Web开发进阶

教学内容

- 第一节 静态资源访问
- 第二节 文件上传
- 第三节 拦截器

静态资源访问

- 使用IDEA创建Spring Boot项目，会默认创建出classpath:/static/目录，静态资源一般放在这个目录下即可。
- 如果默认的静态资源过滤策略不能满足开发需求，也可以自定义静态资源过滤策略。
- 在application.properties中直接定义过滤规则和静态资源位置：

```
spring.mvc.static-path-pattern=/static/**
spring.web.resources.static-locations=classpath:/static/
```

- 过滤规则为/static/**，静态资源位置为classpath:/static/

文件上传原理

- 表单的enctype 属性规定在发送到服务器之前应该如何对表单数据进行编码。
- 当表单的enctype="application/x-www-form-urlencoded"（默认）时，form表单中的数据格式为：key=value&key=value
- 当表单的enctype="multipart/form-data"时，其传输数据形式如下

SpirngBoot实现文件上传功能

- Spring Boot工程嵌入的tomcat限制了请求的文件大小，每个文件的配置最大为1Mb，单次请求的文件的总数不能大于10Mb。
- 要更改这个默认值需要在配置文件（如application.properties）中加入两个配置

```
spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=10MB
```

SpirngBoot实现文件上传功能

- 当表单的enctype="multipart/form-data"时,可以使用MultipartFile 获取上传的文件数据，再通过transferTo方法将其写入到磁盘中

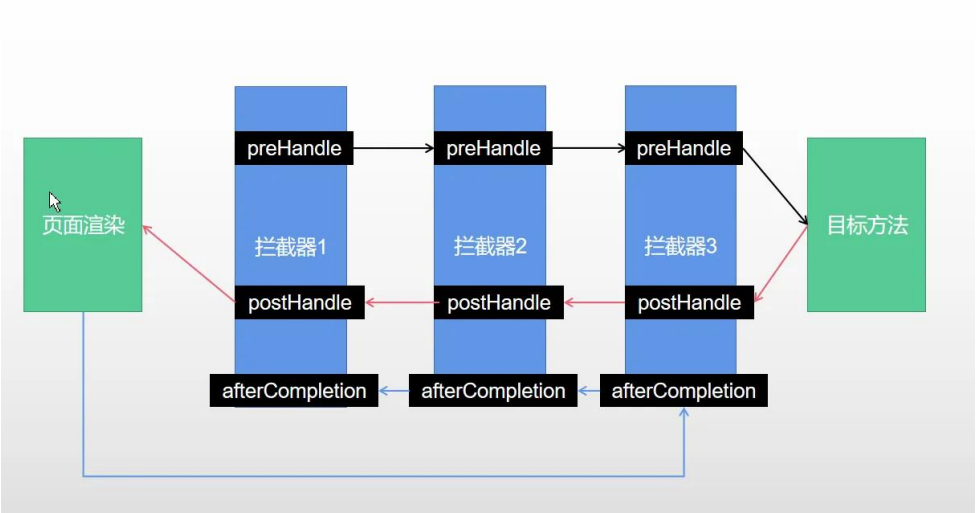
```
@RestController
public class FileController {
    private static final String UPLOADED_FOLDER = System.getProperty("user.dir")+"/upload/";

    @PostMapping("/up")
    public String upload(String nickname,MultipartFile f) throws IOException {
        System.out.println("文件大小:"+f.getSize());
        System.out.println(f.getContentType());
        System.out.println(f.getOriginalFilename());
        System.out.println(System.getProperty("user.dir"));
        saveFile(f);
        return "上传成功";
    }
    public void saveFile(MultipartFile f) throws IOException {
        File upDir = new File(UPLOADED_FOLDER);
        if(!upDir.exists()){
            upDir.mkdir();
        }
        File file = new File(UPLOADED_FOLDER+f.getOriginalFilename());
        f.transferTo(file);
    }
}
```

拦截器

- 拦截器在Web系统中非常常见，对于某些全局统一的操作，我们可以把它提取到拦截器中实现。总结起来，拦截器大致有以下几种使用场景：
- 权限检查：如登录检测，进入处理程序检测是否登录，如果没有，则直接返回登录页面。
- 性能监控：有时系统在某段时间莫名其妙很慢，可以通过拦截器在进入处理程序之前记录开始时间，在处理完后记录结束时间，从而得到该请求的处理时间
- 通用行为：读取cookie得到用户信息并将用户对象放入请求，从而方便后续流程使用，还有提取Locale、Theme信息等，只要是多个处理程序都需要的，即可使用拦截器实现。

拦截器

- Spring Boot定义了HandlerInterceptor接口来实现自定义拦截器的功能
- HandlerInterceptor接口定义了preHandle、postHandle、afterCompletion三种方法，通过重写这三种方法实现请求前、请求后等操作



拦截器定义

```
public class LoginInterceptor extends HandlerInterceptor {
    /**
     * 在请求处理之前进行调用（Controller方法调用之前）
     */
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
throws Exception {
        if (条件) {
            System.out.println("通过");
            return true;
        }else{
             System.out.println("不通过");
            return false;
        }
    }
}
```

拦截器注册

- addPathPatterns方法定义拦截的地址
- excludePathPatterns定义排除某些地址不被拦截
- 添加的一个拦截器没有addPathPattern任何一个url则默认拦截所有请求
- 如果没有excludePathPatterns任何一个请求，则默认不放过任何一个请求。

```
@Configuration
public class WebConfigurer implements WebMvcConfigurer {
     @Override
     public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor( new LoginInterceptor()).addPathPatterns("/user/**");
     }
}
```