一、开发web应用

Spring Boot Web 开发非常的简单，其中包括常用的 json 输出、filters、property、log 等

二、json接口开发

以前使用spring开发项目，需要提供json接口时需要做什么呢?

1、 添加jackjson等相关的jar包

2、配置spring controller扫描包

3、对接的方法添加 @ResponseBody

我们经常由于配置错误出现406错误，由于扫描包理解不到位的配置错误（SSM框架就遇到了，搞了好几天才解决掉！）等等，springboot如何做呢，只需要类添加 `@RestController` 即可，默认类中的方法都会以json的格式返回

```
@RestController
public class HelloController {
    @RequestMapping("/getUser")
    public User getUser() {
        User user=new User();
        user.setUserName("素小暖");
        user.setPassWord("xxxx");
        return user;
    }
}
```

如果需要使用页面开发只要使用`@Controller`注解即可，下面会结合模板来说明。

三、自定义filter

我们常常在项目中会使用filters用于调用日志、排除有XXS威胁的字符、执行权限验证等等。

springboot自动添加了 OrderedCharacterEncodingFilter 和 HiddenHttpMethodFilter， 并且我们可以自定义filter。

两个步骤

1、实现filter接口，实现filter方法

2、添加 @Configuration 注解，将自定义filter加入过滤链

```
@Configuration
public class WebConfiguration {
    @Bean
    public RemoteIpFilter remoteIpFilter() {
        return new RemoteIpFilter();
    }

    @Bean
    public FilterRegistrationBean testFilterRegistration() {

        FilterRegistrationBean registration = new FilterRegistrationBean();
        registration.setFilter(new MyFilter());
        registration.addUrlPatterns("/*");
```

```
        registration.addInitParameter("paramName", "paramValue");
        registration.setName("MyFilter");
        registration.setOrder(1);
        return registration;
    }

    public class MyFilter implements Filter {
            @Override
            public void destroy() {
                // TODO Auto-generated method stub
            }

            @Override
            public void doFilter(ServletRequest srequest, ServletResponse sresponse,
FilterChain filterChain)
                    throws IOException, ServletException {
                // TODO Auto-generated method stub
                HttpServletRequest request = (HttpServletRequest) srequest;
                System.out.println("this is MyFilter,url :"+request.getRequestURI());
                filterChain.doFilter(srequest, sresponse);
            }

            @Override
            public void init(FilterConfig arg0) throws ServletException {
                // TODO Auto-generated method stub
            }
    }
}
```

3、log配置

配置输出的地址和输出级别

```
logging.path=/user/local/log
logging.level.com.favorites=DEBUG
logging.level.org.springframework.web=INFO
logging.level.org.hibernate=ERROR
```

path 为本机的 log 地址，`logging.level` 后面可以根据包路径配置不同资源的 log 级别

五、数据库操作

这里我们重点讲述MySQL、spring data jpa的使用。jpa是利用hibernate生成各种自动化的sql，如果只是简单的增删改查，基本不用手写，spring内部已经封装好了。

下面介绍一下在springboot中的使用

1、添加相关jar包

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
 <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

2、添加配置文件

```
spring.datasource.url=jdbc:mysql://localhost:3306/test
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.jpa.properties.hibernate.hbm2ddl.auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.show-sql= true
```

Hibernate中的hibernate.hbm2ddl.auto 参数说明

hibernate.hbn2ddl.auto:是用于在操作数据库时是否创建表的取值：

- create：运行时创建表（若原始有表存在，则会删除表之后再创建表，再插入数据，说明只能插入最后一条数据，在插入多条数据时候不能使用）
- update：运行时创建表（若原始有表存在，不会删除表，直接进行数据更新）
- none：不创建表，若原始没有表，则会报错。

验证数据库表结构有四个值：

- create：每次加载hibernate时都会删除上一次生成的表，然后根据你的model类再重新来生成新表，哪怕；两次没有任何改变也要这样执行，这就是导致了数据库表数据丢失的一个重要原因。
- create-drop：每次加载hibernate时根据model类生成表，但是sessionFactory一关闭，表就自动删除了。
- update：最常用的属性，第一次加载hibernate时根据model类会自动建立标的结构，以后加载hibernate时根据model类自动更新表结构，即使表结构改变了但表中的行仍然存在不会删除之前的行。要注意的是当部署到服务器后，表结构是不会被马上建立起来的，是要等应用第一次运行起来后才会。
- validate：每次加载hibernate时，验证创建数据库表结构，只会和数据库中的表践行比较，不会创建新表，但是会插入新值。

`dialect` 主要是指定生成表名的存储引擎为 InnoDBD

`show-sql` 是否打印出自动生成的 SQL，方便调试的时候查看

3、添加实体类和dao

```java
@Entity
public class User implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue
    private Long id;
    @Column(nullable = false, unique = true)
    private String userName;
    @Column(nullable = false)
    private String passWord;
    @Column(nullable = false, unique = true)
    private String email;
    @Column(nullable = true, unique = true)
    private String nickName;
    @Column(nullable = false)
    private String regTime;

    //省略getter settet方法、构造方法

}
```

复制

dao 只要继承 JpaRepository 类就可以，几乎可以不用写方法，还有一个特别有尿性的功能非常赞，就是可以根据方法名来自动的生成 SQL，比如findByUserName 会自动生成一个以 userName 为参数的查询方法，比如 findA111 自动会查询表里面的所有数据，比如自动分页等等。。

Entity中不应是成列的字段得加 @Transient 注解，不加注解也会映射成列

```
public interface UserRepository extends JpaRepository<User, Long> {
    User findByUserName(String userName);
    User findByUserNameOrEmail(String username, String email);
}
```

测试：

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(Application.class)
public class UserRepositoryTests {

        @Autowired
        private UserRepository userRepository;

        @Test
        public void test() throws Exception {
                Date date = new Date();
                DateFormat dateFormat =
DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG);
                String formattedDate = dateFormat.format(date);

                userRepository.save(new User("aa1", "aa@126.com", "aa",
"aa123456",formattedDate));
                userRepository.save(new User("bb2", "bb@126.com", "bb",
"bb123456",formattedDate));
                userRepository.save(new User("cc3", "cc@126.com", "cc",
"cc123456",formattedDate));
```

```
        Assert.assertEquals(9, userRepository.findAll().size());
        Assert.assertEquals("bb", userRepository.findByUserNameOrEmail("bb",
"cc@126.com").getNickName());
        userRepository.delete(userRepository.findByUserName("aa1"));
    }

}
```

六、Thymeleaf 模板

Spring Boot 推荐使用 Thymeleaf 来代替 Jsp。

1、Thymeleaf介绍

Thymeleaf是一款用于渲染 XML/XHTML/HTML5 内容的模板引擎。 类似于JSP。它可以轻易的与springMVC框架进行集成作为web应用的模板引擎。与其他模板引擎相比， Thymeleaf 最大的特点是能够直接在浏览器中打开并正确显示模板页面，而不需要启动整个web应用。

Thymeleaf 使用了自然的模板技术，这意味着 Thymeleaf 的模板语法不会破坏文档的结构，模板依旧是有效的XML文档。模板还可以用作工作原型， Thymeleaf 会在运行期替换掉静态值。

URL 在 Web 应用模板中占据着十分重要的地位，需要特别注意的是 Thymeleaf 对于 URL 的处理是通过语法 @{...} 来处理的。Thymeleaf 支持绝对路径 URL：

```
<a th:href="@{http://www.thymeleaf.org}">Thymeleaf</a>
```

条件求值

```
<a th:href="@{/login}" th:unless=${session.user != null}>Login</a>
```

for循环

```
<tr th:each="prod : ${prods}">
      <td th:text="${prod.name}">Onions</td>
      <td th:text="${prod.price}">2.41</td>
      <td th:text="${prod.inStock}? #{true} : #{false}">yes</td>
</tr>
```

2、页面即原型

在传统 Java Web 开发过程中，前端工程师和后端工程师一样，也需要安装一套完整的开发环境，然后各类 Java IDE 中修改模板、静态资源文件，启动/重启/重新加载应用服务器，刷新页面查看最终效果。

但实际上前端工程师的职责更多应该关注页面，使用JSP很难做到这点，因为JSP必须在应用服务器中渲染完成后才能在浏览器中看到效果，而 Thymeleaf从跟不上解决了这个问题，通过属性进行模板渲染不会引入任何新的浏览器不能识别的标签，例如JSP中，不会再Tag内部写表达式。 整个页面直接作为 HTML 文件用浏览器打开，几乎就可以看到最终的效果，这大大解放了前端工程师的生产力，它们的最终交付物就是纯的 HTML/CSS/JavaScript 文件。

七、Gradle构建工具

spring项目建议使用 Maven/Gradle 进行构建项目，相比Maven来讲Gradle更简洁，而且Gradle更适合大型复杂项目的构建。Gradle吸收了maven和ant的特点，不过目前maven仍然是java界的主流，先了解一下嘛。

一个使用 Gradle 配置的项目：

```
buildscript {
    repositories {
        maven { url "http://repo.spring.io/libs-snapshot" }
        mavenLocal()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-
plugin:1.3.6.RELEASE")
    }
}

apply plugin: 'java'  //添加 Java 插件, 表明这是一个 Java 项目
apply plugin: 'spring-boot' //添加 Spring-boot支持
apply plugin: 'war'  //添加 War 插件, 可以导出 War 包
apply plugin: 'eclipse' //添加 Eclipse 插件, 添加 Eclipse IDE 支持, Intellij Idea 为 "idea"

war {
    baseName = 'favorites'
    version =  '0.1.0'
}

sourceCompatibility = 1.7  //最低兼容版本 JDK1.7
targetCompatibility = 1.7  //目标兼容版本 JDK1.7

repositories {     //  Maven 仓库
    mavenLocal()        //使用本地仓库
    mavenCentral()      //使用中央仓库
    maven { url "http://repo.spring.io/libs-snapshot" } //使用远程仓库
}

dependencies {   // 各种 依赖的jar包
    compile("org.springframework.boot:spring-boot-starter-web:1.3.6.RELEASE")
    compile("org.springframework.boot:spring-boot-starter-
thymeleaf:1.3.6.RELEASE")
    compile("org.springframework.boot:spring-boot-starter-data-jpa:1.3.6.RELEASE")
    compile group: 'mysql', name: 'mysql-connector-java', version: '5.1.6'
    compile group: 'org.apache.commons', name: 'commons-lang3', version: '3.4'
    compile("org.springframework.boot:spring-boot-devtools:1.3.6.RELEASE")
    compile("org.springframework.boot:spring-boot-starter-test:1.3.6.RELEASE")
    compile 'org.webjars.bower:bootstrap:3.3.6'
        compile 'org.webjars.bower:jquery:2.2.4'
```

```
    compile("org.webjars:vue:1.0.24")
        compile 'org.webjars.bower:vue-resource:0.7.0'

}

bootRun {
    addResources = true
}
```

八、 WebJars

WebJars 是一个很神奇的东西，可以让大家以jar包的形式来使用前端的各种框架、组件。

1、什么是 WebJars

WebJars是将客户端资源打包成jar包文件，以对资源进行统一依赖管理。 WebJars 的jar包部署在maven中央仓库上。

2、为什么使用

我们在开发java web项目的时候会使用Maven、 Gradle等构建工具以实现对jar包版本依赖管理，以及项目的自动化管理，但是对于JavaScript、css等前端资源包，我们只能采用拷贝到webapp下的方式，这样做就无法对这些资源进行依赖管理，那么WebJars就提供给我们这些前端资源的jar包资源，我们就可以进行依赖管理了。

3、如何使用

（1）添加依赖

```
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>vue</artifactId>
    <version>2.5.16</version>
</dependency>
```

（2）页面引入

```
<link th:href="@{/webjars/bootstrap/3.3.6/dist/css/bootstrap.css}" rel="stylesheet"></link>
```

扩展SpringMVC的功能

```
//使用WebMvcConfigurerAdapter可以来扩展SpringMVC的功能
//@EnableWebMvc   不要接管SpringMVC
@Configuration
public class MyMvcConfig extends WebMvcConfigurerAdapter {
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        // super.addViewControllers(registry);
        //浏览器发送 /atguigu 请求来到 success
        registry.addViewController("/atguigu").setViewName("success");
    }

    //所有的WebMvcConfigurerAdapter组件都会一起起作用
    @Bean //将组件注册在容器
```

```
    public WebMvcConfigurerAdapter webMvcConfigurerAdapter(){
        WebMvcConfigurerAdapter adapter = new WebMvcConfigurerAdapter() {
            @Override
            public void addViewControllers(ViewControllerRegistry registry) {
                registry.addViewController("/").setViewName("login");
                registry.addViewController("/index.html").setViewName("login");
                registry.addViewController("/main.html").setViewName("dashboard");
            }

            //注册拦截器
            @Override
            public void addInterceptors(InterceptorRegistry registry) {
                //super.addInterceptors(registry);
                //静态资源；  *.css , *.js
                //SpringBoot已经做好了静态资源映射
//                registry.addInterceptor(new
LoginHandlerInterceptor()).addPathPatterns("/**")
//                        .excludePathPatterns("/index.html","/","/user/login");
            }
        };
        return adapter;
    }

    @Bean
    public LocaleResolver localeResolver(){

        return new MyLocaleResolver();
    }

}
```

controller

```
package com.springboot.controller;

import com.springboot.dao.DepartmentDao;
import com.springboot.dao.EmployeeDao;
import com.springboot.entities.Department;
import com.springboot.entities.Employee;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

@Controller
public class EmployeeController {
```

```
    @Autowired
    EmployeeDao employeeDao;

    @Autowired
    DepartmentDao departmentDao;

    //查询所有员工返回列表页面
    @GetMapping("/emps")
    public String  list(Model model){
        Collection<Employee> employees = employeeDao.getAll();

        //放在请求域中
        model.addAttribute("emps",employees);
        // thymeleaf默认就会拼串
        // classpath:/templates/xxxx.html
        return "emp/list";
    }

    //来到员工添加页面
    @GetMapping("/emp")
    public String toAddPage(Model model){
        //来到添加页面,查出所有的部门，在页面显示
        Collection<Department> departments = departmentDao.getDepartments();
        model.addAttribute("depts",departments);
        return "emp/add";
    }

    //员工添加
    //SpringMVC自动将请求参数和入参对象的属性进行一一绑定；要求请求参数的名字和
javaBean入参的对象里面的属性名是一样的
    @PostMapping("/emp")
    public String addEmp(Employee employee){
        //来到员工列表页面

        System.out.println("保存的员工信息："+employee);
        //保存员工
        employeeDao.save(employee);
        // redirect: 表示重定向到一个地址  /代表当前项目路径
        // forward: 表示转发到一个地址
        return "redirect:/emps";
    }

    //来到修改页面，查出当前员工，在页面回显
    @GetMapping("/emp/{id}")
    public String toEditPage(@PathVariable("id") Integer id,Model model){
        Employee employee = employeeDao.get(id);
        model.addAttribute("emp",employee);
```

```
        //页面要显示所有的部门列表
        Collection<Department> departments = departmentDao.getDepartments();
        model.addAttribute("depts",departments);
        //回到修改页面(add是一个修改添加二合一的页面);
        return "emp/add";
    }

    //员工修改；需要提交员工id；
    @PutMapping("/emp")
    public String updateEmployee(Employee employee){
        System.out.println("修改的员工数据："+employee);
        employeeDao.save(employee);
        return "redirect:/emps";
    }

    //员工删除
    @DeleteMapping("/emp/{id}")
    public String deleteEmployee(@PathVariable("id") Integer id){
        employeeDao.delete(id);
        return "redirect:/emps";
    }

}
```

```
<div class="container-fluid">
  <div class="row">
        <!--引入侧边栏-->
        <div th:replace="commons/bar::#sidebar(activeUri='emps')"></div>

        <main role="main" class="col-md-9 ml-sm-auto col-lg-10 pt-3 px-4">
            <h2><a class="btn btn-sm btn-success" href="emp" th:href="@{/emp}">
员工添加</a></h2>
            <div class="table-responsive">
              <table class="table table-striped table-sm">
                <thead>
                    <tr>
                        <th>#</th>
                        <th>lastName</th>
                        <th>email</th>
                        <th>gender</th>
                        <th>department</th>
                        <th>birth</th>
                        <th>操作</th>
                    </tr>
                </thead>
```

```
                <tbody>
                    <tr th:each="emp:${emps}">
                        <td th:text="${emp.id}"></td>
                        <td>[[${emp.lastName}]]</td>
                        <td th:text="${emp.email}"></td>
                        <td th:text="${emp.gender}==0?'女':'男'"></td>
                        <td th:text="${emp.department.departmentName}"></td>
                        <td th:text="${#dates.format(emp.birth, 'yyyy-MM-dd
HH:mm')}"></td>
                        <td>
                          <a class="btn btn-sm btn-primary"
th:href="@{/emp/}+${emp.id}">编辑</a>
                          <button th:attr="del_uri=@{/emp/}+${emp.id}"
class="btn btn-sm btn-danger deleteBtn">删除</button>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
    </main>
    <form id="deleteEmpForm"  method="post">
        <input type="hidden" name="_method" value="delete"/>
    </form>
  </div>
</div>
```

jquery

```
<script>
    $(".deleteBtn").click(function(){
        //删除当前员工的
        $("#deleteEmpForm").attr("action",$(this).attr("del_uri")).submit();
            return false;
    });
</script>
```