Spring Boot in Action

Le Thi Thuy Trang – SSE

May 2018

NASH TECH

The Power to Innovate

Preparation

Preparations

- You have to learn some main concepts before joining in the workshop.
- We recommend you to do all the tutorials listed in the next slide. By doing them, you will have knowledge of
  - IoC & DI concept in Spring boot application development
  - How to build a spring boot app by using Spring Initializr / STS
  - How to develop REST APIs with Spring Boot
  - Integrate with database server by using JPA/Hibernate
  - How to do REST API unit testing in Spring Boot application
  - Integrate with Spring Actuator
- Quick understanding: Building Microservices with Spring Boot, Second Edition (can use trial account)
  - Introduction
  - Before You Begin (2)
  - Lesson 1: exclude 1.5
  - Lesson 2: exclude 2.7, 2.8
  - Lesson 4: exclude 4.5, 4.6, 4.7
  - Lesson 5: 5.1, 5.2
  - Lesson 11: 11.1, 11.2
  - Lesson 12: exclude 12.8, 12.9, 12.10
- To have more in-depth knowledge of Spring Boot, we recommend you to read the Spring Boot in action book. You should read the below section first to have good understanding of Spring Boot. (The other chapters you can read later on)
  - Foreword
  - Preface
  - About this book
  - Chapter: 1, 2, 3, 4, 8

Useful Tutorials

- AOP understanding
- DI vs IoC understanding
- How to initialize Spring Boot application (Part II.3 only)
- How to create an embedded server & Rest API then Execute it
- How to integrate with DB using Spring Data JPA
- Testing Tutorial: unit test with mocked services & integration test
- Actuator Integration
- Spring Boot in Action – Craig Walls
- How to building microservice with Spring Boot

Agenda

- Course overview
- Course objectives
- Main contents
- Appendix: Keywords & Techniques
- Q&A
- Exercise
- Assignment

Course overview

Course overview

- Sharing some useful features of Spring Boot
- Sharing keywords and techniques to apply and learn Spring Boot
- Small exercise and assignment to verify our understanding

Course Objectives

Objectives

- Trainees understand main concepts of Spring Boot, and be able to build a Spring Boot application (web / REST APIs) from scratch without support
- Trainees can utilize Spring Boot strength: configuration
- Trainees can apply: unit test, security, spring-data-jpa into Spring Boot application
- Trainees know how to host Spring Boot application
- Trainees can work with Spring Boot in real-world project without any additional training

Main contents

Big questions

- What’s Spring?
- Why Spring?
- Why not Spring?
- Why Spring… again?
- How does it look like?

What’s SPRING?

Spring idea

- Application framework
- Integration of many technologies
- Consistent programming style

Spring components

A combo of

- Web
- Data access
- Caching
- Security
- Cloud
- And so on

Spring core concepts

What're they?

Spring core concepts

AOP

Aspect Oriented Programming

DI

Dependency Injection

AOP

- **Adding** behavior to existing code
- **Without modifying** that code

AOP sample

Lets see sample: **aop-advice-annotation**

- A Java annotation to log the execution time of a method
- @LogExecutionTime

Dependency Injection

- Before

```
public class StudentService {

      private StudentRepository studentRepository;

      public StudentService() {
            this.studentRepository = new StudentRepository();
      }

      public List<Student> getAllStudents() {
            List<Student> students = studentRepository.findAll();
            return students;
      }
}
```

Dependency Injection

- After (1 of 3 ways: setter, constructor, and field)

```
@Service
public class StudentService {

      @Autowired
      private StudentRepository studentRepository;

      public List<Student> getAllStudents() {
            List<Student> students = studentRepository.findAll();
            return students;
      }
}
```

Dependency Injection

How HELPFUL?

Dependency Injection

- Implementation switchable
- Testable (injectable -> isolatable)
- AOP

Dependency Injection

- Class instances are managed by Spring: beans
- When we need them, just tell Spring, then let Spring do anything necessary to prepare them

Ambiguous terminology usage

Dependency Injection (DI)

Vs

Inversion of Control (IoC)

Ambiguous terminology usage

Dependency Injection (DI)

Vs

Inversion of Control (IoC)

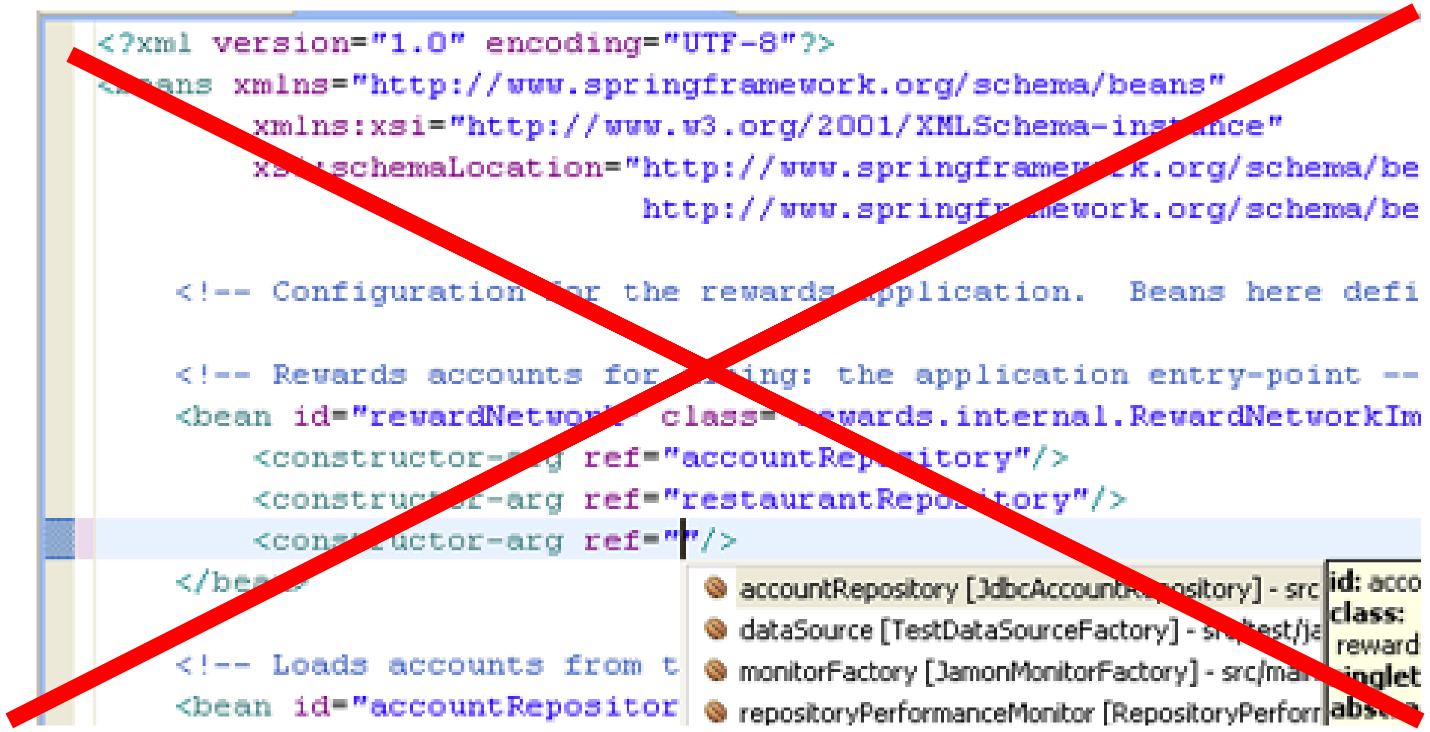
IoC: idea of inverting flow of application – work with contracts

DI: one of patterns implementing IoC

Why not Spring?

Before Spring Boot

- Component code - easier
- **Configuration - headache**



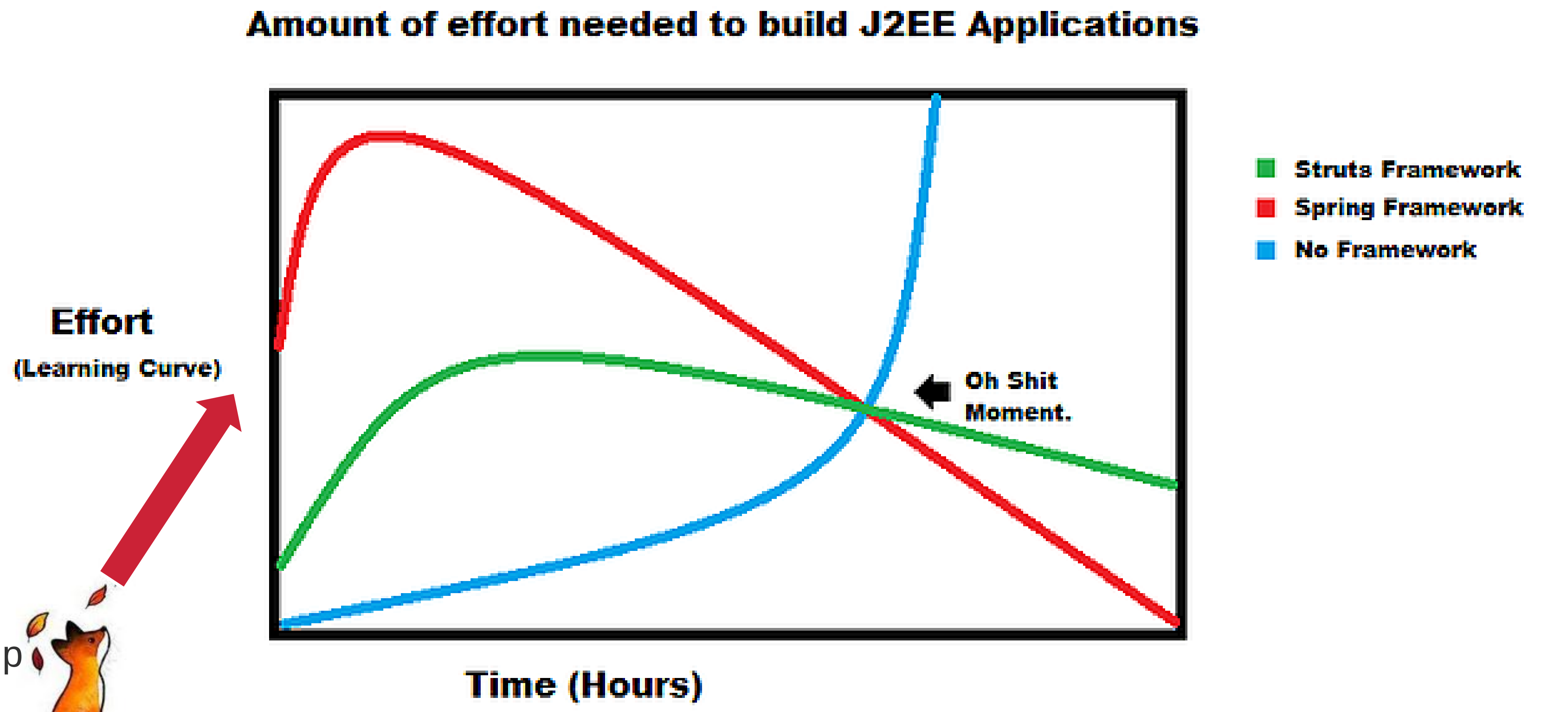
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/be
                             http://www.springframework.org/schema/be

    <!-- Configuration for the rewards application.  Beans here defi

    <!-- Rewards accounts for testing: the application entry-point --
    <bean id="rewardNetwork" class="rewards.internal.RewardNetworkIm
        <constructor-arg ref="accountRepository"/>
        <constructor-arg ref="restaurantRepository"/>
        <constructor-arg ref=""/>
    </bean>

    <!-- Loads accounts from t
    <bean id="accountRepositor
```

```
accountRepository [JdbcAccountRepository] - src
dataSource [TestDataSourceFactory] - src/test/ja
monitorFactory [JamonMonitorFactory] - src/mai
repositoryPerformanceMonitor [RepositoryPerforr
```

```
id: acco
class:
 reward
singlet
abs
```

Why not Spring?

Before Spring Boot



Where I gave up
on the dream

Why Spring …again?

SPRING BOOT

What spring boot DOES?

What’s Spring Boot

- Stand-alone Spring applications
  - Say goodbye to web container: Tomcat, Jboss, etc.
  - Say hello to embedded server: Tomcat, Jetty, Undertow
  - Possible to say hello again if you want
- Quick start with useful starter packages data, security, health check, logging, test, etc.
- Auto-configuration as much as possible -> say goodbye to xml

Why’s Spring Boot

Sample **hello-spring-boot**:

- **Quick to initialize project:** from http://start.spring.io/

Why’s Spring Boot

Sample **hello-spring-boot**:

- **Starter dependencies: spring-boot-starter-* (web, data-jpa, security):** rest API, data access, JWT

Why’s Spring Boot

Sample **hello-spring-boot**:

- **Auto configuration: rest API, datasource (database access)**

Why’s Spring Boot

Sample **hello-spring-boot**:

- **Standalone server / Self-contained runnable application / Fat jar**

Why's Spring Boot

Sample **hello-spring-boot**:

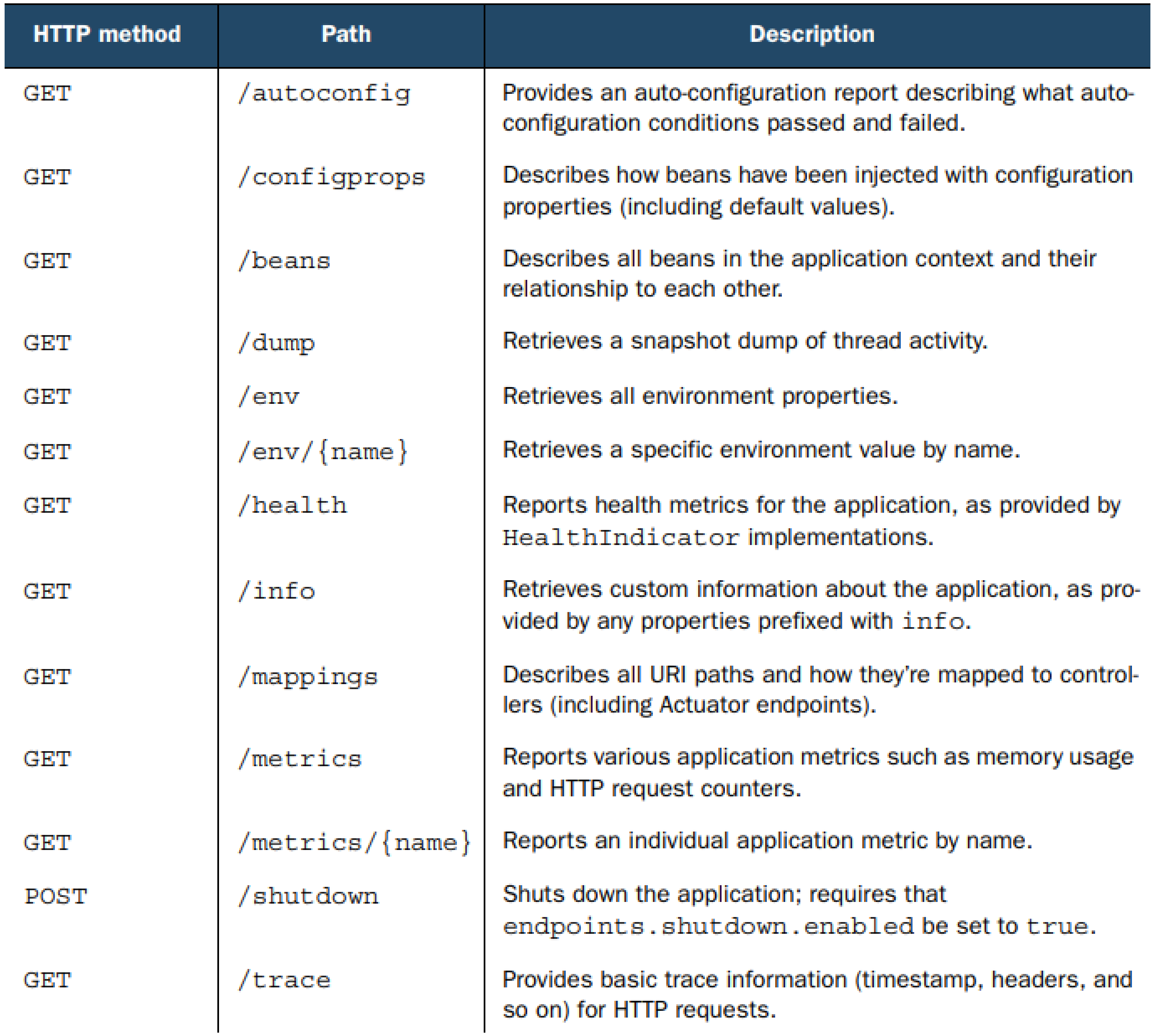
- **Create a custom configuration:** log version of Tomcat
  - If embedded server is Tomcat
  - If log-tomcat-version is not disable
  - Change behavior without modification on code

Why’s Spring Boot

Sample **hello-spring-boot**:

- **Actuator: take a look into application at runtime**
  - actuator/health: health check
  - actuator/info
  - View all configuration: /actuator/configprops
  - Use another port to expose actuator

Why's Spring Boot



HTTP method	Path	Description
GET	/autoconfig	Provides an auto-configuration report describing what auto-configuration conditions passed and failed.
GET	/configprops	Describes how beans have been injected with configuration properties (including default values).
GET	/beans	Describes all beans in the application context and their relationship to each other.
GET	/dump	Retrieves a snapshot dump of thread activity.
GET	/env	Retrieves all environment properties.
GET	/env/{name}	Retrieves a specific environment value by name.
GET	/health	Reports health metrics for the application, as provided by `HealthIndicator` implementations.
GET	/info	Retrieves custom information about the application, as provided by any properties prefixed with `info`.
GET	/mappings	Describes all URI paths and how they're mapped to controllers (including Actuator endpoints).
GET	/metrics	Reports various application metrics such as memory usage and HTTP request counters.
GET	/metrics/{name}	Reports an individual application metric by name.
POST	/shutdown	Shuts down the application; requires that `endpoints.shutdown.enabled` be set to `true`.
GET	/trace	Provides basic trace information (timestamp, headers, and so on) for HTTP requests.

How’s Spring Boot - Testing

- Unit test
- Integration test: @RunWith(SpringJUnit4ClassRunner.class), @SpringBootTest
- Isolate tested module - mocking supporting: @MockBean

How’s Spring Boot - Deployment

- Web container vs standalone server: war vs jar
- Database: validate, auto generate DDL, evolve database using flyway
- Cloud: portable (VMs, AWS, Pivotal Cloudfoundry)

Q&A

Appendix: Keywords & Techniques

To discover Spring Boot

Recommended sites

- Tutorials with explanation & some advice: baeldung
- Quick tutorial & related useful resources - official: spring.io
- Technical review & explanation: martinfowler
- Shortcut to an overview of application: online course
- To have fun with Spring in microservice & cloud: Spring Microservices (Rajesh RV)

My own experience

1. Learn general concepts
2. Find related Spring plugins/starters
3. Configure

Example

Apply Spring Security & OAuth2

1. Learn general concept of OAuth2 on official site
2. Search for Spring OAuth2 plugin(s)
3. Search for user guide(s) and **try to configure**

Exercise

Assignment

Resources

- Trainees walk through all following milestones
- Trainees upload source code to a BitBucket account
- After 5th milestone, trainees deploy artifacts to testing server
- Examiners will check these result
- Trainees are passed with at least 5/6 milestones passed

Milestones

1. Using Initializr create a REST service that:
   - Read data from MySQL
   - Return json objects
2. Implement a REST service that
   - Receive user id, return user name **(1)**
   - Receive user id, return user membership start date **(2)**
   - Receive user id, return user (id, name, start_date, loyal_point), with loyal_point = (today – start_date + 1) * 5 **(3)**
3. Integrate security by JWT without touching legacy code
   - Manage username/password of an administrator account, and a moderator account in DB
   - Apply JWT to authorize user
   - Apply Spring security to configure: all users can use (1), only admin & mod can use (2), only admin can use (3)
4. Implement test:
   - Integration test for DB connection – is it possible to connect, can read & write
   - Unit test for Service that get data from from Repository, perform some logical calculation, and return abstract data
   - Unit test for REST API – does it return expected JSON
5. Implement healthcheck API for service
6. Deploy application to testing server with
   - A release package
   - Run script that others can use to: start application with Jetty server with a customized DB & port, stop application by external application.properties

THANK YOU

www.nashtechglobal.com