GBA培训项目课程材料-SpringBoot框架

国际商业机器（中国）有限公司

2020年8月

目 录

SpringBoot简介

Spring Boot来简化Spring应用开发，约定大于配置， 去繁从简，能创建一个独立的，产品级别的应用。

背景:

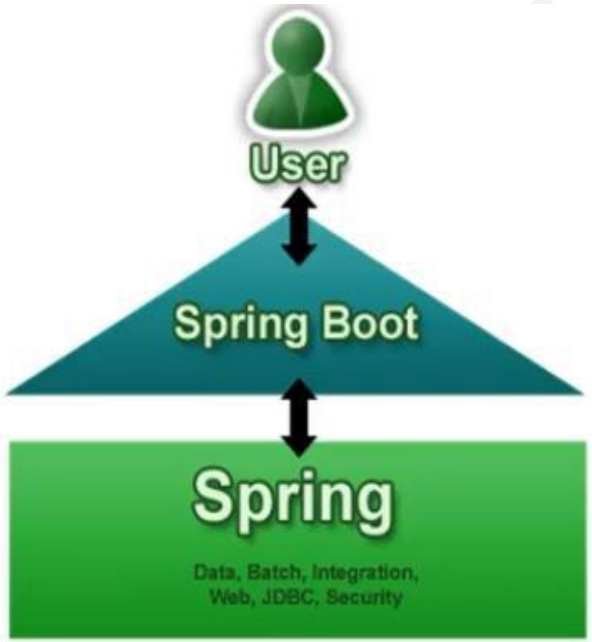
J2EE笨重的开发、繁多的配置、低下的开发效率、 复杂的部署流程、第三方技术集成难度大。

解决:

“Spring全家桶”时代。

Spring Boot--J2EE一站式解决方案

Spring Cloud--分布式微服务架构整体解决方案



SpringBoot简介

Spring boot的优点:

1. 轻松创建独立的Spring应用程序。

2.内嵌Tomcat等web容器，不需要部署WAR文件。

3.提供一系列的“starter” 来简化的Maven配置，不需要添加很多依赖。

4.开箱即用，尽可能自动配置Spring。

SpringBoot简介

Spring Boot 四大核心机制:

1.起步依赖机制：通过起步依赖机制（Starter）,简化jar包的引用，解决jar版本冲突问题。

2.自动配置：可以实现简单配置，甚至是零配置，就能搭建整套框架。

3.StringBoot CLI：一种命令工具。

4.Actuator：是SpringBoot的程序监控器, 可检测应用程序健康状况等。

SpringBoot简介

提问：

SpringMVC是一种基于Java的实现了Web MVC设计模式的请求驱动类型的轻量级Web框架，那么Springboot和SpringMVC有什么区别和联系呢?

目 录

- SpringBoot简介
- **SpringBoot配置**
- SpringBoot与日志
- SpringBoot与数据访问
- SpringBoot与启动配置原理
- SpringBoot与任务

SpringBoot配置文件

SpringBoot使用一个全局的配置文件，配置文件名是固定的：application.yml 和 application.properties。

配置文件的作用：修改SpringBoot自动配置的默认值；SpringBoot在底层是自动配置好；

.properties和.yml两种配置文件对比；

1 .yml拥有天然的树状结构；

2 在properties文件中是以”.”进行分割的， 在.yml中是用”:”进行分割；

3 .yml的数据格式是K-V格式（和json很像），并且通过”:”进行赋值；

4 在.yml中缩进一定不能使用TAB，否则会报很奇怪的错误；

5 每个k的冒号后面一定都要加一个空格；

6 .yml比.properties对中文的支持更友好。

SpringBoot配置文件

SpringBoot中的application.yml配置文件

yml是YAML（YAML Ain't Markup Language）语言的文件，以数据为中心，比json、xml等更适合做配置文件。

YAML 语法

1. 大小写敏感
2. 使用缩进表示层级关系
3. 缩进不允许使用tab，只允许空格
4. 缩进的空格数不重要，只要相同层级的元素左对齐即可
5. '#'表示注释

SpringBoot配置文件

YAML数据类型

1. 字面量：普通的值

字面量：普通的值 [数字，布尔值，字符串]。字面量直接写在后面就可以 ， 字符串默认不用加上双引号或者单引号；

2. 对象、Map（属性和值）（键值对）：

对象键值对使用冒号结构表示 key: value，冒号后面要加一个空格。也可以使用 key:{key1: value1, key2: value2, ...}。还可以使用缩进表示层级关系；比如说:

```
key:
    child-key: value
    child-key2: value2
```

3. 数组（List、Set）：

以 - 开头的行表示构成一个数组：

```
pets:
 - cat
 -cattle
```

行内写法 例如:pets: [cat,cattle]

SpringBoot配置文件

springboot项目springboot_config为例

1. 配置application.yml如下：

```
person:
  last-name: 五指山
  sex: 男
  age: 20
  birth: 2000/01/01
  map:
    property: v1
    form: v2
  list:
    - list1
    - list2
  Cattle:
    name: 老黄牛
    age: 5
```

SpringBoot配置文件

springboot项目springboot_config为例

2.编写实体类**Person**，首先把这个类标记为组件，使用@Component，只有这个组件是容器中的组件，才能容器提供的**@ConfigurationProperties功能**。

@ConfigurationProperties：告诉SpringBoot将本类中的所有属性和配置文件中相关的配置进行绑定；prefix = "person"：配置文件中哪个下面的所有属性进行一一映射。然后springboot会自动将配置文件中配置的每一个属性的值，映射到这个组件中。

```
@Component
@ConfigurationProperties(prefix = "person")
public class Person {  //package com.ibm.demo.config;包下
    private String lastName;
    private String sex;
    private Integer age;
    private Date birth;
    private Map<String, String> map;
    private List<Object> list;
    private Cattle cattle;  //类对象  set和get方法此处略去，实践是需加的
```

SpringBoot配置文件

springboot项目springboot_config为例

3.测试@ConfigurationProperties获取值输出　结合项目测试

```
package com.ibm.demo;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import com.ibm.demo.config.Person;
@SpringBootTest
class SpringbootConfigApplicationTests {
    @Autowired
     Person person;
            @Test
            void contextLoads() {
                    System.out.println(person.getCattle().getName());
            }
}
```

SpringBoot配置文件

springboot项目springboot_config为例

4.通过**@Value获取值**测试输出　结合项目测试

```
package com.ibm.demo;

    @Value("${person.sex}")
    String sex;
    @Test
    void valueGet() {
            System.out.println(sex);
    }
```

springboot中从application.yml配置文件中获取获取值：@ConfigurationProperties支持复杂类型封装，而@Value不支持，@Value只能一个一个的获取配置文件的值。

SpringBoot配置文件

springboot项目springboot_config为例

@PropertySource加载指定的YAML

首先新建一个PropertySourceFactory实体类，用于处理 yml 配置文件的属性资源工程

```
package com.ibm.demo.config;
public class PropertySourceFactory extends DefaultPropertySourceFactory {
 @Override
 public PropertySource<?> createPropertySource(String name, EncodedResource resource)
throws IOException {
        if (resource == null) {
            return super.createPropertySource(name, resource);
        }
        List<PropertySource<?>> sources = new
YamlPropertySourceLoader().load(resource.getResource().getFilename(), resource.getResource());
        return sources.get(0);
    }
}
```

SpringBoot配置文件

springboot项目springboot_config为例

@PropertySource加载指定的YAML

新建一个YAML配置文件 person.yml

```
person:
LAST_NaME: 张三
sex: 男
age: 20
birth: 2000/01/01
map:
  property: v1
  form: v2
list:
  - list1
  - list2
Cattle:
  name: 小牛
  age: 2
```

SpringBoot配置文件

springboot项目springboot_config为例

@PropertySource加载指定的YAML

新建**YmlPerson**实体类改为如下：注意：导入**PropertySourceFactory实体类**路径

```
@Component
@PropertySource(value = {"classpath:person.yml"}, factory = PropertySourceFactory.class)
@ConfigurationProperties(prefix = "person")
public class YmlPerson {
    private String lastName;
    private String sex;
    private Integer age;
    private Date birth;
    private Map<String, String> map;
    private List<Object> list;
    private Cattle cattle;
    //此处get和set略去，实际开发中需加上的
}
```

SpringBoot配置文件

springboot项目springboot_config为例

@PropertySource加载指定的YAML

在测试类下SpringbootConfigApplicationTests注入配置类

```
@Autowired
YmlPerson personConfig;
 //测试方法
 @Test
 void getValueByPropertySource() {
 System.out.println(personConfig.getCattle().getName()+personConfig.getLastName());
 }
```

属性配置占位符

修改YAML文件 (person.yml)　加入${...}

配置文件还可以编写占位符生成随机数

${random.value}、${random.int}、${random.long}

${random.int(10)}、${random.int[1024,65536]}

SpringBoot配置文件

Spring Boot最常用的3种读取properties配置文件中数据的方法

1、**使用@Value注解读取**　springboot项目springboot_config为例

读取properties配置文件时，默认读取的是application.properties

application.properties：

```
springboot.demo.name=zhangsan
springboot.demo.age=18
```

Java代码:

```
@RestController
public class TestController {//package com.ibm.demo.controller包下
    @Value("${springboot.demo.name}")
    private String name;
    @Value("${springboot.demo.age}")
    private String age;
    @RequestMapping(value = "/get/value")
    public String getByValue() {
        return "get properties value by ''@Value'' :" + " name=" + name +" , age=" + age;
    }
}
```

浏览器测试: http://localhost:8080/get/value

SpringBoot配置文件

Spring Boot最常用的3种读取properties配置文件中数据的方法

2、使用Environment读取

读取properties配置文件时，默认读取的是application.properties

application.properties：

```
springboot.demo.name=zhangsan
springboot.demo.age=18
```

在**TestController**类下添加

```
  @Autowired
  private Environment environment;

 @RequestMapping(value = "/get/environment")
  public String getByEnv() {
     return "get properties value by  ''Environment'' :" +
          //1、使用@Value注解读取
          " name=" + environment.getProperty("springboot.demo.sex") +
          " , age=" + environment.getProperty("springboot.demo.address");
  }
```

启动服务后浏览器输入 http://localhost:8080/get/environment 进行测试

SpringBoot配置文件

Spring Boot最常用的3种读取properties配置文件中数据的方法

3、使用@ConfigurationProperties注解读取

在实际项目中，当项目需要**注入的变量值很多**时，上述所述的两种方法工作量会变得比较大，这时候我们通常使用基于类型安全的配置方式，将properties属性和一个Bean关联在一起，即用注解@ConfigurationProperties读取配置文件数据。

(1). 在src\main\resources下新建**config.properties**配置文件

```
springboot.demo.phone=10086
springboot.demo.hobby=旅游
```

SpringBoot配置文件

Spring Boot最常用的3种读取properties配置文件中数据的方法

3、使用@ConfigurationProperties注解读取

(2).创建ConfigBeanProperty并注入config.properties中的值

```
@Component
@ConfigurationProperties(prefix = "springboot.demo")
@PropertySource(value = "config.properties")
public class ConfigBeanProperty {   //package com.ibm.demo.config;包下
        private String phone;
        private String hobby;
        public String getPhone() {
                return phone;
        }
        public void setPhone(String phone) {
                this.phone = phone;
        }
        public String getHobby() {
                return hobby;
        }
        public void setHobby(String hobby) {
                this.hobby = hobby;
        }
}
```

SpringBoot配置文件

Spring Boot最常用的3种读取properties配置文件中数据的方法

3、使用@ConfigurationProperties注解读取

(3).使用时，先使用@Autowired自动装载ConfigBeanProperty，然后再进行取值，示例如下

在TestController类下注入

```
@Autowired
private ConfigBeanProperty configBeanProperty;


@RequestMapping(value = "/get/config")
  public String getByConfigBean() {
     return "get properties value by  ''@ConfigurationProperties'' :" +
                  " phone=" + configBeanProperty.getPhone() +
           " , hobby=" +configBeanProperty.getHobby();
  }
```

启动服务，浏览器输入:http://localhost:8080/get/config

SpringBoot配置文件

结论：

当application.properties和yml文件在并存时（同一目录下），application.properties优先级更好，会先读它，若它没有，再去读yml中的值。

随堂练习作业：

创建Springboot项目，在src\main\resources同时建application.properties和application.yml文件，application.properties文件配置端口为9091，application.yml中配置端口为9092，启动项目查看端口验证优先级。

SpringBoot配置文件

SpringBoot 配置文件存放位置及读取顺序

存放目录 以application.properties为例

SpringBoot配置文件默认可以放到以下目录中，可以自动读取到：

项目根目录下

项目根目录中config目录下

项目的resources目录下

项目resources目录中config目录下

读取顺序

SpringBoot配置文件默认可以放到以下目录中，可以自动读取到：

1.项目根目录中config目录下

2.项目根目录下

3.项目resources目录中config目录下

4.项目的resources目录下

备注:如果同一个配置属性，在多个配置文件都配置了，默认使用第1个读取到的，后面读取的不覆盖前面读取到的。

Profile

Profile是Spring对不同环境提供不同配置功能的支持，可以通过激活、
指定参数等方式快速切换环境

1、多profile文件形式:

– 格式:application-{profile}.properties/yml:
  application-dev.properties、application-prod.properties

2、多profile文档块模式:

3、激活方式:

– 命令行 --spring.profiles.active=dev

– 配置文件 spring.profiles.active=dev

```yaml
spring:
  profiles:
    active: prod # profiles.active: 激活指定配置
---
spring:
  profiles: prod
server:
  port: 80
--- #三个短横线分割多个profile区（文档块）
spring:
  profiles: default # profiles: default表示未指定默认配置
server:
  port: 8080
```

SpringBoot外部配置文件

外部Jar包方式读取配置文件 --启动命令读取

例如:
使用命令 java -jar springboot_config-0.0.1-SNAPSHOT.jar --spring.config.location=C:\JMPX\config\application.properties (配置文件路径)

SpringBoot自动配置原理

Springboot的自动配置，是指springboot会自动将一些配置类的bean注册进ioc容器，可在需要的地方使用@autowired或者@resource等注解来使用它。“自动”的表现形式就是只需要引入想用功能的包，相关的配置完全不用管，springboot会自动注入这些配置bean，直接使用这些bean即可。

@SpringBootApplication注解是Spring Boot的核心注解，它其实是一个组合注解

虽然定义使用了多个Annotation进行了原信息标注，但实际上重要的只有三个Annotation：

@Configuration（@SpringBootConfiguration点开查看发现里面是应用了@Configuration）

@EnableAutoConfiguration

@ComponentScan

所以，如果我们使用如下的SpringBoot启动类，整个SpringBoot应用依然可以与之前的启动类功能对等。

SpringBoot自动配置原理

1、@Configuration

@Configuration就是JavaConfig形式的Spring Ioc容器的配置类使用的那个@Configuration，SpringBoot社区推荐使用基于JavaConfig的配置形式，所以，这里的启动类标注了@Configuration之后，本身其实也是一个IoC容器的配置类。

@Configuration：提到@Configuration就要提到他的搭档**@Bean**。使用这两个注解就可以创建一个简单的spring**配置类**，可以用来替代相应的xml配置文件。

SpringBoot自动配置原理

2、@ComponentScan

@ComponentScan这个注解在Spring中很重要，它对应XML配置中的元素，@ComponentScan的功能其实就是自动扫描并加载符合条件的组件（比如@Component和@Repository等）或者bean定义，最终将这些bean定义加载到IoC容器中。

可以通过basePackages等属性来细粒度的定制@ComponentScan自动扫描的范围，如果不指定，则默认Spring框架实现会从声明@ComponentScan所在类的package进行扫描。

注：所以SpringBoot的启动类最好是放在root package下，因为默认不指定basePackages。

SpringBoot自动配置原理

3、@EnableAutoConfiguration

@EnableAutoConfiguration会根据类路径中的jar依赖为项目进行自动配置，如：添加了spring-boot-starter-web依赖，会自动添加Tomcat和Spring MVC的依赖，Spring Boot会对Tomcat和Spring MVC进行自动配置。

借助于Spring框架原有的一个工具类：SpringFactoriesLoader的支持，@EnableAutoConfiguration可以智能的自动化配置。

目 录

- SpringBoot简介
- SpringBoot配置
- **SpringBoot与日志**
- SpringBoot与数据访问
- SpringBoot与启动配置原理
- SpringBoot与任务

日志框架

市场上存在非常多的日志框架。JUL(java.util.logging)，JCL(Apache Commons Logging)，Log4j，**Log4j2**，Logback、**SLF4j**、jboss-logging等。Spring Boot在框架内容部使用JCL，spring-boot-starter-logging采用了 slf4j+logback的形式，Spring Boot也能自动适配(jul、log4j2、logback) 并 简化配置 。

日志门面	日志实现
JCL(Jakarta Commons Logging) SLF4j(Simple Logging Facade for Java) jboss-logging	Log4j JUL(java.util.logging) Log4j2 Logback

默认配置

1、全局常规设置(格式、路径、级别)
2、指定日志配置文件位置
3、切换日志框架

`spring-boot-starter-log4j2`	Starter for using Log4j2 for logging. An alternative to `spring-boot-starter-logging`
`spring-boot-starter-logging`	Starter for logging using Logback. Default logging starter

日志使用（springboot-demo工程）

1、默认配置

```
@RestController
public class LoggingController {
private Logger logger = LoggerFactory.getLogger(LoggingController.class);
@GetMapping("/logging/test")
public String loggingTest() {
// 日志的级别;
// 由低到高 trace<debug<info<warn<error
// 可以调整输出的日志级别；日志就只会在这个级别和以后的高级别生效
logger.trace("这是trace日志...");
logger.debug("这是debug日志...");
// SpringBoot默认给我们使用的是info级别的，没有指定级别的就用SpringBoot默认规定的级别：root级别
logger.info("这是info日志...");
logger.warn("这是warn日志...");
logger.error("这是error日志...");
return "success";
}
}
```

运行结果如下，可以看到默认配置时只会输出info及info之后级别的日志

```
2020-09-01 [http-nio-8090-exec-1] INFO  com.ibm.springboot.controller.LoggingController - 这是info日志...
2020-09-01 [http-nio-8090-exec-1] WARN  com.ibm.springboot.controller.LoggingController - 这是warn日志...
2020-09-01 [http-nio-8090-exec-1] ERROR com.ibm.springboot.controller.LoggingController - 这是error日志...
```

2.修改日志的输出级别

在配置文件application.yml中添加配置

```
logging:
  level:
    com.ibm.springboot.controller.LoggingController: trace #设置输出级别
```

再次运行该接口，结果如下，可以看到输出的为自定义级别及该级别之后的日志

```
2020-09-01 [http-nio-8090-exec-1] TRACE com.ibm.springboot.controller.LoggingController - 这是trace日志...
2020-09-01 [http-nio-8090-exec-1] DEBUG com.ibm.springboot.controller.LoggingController - 这是debug日志...
2020-09-01 [http-nio-8090-exec-1] INFO  com.ibm.springboot.controller.LoggingController - 这是info日志...
2020-09-01 [http-nio-8090-exec-1] WARN  com.ibm.springboot.controller.LoggingController - 这是warn日志...
2020-09-01 [http-nio-8090-exec-1] ERROR com.ibm.springboot.controller.LoggingController - 这是error日志...
```

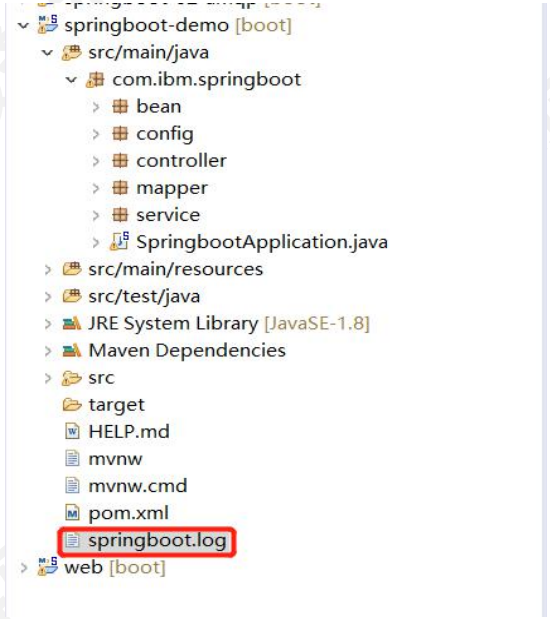
3.日志输出到文件

3.1 logging.file

配置文件中添加如下配置

```
logging:
  # 1.不指定路径时会在当前项目下生成springboot.log日志
  # 2.可以指定完整的路径,如：D:/springboot.log
  file: springboot.log
```

启动项目可以看到生成的日志文件



3.2 logging.path

配置文件中添加如下配置

```
logging:
  #在当前磁盘的根路径下创建spring文件夹和里面的log文件夹;
  #使用 spring.log 作为默认文件;
  path: /spring/log
```

启动项目可以看到生成的日志文件

备注：当file和path同时使用时,以file为准

4.设置日志的输出格式

配置文件中添加如下配置

```
logging:
  #  在控制台输出的日志的格式
  pattern:
     console: '%d{yyyy-MM-dd} [%thread] %-5level %logger{50} - %msg%n'
  # 指定文件中日志输出的格式
     file: '%d{yyyy-MM-dd} === [%thread] === %-5level === %logger{50} === - %msg%n'
```

日志输出格式：

%d表示日期时间，

%thread表示线程名，

%-5level：级别从左显示5个字符宽度

%logger{50} 表示logger名字长50个字符，否则按照句点分割。

%msg：日志消息，

%n是换行

练习作业：

根据以上示例，配置日志信息，将日志信息输入到指定目录下C:\JMPX\log（此目录自建）

目 录

- SpringBoot简介
- SpringBoot配置
- SpringBoot与日志
- **SpringBoot与数据访问**
- SpringBoot与启动配置原理
- SpringBoot与任务

Springboot与数据访问

Springboot与数据访问介绍

- 对于数据访问层，无论是SQL还是NOSQL，Spring Boot默认采用整合
- Spring Data的方式进行统一处理，添加大量自动配置，屏蔽了很多设置。引入 各种xxxTemplate，xxxRepository来简化我们对数据访问层的操作。对我们来 说只需要进行简单的设置即可。我们将在数据访问章节测试使用SQL。
- – JDBC
- – MyBatis
- – JPA

`spring-boot-starter-jdbc`	Starter for using JDBC with the Tomcat JDBC connection pool
`spring-boot-starter-data-cassandra`	Starter for using Cassandra distributed database and Spring Data Cassandra
`spring-boot-starter-data-cassandra-reactive`	Starter for using Cassandra distributed database and Spring Data Cassandra Reactive
`spring-boot-starter-data-couchbase`	Starter for using Couchbase document-oriented database and Spring Data Couchbase
`spring-boot-starter-data-elasticsearch`	Starter for using Elasticsearch search and analytics engine and Spring Data Elasticsearch
`spring-boot-starter-data-jpa`	Starter for using Spring Data JPA with Hibernate
`spring-boot-starter-data-ldap`	Starter for using Spring Data LDAP
`spring-boot-starter-data-mongodb`	Starter for using MongoDB document-oriented database and Spring Data MongoDB
`spring-boot-starter-data-mongodb-reactive`	Starter for using MongoDB document-oriented database and Spring Data MongoDB Reactive
`spring-boot-starter-data-neo4j`	Starter for using Neo4j graph database and Spring Data Neo4j
`spring-boot-starter-data-redis`	Starter for using Redis key-value data store with Spring Data Redis and the Jedis client
`spring-boot-starter-data-redis-reactive`	Starter for using Redis key-value data store with Spring Data Redis reactive and the Lettuce client
`spring-boot-starter-data-rest`	Starter for exposing Spring Data repositories over REST using Spring Data REST
`spring-boot-starter-data-solr`	Starter for using the Apache Solr search platform with Spring Data Solr

Springboot与数据访问

Springboot与Mybatis整合

1、引入mybatis-starter - mybatis-spring-boot-starter

2、配置文件模式

3、注解模式实现

4、测试

Springboot与数据访问

Springboot与Mybatis整合步骤：

第一步：导入maven依赖

```
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>1.3.1</version>
</dependency>
```

第二步：添加配置

```
spring:
  datasource:
    username: root
    password: root  #修改为自己的数据库密码
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/test_db?useUnicode=true&characterEncoding=UTF-8&useSSL=false
```

Springboot与数据访问

Springboot与Mybatis整合步骤：

第三步：代码示例（springboot-demo工程）

1.注解模式

启动类

```
@MapperScan(value = “com.ibm.springboot.mapper”) //@MapperScan将包下的接口扫描装配到容器中
@SpringBootApplication
public class SpringbootApplication {
public static void main(String[] args) {
SpringApplication.run(SpringbootApplication.class, args);
}
}
```

或者在对应接口上添加@Mapper直接来指是操作数据库的mapper

Springboot与数据访问

Springboot与Mybatis整合步骤：

第三步：代码示例（springboot-demo工程）

controller层：

```
package com.ibm.springboot.controller;
@RestController
public class MyBatisController {
        @Autowired
        private DepartmentMapper departmentMapper;
        @Autowired
        private EmployeeMapper employeeMapper;
    //增加部门
    @PostMapping("/department/insert")
    public Department insertDept(@RequestBody Department department){
        departmentMapper.insertDept(department);
        return department;
    }
```

Springboot与数据访问

Springboot与Mybatis整合步骤：

controller层：

```
 //修改部门
@PutMapping("/department/update")
public void updateDept(@RequestBody Department department){
        departmentMapper.updateDept(department);
    }
 //查询部门
@GetMapping("/department/query/{id}")
    public Department getDepartment(@PathVariable("id") Integer id){
        return departmentMapper.getDeptById(id);
    }
@DeleteMapping("/department/delete/{id}")   //删除部门
    public void deleteDepartment(@PathVariable("id") Integer id){
        departmentMapper.deleteDeptById(id);
    }
}
```

Springboot与数据访问

接口层：

```
//指定这是一个操作数据库的mapper
//@Mapper或者@MapperScan将接口扫描装配到容器中
public interface DepartmentMapper {
    @Select("select * from department where id=#{id}")
    public Department getDeptById(Integer id);

    @Delete("delete from department where id=#{id}")
    public int deleteDeptById(Integer id);

    @Options(useGeneratedKeys = true,keyProperty = "id")
    @Insert("insert into department(departmentName) values(#{departmentName})")
    public int insertDept(Department department);

    @Update("update department set departmentName=#{departmentName} where id=#{id}")
    public int updateDept(Department department);
}
```

Springboot与数据访问

自定义MyBatis的配置规则；给容器中添加一个ConfigurationCustomizer来匹配javaBean中属性字段的驼峰形式

```
package com.ibm.springboot.config;
import org.apache.ibatis.session.Configuration;
import org.mybatis.spring.boot.autoconfigure.ConfigurationCustomizer;
import org.springframework.context.annotation.Bean;
@org.springframework.context.annotation.Configuration
public class MyBatisConfig {
    @Bean
    public ConfigurationCustomizer configurationCustomizer(){
        return new ConfigurationCustomizer(){
            @Override
            public void customize(Configuration configuration) {
                configuration.setMapUnderscoreToCamelCase(true);
            }
        };
    }
}
```

Springboot与数据访问

2.配置文件模式

2.1.创建一个mybatis的全局配置文件，在文件中添加配置来匹配javaBean中属性字段的驼峰形式：

文件名： classpath:mybatis/mybatis-config.xml

文件内容：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
 <configuration>
  <settings>
      <setting name="mapUnderscoreToCamelCase" value="true"/>
  </settings>
</configuration>
```

2.2.需要在配置文件中添加如下配置：

```
mybatis:
    # 指定全局配置文件位置
    config-location: classpath:mybatis/mybatis-config.xml
    # 指定sql映射文件位置
    mapper-locations: classpath:mybatis/mapper/*.xml
```

Springboot与数据访问

2.3.代码示例

controller层:

```
package com.ibm.springboot.controller;

@RestController
public class MyBatisController {
    @Autowired
    private DepartmentMapper departmentMapper;
    @Autowired
    private EmployeeMapper employeeMapper;

    /**
    * 新增员工
    * @param employee
    */
    @PostMapping("/employee/insert")
    public void insertEmployee(@RequestBody Employee employee) {
        employeeMapper.insertEmp(employee);
    }
```

Springboot与数据访问

```
/**
 * 查询员工
 * @param id
 * @return
 */
@GetMapping("/employee/query/{id}")
    public Employee getEmployee(@PathVariable("id") Integer id){
        return employeeMapper.getEmpById(id);
    }
}
```

Mapper层：

```
package com.ibm.springboot.mapper;
import com.ibm.springboot.bean.Employee;

//@Mapper或者@MapperScan将接口扫描装配到容器中
public interface EmployeeMapper {
    public Employee getEmpById(Integer id);
    public void insertEmp(Employee employee);
}
```

Springboot与数据访问

Mapper层接口对应的sql映射文件：

文件名：classpath:mybatis/mapper/*EmployeeMapper*.xml

内容实现：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.ibm.springboot.mapper.EmployeeMapper">
    <select id="getEmpById" resultType="com.ibm.springboot.bean.Employee">
        SELECT * FROM employee WHERE id=#{id}
    </select>

    <insert id="insertEmp">
        INSERT INTO employee(lastName,email,gender,d_id) VALUES
(#{lastName},#{email},#{gender},#{dId})
     </insert>
  </mapper>
```

随堂练习作业：

根据上面的示例，实现Springboot和Mybatis的集成，连接mysql数据库。通过浏览器请求输出值。(值不限)

目 录

Springboot与启动配置原理

- SpringApplication.run(主程序类)
  - **new SpringApplication(主程序类)**
    - 判断是否web应用
    - 加载并保存所有ApplicationContextInitializer（META-INF/spring.factories），
    - 加载并保存所有ApplicationListener
    - 获取到主程序类
  - **run()**
    - 回调所有的SpringApplicationRunListener（META-INF/spring.factories）的starting
    - 获取ApplicationArguments
    - 准备环境&回调所有监听器（ SpringApplicationRunListener ）的environmentPrepared
    - 打印banner信息
    - 创建ioc容器对象（
      - AnnotationConfigEmbeddedWebApplicationContext（web环境容器）
      - AnnotationConfigApplicationContext（普通环境容器）

Springboot与启动配置原理

- **run()**
  - 准备环境
    - 执行**ApplicationContextInitializer**. initialize()
    - 监听器**SpringApplicationRunListener**回调contextPrepared
    - 加载主配置类定义信息
    - 监听器**SpringApplicationRunListener**回调contextLoaded
  - 刷新启动IOC容器；
    - 扫描加载所有容器中的组件
    - 包括从META-INF/spring.factories中获取的所有**EnableAutoConfiguration**组件
  - 回调容器中所有的**ApplicationRunner**、**CommandLineRunner**的run方法
  - 监听器SpringApplicationRunListener回调finished

Springboot与启动配置原理

- Spring Boot启动扫描所有jar包的META-INF/spring.factories中配置的EnableAutoConfiguration组件
- spring-boot-autoconfigure.jar\META-INF\spring.factories有启动时需要加载的EnableAutoConfiguration组件配置
- 配置文件中使用debug=true可以观看到当前启用的自动配置的信息
- 自动配置会为容器中添加大量组件
- Spring Boot在做任何功能都需要从容器中获取这个功能的组件
- Spring Boot 总是遵循一个标准；容器中有我们自己配置的组件就用我们配置的，没有就用自动配置默认注册进来的组件；

目 录

一、异步任务

在Java应用中，绝大多数情况下都是通过同步的方式来实现交互处理的；但是在处理与第三方系统交互的时候，容易造成响应迟缓的情况，之前大部分都是使用多线程来完成此类任务，其实，在Spring 3.x之后，就已经内置了@Async来完美解决这个问题。

两个注解：

@EnableAysnc、@Aysnc

SpringBoot异步任务

代码示例：启动类：

```
@EnableAsync  //开启异步注解功能
@SpringBootApplication
public class SpringbootTaskApplication {

public static void main(String[] args) {
SpringApplication.run(SpringbootTaskApplication.class, args);
}
}
```

controller层：

```
@RestController
public class AsyncController {
@Autowired
 AsyncService asyncService;
@GetMapping("/async/hello")
 public String hello(){
    asyncService.hello();
    return "success";
}
}
```

SpringBoot异步任务

Service层:

```
package com.ibm.springboot.demo.service;

import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

@Service
public class AsyncService {

    //告诉Spring这是一个异步方法
    @Async
    public void hello(){
        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("处理数据中...");
    }
}
```

SpringBoot定时任务

二、定时任务

项目开发中经常需要执行一些定时任务，比如需要在每天凌晨时候，分析一次前一天的日志信息。Spring为我们提供了异步执行任务调度的方式，提供TaskExecutor 、TaskScheduler 接口。

两个注解：@EnableScheduling、@Scheduled

cron表达式：

字段	允许值	允许的特殊字符
秒	0-59	, - * /
分	0-59	, - * /
小时	0-23	, - * /
日期	1-31	, - * ? / L W C
月份	1-12或JAN-DEC	, - * /
星期	1-7或SUN-SAT	, - * ? / L C #
年（可选）	空,1970-2099	, - * /

特殊字符	代表含义
,	枚举
-	区间
*	任意
/	步长
?	日/星期冲突匹配
L	最后
W	工作日
C	和calendar联系后计算过的值
#	星期，4#2，第2个星期三

SpringBoot定时任务

代码示例：

启动类：

```
@EnableScheduling //开启基于注解的定时任务
@SpringBootApplication
public class SpringbootTaskApplication {
        public static void main(String[] args) {
        SpringApplication.run(SpringbootTaskApplication.class, args);
  }
}
```

服务层：

```
package com.ibm.springboot.service;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;

@Service
public class ScheduledService {
```

SpringBoot定时任务

代码示例：（接上页）

```
/**
   * second(秒), minute（分）, hour（时）, day of month（日）, month（月）, day of week
(周几）.
   * 0 * * * * MON-FRI
   *  【0 0/5 14,18 * * ?】 每天14点整，和18点整，每隔5分钟执行一次
   *  【0 15 10 ? * 1-6】 每个月的周一至周六10:15分执行一次
   *  【0 0 2 ? * 6L】每个月的最后一个周六凌晨2点执行一次
   *  【0 0 2 LW * ?】每个月的最后一个工作日凌晨2点执行一次
   *  【0 0 2-4 ? * 1#1】每个月的第一个周一凌晨2点到4点期间，每个整点都执行一次；
   */
  // @Scheduled(cron = "0 * * * * MON-SAT")
  // @Scheduled(cron = "0,1,2,3,4 * * * * MON-SAT")
  // @Scheduled(cron = "0-4 * * * * MON-SAT")
   @Scheduled(cron = "0/4 * * * * MON-SAT") //每4秒执行一次
   public void hello(){
       System.out.println("hello ... ");
   }
}
```

三、邮件任务

- 邮件发送需要引入spring-boot-starter-mail
- Spring Boot 自动配置MailSenderAutoConfiguration
- 定义MailProperties内容，配置在application.yml中
- 自动装配JavaMailSender
- 测试邮件发送

SpringBoot集成邮件功能

邮件功能的应用场景

邮件功能的应用场景可谓十分广泛，比如注册用户、密码找回，消息通知、以及一些程序异常报警通知等都需要使用到该功能。

正是由于邮件功能的使用广泛，因此springboot也加在它的组件中添加了邮件。

springboot知识点简单回顾

springboot是spring家族中微型框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。springboot通过自动配置和起步依赖可快速构建项目，敏捷式开发。

本节课学习目标

利用springboot框架集成邮件功能，实现发送邮件的效果。

SpringBoot集成邮件功能

Springboot实现发送邮件功能步骤

1.获取邮箱的授权码，用来发送邮件 (以163邮箱为例)

163邮箱- > 设置 -> POP3/SMTP/IMAP -> 开启IMAP/SMTP服务和POP3/SMTP服务 -> 开启客户端授权码

IMAP全称为Internet Message Access Protocol（互联网邮件访问协议），IMAP允许从邮件服务器上获取邮件的信息、下载邮件等。

SMTP全称为Simple Mail Transfer Protocol（简单邮件传输协议），它是一组用于从源地址到目的地址传输邮件的规范，通过它来控制邮件的中转方式。

POP3全称为Post Office Protocol 3（邮局协议），POP3支持客户端远程管理服务器端的邮件。

SpringBoot集成邮件功能

Springboot实现发送邮件功能步骤　项目springboot_integrate_mail为例

2.配置邮件服务 在项目pom.xml文件中添加spring-boot-starter-mail依赖

```
<!--邮件模块依赖-->
<dependency>
  <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-mail</artifactId>
 </dependency>
```

3.配置application.properties文件

```
spring.mail.host=smtp.163.com
spring.mail.username=Chang168daydayup@163.com
spring.mail.password=LYNZGLGDOMEAFTLO
spring.mail.default-encoding=utf-8
spring.mail.protocol=smtps
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.required=true
spring.mail.to=Chang168daydayup@163.com
spring.mail.cc=Chang168daydayup@163.com
```

SpringBoot集成邮件功能

Springboot实现发送邮件功能步骤　项目springboot_integrate_mail为例

4.代码实现 EmailController

```
package com.ibm.controller;
import java.io.File;
import javax.mail.MessagingException;
import javax.mail.internet.MimeMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EmailController {
```

SpringBoot集成邮件功能

Springboot实现发送邮件功能步骤　项目springboot_integrate_mail为例

4.代码实现 EmailController类

```
@Autowired
private JavaMailSender mailSender;

@Value("${spring.mail.username}")
private String mailFrom;

@Value("${spring.mail.to}")
private String mailTo;

@Value("${spring.mail.cc}")
private String mailCc;
```

SpringBoot集成邮件功能

Springboot实现发送邮件功能步骤　项目springboot_integrate_mail为例

4.代码实现 EmailController类 简单邮件发送方法sendMail()

```
@GetMapping("/simple/mail/send")
 public String sendMail() {
    SimpleMailMessage message =  new SimpleMailMessage();
    //发送者
    message.setFrom(mailFrom);
    //主题
    message.setSubject("简单测试邮件");
    message.setTo(mailTo);
    message.setCc(mailCc);
    message.setText("这个是简单测试邮件！");
    mailSender.send(message);
    return "send simpleMailMessage success";
 }
```

SpringBoot集成邮件功能

Springboot实现发送邮件功能步骤　项目springboot_integrate_mail为例

4.代码实现 EmailController类 邮件发送附件方法complexMailSend()

```
@GetMapping("/complex/mail/send")
  public String complexMailSend() throws MessagingException {
                    //创建一个复杂的消息邮件
                    MimeMessage mimeMessage = mailSender.createMimeMessage();
                    MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, true);
                    //邮件设置
                    helper.setSubject("邮件附件发送测试");
                    helper.setText("<b style='color:red'>发送邮件附件成功</b>",true);
                    helper.setTo(mailTo);
                    helper.setFrom(mailFrom);
                    //上传文件
                    helper.addAttachment("1.jpg",new File("src/main/resources/uploadFile/1.jpg"));
                    helper.addAttachment("2.jpg",new File("src/main/resources/uploadFile/2.jpg"));
                    mailSender.send(mimeMessage);
                    return "send complexMailMessage success";
          }
}
```

SpringBoot集成邮件功能

Springboot实现发送邮件功能步骤　项目springboot_integrate_mail为例

5. 测试邮件发送

简单邮件测试：启动项目服务，打开浏览器输入http://localhost:8080/simple/mail/send
页面返回正确信息：send simpleMailMessage success

复杂邮件测试：启动项目服务，打开浏览器输入http://localhost:8080/complex/mail/send
页面返回正确信息：send complexMailMessage success

练习作业：

根据上面的示例，实现配置QQ邮件，实现QQ邮箱定时任务发送邮件的功能。

SpringBoot核心功能总结

- 一、独立运行Spring项目

Spring boot 可以以jar包形式独立运行，运行一个Spring Boot项目只需要通过java -jar xx.jar来运行。

- 二、内嵌servlet容器

Spring Boot可以选择内嵌Tomcat、jetty或者Undertow,这样我们无须以war包形式部署项目。

- 三、提供starter简化Maven配置

spring提供了一系列的start pom来简化Maven的依赖加载，例如，当你使用了spring-boot-starter-web，会自动加入所需依赖包。

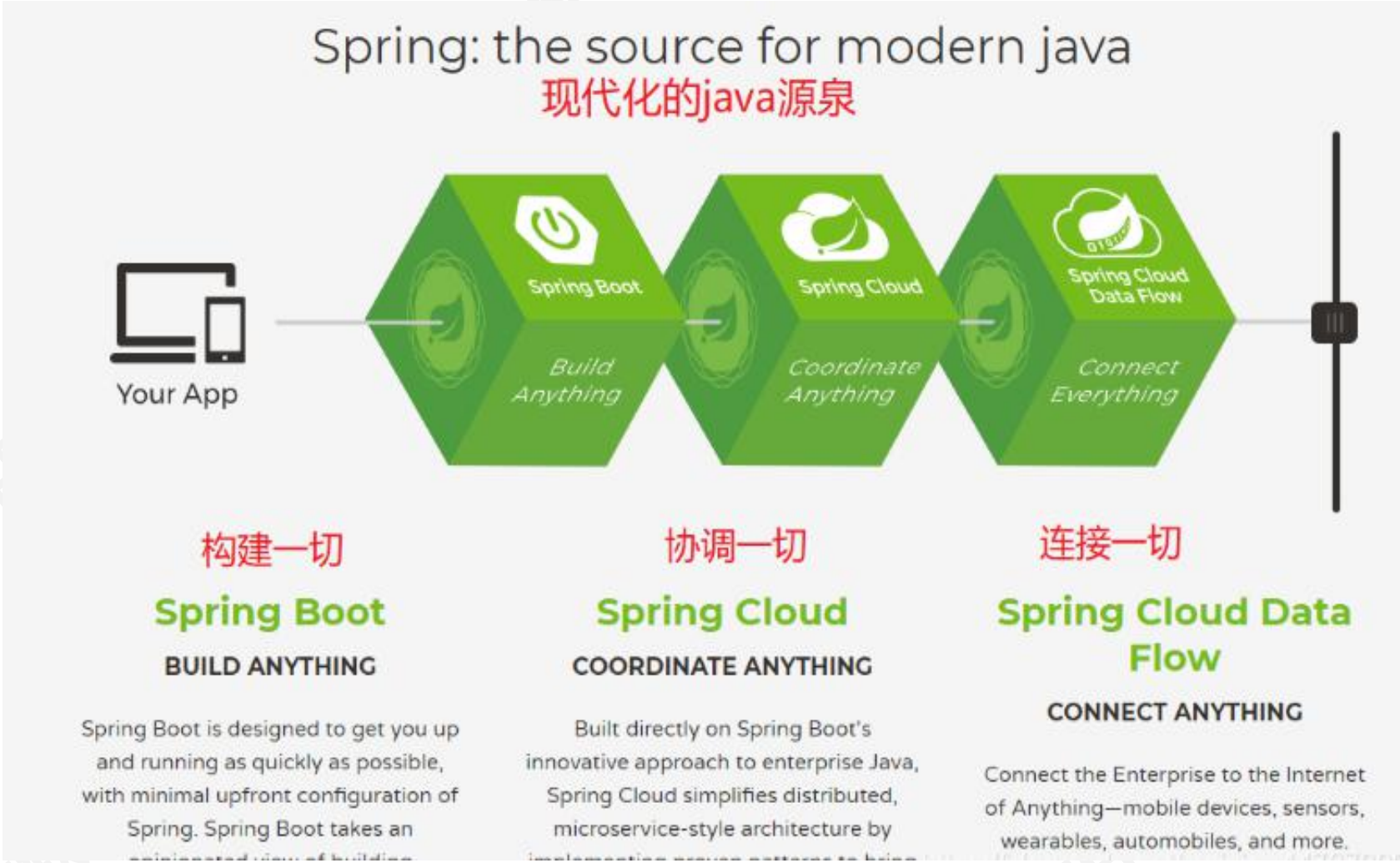
SpringBoot核心功能总结

- 四、自动装配Spring
- SpringBoot会根据在类路径中的jar包，类、为jar包里面的类自动配置
- Bean，这样会极大地减少我们要使用的配置。当然，SpringBoot只考
- 虑大多数的开发场景，并不是所有的场景，若在实际开发中我们需要
- 配置Bean，而SpringBoot灭有提供支持，则可以自定义自动配置。

- 五、准生产的应用监控
- SpringBoot提供基于http ssh telnet对运行时的项目进行监控。

- 六、无代码生产和xml配置
- SpringBoot不是借助与代码生成来实现的，而是通过条件注解来实现的，
- 这是Spring4.x提供的新特性。

SpringBoot框架知识拓展



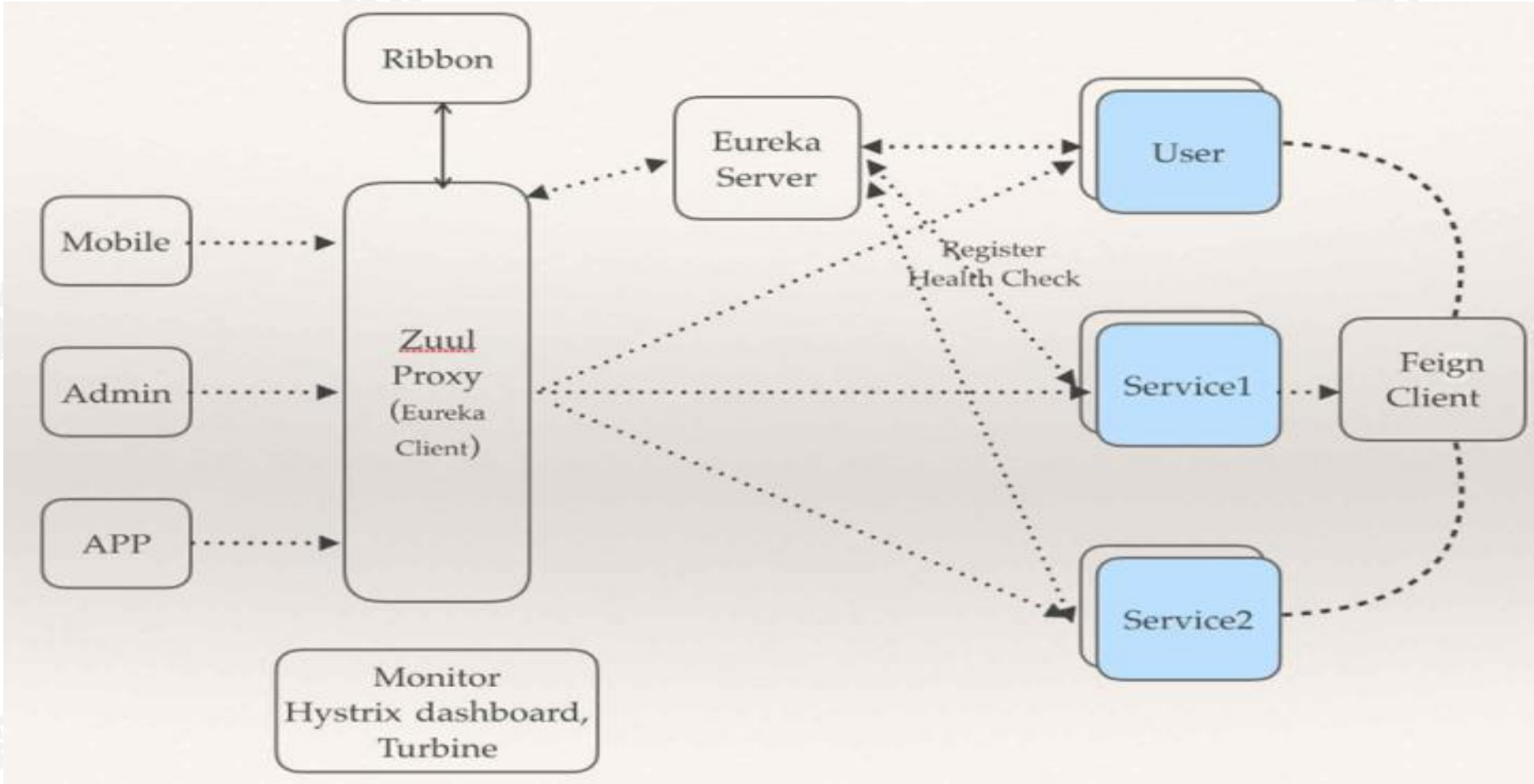
SpringBoot框架知识拓展

Spring Boot 在简化配置、打包和集成第三方工具方面确实做得很好，可以减低 Spring 开发人员的入门门槛。

Spring Cloud 是 Pivotal 推出的基于Spring Boot的一系列框架的集合，旨在帮助开发者快速搭建一个分布式的服务或应用。Spring Cloud 由众多子项目组成，如Spring Cloud Config、Spring Cloud Netflix、Spring Cloud Consul等，提供了搭建分布式系统及微服务常用的工具，如配置管理、服务发现、服务容错、服务路由等。

SpringBoot框架知识拓展

基于Spring Cloud微服务技术框架关系图



SpringBoot框架知识拓展

网易云轻舟微服务就是基于这样的理念设计的，并且是基于开源、兼容开源的。在微服务框架层面，轻舟微服务基于 Spring Cloud 优化，并兼容 Dubbo。

思无极 行有方 达天下