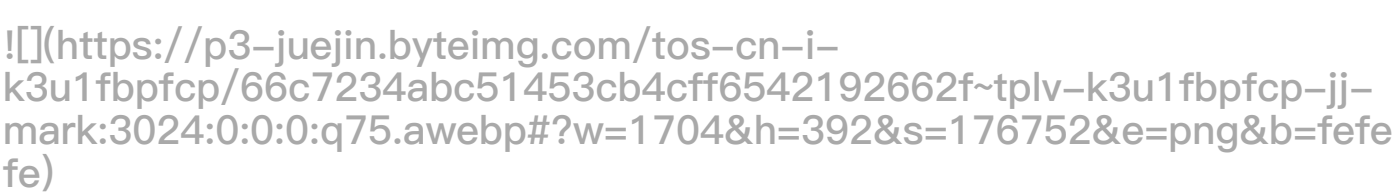
```
on.setCode(ResponseStatusEnum.BAD_REQUEST.getCode());
            }
            return ResponseVO.failure(bizException.getCode(),
bizException.getMessage());
        } else if (e instanceof MethodArgumentNotValidException) {
            // 参数检验异常
            MethodArgumentNotValidException
methodArgumentNotValidException =
(MethodArgumentNotValidException) e;
            Map<String, String> map = new HashMap<>();
            BindingResult result =
methodArgumentNotValidException.getBindingResult();
            result.getFieldErrors().forEach((item)->{
                String message = item.getDefaultMessage();
                String field = item.getField();
                map.put(field, message);
            });
            log.error("数据校验出现错误：", e);
            return
ResponseVO.failure(ResponseStatusEnum.BAD_REQUEST, map);
        } else if (e instanceof HttpRequestMethodNotSupportedException)
{
            log.error("请求方法错误：", e);
            return
ResponseVO.failure(ResponseStatusEnum.BAD_REQUEST.getCode(), "请
求方法不正确");
        } else if (e instanceof MissingServletRequestParameterException) {
            log.error("请求参数缺失：", e);
            MissingServletRequestParameterException ex =
(MissingServletRequestParameterException) e;
            return
ResponseVO.failure(ResponseStatusEnum.BAD_REQUEST.getCode(), "请
求参数缺少: " + ex.getParameterName());
        } else if (e instanceof MethodArgumentTypeMismatchException) {
            log.error("请求参数类型错误：", e);
            MethodArgumentTypeMismatchException ex =
(MethodArgumentTypeMismatchException) e;
            return
ResponseVO.failure(ResponseStatusEnum.BAD_REQUEST.getCode(), "请
求参数类型不正确：" + ex.getName());
        } else if (e instanceof NoHandlerFoundException) {
            NoHandlerFoundException ex = (NoHandlerFoundException) e;
            log.error("请求地址不存在：", e);
            return ResponseVO.failure(ResponseStatusEnum.NOT_EXIST,
ex.getRequestURL());
```

```
        } else {
            //如果是系统的异常，比如空指针这些异常
            log.error("【系统异常】", e);
            return
ResponseVO.failure(ResponseStatusEnum.SYSTEM_ERROR.getCode(),
ResponseStatusEnum.SYSTEM_ERROR.getMsg());
        }
    }

}
```

再次调用接口结果如下：



这时候正常返回统一的格式，方便前端处理。关于接口返回结果格式全局统一和异常统一处理详解请看之前总结的：[Spring Boot如何优雅实现结果统一封装和异常统一处理](http://cxyroad.com/"https://mp.weixin.qq.com/s?__biz=Mzg5MDY1NzI0MQ==&mid=2247486185&idx=1&sn=43b44a937a481d0b6539aad487f5014f&chksm=cfd80a5ff8af834950e923cb2cc8a9ab5caf206c578fea0f95a03309eb9a9ea25ba6e1994d24&token=1842767138&lang=zh_CN#rd")

3.@InitBinder

该注解作用于方法上,用于将前端请求的特定类型的参数在到达controller之前进行处理，从而达到转换请求参数格式的目的。

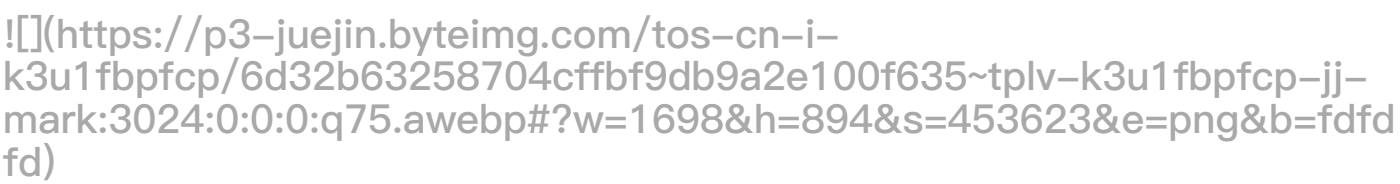
先来看看我们接口示例：

```
    @GetMapping("/222")
    public void test222(User user) {
        System.out.println(user);
    }
```

```

```
@Data
public class User {
    private Long id;
    private Date birthday;
}
```

postman调接口：



报错了：

```
org.springframework.validatio
mappedHandler.applyAfterConcurrentHandlingStarted(processedRequest
, response);
        }
      }
      else {
        // Clean up any resources used by a multipart request.
        if (multipartRequestParsed) {
          cleanupMultipart(processedRequest);
        }
      }
    }
  }
```

Spring MVC是通过处理器适配器来进行具体方法的调用执行的，这时候来到适配器`RequestMappingHandlerAdapter`
```

```
@Override
public void afterPropertiesSet() {
  // Do this first, it may add ResponseBody advice beans
  //  初始化 ControllerAdvice 相关
  initControllerAdviceCache();

  // 初始化 argumentResolvers 属性
  if (this.argumentResolvers == null) {
    List<HandlerMethodArgumentResolver> resolvers =
getDefaultArgumentResolvers();
    this.argumentResolvers = new
HandlerMethodArgumentResolverComposite().addResolvers(resolvers);
  }
  // 初始化 initBinderArgumentResolvers 属性
  if (this.initBinderArgumentResolvers == null) {
    List<HandlerMethodArgumentResolver> resolvers =
getDefaultInitBinderArgumentResolvers();
    this.initBinderArgumentResolvers = new
HandlerMethodArgumentResolverComposite().addResolvers(resolvers);
  }
  // 初始化 returnValueHandlers 属性
  if (this.returnValueHandlers == null) {
    List<HandlerMethodReturnValueHandler> handlers =
getDefaultReturnValueHandlers();
    this.returnValueHandlers = new
HandlerMethodReturnValueHandlerComposite().addHandlers(handlers);
  }
}

```

```


private void initControllerAdviceCache() {
  if (getApplicationContext() == null) {
    return;
  }

  // <1> 扫描 @ControllerAdvice 注解的 Bean 们，并将进行排序
  List<ControllerAdviceBean> adviceBeans =
ControllerAdviceBean.findAnnotatedBeans(getApplicationContext());
  AnnotationAwareOrderComparator.sort(adviceBeans);

  List<Object> requestResponseBodyAdviceBeans = new ArrayList<>();

  //  遍历 ControllerAdviceBean 数组
```

```
  for (ControllerAdviceBean adviceBean : adviceBeans) {
    Class<?> beanType = adviceBean.getBeanType();
    if (beanType == null) {
      throw new IllegalStateException("Unresolvable type for
ControllerAdviceBean: " + adviceBean);
    }
    // 扫描有 @ModelAttribute ，无 @RequestMapping 注解的方法，添加到
modelAttributeAdviceCache 中
    Set<Method> attrMethods =
MethodIntrospector.selectMethods(beanType,
MODEL_ATTRIBUTE_METHODS);
    if (!attrMethods.isEmpty()) {
      this.modelAttributeAdviceCache.put(adviceBean, attrMethods);
    }
    // 扫描有 @InitBinder 注解的方法，添加到 initBinderAdviceCache 中
    Set<Method> binderMethods =
MethodIntrospector.selectMethods(beanType, INIT_BINDER_METHODS);
    if (!binderMethods.isEmpty()) {
      this.initBinderAdviceCache.put(adviceBean, binderMethods);
    }
    // 如果是 RequestBodyAdvice 或 ResponseBodyAdvice 的子类，添加到
requestResponseBodyAdviceBeans 中
    if (RequestBodyAdvice.class.isAssignableFrom(beanType)) {
      requestResponseBodyAdviceBeans.add(adviceBean);
    }
    if (ResponseBodyAdvice.class.isAssignableFrom(beanType)) {
      requestResponseBodyAdviceBeans.add(adviceBean);
    }
  }

  // 将 requestResponseBodyAdviceBeans 添加到
this.requestResponseBodyAdvice 属性种
  if (!requestResponseBodyAdviceBeans.isEmpty()) {
    this.requestResponseBodyAdvice.addAll(0,
requestResponseBodyAdviceBeans);
  }
}

```
```

这就是@ControllerAdvice的实现原理底层分析咯。

原文链接: https://juejin.cn/post/7383548892626845711
```