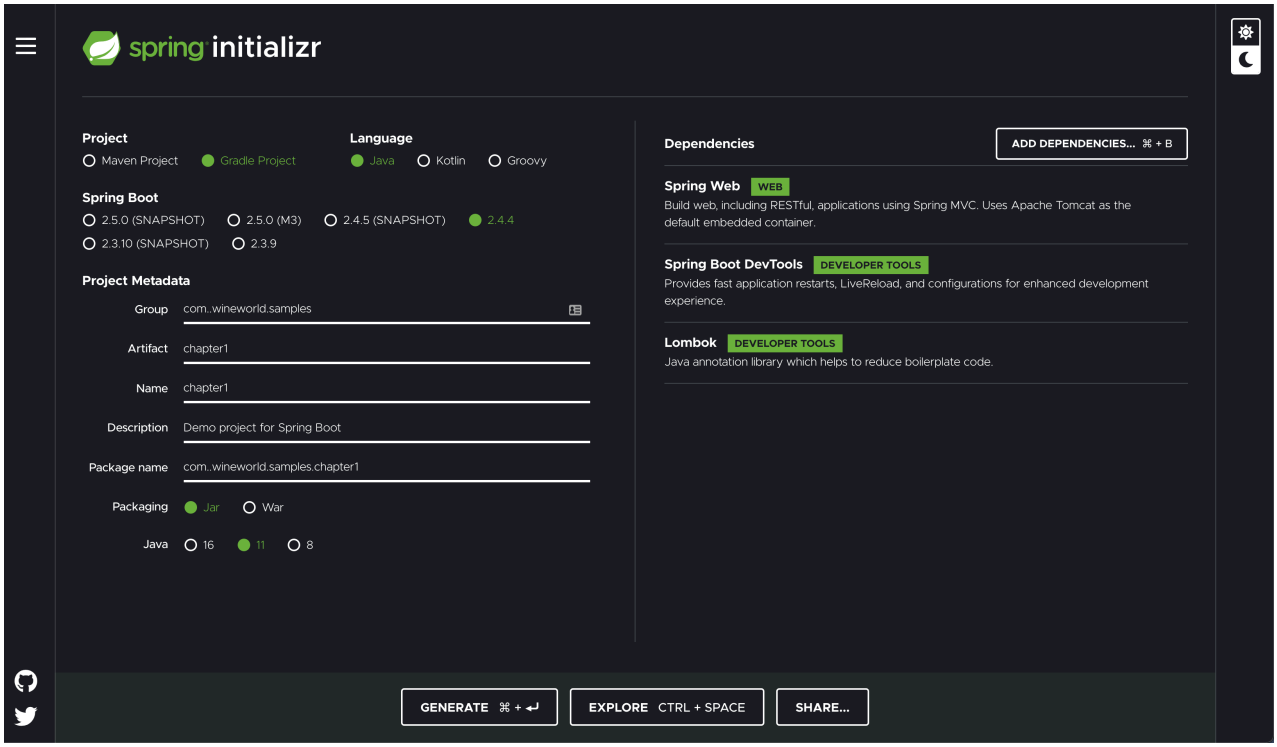
Spring Boot Quick Start

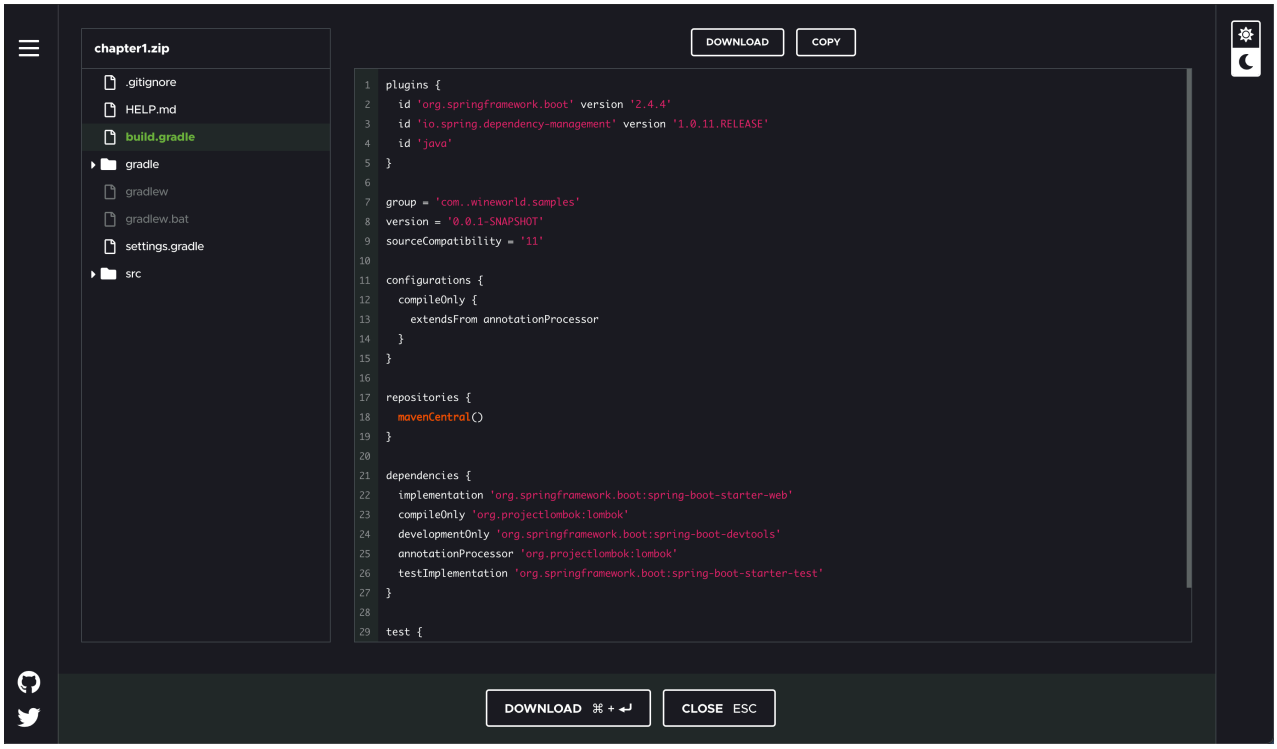
1 初始化工具

为了方便开发者快速构建应用，Spring 官方提供SpringBoot应用初始化工具，键入网址： https://start.spring.io 进入项目初始化，输入相关配置并选择模块，演示项目选择如下模块：

- Spring Web: 使用SpringMVC构建WEB服务
- Srping Boot DevTools: 支持 Spring Boot 开发热部署
- Lombok: Java代码增强工具

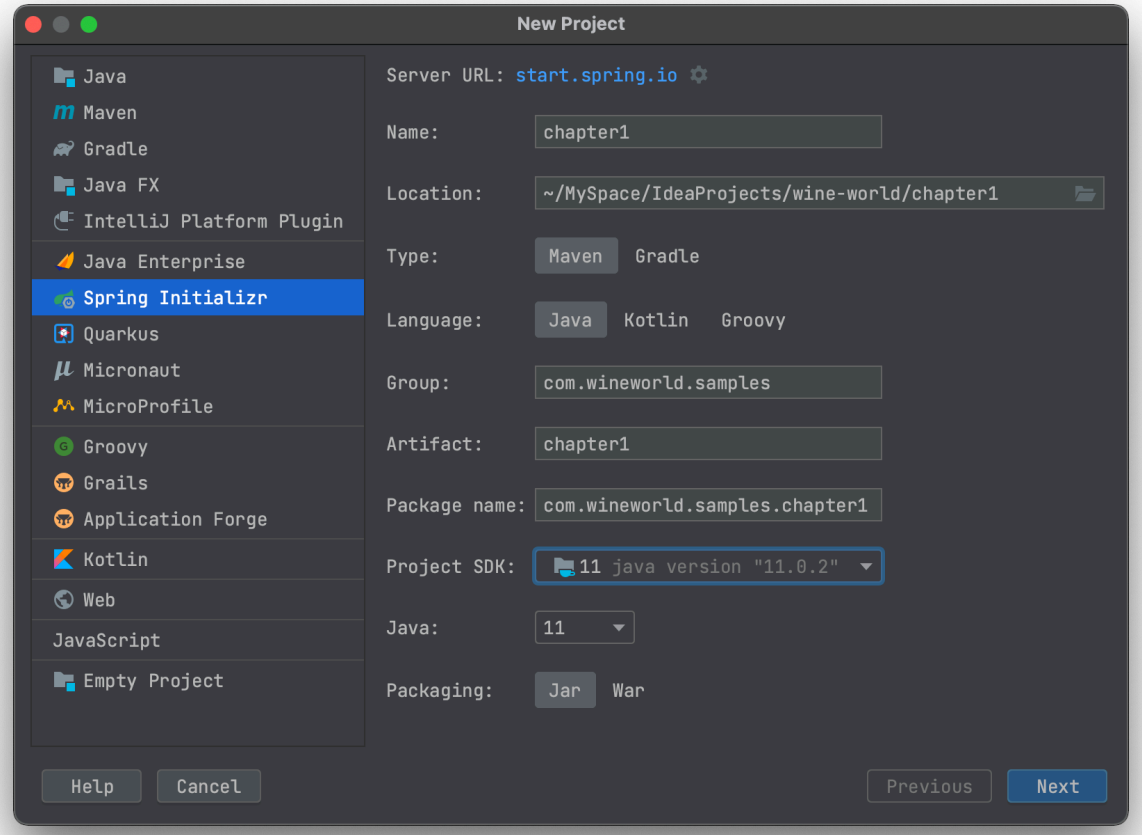


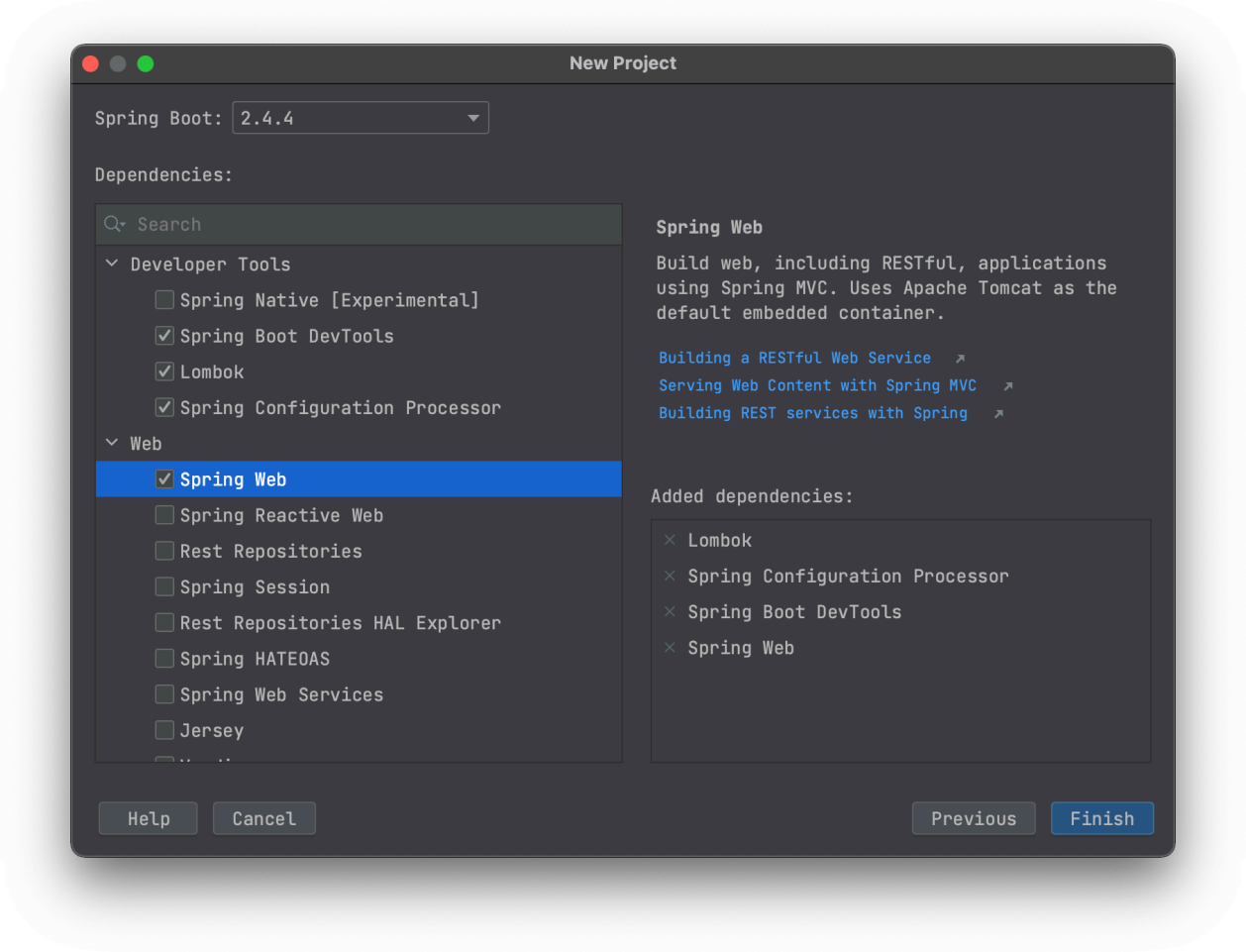
点击 "EXPLORE" 可预览生成的源文件，并支持下载。



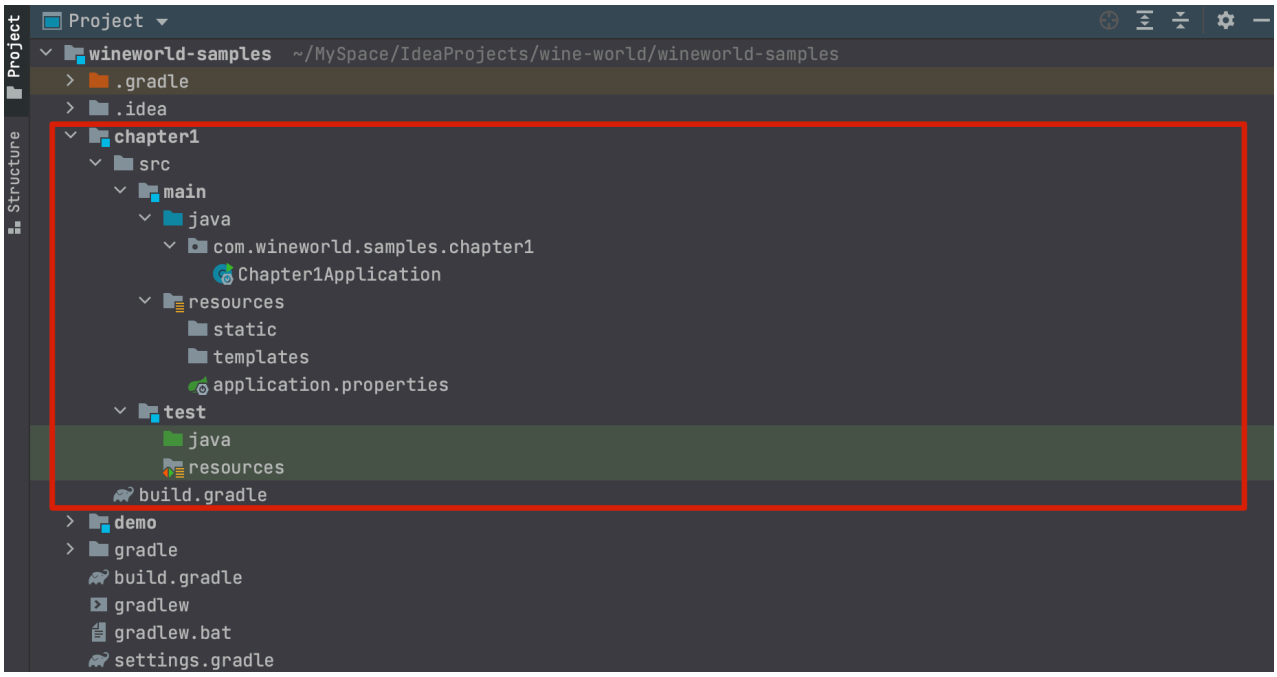
下载后导入 JetBrains IDEA。

另外，JetBrains Idea 创建新项目也提供 Spring Initializr 快速创建 Sping Boot 应用：





Spring Intitializr 生成的源代码代码如下所示：



- Chapter1Application: Spring Boot 应用 Main 程序
  - @SpringBootApplication: 标注 Chapter1Application 是应用入口启动类，关于此注解详细说明可参考 https://www.jianshu.com/p/4e1cab2d8431

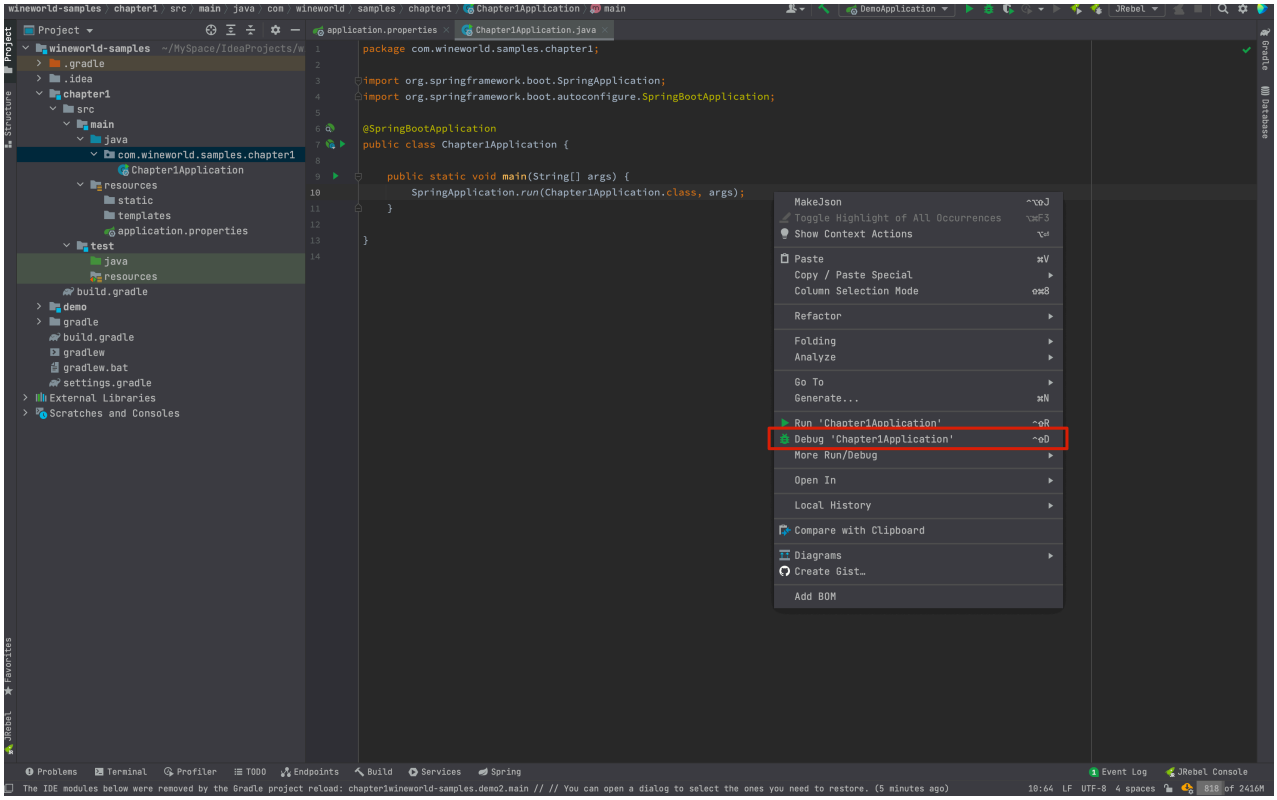
```
@SpringBootApplication
public class Chapter1Application {

    public static void main(String[] args) {
        SpringApplication.run(Chapter1Application.class, args);
    }

}
```

- application.properties: 项目配置文件，当前文件内容为空
- static/templates目录：静态资源或JSP/thymeleaf/freemarker模版文件目录，演示构建restful web应用，可忽略

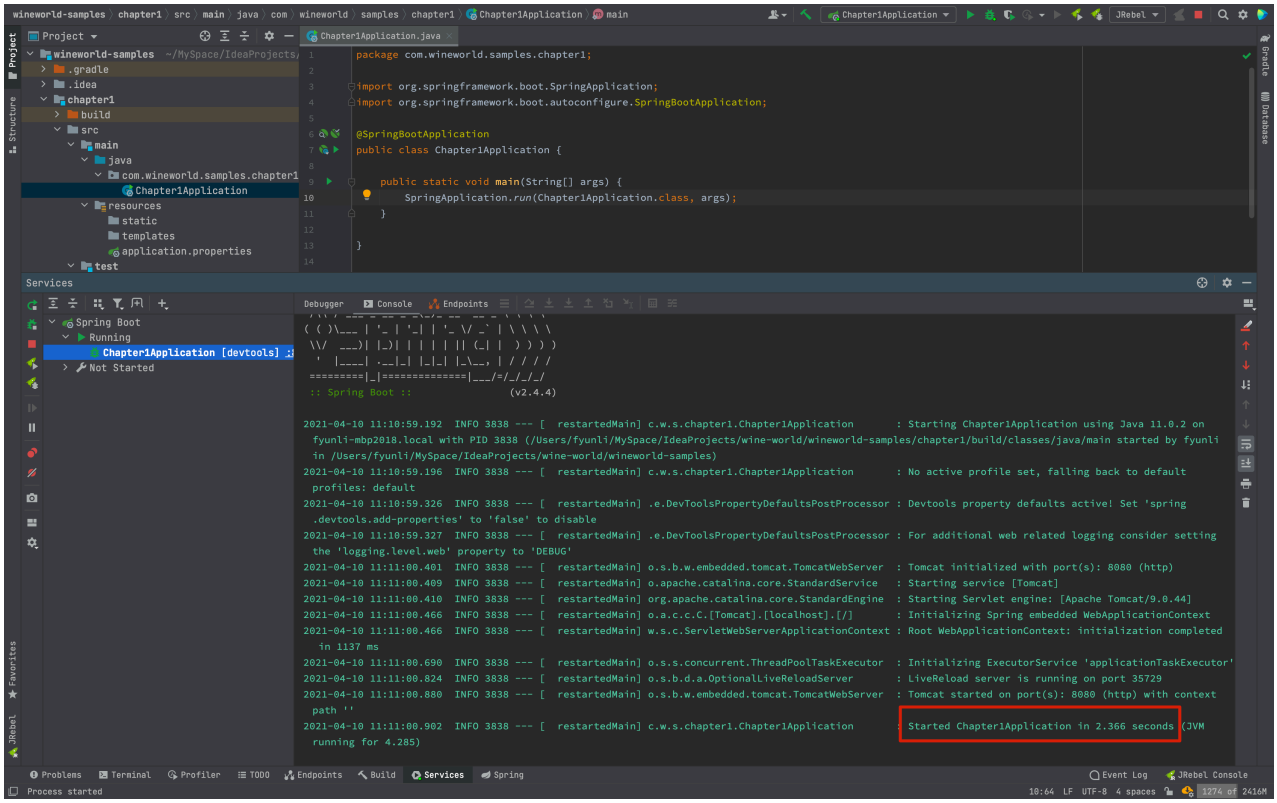
因国内网络环境，建议 build.gradle 增加如下配置：

```
// 必须增加开头
buildscript {
    repositories {
        maven { url 'https://maven.aliyun.com/repository/public' }
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:2.4.4")
    }
}
...
// repositories 增加阿里云 maven 仓库
repositories {
    maven { url 'https://maven.aliyun.com/repository/public' }
    jcenter()
    mavenCentral()
}
```

等待 gradle 下载依赖完成后，打开 Chapter1Application，右键菜单选择 “Debug Chapter1Application” 运行。



出现 "Started Chapter1Application in xxx seconds" 表明应用已经成功启动。



2 第一个 Controller

完成上述工作之后，还无法提供任何一个URL访问点供用户访问，下面我们开始来发布一个经典"Hello World"。

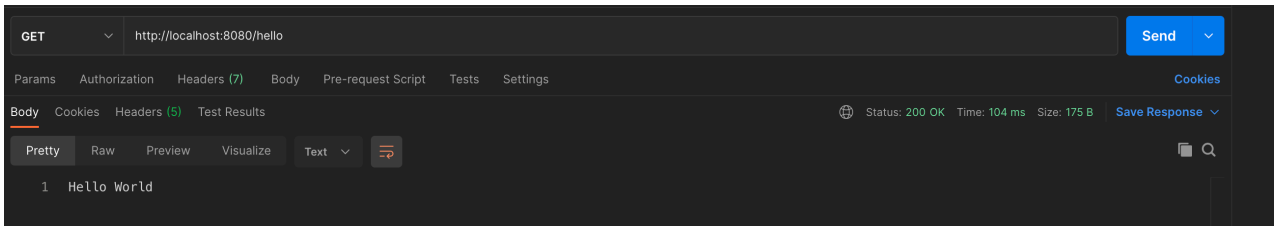
按照经典的 entity/dao/service/controller 分层结构，创建 controller 包，并在其下创建 HelloController，内容如下：

```
@RestController
public class HelloController {

    @GetMapping("/hello")
    public String hello() {
        return "Hello World";
    }

}
```

其中：

- @RestController: 标注 Controller 是一个 SpringMVC Controller，并引入 @ResponseBody 注解表示 Controller 的返回的结果直接写入 HTTP 响应正文中。
- @GetMapping: 声明一个 URL "/hello"，并允许 HTTP GET 方法访问
    - SpringMVC 支持 HTTP 标准方法，参考：
        - https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods
        - https://www.baeldung.com/spring-mvc-annotations(此系列包含Spring各组件的注解说明，推荐)

编译之后，Spring Boot Devtools 会自动重启应用，使用 Postman 或浏览器输入 http://localhost:8080/hello，结果如下：



3 Servlet 容器配置

Spring Boot 默认使用嵌入式 Tomcat 容器，并运行在 8080 端口，如果需要改变容器端口，Context Path，或调整容器线程设置等，可在 application.properties 文件配置：

```
server.port=8088
server.servlet.context-path=/chapter1
```

重启应用后，则 http://localhost:8080/hello 不再可访问，URL变更为 http://localhost:8088/chapter1/hello

更多配置请参考：

https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html#server.port

https://www.baeldung.com/spring-boot-configure-tomcat

4. 日志配置

Spring Boot 使用 Commons Logging 处理日志输出，支持 java util logging, log4j, logback 等，默认情况下，Spring Boot 使用 logback 输出日志，可通过 application.properties 或者 logback.xml 等调整日志设置：

https://www.jianshu.com/p/f67c721eea1b

https://docs.spring.io/spring-boot/docs/2.4.4/reference/html/spring-boot-features.html#boot-features-logging

推荐使用 logback，并通过 logback.xml 配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <include resource="org/springframework/boot/logging/logback/defaults.xml"/>
    <property name="LOG_FILE" value="logs/samples-chapter1.log"/>
    <property name="LOG_PATTERN" value="%d{yyyy-MM-dd HH:mm:ss.SSS} -%5p ${PID:-}
[%15.15t] %logger{39}.%M:%L : %m%n"/>
    <property name="CONSOLE_LOG_PATTERN" value="${LOG_PATTERN}"/>
    <property name="FILE_LOG_PATTERN" value="${LOG_PATTERN}"/>

    <include resource="org/springframework/boot/logging/logback/console-appender.xml"/>
    <include resource="org/springframework/boot/logging/logback/file-appender.xml"/>

    <logger name="org.springframework" level="WARN"/>
    <logger name="org.apache" level="WARN"/>

    <root level="DEBUG">
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="FILE"/>
    </root>

</configuration>
```

5 第一个CRUD(Data JPA)

以上我们对 Spring Boot 构建应用有了初步的理解，大多数情况下，构建的应用都需要和数据库打交道，基于 Spring Data JPA，我们构建一个单表的 CRUD Rest API。

5.1 导入 starter

build.gradle 增加:

```
implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
implementation 'com.h2database:h2'
```

Spring Data JPA 默认使用 Hibernate 作为 JPA 实现。demo 项目使用 h2 嵌入式数据库。

5.2 配置数据源

在 application.properties 中添加：

```
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:test
spring.datasource.username=root
spring.datasource.password=test
```

5.3 Entity

新建 entity 包，并创建一个 Person 实体：

```
@Entity
@Getter
@Setter
public class Person implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String name;

    @Temporal(TemporalType.DATE)
    private Date birthday;

}
```

Spring Data jpa 注解请参考： https://www.baeldung.com/spring-data-annotations

5.4 Repository (Dao)

```
@Repository
public interface PersonRepository extends JpaRepository<Person, Long> {

}
```

5.5 Controller

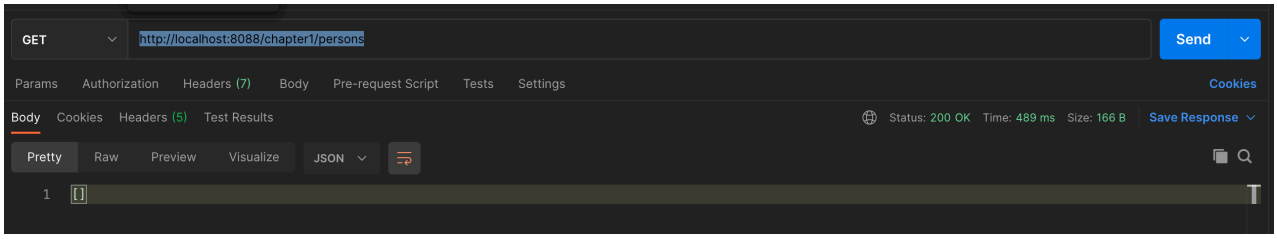
Person API 按照 REST API 要求，如下：

```
@RestController
@RequestMapping("/persons")
public class PersonController {

    @Autowired
    private PersonRepository personRepository;

    @GetMapping
    public Iterable<Person> list() {
        return personRepository.findAll();
    }

    @GetMapping("/{id}")
    public Person list(@PathVariable Long id) {
        return personRepository.getOne(id);
    }

    @PostMapping
    public Person create(@RequestBody Person person) {
        return personRepository.save(person);
    }

    @PutMapping("/{id}")
    public Person update(@PathVariable Long id, @RequestBody Person person) {
        Assert.notNull(id, "非法参数");
        return personRepository.save(person);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity delete(@PathVariable Long id) {
        personRepository.deleteById(id);
        return ResponseEntity.ok("Deleted");
    }

}
```

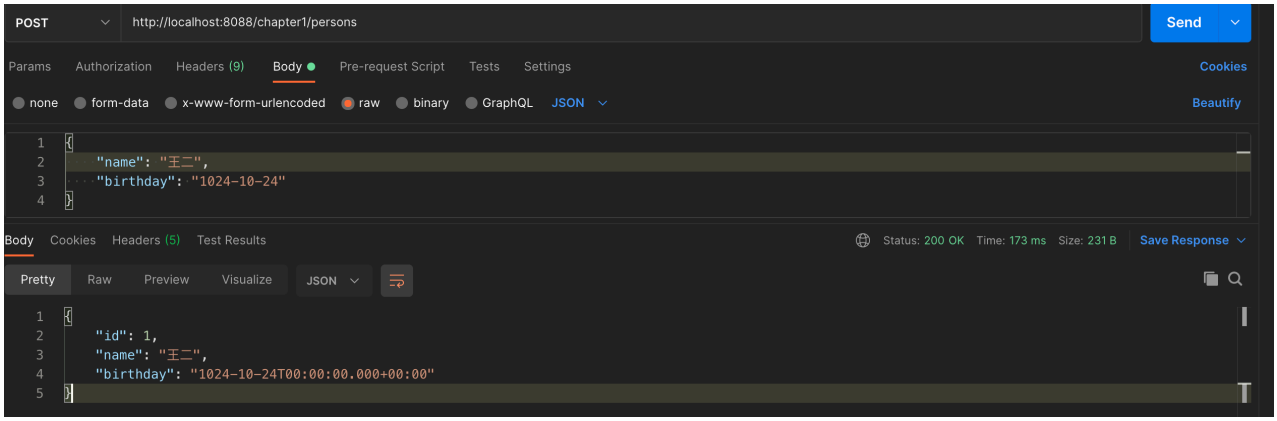
了解更多 REST API，可参考 http://www.ruanyifeng.com/blog/2014/05/restful_api.html

5.6 测试

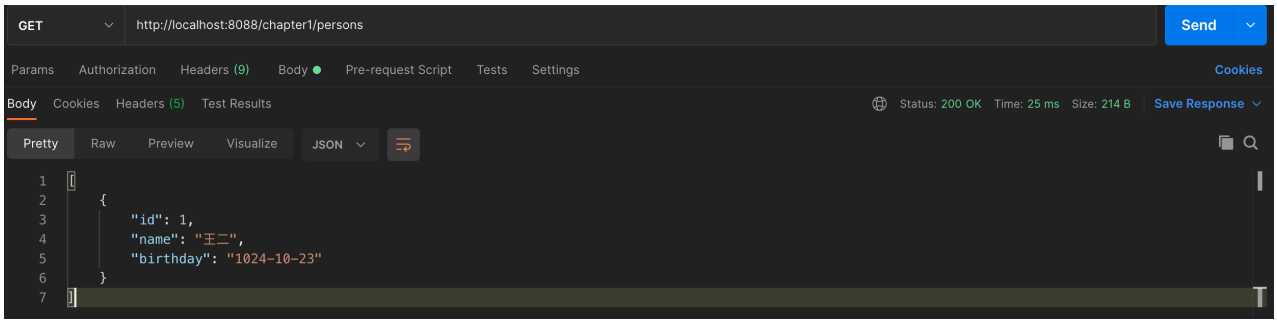
- Postman 输入 http://localhost:8088/chapter1/persons



因当前无数据，因此输出空的集合，现在创建一个Person:



再次查询：



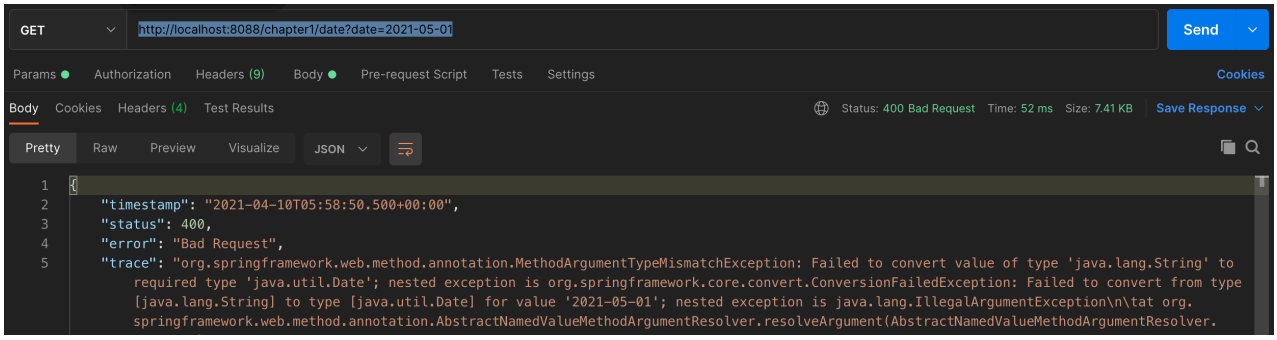
请自行进行更新和删除的测试。

6 PropertyEditor

考虑这样一个请求：

```
@GetMapping("/date")
public Date hello(Date date) {
    return date;
}
```

Postman 请求 http://localhost:8088/chapter1/date?date=2021-05-01，结果如下：



那么项目中经常需要接收用户输入的日期，该如何处理呢？

Spring 支持通过 PropertyEditorSupport 进行转换。

```
public class DateEditor extends PropertyEditorSupport {
```

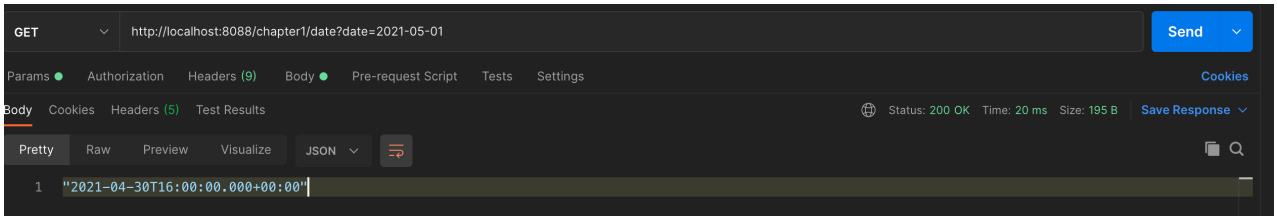
```
    public static StdDateFormat STD_DATE_FORMAT = new StdDateFormat();
    private boolean emptyAsNull;

    static {
        STD_DATE_FORMAT.setTimeZone(TimeZone.getTimeZone("Asia/Shanghai"));
    }

    public DateEditor(boolean emptyAsNull) {
        this.emptyAsNull = emptyAsNull;
    }

    @Override
    public String getAsText() {
        Date value = (Date) getValue();
        return value != null ? STD_DATE_FORMAT.format(value) : "";
    }

    @Override
    public void setAsText(String text) {
        if (text == null) {
            setValue(null);
        } else {
            String value = text.trim();
            if (emptyAsNull && "".equals(value)) {
                setValue(null);
            } else {
                try {
                    value = StringUtils.replace(value, " ", "T");
                    Date time = STD_DATE_FORMAT.parse(value);
                    setValue(time);
                } catch (ParseException e) {
                    setValue(null);
                }
            }
        }
    }

}
```

注册自定义 PropertyEditor：

```
@RestController
public class HelloController {

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        binder.registerCustomEditor(Date.class, new DateEditor(true));
    }

    @GetMapping("/hello")
    public String hello() {
```

```
        return "Hello World";
    }

    @GetMapping("/date")
    public Date hello(Date date) {
        return date;
    }

}
```

重新请求 http://localhost:8088/chapter1/date?date=2021-05-01



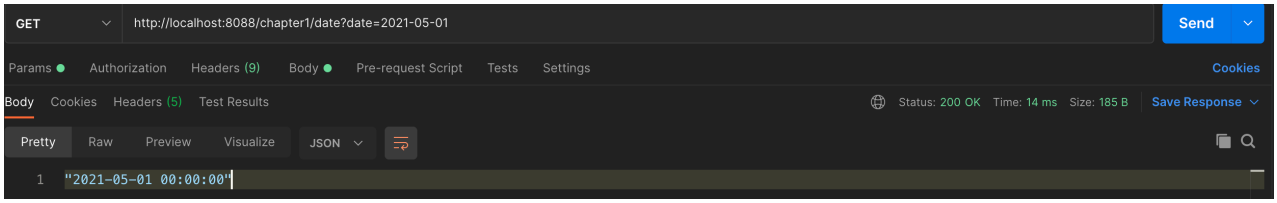
7 JSON 格式化

观察上一节输出的日期格式，会发现默认是以 ISO8601 规范 https://zh.wikipedia.org/zh-hans/ISO_8601 输出0时区时间。对于中国用户而言，如何输出对国人更友好的格式?

7.1 application.properteis 设置

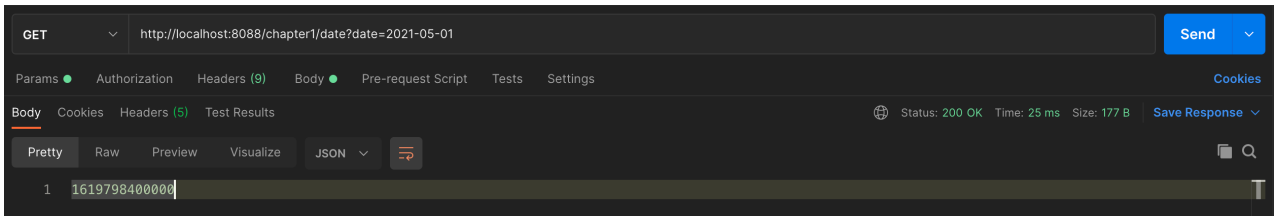
```
spring.jackson.date-format=yyyy-MM-dd HH:mm:ss
spring.jackson.time-zone=GMT+8
```

请求：http://localhost:8088/chapter1/date?date=2021-05-01



或者可以以时间戳方式输出日期，配置:

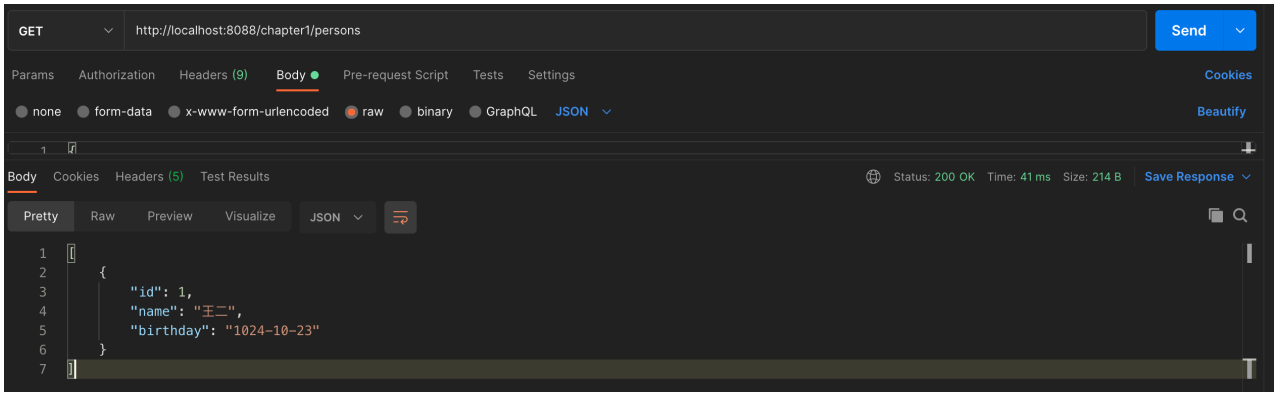
```
spring.jackson.serialization..write-dates-as-timestamps=true
```



7.2 JsonFormat

同时，也可使用 @JsonFormat 标注具体对象的成员输出格式：

```
public class Person implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String name;

    @JsonFormat(pattern = "yyyy-MM-dd")
    @Temporal(TemporalType.DATE)
    private Date birthday;

}
```



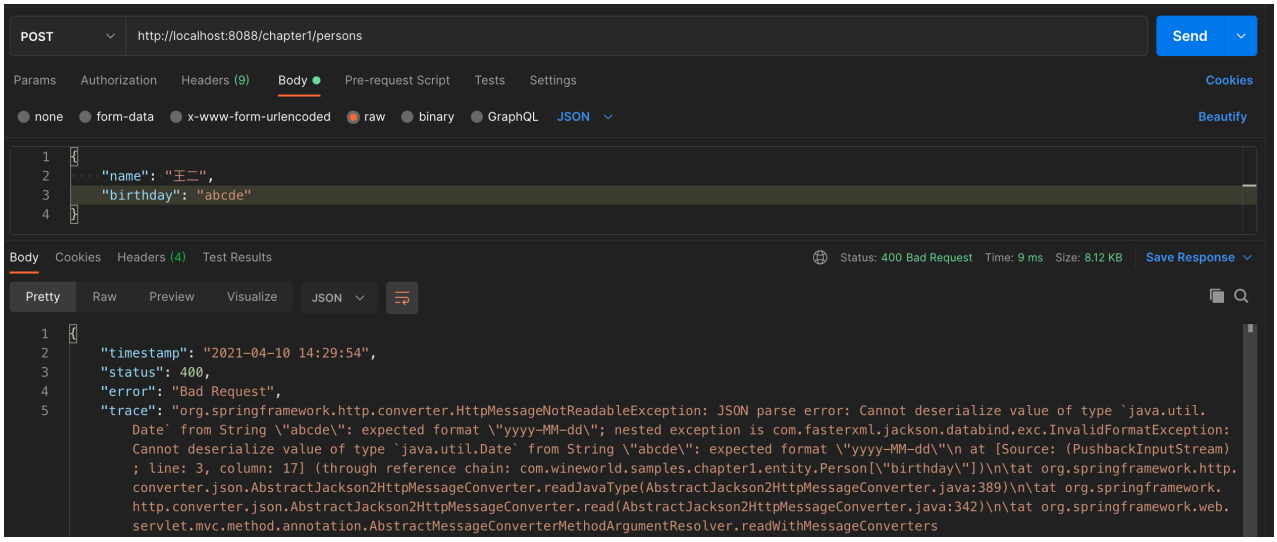
7.3 重定义 ObjectMapper

另外，还可通过自定义系列化器处理输出，如下所示即为实际项目中经常使用的通过自定义系列化器防止XSS注入：

```
    @Bean
    @Primary
    @ConditionalOnMissingBean(ObjectMapper.class)
    public ObjectMapper objectMapper(Jackson2ObjectMapperBuilder builder) {
        ObjectMapper objectMapper = builder.createXmlMapper(false).build();
        SimpleModule xssModule = new
SimpleModule(XssStringJsonSerializer.class.getSimpleName());
        xssModule.addSerializer(new XssStringJsonSerializer());
        objectMapper.registerModule(xssModule);
        return objectMapper;
    }
```

8 输入校验

输入 birthday=abcde，结果如下：



因为期望生日是个有效的日期，但输入不合规，导致程序异常。

Spring Boot 支持基于 JSR 303 校验框架进行输入校验。

8.1 导入依赖

```
implementation 'org.hibernate.validator:hibernate-validator'
```

8.2 标注校验规则

```
    @NotBlank(message = "姓名不可为空")
    private String name;
```

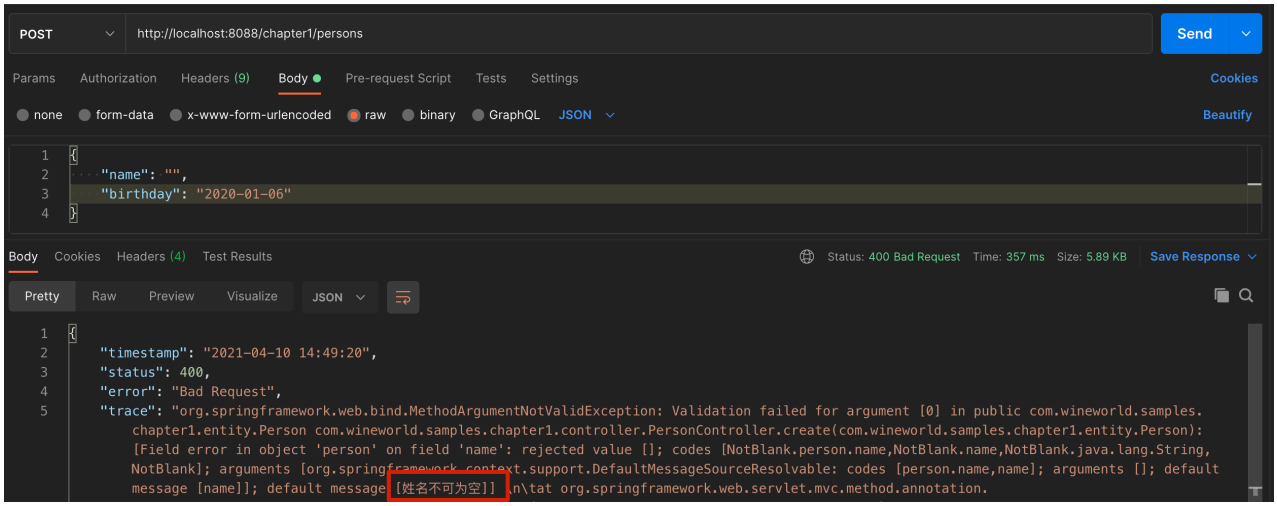
支持的校验规则及自定义校验可参考后文链接。

8.3 @Valid

Controller 方法中声明:

```
    @PostMapping
    public Person create(@Valid @RequestBody Person person) {
        return personRepository.save(person);
    }
```

8.4 测试



https://segmentfault.com/a/1190000023471742

9 统一异常

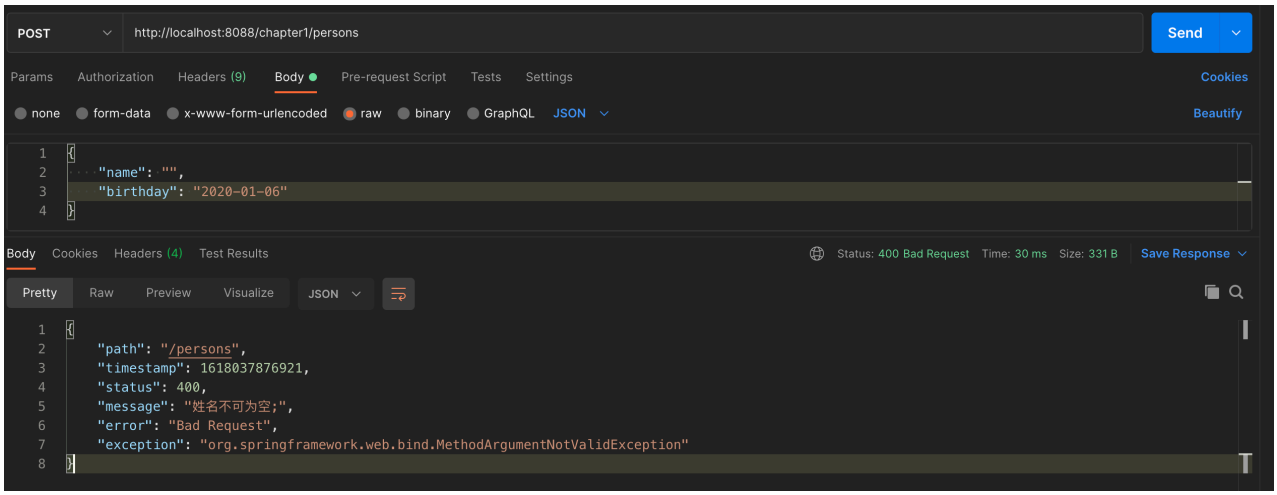
如上所示，项目当前输出的异常信息相当不友好，Spring Boot 支持全局异常处理。

- 标注 RestControllerAdvice
- 继承 ResponseEntityExceptionHandler
- 将异常规范为 ResponseWrapper 格式输出

参考: https://www.jianshu.com/p/ebb8f705863d

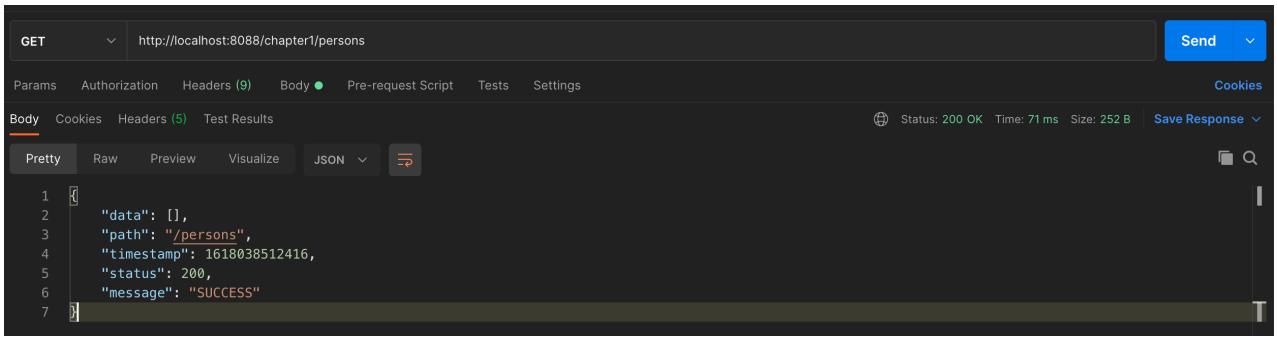
```
@Order
@RestControllerAdvice
@Slf4j
public class DemoRestControllerAdvice extends ResponseEntityExceptionHandler {

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        binder.registerCustomEditor(String.class, new StringTrimmerEditor(false));
        binder.registerCustomEditor(Date.class, new DateEditor(true));
    }

    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(
            MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus status,
WebRequest request) {
        final List<ObjectError> errors = ex.getBindingResult().getAllErrors();
        StringBuffer sb = new StringBuffer();
        if (!CollectionUtils.isEmpty(errors)) {
            for (ObjectError error : errors) {
                sb.append(error.getDefaultMessage());
                sb.append(";");
            }
        }
        return handleExceptionInternal(ex, status, sb.toString(), request);
```

```
    }

    protected ResponseEntity<Object> handleExceptionInternal(Exception ex, HttpStatus
status, String message,
                                                                WebRequest request) {
        String path = request.getDescription(false)
                .replace("uri=", "")
                .replace(request.getContextPath(), "");
        ResponseWrapper responseWrapper = ResponseWrapper.error(ex, status.value(),
message, path);
        return super.handleExceptionInternal(ex, responseWrapper, new HttpHeaders(),
                HttpStatus.valueOf(responseWrapper.getStatus()), request);
    }

}
```



10 统一返回

上一节链接 https://www.jianshu.com/p/ebb8f705863d 有详述，请自行参考：

```
@RestControllerAdvice
@Slf4j
public class WrapperResponseBodyAdvice implements ResponseBodyAdvice<Object> {

    @Autowired
    private ObjectMapper objectMapper;

    public boolean supports(MethodParameter returnType, Class converterType) {
        String typeName = returnType.getGenericParameterType().getTypeName();
        return !typeName.startsWith(ResponseEntity.class.getName());
    }

    public Object beforeBodyWrite(@Nullable Object body, MethodParameter returnType,
                                  MediaType selectedContentType, Class
selectedConverterType,
```

```
                                    ServerHttpRequest request, ServerHttpResponse
response) {
        HttpServletRequest req = ((ServletServerHttpRequest)
request).getServletRequest();
        String uri = StringUtils.replace(req.getRequestURI(), req.getContextPath(),
"");

        if (body == null) {
            response.getHeaders().add(HttpHeaders.CONTENT_TYPE,
MediaType.APPLICATION_JSON_VALUE);
            return ResponseWrapper.success(null, uri);
        }

        if (body instanceof String) {
            return populateString(body, uri, response);
        }

        return ResponseWrapper.success(body, uri);
    }

    private String populateString(Object body, String uri, ServerHttpResponse response)
{
        try {
            response.getHeaders().add(HttpHeaders.CONTENT_TYPE,
MediaType.APPLICATION_JSON_VALUE);
            ResponseWrapper responseBody = ResponseWrapper.success(body, uri);
            return objectMapper.writeValueAsString(responseBody);
        } catch (JsonProcessingException e) {
            log.warn(e.getMessage());
            throw new DemoException("服务器内部错错");
        }
    }

}
```