract class AbstractErrorWebExceptionHandler implements ErrorWebExceptionHandler, InitializingBean {

/\*\*

\* Currently duplicated from Spring WebFlux HttpWebHandlerAdapter.

\*/

private static final Set<String> DISCONNECTED\_CLIENT\_EXCEPTIONS;

static {

Set<String> exceptions = new HashSet<>();

exceptions.add("AbortedException");

exceptions.add("ClientAbortException");

exceptions.add("EOFException");

exceptions.add("EofException");

DISCONNECTED\_CLIENT\_EXCEPTIONS = Collections.unmodifiableSet(exceptions);

}

private static final Log logger = HttpLogging.forLogName(org.springframework.boot.autoconfigure.web.reactive.error.AbstractErrorWebExceptionHandler.class);

private final ApplicationContext applicationContext;

private final ErrorAttributes errorAttributes;

private final ResourceProperties resourceProperties;

private final TemplateAvailabilityProviders templateAvailabilityProviders;

private List<HttpMessageReader<?>> messageReaders = Collections.emptyList();

private List<HttpMessageWriter<?>> messageWriters = Collections.emptyList();

private List<ViewResolver> viewResolvers = Collections.emptyList();

public AbstractErrorWebExceptionHandler(ErrorAttributes errorAttributes, ResourceProperties resourceProperties,

ApplicationContext applicationContext) {

Assert.notNull(errorAttributes, "ErrorAttributes must not be null");

Assert.notNull(resourceProperties, "ResourceProperties must not be null");

Assert.notNull(applicationContext, "ApplicationContext must not be null");

this.errorAttributes = errorAttributes;

this.resourceProperties = resourceProperties;

this.applicationContext = applicationContext;

this.templateAvailabilityProviders = new TemplateAvailabilityProviders(applicationContext);

}

/\*\*

\* Configure HTTP message writers to serialize the response body with.

\* @param messageWriters the {@link HttpMessageWriter}s to use

\*/

public void setMessageWriters(List<HttpMessageWriter<?>> messageWriters) {

Assert.notNull(messageWriters, "'messageWriters' must not be null");

this.messageWriters = messageWriters;

}

/\*\*

\* Configure HTTP message readers to deserialize the request body with.

\* @param messageReaders the {@link HttpMessageReader}s to use

\*/

public void setMessageReaders(List<HttpMessageReader<?>> messageReaders) {

Assert.notNull(messageReaders, "'messageReaders' must not be null");

this.messageReaders = messageReaders;

}

/\*\*

\* Configure the {@link ViewResolver} to use for rendering views.

\* @param viewResolvers the list of {@link ViewResolver}s to use

\*/

public void setViewResolvers(List<ViewResolver> viewResolvers) {

this.viewResolvers = viewResolvers;

}

/\*\*

\* Extract the error attributes from the current request, to be used to populate error

\* views or JSON payloads.

\* @param request the source request

\* @param includeStackTrace whether to include the error stacktrace information

\* @return the error attributes as a Map.

\*/

protected Map<String, Object> getErrorAttributes(ServerRequest request, boolean includeStackTrace) {

return this.errorAttributes.getErrorAttributes(request, includeStackTrace);

}

/\*\*

\* Extract the original error from the current request.

\* @param request the source request

\* @return the error

\*/

protected Throwable getError(ServerRequest request) {

return this.errorAttributes.getError(request);

}

/\*\*

\* Check whether the trace attribute has been set on the given request.

\* @param request the source request

\* @return {@code true} if the error trace has been requested, {@code false} otherwise

\*/

protected boolean isTraceEnabled(ServerRequest request) {

String parameter = request.queryParam("trace").orElse("false");

return !"false".equalsIgnoreCase(parameter);

}

/\*\*

\* Render the given error data as a view, using a template view if available or a

\* static HTML file if available otherwise. This will return an empty

\* {@code Publisher} if none of the above are available.

\* @param viewName the view name

\* @param responseBody the error response being built

\* @param error the error data as a map

\* @return a Publisher of the {@link ServerResponse}

\*/

protected Mono<ServerResponse> renderErrorView(String viewName, ServerResponse.BodyBuilder responseBody,

Map<String, Object> error) {

if (isTemplateAvailable(viewName)) {

return responseBody.render(viewName, error);

}

Resource resource = resolveResource(viewName);

if (resource != null) {

return responseBody.body(BodyInserters.fromResource(resource));

}

return Mono.empty();

}

private boolean isTemplateAvailable(String viewName) {

return this.templateAvailabilityProviders.getProvider(viewName, this.applicationContext) != null;

}

private Resource resolveResource(String viewName) {

for (String location : this.resourceProperties.getStaticLocations()) {

try {

Resource resource = this.applicationContext.getResource(location);

resource = resource.createRelative(viewName + ".html");

if (resource.exists()) {

return resource;

}

}

catch (Exception ex) {

// Ignore

}

}

return null;

}

/\*\*

\* Render a default HTML "Whitelabel Error Page".

\* <p>

\* Useful when no other error view is available in the application.

\* @param responseBody the error response being built

\* @param error the error data as a map

\* @return a Publisher of the {@link ServerResponse}

\*/

protected Mono<ServerResponse> renderDefaultErrorView(ServerResponse.BodyBuilder responseBody,

Map<String, Object> error) {

StringBuilder builder = new StringBuilder();

Date timestamp = (Date) error.get("timestamp");

Object message = error.get("message");

Object trace = error.get("trace");

builder.append("<html><body><h1>Whitelabel Error Page</h1>")

.append("<p>This application has no configured error view, so you are seeing this as a fallback.</p>")

.append("<div id='created'>").append(timestamp).append("</div>")

.append("<div>There was an unexpected error (type=").append(htmlEscape(error.get("error")))

.append(", status=").append(htmlEscape(error.get("status"))).append(").</div>");

if (message != null) {

builder.append("<div>").append(htmlEscape(message)).append("</div>");

}

if (trace != null) {

builder.append("<div style='white-space:pre-wrap;'>").append(htmlEscape(trace)).append("</div>");

}

builder.append("</body></html>");

return responseBody.syncBody(builder.toString());

}

private String htmlEscape(Object input) {

return (input != null) ? HtmlUtils.htmlEscape(input.toString()) : null;

}

@Override

public void afterPropertiesSet() throws Exception {

if (CollectionUtils.isEmpty(this.messageWriters)) {

throw new IllegalArgumentException("Property 'messageWriters' is required");

}

}

/\*\*

\* Create a {@link RouterFunction} that can route and handle errors as JSON responses

\* or HTML views.

\* <p>

\* If the returned {@link RouterFunction} doesn't route to a {@code HandlerFunction},

\* the original exception is propagated in the pipeline and can be processed by other

\* {@link org.springframework.web.server.WebExceptionHandler}s.

\* @param errorAttributes the {@code ErrorAttributes} instance to use to extract error

\* information

\* @return a {@link RouterFunction} that routes and handles errors

\*/

protected abstract RouterFunction<ServerResponse> getRoutingFunction(ErrorAttributes errorAttributes);

@Override

public Mono<Void> handle(ServerWebExchange exchange, Throwable throwable) {

if (exchange.getResponse().isCommitted() || isDisconnectedClientError(throwable)) {

return Mono.error(throwable);

}

this.errorAttributes.storeErrorInformation(throwable, exchange);

ServerRequest request = ServerRequest.create(exchange, this.messageReaders);

if (throwable instanceof ResponseStatusException){

HttpStatus status = ((ResponseStatusException) throwable).getStatus();

if (HttpStatus.NOT\_FOUND.equals(status)) {

// TODO 处理你自己的业务

// TODO 处理你自己的业务

}

}

return getRoutingFunction(this.errorAttributes).route(request).switchIfEmpty(Mono.error(throwable))

.flatMap((handler) -> handler.handle(request))

.doOnNext((response) -> logError(request, response, throwable))

.flatMap((response) -> write(exchange, response));

}

private boolean isDisconnectedClientError(Throwable ex) {

return DISCONNECTED\_CLIENT\_EXCEPTIONS.contains(ex.getClass().getSimpleName())

|| isDisconnectedClientErrorMessage(NestedExceptionUtils.getMostSpecificCause(ex).getMessage());

}

private boolean isDisconnectedClientErrorMessage(String message) {

message = (message != null) ? message.toLowerCase() : "";

return (message.contains("broken pipe") || message.contains("connection reset by peer"));

}

private void logError(ServerRequest request, ServerResponse response, Throwable throwable) {

if (logger.isDebugEnabled()) {

logger.debug(request.exchange().getLogPrefix() + formatError(throwable, request));

}

if (response.statusCode().equals(HttpStatus.INTERNAL\_SERVER\_ERROR)) {

logger.error(request.exchange().getLogPrefix() + "500 Server Error for " + formatRequest(request),

throwable);

}

}

private String formatError(Throwable ex, ServerRequest request) {

String reason = ex.getClass().getSimpleName() + ": " + ex.getMessage();

return "Resolved [" + reason + "] for HTTP " + request.methodName() + " " + request.path();

}

private String formatRequest(ServerRequest request) {

String rawQuery = request.uri().getRawQuery();

String query = StringUtils.hasText(rawQuery) ? "?" + rawQuery : "";

return "HTTP " + request.methodName() + " \"" + request.path() + query + "\"";

}

private Mono<? extends Void> write(ServerWebExchange exchange, ServerResponse response) {

// force content-type since writeTo won't overwrite response header values

exchange.getResponse().getHeaders().setContentType(response.headers().getContentType());

return response.writeTo(exchange, new ResponseContext());

}

private class ResponseContext implements ServerResponse.Context {

@Override

public List<HttpMessageWriter<?>> messageWriters() {

return org.springframework.boot.autoconfigure.web.reactive.error.AbstractErrorWebExceptionHandler.this.messageWriters;

}

@Override

public List<ViewResolver> viewResolvers() {

return org.springframework.boot.autoconfigure.web.reactive.error.AbstractErrorWebExceptionHandler.this.viewResolvers;

}

}

}