



匠人学院

jiangren.com.au

Spring Boot Introduction

背景

- 从 Spring 说起，2000 年左右 Java 行业中都是 EJB 的天下，但是 EJB 本身比较庞大复杂，各企业使用起来并不是很便利。
- Rod Johnson 认为企业开发应该更简单，没有必要全部使用 EJB，企业开发应该是一个统一的、高效的方式构造整个应用，并且可以将单层框架以最佳的组合揉和在一起建立一个连贯的体系。
- Rod Johnson 2002 年编写了《Expert One-on-One J2EE Design and Development》。这本书是 Rod Johnson 的成名著作，非常经典，从这本书中的代码诞生了 Spring Framework。
- Spring 在不断发展的过程中也出现了一些问题，随着 Spring 边界不断扩张，需要的配置文件也越来越多，使用起来也越复杂，项目中也经常因为配置文件配置错误产生很多问题。慢慢 Spring 变成了一个大而全的框架，背离它简洁开发的理念。随着微服务的概念兴起，Spring 开发了一个全新的技术栈：**Spring Boot**。



Spring Boot

Spring Boot 是由 Pivotal 团队提供的全新框架，其设计目的是用来简化新 Spring 应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。采用 Spring Boot 可以大大的简化开发模式，所有你想集成的常用框架，它都有对应的组件支持。

<https://spring.io/projects/spring-boot> 官网介绍:

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.

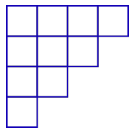
Spring Boot 是一套全新的框架，它来自于 Spring 大家族，因此 Spring 所有具备的功能它都有，而且更容易使用；Spring Boot 以约定大于配置的核心思想，默认帮我们进行了很多设置，多数 Spring Boot 应用只需要很少的 Spring 配置。Spring Boot 开发了很多的应用集成包，支持绝大多数开源软件，让我们以很低的成本去集成其他主流开源软件。

Spring Boot 特性

- 使用 Spring 项目引导页面可以在几秒构建一个项目
- 方便对外输出各种形式的服务，如 REST API、WebSocket、Web、Streaming、Tasks
- 非常简洁的安全策略集成
- 支持关系数据库和非关系数据库
- 支持运行期内嵌容器，如 Tomcat、Jetty
- 强大的开发包，支持热启动
- 自动管理依赖
- 自带应用监控
- 支持各种 IDE，如 IntelliJ IDEA、NetBeans

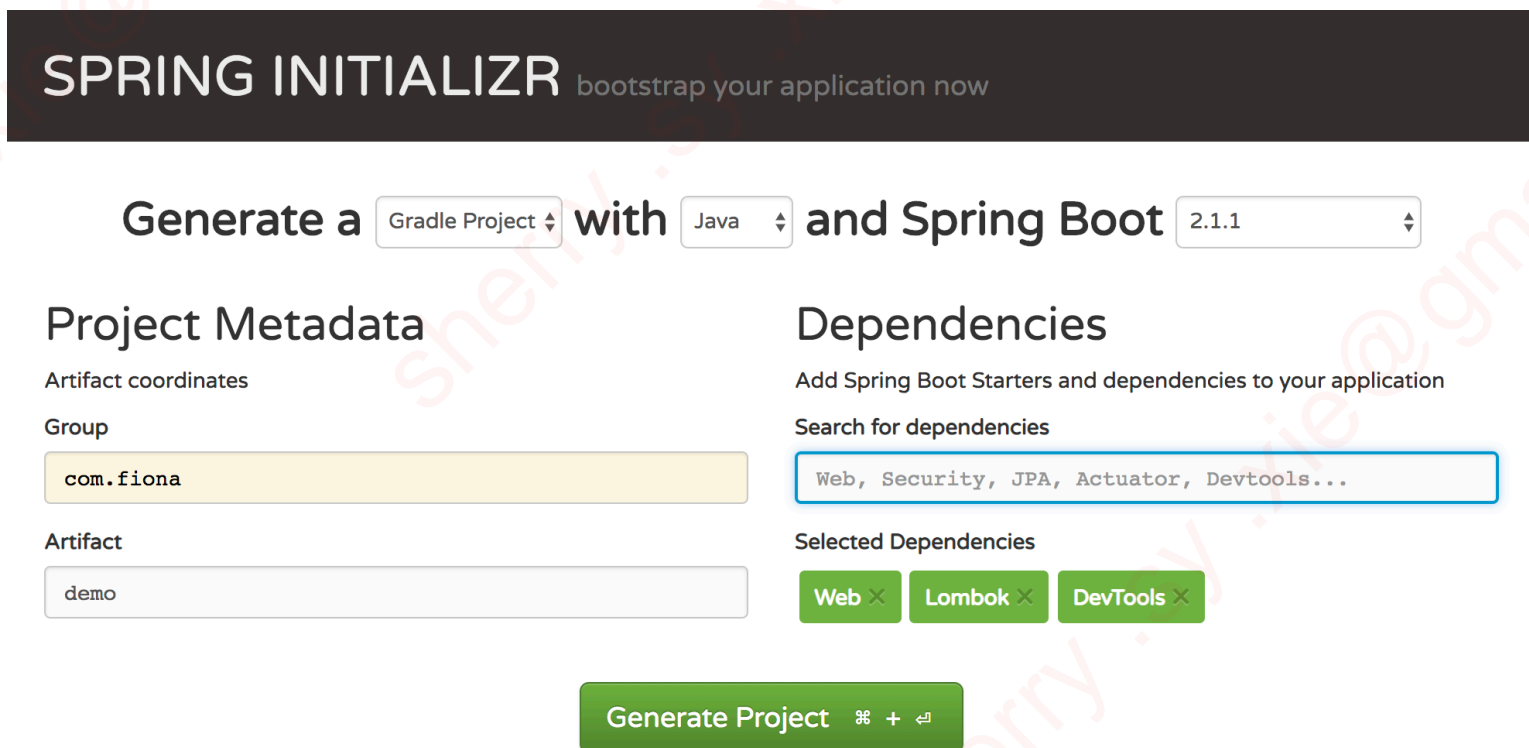
为什么学习Spring Boot

- 从软件发展的角度来讲，越简单的开发模式越会流行。简单的开发模式解放出更多生产力，让开发人员可以将精力集中在业务上，而不是各种配置、语法所设置的门槛上。Spring Boot 就是尽可能的简化应用开发的门槛。
- Spring Boot 所集成的技术栈，几乎都是各互联网公司在使用的技术，按照 Spring Boot 的路线去学习，基本可以了解国内外互联网公司的技术特点。
- Spring Boot 和微服务架构都是未来软件开发的一个大趋势，越早参与其中受益越大。



开始第一个项目

- (1) 访问 SPRING INITIALIZR <http://start.spring.io/>。
- (2) 选择构建工具 Gradle, Spring Boot 版本 2.1.1 及一些工程基本信息，可参考下图



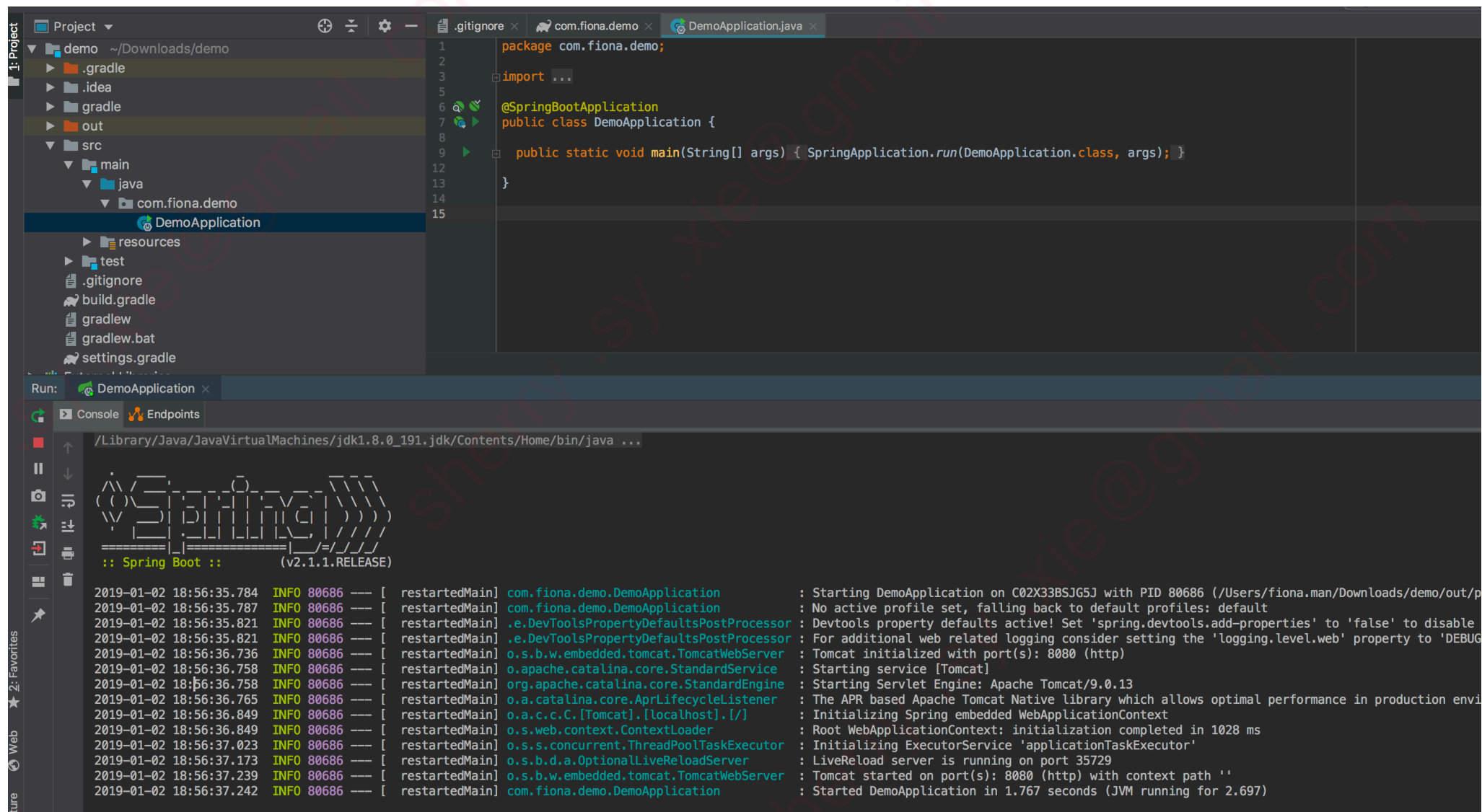
The screenshot shows the Spring Initializr web application. At the top, a dark banner reads "SPRING INITIALIZR bootstrap your application now". Below this, the main heading is "Generate a" followed by a dropdown menu set to "Gradle Project", then "with" followed by a dropdown menu set to "Java", and "and Spring Boot" followed by a dropdown menu set to "2.1.1".

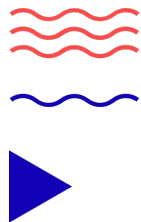
Under the heading, there are two main sections:

- Project Metadata:** This section contains two input fields. The first is labeled "Artifact coordinates" and has a sub-label "Group". The input field contains the text "com.fiona". The second input field is labeled "Artifact" and contains the text "demo".
- Dependencies:** This section contains a text input field labeled "Search for dependencies" with the placeholder text "Web, Security, JPA, Actuator, Devtools...". Below this, there is a section titled "Selected Dependencies" which shows three green buttons: "Web", "Lombok", and "DevTools", each with a small 'x' icon to its right.

At the bottom of the form, there is a large green button labeled "Generate Project" with a small icon of a gear, a plus sign, and a download arrow.

导入项目到IntelliJ，可以直接运行





命令行运行

- (1) ./gradlew build
- (2) ./gradlew tasks
- (3) ./gradlew test
- (4) ./gradlew bootJar
- (5) ./gradlew bootRun

```
drwxr-xr-x@ 12 fiona.man MYOB\Domain Users 384 2 Jan 18:56 .
drwx-----+ 29 fiona.man MYOB\Domain Users 928 2 Jan 16:00 ..
-rw-r--r--@ 1 fiona.man MYOB\Domain Users 276 1 Jan 21:29 .gitignore
drwxr-xr-x 4 fiona.man MYOB\Domain Users 128 2 Jan 08:31 .gradle
drwxr-xr-x 6 fiona.man MYOB\Domain Users 192 2 Jan 18:55 .idea
-rw-r--r--@ 1 fiona.man MYOB\Domain Users 712 1 Jan 21:29 build.gradle
drwxr-xr-x@ 3 fiona.man MYOB\Domain Users 96 1 Jan 21:29 gradle
-rwxr-xr-x@ 1 fiona.man MYOB\Domain Users 5296 1 Jan 21:29 gradlew
-rw-r--r--@ 1 fiona.man MYOB\Domain Users 2260 1 Jan 21:29 gradlew.bat
drwxr-xr-x 3 fiona.man MYOB\Domain Users 96 2 Jan 18:56 out
-rw-r--r--@ 1 fiona.man MYOB\Domain Users 26 1 Jan 21:29 settings.gradle
drwxr-xr-x@ 4 fiona.man MYOB\Domain Users 128 1 Jan 21:29 src
```


构建工具

(1) 为什么需要构建工具？

- a. 管理依赖
- b. 管理构建过程



maven



Apache Ant

XML
(build.xml)

Apache Maven

XML
(pom.xml)

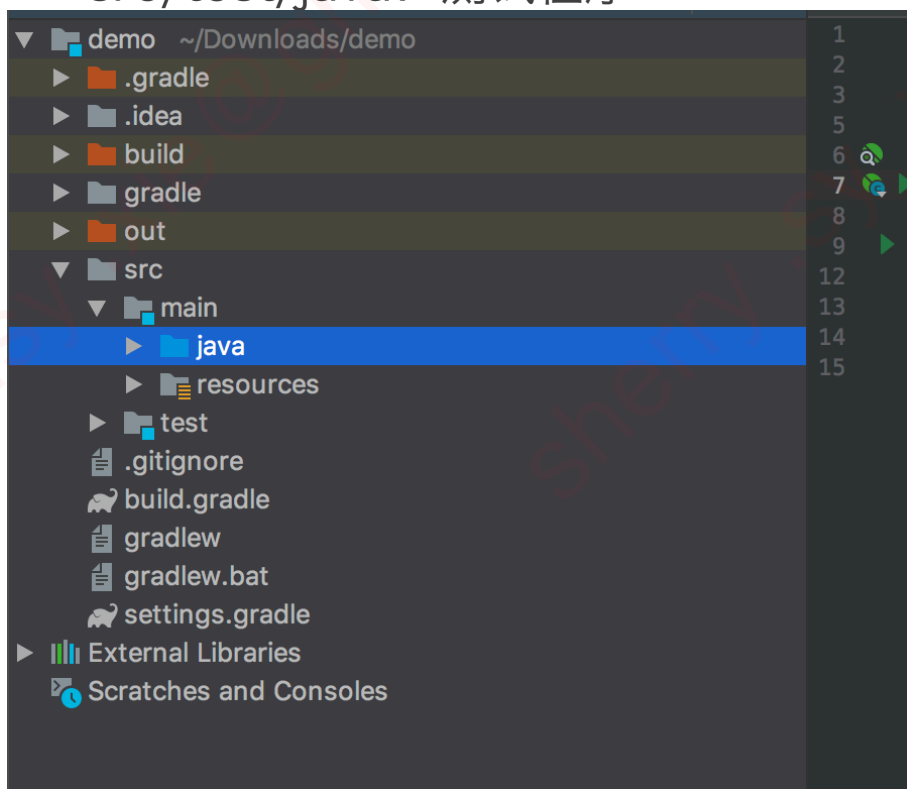
Gradle

Groovy DSL
(build.gradle)

代码目录结构

Spring Boot 的基础结构共三个目录：

- src/main/java: 程序开发以及主程序入口
- src/main/resources: 配置文件
- src/test/java: 测试程序



com.company.project 目录下：

- ProjectNameApplication.java: 建议放到根目录下面，是项目的启动类，Spring Boot 项目只能有一个 main() 方法。
- util/comm: 目录建议放置公共的类，如全局的配置文件、工具类等。
- domain/model/entity: 目录主要用于实体 (Entity) 与数据访问层 (Repository) 。
- repository: 数据库访问层代码。
- service: 该层主要是业务类代码。
- web/controller: 该层负责页面访问控制。
- factory, handler, builder...
- resources 目录下:
 - (1) static: 目录存放 Web 访问的静态资源，如 JS、CSS、图片等。
 - (2) templates: 目录存放页面模板。
 - (3) application.properties: 项目的配置信息。
- test 目录存放单元测试的代码

先来个RESTful API

(1) in build.gradle

```
implementation('org.springframework.boot:spring-boot-starter-web')
```

(2) Controller: @RestController 的意思就是 controller 里面的方法都以 json 格式输出

```
@RestController
public class HelloWorldController {
    @RequestMapping("/hello")
    public String hello() {
        return "Hello World";
    }
}
```

(3) 运行 并访问：<http://localhost:8080/hello>

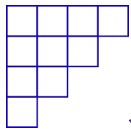


继续RESTful API

(1) 输入参数：

```
@RequestMapping("/hello")  
public String hello(String name) {  
    return "Hello " + name;  
}
```

(2) 运行并访问：<http://localhost:8080/hello?name=fiona>



单元测试

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class DemoApplicationTests {

    @Test
    public void contextLoads() {
    }

}
```

使用 mockmvc 进行 web 测试

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class HelloWorldControllerTest {
    private MockMvc mockMvc;

    @Before
    public void setUp() throws Exception {
        mockMvc = MockMvcBuilders.standaloneSetup(new
        HelloWorldController()).build();
    }

    @Test
    public void getHello() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.post("/hello?
        name=fiona").accept(MediaType.APPLICATION_JSON_UTF8)).andExpect(print(
        ));
    }
}
```

来个复杂一点点的RESTful API

```
@RestController
public class UserController {

    @GetMapping("/users/{id}")
    public User getUser(@PathVariable int id) {
        return new User(id, "user@test.com", "testPassword");
    }

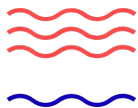
    @PostMapping("/users")
    @ResponseStatus(HttpStatus.CREATED)
    public User create(@Valid @RequestBody User user) {
        return new User(12344, user.getEmail(), user.getPassword());
    }

    @GetMapping(value = "/users")
    public User getByEmail(@RequestParam("email") String email) {
        return new User(123, email, "testPassword");
    }
}
```

```
@AllArgsConstructor
@NoArgsConstructor
@Data
public class User {
    private int id;
    @NotBlank
    private String email;
    @NotBlank
    private String password;
}
```

<http://localhost:8080/users/123>

<http://localhost:8080/users?email=test@test.com>



Controller 层错误处理

@ControllerAdvice + @ExceptionHandler 全局处理 Controller 层异常

@AllArgsConstructor

@NoArgsConstructor

@Data

```
public class ErrorResponse {  
    private String message;  
    private String details;  
}
```

@RestControllerAdvice

```
public class ControllerExceptionHandler {
```

```
    @ExceptionHandler(Exception.class)
```

```
    public ResponseEntity<ErrorResponse> handleBadRequest(Exception exception, WebRequest request) {  
        return new ResponseEntity(new ErrorResponse(exception.getMessage(), request.getDescription(false)),  
            HttpStatus.BAD_REQUEST);  
    }  
}
```

参数校验

参数校验在日常开发中非常常见，最基本的校验有判断属性是否为空、长度是否符合要求等，在传统的开发模式中需要写一堆的 if else 来处理这些逻辑，很繁琐，效率也低。

使用 @Valid + BindingResult 就可以优雅地解决这些问题。

```
public class User {  
    private int id;  
    @NotEmpty(message="Email cannot be empty")  
    private String email;  
    private String password;  
    @Max(value = 100, message = "Age should not be bigger than 100")  
    @Min(value = 18, message = "Age should be at least 18")  
    private int age;  
}
```

<http://localhost:8080/users>

```
{  
    "email": "",  
    "password": "testPassword",  
    "age": 12  
}
```

```
@PostMapping("/users")  
@ResponseStatus(HttpStatus.CREATED)  
public User create(@Valid @RequestBody User user) {  
    return new User(12344, user.getEmail(), user.getPassword(),  
        user.getAge());  
}
```


参数校验

```
@PostMapping("/users")
@ResponseStatus(HttpStatus.CREATED)
public User create(@Valid @RequestBody User user, BindingResult result) {
    System.out.println("user:" + user);
    if (result.hasErrors()) {
        List<ObjectError> list = result.getAllErrors();
        for (ObjectError error : list) {
            System.out.println(error.getCode() + "-" + error.getDefaultMessage());
        }
    }
    return new User(12344, user.getEmail(), user.getPassword(), user.getAge());
}
```

<http://localhost:8080/users>

```
{
  "email": "",
  "password":
    "testPassword",
  "age": 29
}
```

自定义Filter

Filters 通常用于用于记录请求日志、排除有 XSS 威胁的字符、执行权限验证等等。Spring Boot 自动添加了 OrderedCharacterEncodingFilter 和 HiddenHttpMethodFilter, 并且可以自定义 Filter。

自定义 Filter 两个步骤:

- 实现 Filter 接口, 实现 Filter 方法添加@Configuration 注解,
- 将自定义 Filter 加入过滤链

```
public class TestFilter implements Filter {  
    @Override  
    public void init(FilterConfig filterConfig) throws  
        ServletException {  
    }  
    @Override  
    public void doFilter(ServletRequest request,  
        ServletResponse response, FilterChain chain) throws  
        IOException, ServletException {  
        HttpServletRequest httpRequest = (HttpServletRequest)  
            request;  
        System.out.println("this is a Test Filter ,url :"+  
            httpRequest.getRequestURI());  
        chain.doFilter(request, response);  
    }  
    @Override  
    public void destroy() {  
    }  
}
```

自定义Filter

自定义 Filter 两个步骤:

- 实现 Filter 接口，实现 Filter 方法
添加@Configuration 注解，
- 将自定义 Filter 加入过滤链

@Configuration

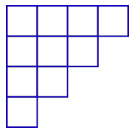
```
public class WebConfiguration {
```

@Bean

```
public RemoteIpFilter remotepFilter() {  
    return new RemoteIpFilter();  
}
```

@Bean

```
public FilterRegistrationBean testFilterRegistration() {  
    FilterRegistrationBean registration = new FilterRegistrationBean();  
    registration.setFilter(new TestFilter());  
    registration.addUrlPatterns("/");  
    registration.addInitParameter("paramName", "paramValue");  
    registration.setName("TestFilter");  
    registration.setOrder(1);  
    return registration;  
}
```



加点properties

配置在 application.properties 中

```
com.fiona.demo.className=JavaSpringBoot  
com.fiona.demo.description=description
```

```
@Component  
@ConfigurationProperties(prefix="com.fiona.demo")  
@Data  
public class DemoProperties {  
    private String className;  
    private String description;  
}
```

```
@RunWith(SpringRunner.class)  
@SpringBootTest  
public class DemoPropertiesTest {  
    @Resource  
    private DemoProperties properties;  
  
    @Test  
    public void testProperties() {  
        assertEquals("JavaSpringBoot", properties.getClassName());  
        assertEquals("description", properties.getDescription());  
    }  
}
```



匠人学院
jiangren.com.au

Q & A