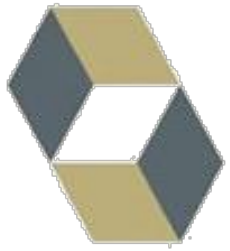


HIBERNATE

Trainer: Nilesh Ghule

Wake up from Hibernate, Spring up!!!

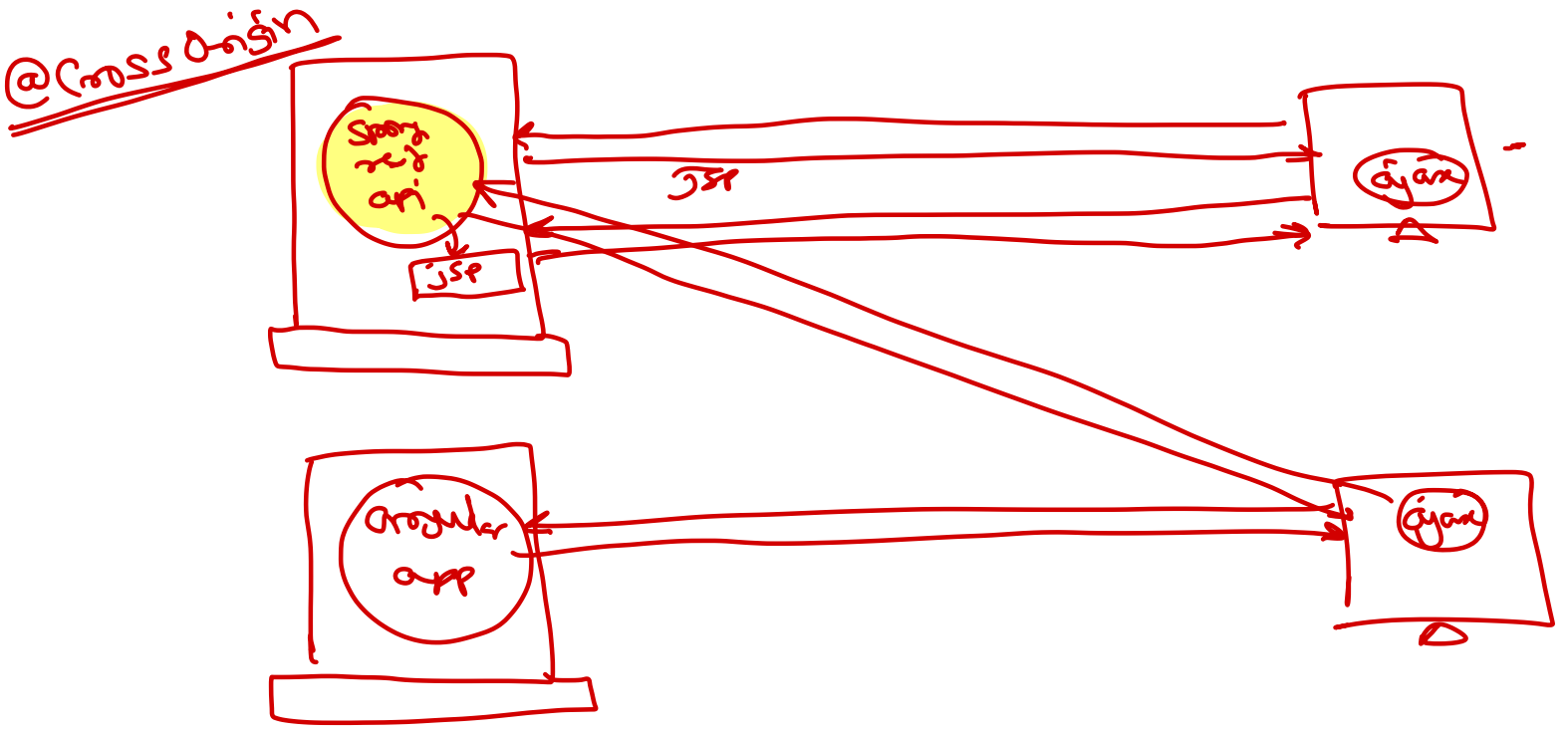
Agenda

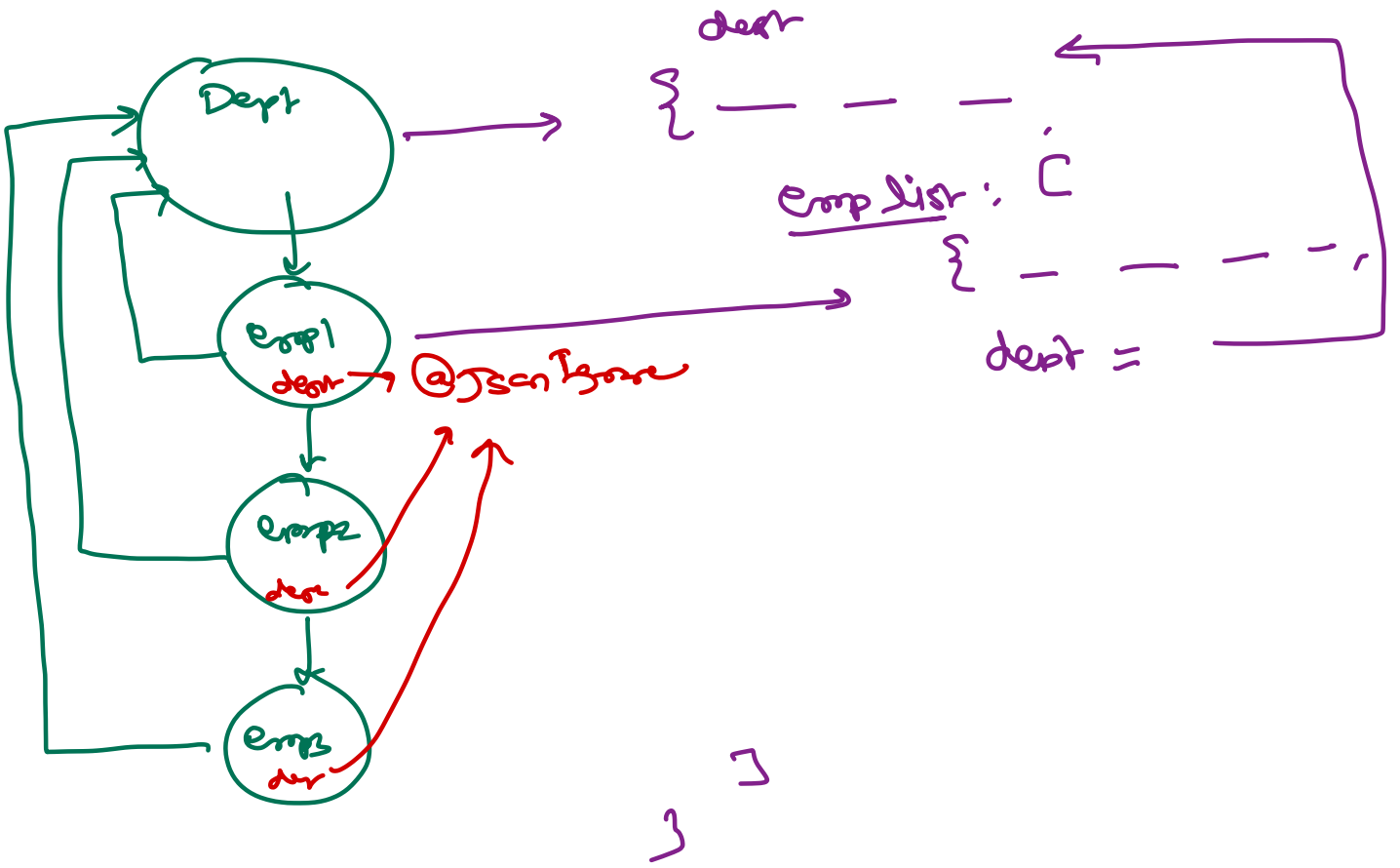
- RestTemplate
- Spring AOP
- Spring Profile
- Spring Conditional configuration
- Spring Boot Introduction

REST clients

1. Mobile phone applns
2. Core Java applns → Java code → RestTemplate.
3. Java EE / Spring MVC applns → Java code → RestTemplate.
4. Another REST service. → Java code → RestTemplate.
5. JS code / library / framework → AJAX → jQuery, Angular, ... $.ajax
6. Linux / Embedded C/C++/Python appln

JS


@CrossOrigin
Spring MVC API
JSP
JSP
Angular app
Ajax
Ajax

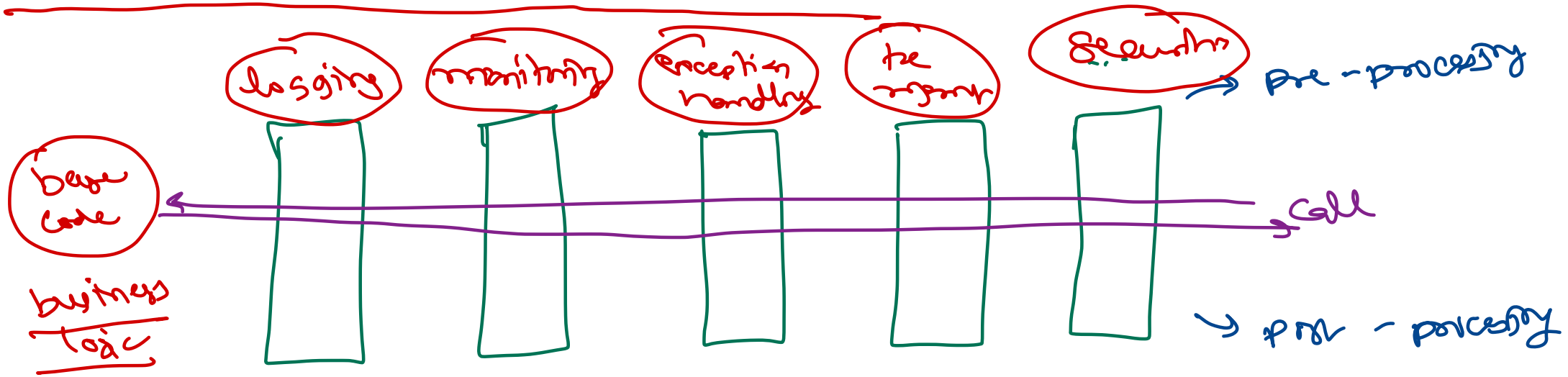

dept
{ — — —
Dept
empList : C
{ — — — —
empl
dept
@JsonIgnore
dept =
empl2
dept
empl3
dept
]
}

Spring REST services

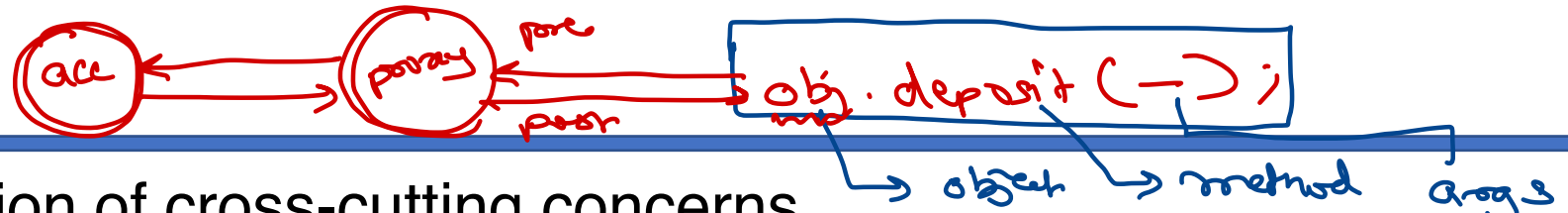
- Spring REST services produce/consume JSON/XML data based on available message converters. Can configure in MVC using configureMessageConverters():
  - MappingJackson2HttpMessageConverter, MappingJackson2XmlHttpMessageConverter.
- Spring REST services can be invoked by RestTemplate.
  - Using this can execute GET, POST, PUT, DELETE, HEAD or OPTIONS requests.
  - restTemplate.optionsForAllow(url);
  - restTemplate.postForEntity(), restTemplate.postForObject()
  - restTemplate.getForEntity(), restTemplate.getForObject()
  - restTemplate.put(), restTemplate.delete()
  - restTemplate.exchange()
- Spring5 introduced WebClient that can be used to invoke REST services.
  - Based on reactive pattern.
  - Can invoke synchronously or asynchronously.

Spring AOP - Aspect Oriented Programming.

- Implementation of cross cutting concerns without modifying core business logic.
  - Pre-processing & Post-processing
  - In Java EE it is implemented using Filters.
  - In Java it can be implemented using Java Proxies or using Aspect/J framework.
- Spring AOP is wrapper on Aspect/J library.



Spring AOP



- Aspect: Implementation of cross-cutting concerns.
  - In Spring AOP, it is a class containing cross-cutting concern implementation - @Aspect.
  - It will contain implementation of one or more advices implementations.
- Advice: Implementation of a pre-processing or post-processing or both.
  - In Spring AOP, it is a method in @Aspect class.
  - Types: @Before, @After, @Around, @AfterThrowing

@AfterReturning → always for post process

- JoinPoint: Point of intersection where advice is applied.
  - In Spring AOP, it is an object containing info about the method on which advice is applied and the object on which that method is invoked.
  - It is argument to the advice methods i.e. JoinPoint or ProceedingJoinPoint.
- Pointcut: Combination of multiple JoinPoints.
- Target: The object on which business logic method is invoked.
- Proxy: The wrapper on the target object.
  - Responsible for executing advices & actual business logic.
  - Proxy can be interface based or cglib.

@Before
security
logging

@After
logging

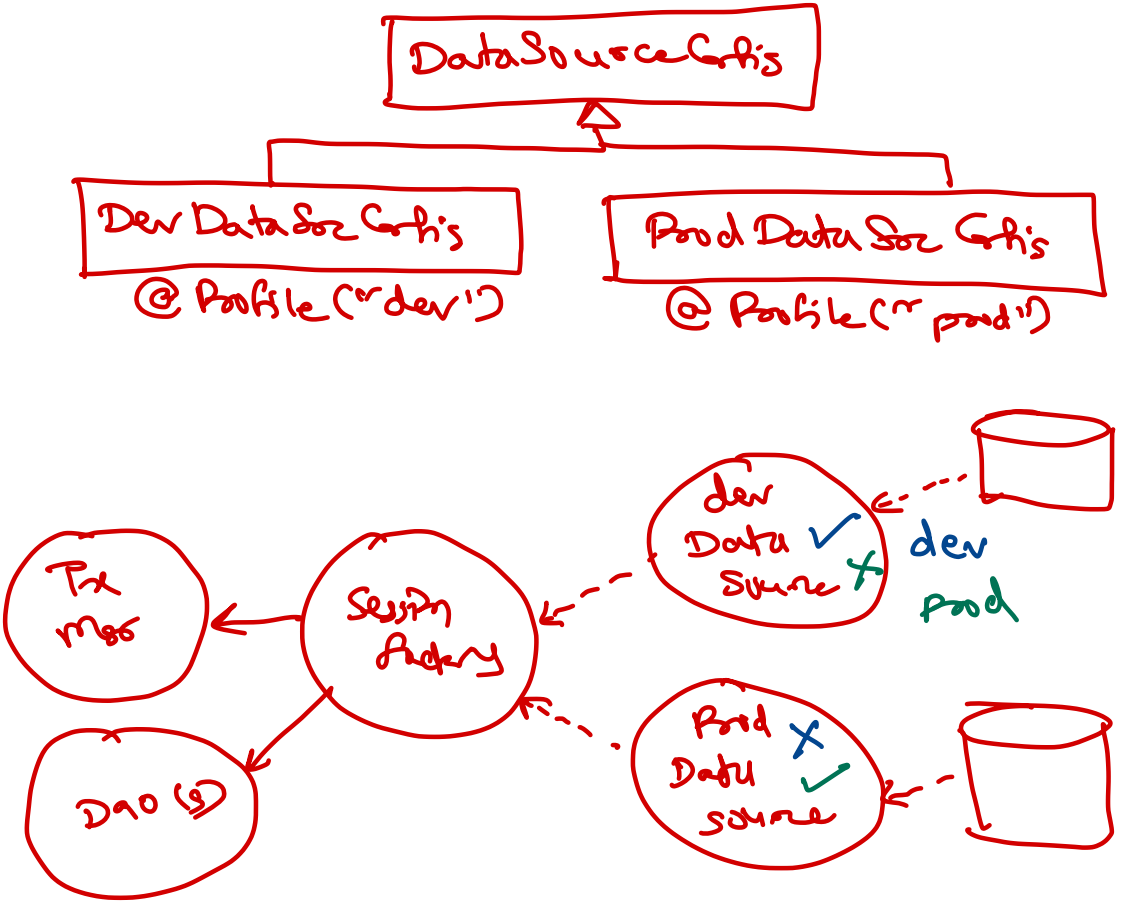
@Around
pre
time1 = get system time
call business logic
post
time2 = get system time
diff = time2 - time1;
time wrapper
performance test

Spring AOP

- Add AOP dependencies in pom.xml i.e. spring-aop, aspectjweaver
- Implement aspect class and advices in it.
  - @Before("execution (* Account.get*(..))")
  - @After("execution (* Account.set*(..))")
  - @Pointcut("execution (* Account.withdraw(..)) || execution (* Account.deposit(..))")
  - @Around("transaction()")
- For XML (mixed) config, in bean config file
  - <aop:aspectj-autoproxy>
- For Annotation config, in config class
  - @EnableAspectJAutoProxy

Spring Profile

- Profile is feature of Spring core.
- Profiles allows to have different application configuration for different environments.
- Enterprise application development is complex and may need different configurations for different stages e.g. DEV, TEST, PRODUCTION, etc.
- @Profile is associated with any spring bean and specify the bean object to be created in which profiles.
  - @Profile("dev") @Component public class MyComponent { … }
- The active profile can be set by one of the following ways.
  - onStartup() → servletContext.setInitParameter("spring.profiles.active", "dev");
  - @Autowired ConfigurableEnvironment env; → env.setActiveProfiles("dev");
  - JVM system parameter → -Dspring.profiles.active=dev
  - maven profile and application properties
  - Test case → @ActiveProfiles("dev")


DataSourceConfig
DevDataSourConfig
@Profile("dev")
ProdDataSourConfig
@Profile("prod")
dev
Data
Source
dev
prod
Tx
Mgr
Session
Factory
Prod
Data
source
DAO

Spring conditional configuration

- Conditional configuration is feature of Spring core.
- It allows only load beans into the application context if some condition is met.
- Custom conditions can be defined using @Conditional and Condition interface.
- @Conditional is used on a bean method, configurations or beans (components).
- Condition interface matches method should be implemented as per requirement.
  - Based on property/environment value or expression
  - Based on JNDI name or resource
  - Based on class in class-path
  - Based on bean in context

Spring Boot

- Spring Boot is NOT a new standalone framework. It is combination of various frameworks on spring platform i.e. spring-core, spring-webmvc, spring-data, …
- Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can “just run”.
- Spring Boot take an "opinionated view" of the Spring platform and third-party libraries. So most of applications need minimum configuration.
- Primary Goals/Features:
  - Provide a radically faster and widely accessible "Quick Start".
  - Opinionated config, yet quickly modifiable for different requirements.
  - Managing versions of dependencies with starter projects.
  - Provide lot of non-functional common features (e.g. security, servers, health checks, ...).
  - No extra code generation and XML config.
  - Embedded Web Server for web applications.
  - No code generation (provide lot of boilerplate code) or XML configuration.
- Not good for porting existing applications.

Spring Boot

- Latest spring boot: 2.3.1
  - spring framework – 5.2.6
- @SpringBootApplication
  - @Configuration + @EnableAutoConfiguration + @ComponentScan
- @EnableAutoConfiguration
  - spring-boot-starter-web → web-mvc + tomcat
- SpringApplication.run(DeptappApplication.class, args)
  - Start embedded web server
  - Deploy MVC application

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>