异常与事务以及故障转移和灰度发布

当不满足条件的时候，应该建议直接返回，还是抛出异常?

例如为null的时候

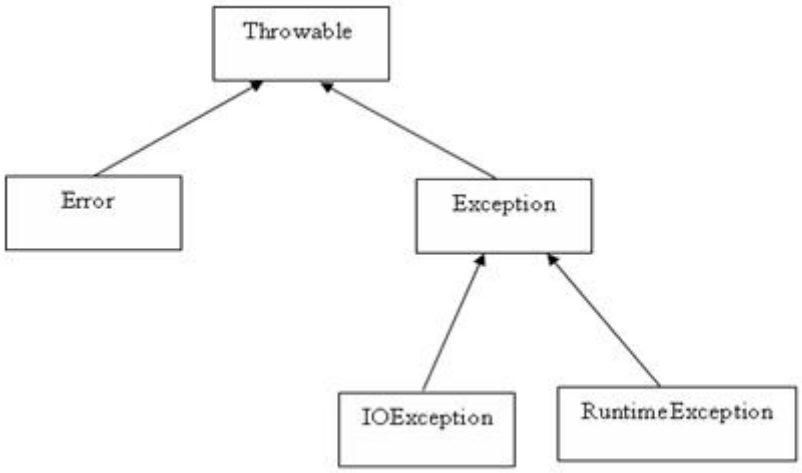
错误:

java.lang.Error 一般java虚拟机处理不了，程序不会从错误中恢复。

异常:

运行时异常 RuntimeException

检查性异常 打开一个不存在的文件 IOException



单个处理

异常处理的原则，要么就处理，要么就抛出。不要处理和抛出同时弄了，这里的处理是向logger.error()记录。

> 适用于不得不处理的异常，例如方法后面标记了异常。

例如:

```
    public final void afterPropertiesSet() throws IOException {
        if (this.singleton) {
            this.singletonInstance = this.createProperties();
        }
    }
```

那么在调用这个方法的时候，要么就try住，要么就抛出。

第一种：再次抛出

```
    @Bean
    public Properties quartzProperties() throws IOException {
     +
        propertiesFactoryBean.afterPropertiesSet();

        return propertiesFactoryBean.getObject();
    }
```

第二种：直接处理了

```
    @Bean
    public Properties quartzProperties() {
        PropertiesFactoryBean propertiesFactoryBean = new PropertiesFactoryBean();
        propertiesFactoryBean.setLocation(new ClassPathResource("/quartz.properties"));

        Properties object = null;
        try {
            propertiesFactoryBean.afterPropertiesSet();
            object = propertiesFactoryBean.getObject();
        } catch (IOException e) {
            log.error("\n====>{}", e.getMessage(), e);
        }
        return object;
    }
```

批量处理

全局异常捕获

> 没有主动在方法后面标记抛出异常的情况

```
try{
    userMapper.insert(userModel);
}catch(RuntimeException e){
    BaseResponse.error("保存失败")
}
```

全局异常捕获，捕获全局异常，然后处理，最后返回给前端。

```
package com.ctcc.misas.common.handler;

import com.ctcc.misas.exception.BaseException;
import com.ctcc.misas.response.BaseResponse;
import lombok.extern.slf4j.Slf4j;
import org.springframework.dao.DuplicateKeyException;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.http.converter.HttpMessageNotReadableException;
```

```
import
org.springframework.messaging.handler.annotation.support.MethodArgumentNotValidException
;
import org.springframework.validation.BindException;
import org.springframework.web.HttpRequestMethodNotSupportedException;
//import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.MissingServletRequestParameterException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

import javax.servlet.http.HttpServletRequest;
import javax.validation.ConstraintViolationException;
import javax.validation.ValidationException;

/**
 * Description:全局异常处理, 采用@Controller + @ExceptionHandler解决
 * <br>自定义异常处理类
 *
 * @author eric
 */
@ControllerAdvice
@Slf4j
@ResponseBody
public class GlobalExceptionHandler {

//    存在注入不了的情况: https://docs.qq.com/doc/DSGJPZnpkV3ZkQlNJ
//    @Autowired(required = false)
//    JavaMailSender javaMailSender;

    /**
     * 业务异常捕获
     *
     * @param request
     * @param e
     * @param <T>
     * @return
     */
    @ExceptionHandler(BaseException.class)
    public <T> BaseResponse<?> baseExcepitonHandler(HttpServletRequest request,
BaseException e) {
        log.error("{} Exception", request.getRequestURI(), e.getMessage(), e);
        return BaseResponse.error(e.getMessage());
    }

    /**
     * ValidationException
     */
    @ExceptionHandler(ValidationException.class)
    public <T> BaseResponse<?> handleValidationException(ValidationException e) {
        log.warn(e.getMessage(), e);
        return BaseResponse.error("校验出错啦! ", e.getMessage());
    }

    /**
     * 参数校验异常捕获 包括各种自定义的参数异常
     *
```

```
     * @param request
     * @param e
     * @param <T>fsFileId
     * @return
     */
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(ConstraintViolationException.class)
    public <T> BaseResponse<?> constraintViolationExceptionHandler(HttpServletRequest
request, ConstraintViolationException e) {
        log.warn("\n====>{} Exception Message: {}", request.getRequestURI(),
e.getMessage(), e);
        return BaseResponse.error("参数错误", e.getMessage());
    }


    /**
     * 方法参数校验
     * https://blog.csdn.net/chengliqu4475/article/details/100834090
     */
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public <T> BaseResponse<?>
handleMethodArgumentNotValidException(MethodArgumentNotValidException e) {
        log.warn(e.getMessage(), e);
        return BaseResponse.error("参数检验出错啦！",
e.getBindingResult().getFieldError().getDefaultMessage());
    }


    /**
     * 处理400参数错误
     *
     * @param e
     * @return
     */
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(HttpMessageNotReadableException.class)
    public BaseResponse
handleHttpMessageNotReadableException(HttpMessageNotReadableException e) {
        log.warn("参数解析失败{}", e.getMessage(), e);
        return BaseResponse.badrequest("参数解析失败", e.getMessage());
    }

    /**
     * 405错误方法不支持
     *
     * @param e
     * @return
     */
    @ResponseStatus(HttpStatus.METHOD_NOT_ALLOWED)
    @ExceptionHandler(HttpRequestMethodNotSupportedException.class)
    public BaseResponse handleHttpRequestMethodNotSupportedException(HttpServletRequest
request, HttpRequestMethodNotSupportedException e) {
        log.warn("[{}]不支持当前[{}]请求方法,应该是[{},{}]", request.getRequestURI(),
e.getMethod(), e.getSupportedHttpMethods(), e.getSupportedMethods(), e);
        return BaseResponse.badrequest("请求方法不支持", e.getMessage());
    }

    @ResponseStatus(HttpStatus.BAD_REQUEST)
```

```
    @ExceptionHandler(BindException.class)
    public BaseResponse bindException(BindException e) {
        log.warn("请求参数错误:{}", e.getMessage(), e);
        return
BaseResponse.badrequest(e.getBindingResult().getFieldError().getDefaultMessage());
    }

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MissingServletRequestParameterException.class)
    public BaseResponse
MissingServletRequestParameterException(MissingServletRequestParameterException e) {
        log.warn("请求参数错误{}", e.getMessage(), e);
        return BaseResponse.badrequest("请求参数错误", e.getMessage());
    }

    @ExceptionHandler(DuplicateKeyException.class)
    public BaseResponse handleDuplicateKeyException(DuplicateKeyException e) {
        log.warn(e.getMessage(), e);
        return BaseResponse.badrequest("请求参数错误", e.getMessage());

    }

    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    @ExceptionHandler(NullPointerException.class)
    public <T> BaseResponse<?> nullPointerExcepitonHandler(HttpServletRequest request,
NullPointerException e) {
        log.error("{} NULL POINT Exception", request.getRequestURI(), e.getMessage(),
e);
        return BaseResponse.error("biu, 踩雷啦！", e.getMessage());
    }

    /**
     * 这个应该放在最下面比较好，最后加载
     * 处理未定义的其他异常信息
     * 参数为空等
     *
     * @param request
     * @param exception
     * @return
     */
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    @ExceptionHandler(value = Exception.class)
    public BaseResponse exceptionHandler(HttpServletRequest request, Exception
exception) {
        log.error("{} Exception Message: {}", request.getRequestURI(),
exception.getMessage(), exception);
        return BaseResponse.error("服务器异常", exception.getMessage());
    }

}
```

事物的原子性在日常编程的时候有哪些体验?

事务

ACID 原子性

https://www.cnblogs.com/caoyc/p/5632963.html

随着系统复杂性增加，事务的原子性需求就慢慢体现出来了。

Spring 事务回滚机制是这样的：当所拦截的方法有指定`异常抛出`，事务才会自动进行回滚！也就是说事务需要异常来配合才能触发。

```
1 @Transactional(rollbackFor=Exception.class) //指定回滚,遇到异常Exception时回滚
2 public void methodName() {
3       throw new Exception("注释");
4 }
```

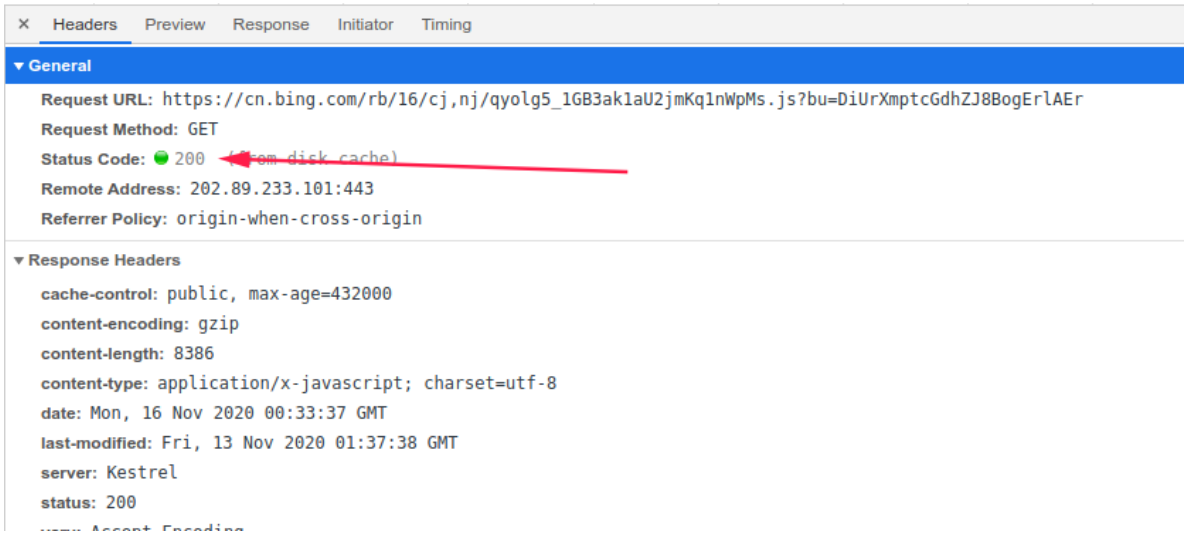
http的状态与返回的code该怎么使用?

code

> 针对业务的

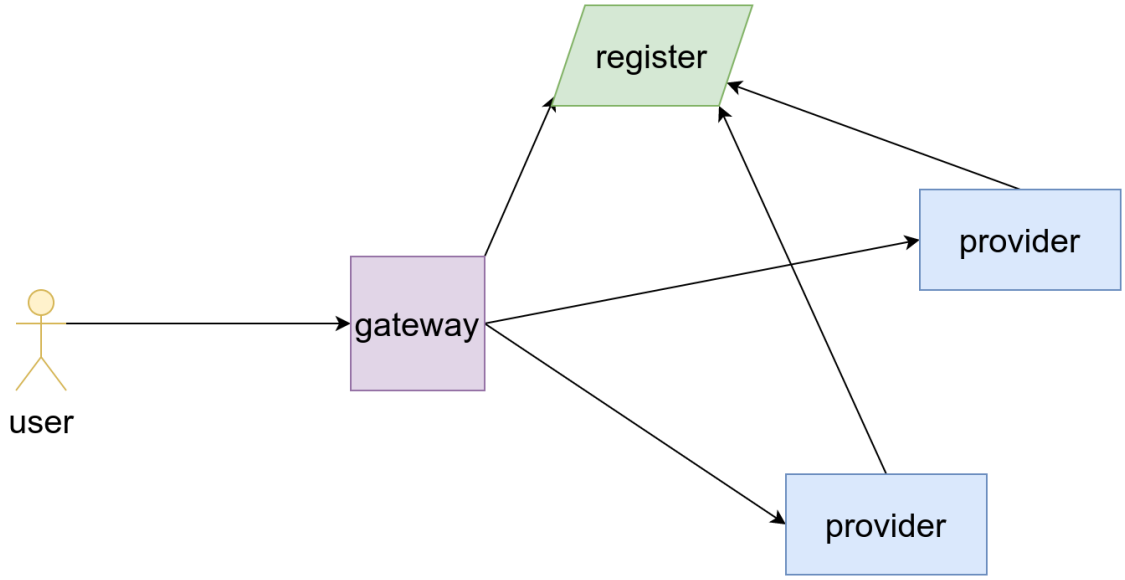
status

> 针对http协议的



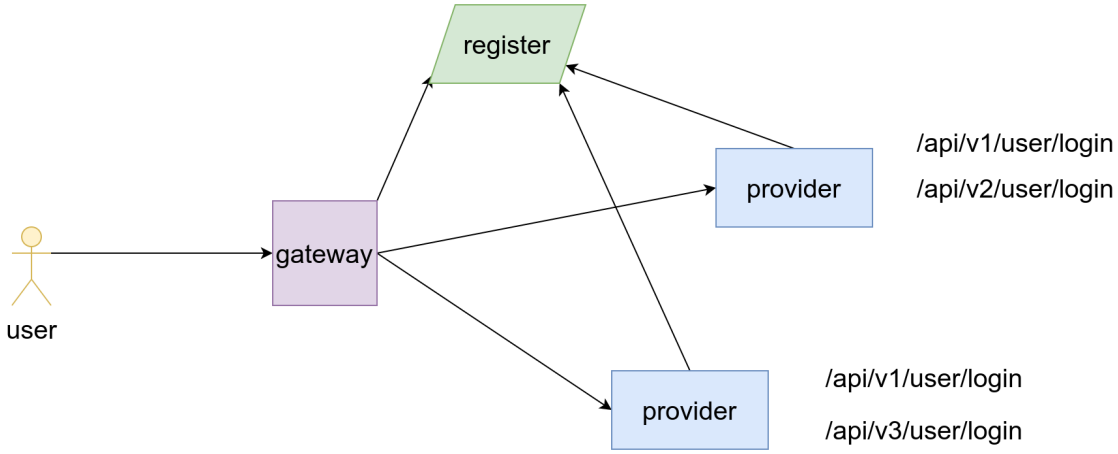
故障转移

故障转移，当gateway发现服务提供者返回状态为预期异常状态的时候就可以重新发起请求。



灰度发布

访问不存在的接口



```
        <!--        500等异常状态的的重试-->
         <dependency>
             <groupId>org.springframework.retry</groupId>
             <artifactId>spring-retry</artifactId>
         </dependency>
```

```
ribbon: #设置ribbon的超时时间小于zuul的超时时间
  ReadTimeout: 100000
  ConnectTimeout: 100000
  maxAutoRetries: 1
  maxAutoRetriesNextServer: 3
  OkToRetryOnAllOperations: true  #默认为false,则只允许GET请求被重试
  retryableStatusCodes: 500,502,404,400 # 500服务器内部错误，502网关超时，404服务没有找到，
400客户端请求错误，例如参数错误
```