基于Spring Boot的Web层服务开发实践

饿了么技术&创新中心

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

https://projects.spring.io/spring-boot/

Spring Boot的设计目标

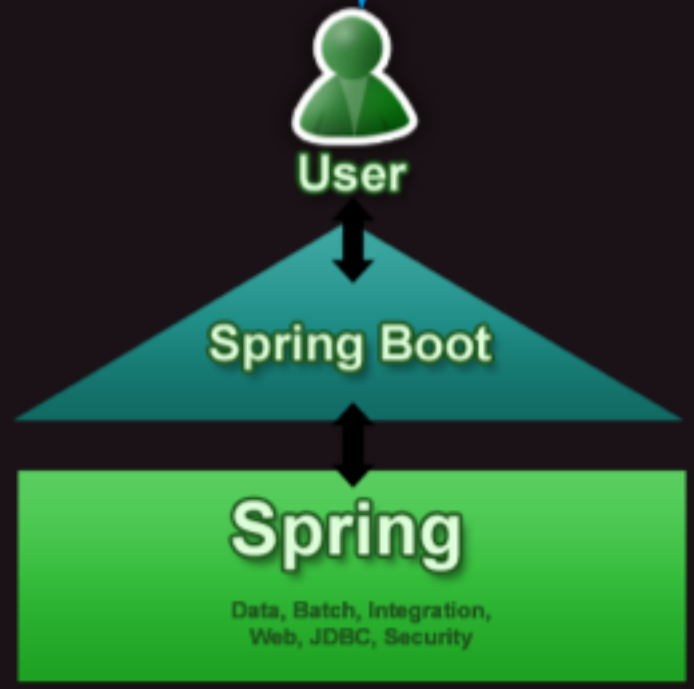
使Spring Framework的应用开发变得简单，更容易上手

非侵入性提供一套常用的配置，但是用户可随时覆盖

提供更多的基础性、非业务的功能（内置Web容器、权限认证机制、监控、应用配置管理等等）

完全不依赖XML配置

Spring Boot 不能取代 Spring Framework。想要用好Spring Boot，必须对Spring Framework有足够的知识。



基于Spring Boot的应用：

独立可执行Jar

内置Servlet容器(Jetty/Tomcat/Undertow)，不用部署war包

自动化配置机制，对常见的Spring基础组件(Data、Security、MVC、MQ...) 提供“即开即用”的默认配置

Cloud Friendly

- 可定制的脚本，适配init.d/systemd
- External Configuration 灵活的配置注入机制
- Actuator 应用管理和监控

嵌入式WEB容器原理

AnnotationConfigEmbeddedWebApplicationContext

- 用来启动EmbeddedServletContainer
- 自动注册Servlet和Filter类型的Bean
- 定制注册过程：ServletRegistrationBean/FilterRegistrationBean

EmbeddedServletContainerAutoConfiguration

- Jetty
- Tomcat
- Undertow

自动化配置
“org.springframework.boot.autoconfigure.web.*AutoConfiguration”

基于 Token的认证机制

根据任意一种认证手段（用户名+密码，手机号+验证码等等）生成Token

更适用于移动端(APP)

无状态, Client Side Session

推荐标准 JWT (http://jwt.io)

应用场景

移动端用户认证解决方案

RESTful API开发

替代传统的基于Session的用户认证登录机制

饿了么 IT大咖说 知识分享平台

Talk is cheap.
Show me the code.

LINUS TORVALDS

Spring Security 配置


```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    SecurityConfig() {
        super(true); //不让Spring Security进行默认的配置
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //不加入默认的认证规则 anyRequest().authenticated()
        http.headers().defaultsDisabled()
                        .cacheControl();//加入Cache相关HTTP头，禁用浏览器缓存
        http.formLogin().disable() //不要UsernamePasswordAuthenticationFilter
                .httpBasic().disable() //不要BasicAuthenticationFilter
                .sessionManagement().sessionCreationPolicy(SessionCreationPo
licy.STATELESS).securityContext().and()
                .servletApi();
        http.addFilterBefore(new TokenAuthenticationFilter(),
UsernamePasswordAuthenticationFilter.class);
    }
}
```

Token认证Filter的实现示意代码

Token认证Filter的实现示意代码


```java
public class TokenAuthenticationFilter extends OncePerRequestFilter {
    public static final String X_AUTHENTICATION_TOKEN = "X-Authentication-Token";
    private RequestMatcher matcher = new AntPathRequestMatcher("/login/**");

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain
filterChain) throws ServletException, IOException {

        if (matcher.matches(request)) {
            //创建Token后，不需要继续执行
            response.addHeader(X_AUTHENTICATION_TOKEN, "1234");
            return;

        }

        if (SecurityContextHolder.getContext().getAuthentication() != null) {
            filterChain.doFilter(request, response);
            //已经完成认证
            return;
        }

        //获取Token
        String token = request.getHeader(X_AUTHENTICATION_TOKEN);
        //校验Token
        if ("1234".equals(token)) {
            List<GrantedAuthority> authorities = new ArrayList<>();
            authorities.add(new SimpleGrantedAuthority("ADMIN"));
            UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken("John", "", authorities);
            SecurityContextHolder.getContext().setAuthentication(authenticationToken);
            filterChain.doFilter(request, response);
        } else {
            throw new BadCredentialsException("Token认证失败！");
        }
    }
}
```

创建Token

校验Token

一个简单的演示

思考

思考

如何强制收回一个有效的TOKEN?

TOKEN在客户端（浏览器或APP）如何保存?

TOKEN有效期设置多长合适?

TOKEN的缺点?

Spring Boot是打开Spring Cloud大门的🔑

饿了么 | IT大咖说 知识分享平台