

# Security

## Spring Boot - Security

- Spring Security가 클래스패스에 있다면, 웹 어플리케이션은 기본값으로 보안처리가 된다
- 스프링 부트는 스프링 시큐리티의 content-negotiation strategy 를 따른다
- 이로써 httpBasic 과 formLogin 방식 중 하나를 채택한다
- 메소드 레벨의 보안을 적용하기 위해서, @EnableGlobalMethodSecurity 어노테이션을 사용할 수 있다.

```
@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true, jsr250Enabled = true)
public class MethodSecurity {

}
```

```
@Secured("ROLE_USER")
public void dashboard() {
    // ...
}
```

## Default user

- 기본적으로 UserDetailsService 는 유저를 하나 만들어주는데, username 은 user 이고, 패스워드는 랜덤으로 생성되어 앱 시작 시기에 콘솔에 INFO 레벨의 로그로 출력된다.

```
Using generated security password: 78fa095d-3f4c-48b1-ad50-e24c31d5cf35
```

- 만약 로그 설정을 따로 적용중이라면 [org.springframework.boot.autoconfigure.security](#) 카테고리를 INFO 레벨로 설정해야만 기본적으로 만들어지는 패스워드가 출력된다.
- name, password 값 설정하는 법

```
# application.yml
spring:
  security:
    user:
      name: username
      password: password
```

## 웹 어플리케이션 디폴트 feature

- `UserService` or (웹플렉스 사용중이라면) `ReactiveUserService` 빈은 인메모리 저장 방식으로 유저를 저장
- `SecurityProperties.User`

```
public static class User {  
  
    /**  
     * Default user name.  
     */  
    private String name = "user";  
  
    /**  
     * Password for the default user name.  
     */  
    private String password = UUID.randomUUID().toString();  
  
    /**  
     * Granted roles for the default user name.  
     */  
    private List<String> roles = new ArrayList<>();  
  
    private boolean passwordGenerated = true;  
  
    // getter  
  
    // setter  
  
}
```


- Form-based 로그인이나 HTTP Basic 보안방식을 사용한다
- 전체 어플리케이션 단위, 액츄에이터가 클래스패스에 있다면 액츄에이터까지 보안이 적용된다
- `DefaultAuthenticationEventPublisher` 가 인증 이벤트를 처리한다
- 이벤트를 커스터마이징하게 처리하고 싶으면 `AuthenticationEventPublisher` 를 사용할 수 있다.

```
public interface AuthenticationEventPublisher {  
  
    void publishAuthenticationSuccess(Authentication authentication);  
  
    void publishAuthenticationFailure(AuthenticationException exception, Authentication authentication);  
  
}
```

## 참고 자료


### Spring Boot Features

Graceful shutdown is supported with all four embedded web servers (Jetty, Reactor Netty, Tomcat, and Undertow) and with both reactive and Servlet-based web applications. It occurs as part of closing the application context and is performed in the earliest phase of stopping SmartLifecycle beans.

 <https://docs.spring.io/spring-boot/docs/2.4.3/reference/html/spring-boot-features.html#boot-features-security>

### Spring Security 기초


Spring Security는 애플리케이션의 보안을 쉽게 구성할 수 있게 도와주는 프레임워크이다. 인증(Authentication)과 인가(Authorization)를 담당한다. 스프링 시큐리티는 이러한 인증과 인가를 Filter 를 이용하여 구현한다. 이러한 인증의 흐름을 그림으로 표현하면

 <https://minkukjo.github.io/framework/2021/01/10/Spring-Security-01/>



### Spring Security의 내부

Security Context Persistence Filter 가 Session에 Security Context가 존재하는지 확인한다. 다음으로는 Logout Filter 로 이동하는데 아무것도 하지 않는다. 다음은 Authentication Filter 로 이동한다. 유저는 로그인 하지 않았으므로 이 필터 역시 아무

 <https://minkukjo.github.io/framework/2021/01/10/Spring-Security-02/>



### Delegating Filter Proxy 탐험


Delegating Filter Proxy 가 정말로 이론으로 배운 것 처럼 구현되어있는지 내부를 구경해보자. GenericFilterBean 을 상속받고 있는데 이전 시간에도 언급한적이 있지만 이 필터는 스프링의 설정 정보를 포함하는 필터이자 추상 클래스이다. 지금부터 IntelliJ IDE의 디

 <https://minkukjo.github.io/framework/2021/01/15/Spring-Security-03/>



### Spring Security AutoConfigurations의 이해와 커스터마이징

우리는 이전 시간까지 스프링 시큐리티의 내부가 어떻게 동작하는지를 이해했다. 이번 시간에는 스프링 시큐리티의 자동 설정이 어떻게 이루어지는가를 살펴보고자 한다. 위 경로로 들어가보면 자동 설정이 엄청나게 많은데, 이것들이 바로 스프링에서 자동 설정해주는

 <https://minkukjo.github.io/framework/2021/01/16/Spring-Security-04/>



# Graceful shutdown

## Graceful shutdown

- 내장 웹 서버 (Jetty, Reactor Netty, Tomcat, Undertow)
- Reactive, 서블릿 베이스의 웹 어플리케이션
- 어플리케이션 컨텍스트를 닫아주게 되는데 `SmartLifecycle` 빈을 멈추는 가장 첫번째 단계에서 수행된다.
- 이 정지 과정은 timeout 을 사용해서 grace period 를 제공한다.
- 이 grace period 기간동안 존재하는 요청들은 모두 완벽하게 수행되지만 새로운 요청들은 더 이상 받지 않는다.
- 새로운 요청들을 수락하지 않는 방식은 사용되는 웹 서버마다 다르다
  - Jetty, Reactor Netty, Tomcat 은 네트워크 레이어에서 요청을 받아오지 않는다
  - Undertow 는 요청을 받아는 오되 즉각적으로 `503 service unavailable response` 를 응답한다
- IDE 에서 작업시, 적절한 SIGTERM signal 을 보내지 않는다면 graceful shutdown가 정상적으로 작동하지 않을 수 있다. → IDE의 레퍼런스 참고할 것!

## 관련 property

- To enable graceful shutdown

```
server:  
  shutdown: graceful
```


- To configure the timeout period

```
spring:  
  lifecycle:  
    timeout-per-shutdown-phase: 20s
```

## 참고 자료


### Spring Boot Features

Graceful shutdown is supported with all four embedded web servers (Jetty, Reactor Netty, Tomcat, and Undertow) and with both reactive and Servlet-based web applications. It occurs as part of closing the application context and is performed in the earliest phase of stopping SmartLifecycle beans.

 <https://docs.spring.io/spring-boot/docs/2.4.3/reference/html/spring-boot-features.html#boot-features-graceful-shutdown>

### Spring boot Graceful Shutdown

시작하며 보통 애플리케이션에서 변경 사항이 있으면, 애플리케이션을 재 시작 해야 변경사항이 반영된다. 물론 배포 방식에 따라 Blue-Green 배포 형식을 따르게 되면, 이전 버전 앱을 굳이 죽일 필요는 없지만, Rolling

 <https://bravenamme.github.io/2020/10/06/graceful-shutdown/>



# Caching

## Caching

- 스프링 프레임워크는 투명한 캐시를 제공한다  
투명한 캐시? <https://moon-seung-chan.tistory.com/17>
- 스프링 코어의 추상화가 메소드에 캐시를 적용한다  
→ 많은 execution 이 줄어드는 효과가 있다.
- 스프링 부트는 @EnableCaching 어노테이션을 사용해서 캐시를 지원가능하게 만들어 캐시 인프라를 자동 설정한다.

## Code

```
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Component;

@Component
public class MathService {

    @Cacheable("piDecimals")
    public int computePiDecimal(int i) {
        // ...
    }

}
```


- 메소드에 @Cacheable, @CacheEvict, @CachePut 과 같은 어노테이션을 붙일 수 있다.
- `computePiDecimal()` 은 값비싼 연산을 수행하는 메소드이다. 이 경우 캐시를 적용해서 성능상 이점을 볼 수 있다. `computePiDecimal()` 내부 코드가 호출되기 전에 `piDecimals` 캐시에서 아규먼트 `i` 에 해당하는 캐시 엔트리를 찾아보고 엔트리가 찾아지면 메소드를 호출하는 것 대신에 캐시 데이터를 즉각적으로 응답 해 준다. 만약 캐시 엔트리가 없다면 메소드를 호출하고 값을 응답하기 전에 캐시를 업데이트 해준다.
- 만약 특정한 캐시 라이브러리를 추가하지 않는다면 스프링 부트는 simple provider 를 사용하도록 자동 설정한다. 그 외 캐시 라이브러리로 JCache (JSR-107), EhCache, Hazelcast, Infinispan, Couchbase, Redis, Caffeine 과 같은 캐시 프로파이드를 사용할 수 있다.
- 캐시가 필요한 경우 이런 프로바이더가 캐시를 만들어 준다.
- Simple provider 는 인메모리에서 concurrent map 을 사용한다. production 레벨에서 simple provider 를 사용하는 것을 별로 추천하지 않지만 스프링 부트의 캐시 feature 를 이해하기 위한 getting started 용으로는 좋다

- Simple provider 대신에 특정 캐시 프로바이더를 사용하기로 결정했으면, 관련 documentation을 읽어라. 그래서 어떻게 앱 내에서 설정하는지 알아봐야 한다. 거의 모든 캐시 프로바이더는 explicitly 한 캐시 관련 설정이 필요하다.
- 디폴트 캐시 커스터마이징 프로퍼티: `spring.cache.cache-names`

## 참고 자료

### Spring Boot Features

Graceful shutdown is supported with all four embedded web servers (Jetty, Reactor Netty, Tomcat, and Undertow) and with both reactive and Servlet-based web applications. It occurs as part of closing the application context and is performed in the earliest phase of stopping SmartLifecycle beans.

 <https://docs.spring.io/spring-boot/docs/2.4.3/reference/html/spring-boot-features.html#boot-features-caching>

### A Guide To Caching in Spring | Baeldung

In this tutorial, we're going to learn how to use the Caching Abstraction in Spring, and generally improve the performance of our system. We'll enable simple caching for some real-world method examples, and we'll discuss how we


 <https://www.baeldung.com/spring-cache-tutorial>

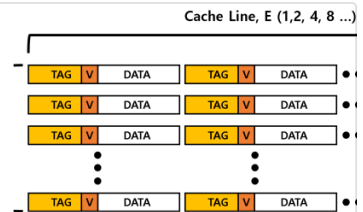


스프링 부트에서 Caching 사용하기

### 캐시 메모리의 구조(Cache Structure)와 캐시 히트(Cache Hit), 캐시 미스(Cache Miss)

월리 · 2021. 3. 28. 13:47 캐시 메모리는 S개의 집합(Set)으로 묶여있으며, 각 집합은 E개 캐시 엔트리를 가지고 있는 캐시라인(Cache Line)으로 구성된다. 필요한 캐시 메모리의 크기는 메모리에 존재하는 캐시 블록의 전체 개수(Blocks)와 블록 하나당의 크기(Block size)로 계산한다 여기서 기억해야


 <https://blog.naver.com/PostView.nhn?blogId=cjsk3113&logNo=222290234374&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView>

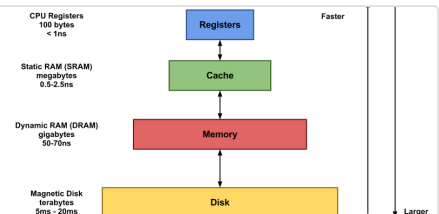


캐시 메모리 구조, 캐시 히트, 캐시 미스

### 캐시 메모리란?

데이터를 저장하는 공간의 속도와 용량은 반비례 관계이다. 대체적으로 속도가 빠른 메모리는 저장공간이 작으며, 느린 메모리는 저장공간이 큰 특징을 가진다. 속도와 공간 두마리 토끼를 잡기에는 비용이 너무 커진다. 그렇기에 데이터 저장 공간은 속도와 용량에 따라 각자 특성에

 <https://moon-seung-chan.tistory.com/17>



### A Guide to ConcurrentMap | Baeldung

Maps are naturally one of the most widely style of Java collection. And, importantly, is not a thread-safe implementation, while Hashtable does provide thread-safety by synchronizing operations. To solve the problem, the Java

 <https://www.baeldung.com/java-concurrent-map>








# Messaging

- 스프링 프레임워크는 통합 메세지 시스템을 지원한다
- JMS API, JmsTemplate
- Spring AMQP for Advanced Message Queuing Protocol
- RabbitMQ, RabbitTemplate
- SpringWebSocket, STOMP
- Apache Kafka

## 참고 자료

### Spring Boot Features

Graceful shutdown is supported with all four embedded web servers (Jetty, Reactor Netty, Tomcat, and Undertow) and with both reactive and Servlet-based web applications. It occurs as part of closing the application context and is performed in the earliest phase of stopping SmartLifecycle beans.

 <https://docs.spring.io/spring-boot/docs/2.4.3/reference/html/spring-boot-features.html#boot-features-messaging>

### [AMQP][RabbitMQ]RabbitMQ를 사용하는 이유와 설치방법 - (1)

AMQP는 <https://ko.wikipedia.org/wiki/AMQP> 여기서 확인하는게 좋을 것 같다. RabbitMQ는 AMQP를 구현한 미들웨어이다. 그래서 RabbitMQ는 서비스와 서비스 사이에 존재하여서 메시지를 전달하는 역할을 하게된다. ...

🔗 <https://kamang-it.tistory.com/entry/AMQPRabbitMQRabbitMQ%EB%A5%BC-%EC%82%AC%EC%9A%A9%ED%95%98%EB%8A%94-%EC%9D%B4%EC%9C%A0%EC%99%80-%EC%84%A4%EC%B9%98%EB%B0%A9%EB%B2%95-1>

```
root@fe2058b06fc6:~# apt-get install rabbitmq-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  cron erlang-asn1 erlang-base erlang-corba erlang-crypto erlang-
erlang-os-mon erlang-parsetools erlang-public-key erlang-
Suggested packages:
  anacron checksecurity exim4 | postfix | mail-transport-agent
The following NEW packages will be installed:
  cron erlang-asn1 erlang-base erlang-corba erlang-crypto erlang-
erlang-os-mon erlang-parsetools erlang-public-key erlang-
rabbitmq-server
0 upgraded, 31 newly installed, 0 to remove and 0 not upgraded
Need to get 24.6 MB of archives.
After this operation, 41.4 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

# Calling REST Services

## Calling REST Services with RestTemplate

- 앱 내에서 원격 REST Service 를 사용할 필요가 있다면, 스프링 프레임워크의 `RestTemplate` 을 사용할 수 있다.
- `RestTemplate` 은 사용하기 전에 커스터마이징해야 하는 경우 많은 이유로, 스프링부트는 `RestTemplate` 에 대해 어느 특정한 하나의 자동 설정을 제공하지 않는다.
- 필요하다면 자동 설정되는 `RestTemplateBuilder` 를 사용해서 `RestTemplate` 빈 인스턴스를 생성할 수는 있다.  
`RestTemplateBuilder` 를 이용해서 쓸만한 `HttpMessageConverters` 가 `RestTemplate` 인스턴스에 사용되도록 할 수 있다.

```
@Service
public class MyService {

    private final RestTemplate restTemplate;

    public MyService(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    public Details someRestCall(String name) {
        return this.restTemplate.getForObject("/{name}/details", Details.class, name);
    }

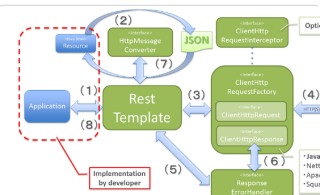
}
```

## 참고 자료

`RestTemplate` (정의, 특징, URLConnection, HttpClient, 동작원리, 사용법, connection pool 적용)

참조문서 : <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html> spring 3.0 부터 지원한다. 스프링에서 제공하는 http 통신에 유용하게 쓸 수 있는 템플릿이며, HTTP 서버와의 통신을 단순화하고 RESTful 원칙을 지킨다.

☞ <https://sjh836.tistory.com/141>



## Calling REST Services with WebClient

- 만약 스프링 웹플렉스가 클래스패스에 있다면, 원격 REST 서비스의 API를 호출할 때 `WebClient` 를 사용할 수도 있다.
- RestTemplate과 비교해보자면, 클라이언트는 더욱 평서널한 기분(?)과 충분히 reactive 같다는 기분을 느낄 수 있다.
- 스프링 부트는 미리 설정된 `WebClient.Builder` 를 제공하는데 이를 컴포넌트에 주입받아서 `WebClient` 인스턴스를 만들 것을 강력하게 조언한다.

```
@Service
public class MyService {

    private final WebClient webClient;
```

```
public MyService(WebClient.Builder webClientBuilder) {  
    this.webClient = webClientBuilder.baseUrl("https://example.org").build();  
}  
  
public Mono<Details> someRestCall(String name) {  
    return this.webClient.get().uri("/{name}/details", name)  
        .retrieve().bodyToMono(Details.class);  
}  
}
```

# Validation

- JSR-303 구현체 (`Hibernate validator` 같은) 가 클래스 패스에 있는 한, 메소드를 벨리데이션 하는 피쳐가 `Bean Validation 1.1` 에 의해 제공된다.
- 메소드의 파라미터나 응답하는 값에 `javax.validation` 제약조건을 어노테이션을 이용해 적용할 수 있다.
- 이런 어노테이션이 붙은 메소드가 있는 Target class 는 tap레벨에 `@Validated` 어노테이션이 붙어야 한다.

```
@Service
@Validated
public class MyBean {

    public Archive findByCodeAndAuthor(
        @Size(min = 8, max = 10) String code, Author author
    ) {
        ...
    }
}
```