Spring Framework Servlet-API

MVC Architecture : Manual

Architecture is implemented strictly, disciplined way

remove lot of Boiler-plate code

abstract the low level complexity

Focus more on business logic

Most popular frameworks to develop java application

J2EE : Java 2 Enterprise Edition : Framework to develop web app using java

Complex in nature

lots of deployment descriptor

lots of interface, abstract classes needs to be created to expose a single service

productivity reduces, reduces efficiency

Rod Johnson

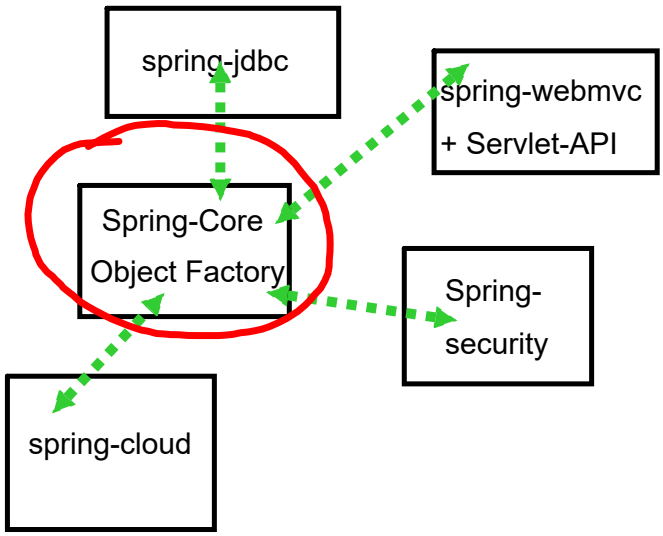
=> Object Factory : Responsible for creating and managing object

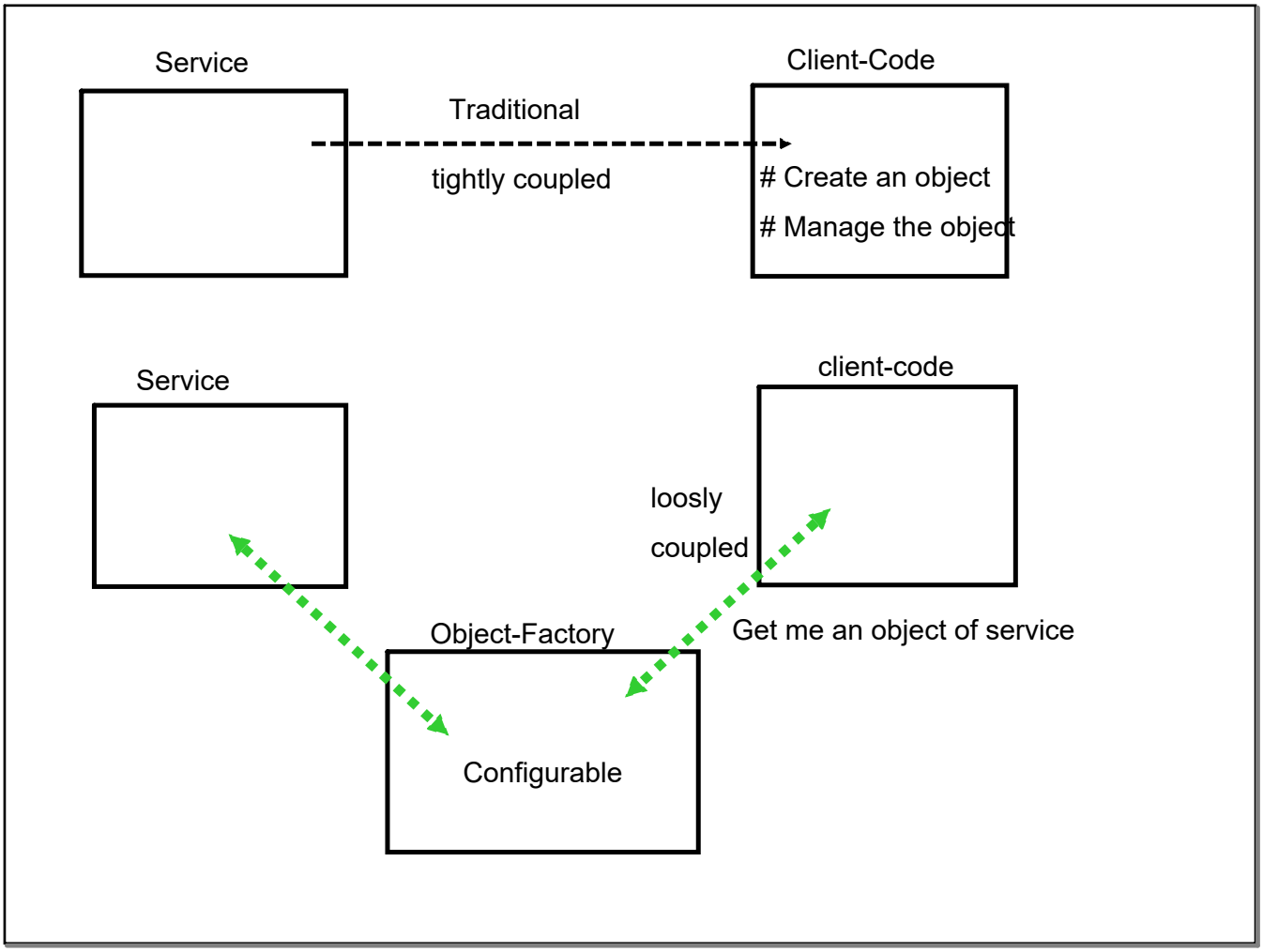
Increased the Productivity

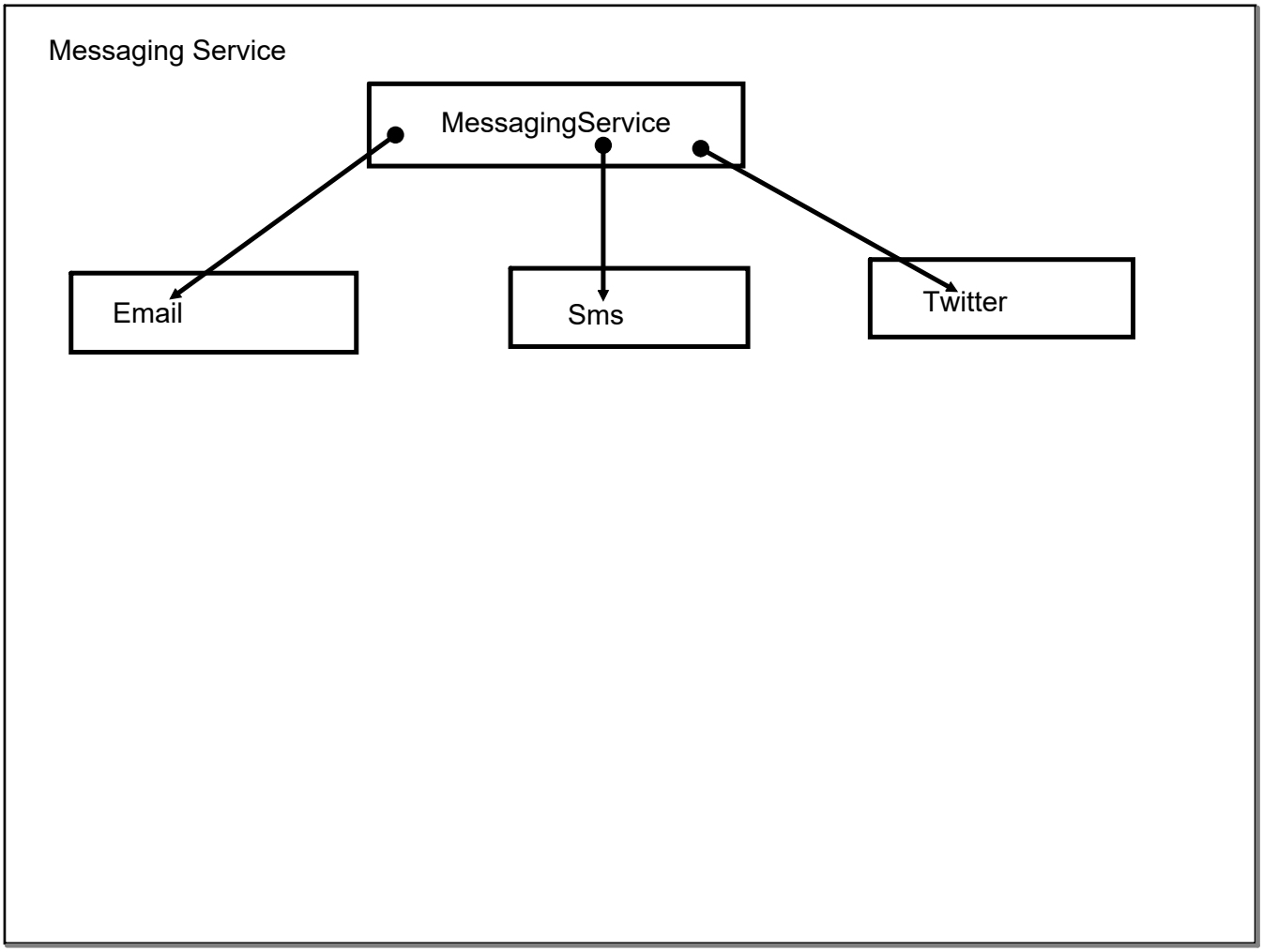
Increased the efficiency

1. Object Factory
2. Highly Modular
3. POJOs

Spring Framework




Service
Client-Code
Traditional
tightly coupled
Create an object
Manage the object
Service
client-code
loosly
coupled
Object-Factory
Get me an object of service
Configurable


Messaging Service
MessagingService
Email
Sms
Twitter

Object Factory | Bean Factory | Application Context

Provided by Spring - Core Module

A Custom Configuration needs to be provided to define the behavior of Object Factory

XML Based Configuration (Legacy)
Annotation Based Configuration (Modern)
Pure Java Based Configuration (Modern)

Std Spring Framework :
bundle of few Modules
=> Core
=> Spring-web-mvc
=> Spring AOP (proxy)

Bean Factory works on two key principals

1. IoC : Inversion of Control
2. DI : Dependency Injection

IoC : Outsourcing the (control of) creation and management of Object

XML Based Config :

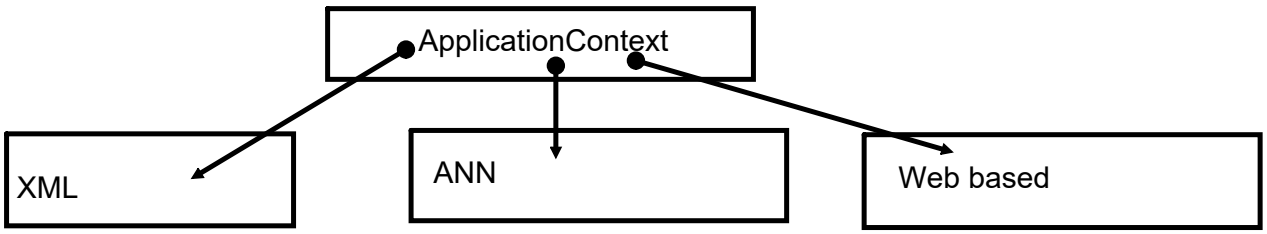
XML file + certain dependencies for support of additional spring tags

BEAN : Container(Object Factory) managed Object

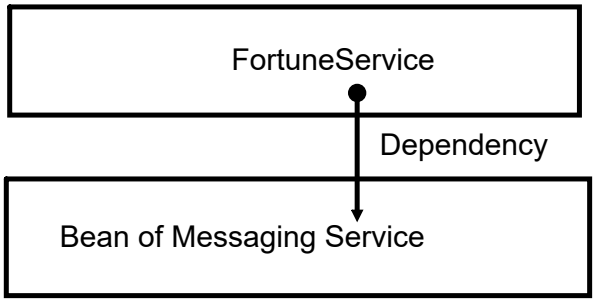
Multiple classes provided for Bean Factory

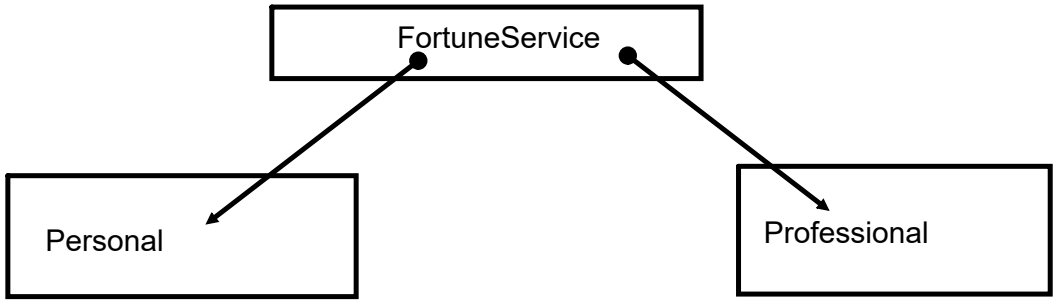
way of config (XML or java)

env for which bean factory (simple java, web app)



Each bean must have a unique id





Need the BEan factory to inject Fortune Service instance into EmailService

Constructor Based DI

Setter BAsed DI

```
<bean id="personal"
class="com.wf.training.spring.factory.service.PersonalFortune"></bean>
    <bean id="professional"
     <!-- Injecting Dependency -->

        <!-- Constructor Based -->
     <bean id="emailservice"
class="com.wf.training.spring.factory.service.EmailService">
        <constructor-arg ref="personal"/>
     </bean>
```

Injecting the literal values :

Delegate them to a text file (properties files)

literal values as key-value pair

need to specify property file in config

Bean Management :

1. Life cycle
2. Scope

=> Scope : Accessibility of bean

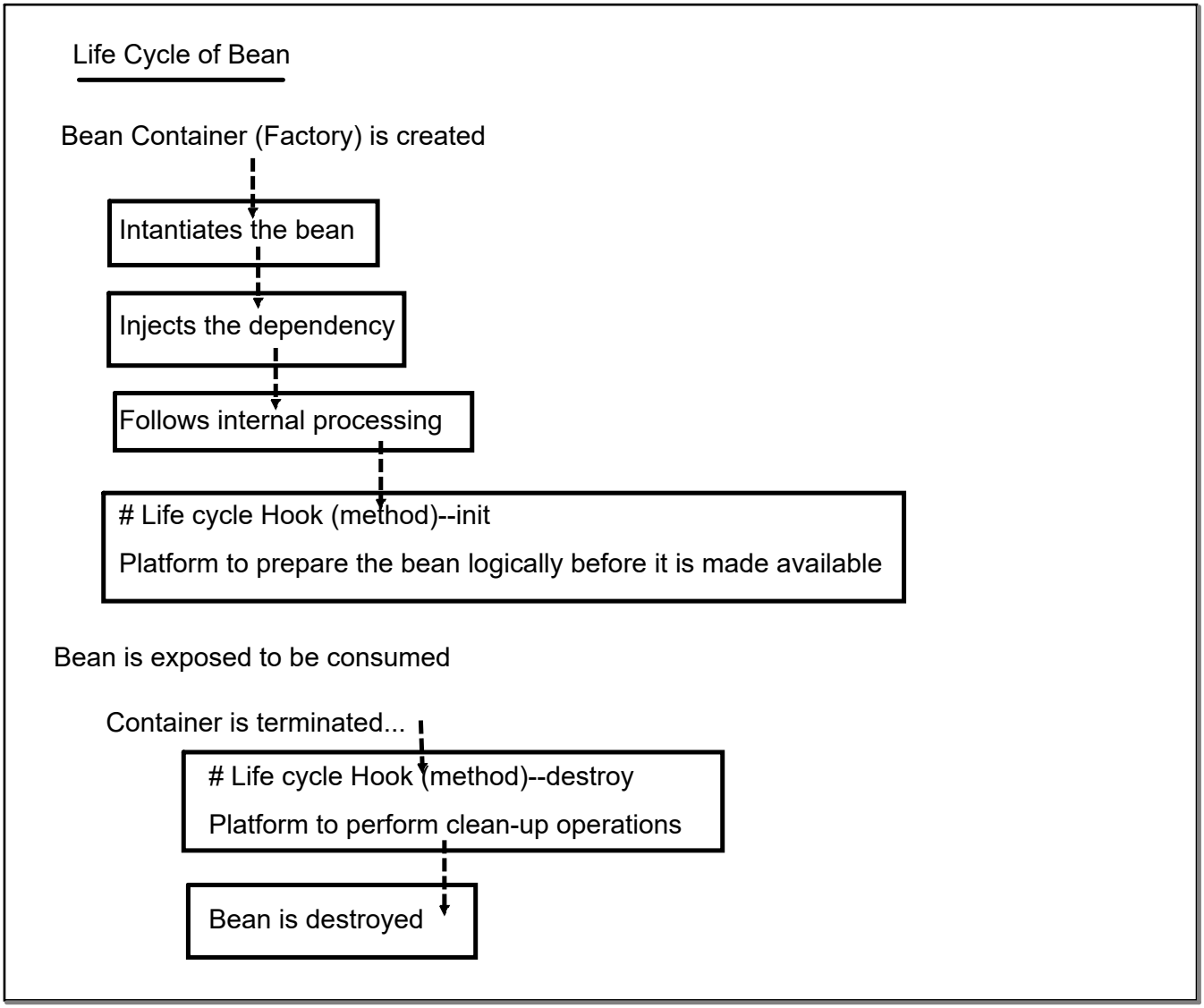
by default scope : Singleton : Single instance will be created

: Prototype : Diff each time

request

session : Web based

application


Life Cycle of Bean
Bean Container (Factory) is created
Intantiates the bean
Injects the dependency
Follows internal processing
Life cycle Hook (method)--init
Platform to prepare the bean logically before it is made available
Bean is exposed to be consumed
Container is terminated...
Life cycle Hook (method)--destroy
Platform to perform clean-up operations
Bean is destroyed

Prototype : BEan container does not maintain life cycle..

Annotation based config

xml file : path reference

Creating the bean

@Component :

Any class decorated with @Component will be initiated by bean factory

By default the class name itself becomes the id , first character being small case...

DI using annotation

1. Constructor
2. Setter
3. Field

@Autowired : search for bean, if found, inject it

Scope : @Scope

Life cycle hook methods : Annotations

Pure Java Based Config :

xml file will be replaced by Java class

Pure Java Config :

Programmatically configure Bean Factory

before 10 am or after 5 pm : personal fortune

else : professional fortune

Expose the bean

@Component	@Bean
Class level	Method level

Spring web-mvc module : MVC architecture

uses Servlet-API : in an abstract

POJO

Controller : Servlet

View : JSP

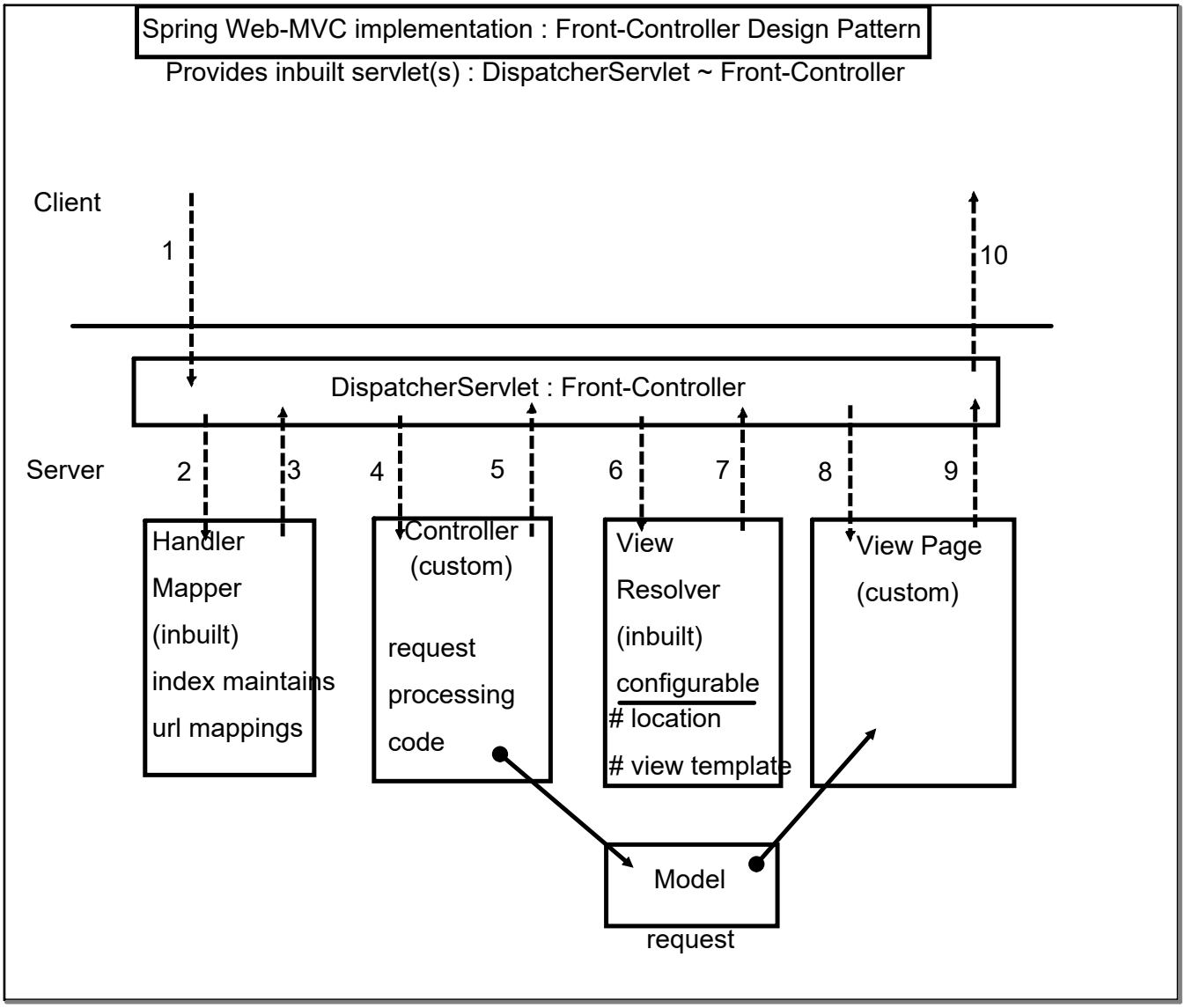
Model : Data Structure

Controller : POJOs (Servlet capabilities)

View : Spring supports multiple view templates

- default : JSP + JSTL
- Thymeleaf
- Mustache
- FreeMarker
- Velocity
- Tiles

Model : Data Structure/Data Container


Spring Web-MVC implementation : Front-Controller Design Pattern
Provides inbuilt servlet(s) : DispatcherServlet ~ Front-Controller
Client
Server
1
2
3
4
5
6
7
8
9
10
DispatcherServlet : Front-Controller
Handler Mapper (inbuilt) index maintains url mappings
Controller (custom) request processing code
View Resolver (inbuilt) configurable # location # view template
View Page (custom)
Model
request

1. Need to register the DispatcherServlet
2. COnfig to target all requests to DS

web.xml (servlet-api)

Spring Based Config : (XML)

+ Bean Factory Config

+ web mvc config

<servlet-name>-servlet.xml

eg: dispatcher-servlet.xml

View Resolver :

eg : "index" (string) name of view page (controller)

prefix : location

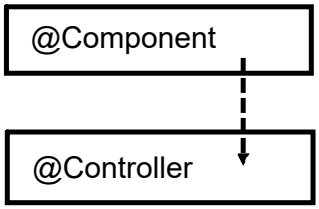
suffix : View Template (extension)

```
<property name="prefix" value="/WEB-INF/views/"/>
<property name="suffix" value=".jsp"/>
```

/WEB-INF/views/index.jsp

Custom Resources

1. Controller : POJO, registered with Handler Mapper



identifying the HTTP Verb

Mapping will take place using getter/setter only

Maven Project

1. archetype : web
2. Add the Server Runtime Library
3. convert java 1.5 to 1.8
4. Adding dependencies
    1. spring framework
    2. servlet for DS
    3. jsp+jstl

Pure Java Config

~ add a maven plugin
web.xml (servlet-api)
~ Java Class

dispatcher-servlet.xml (spring) ~ Java class

Java Class for web.xml

- # Registered DS (auto - inherit inbuilt class)
- # Mapped the url

Java Class for Spring config

- # component scanning path
- # exposed a bean of ViewResolver

Form handling spring-way : Forms are critical

Custom Tag Library : JSP

Need to add the reference of custom tag library

Spring forms : map the forms (UI) with java classes

control the form behavior (UI) through java classes

Validation : Validator API : Hibernate-Validator (dependency)

Client - Side Validation : submission takes place when all constraint are satisfied
HTML5 attribute + JS

Server-Side Validation :

Absolute URL : fetch the context path : predeclared variable in JSP to access the context path

Annotation : interface(skeleton) + implementation class (logic)

@EmployeeCode
Rule
1. Retention policy : compile/runtime
compile : @Override/@FunctionalInterface
runtime : validation
2. Target : where that annotation can be used
method, class, field...

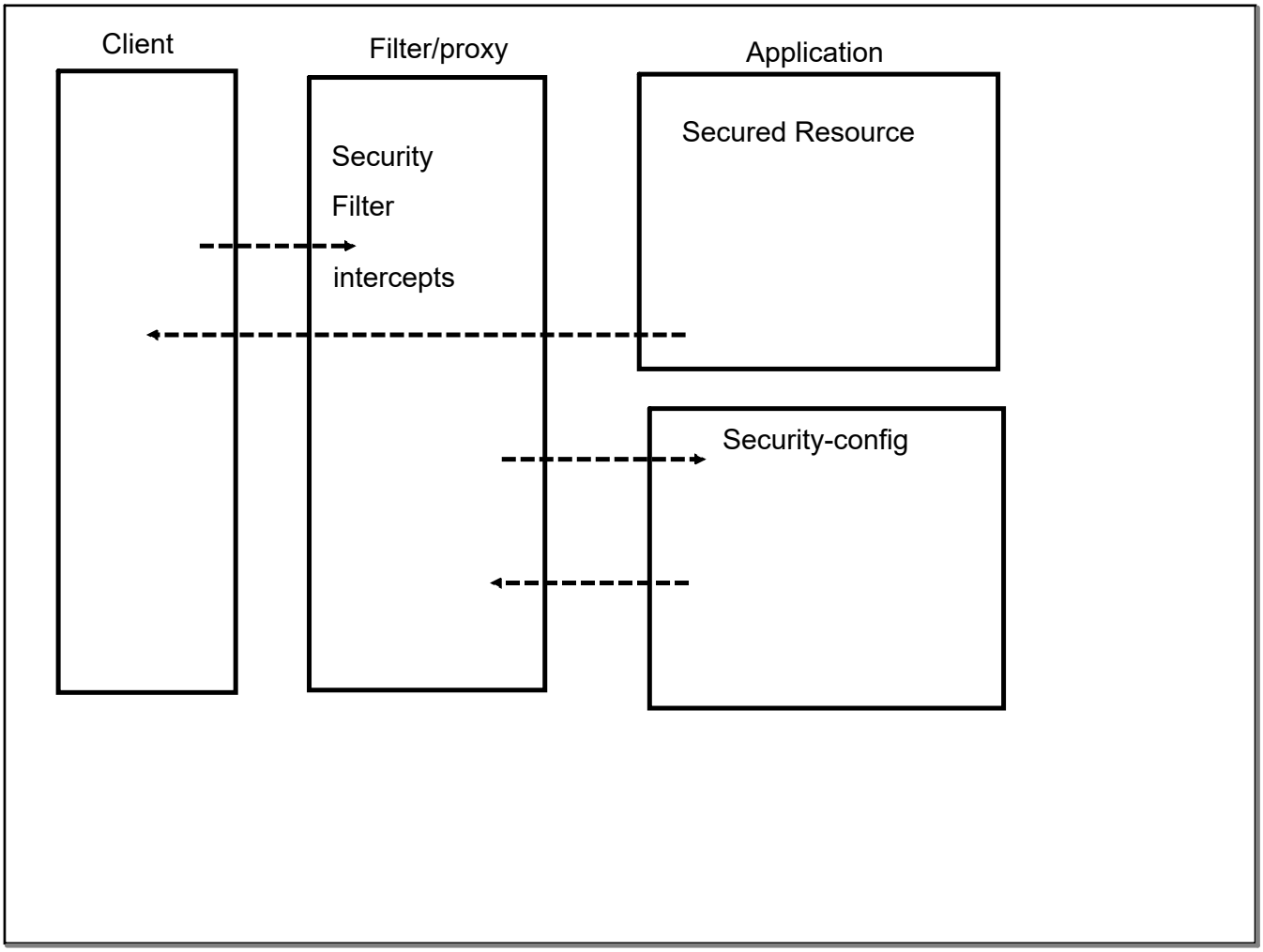
Spring - Security Module

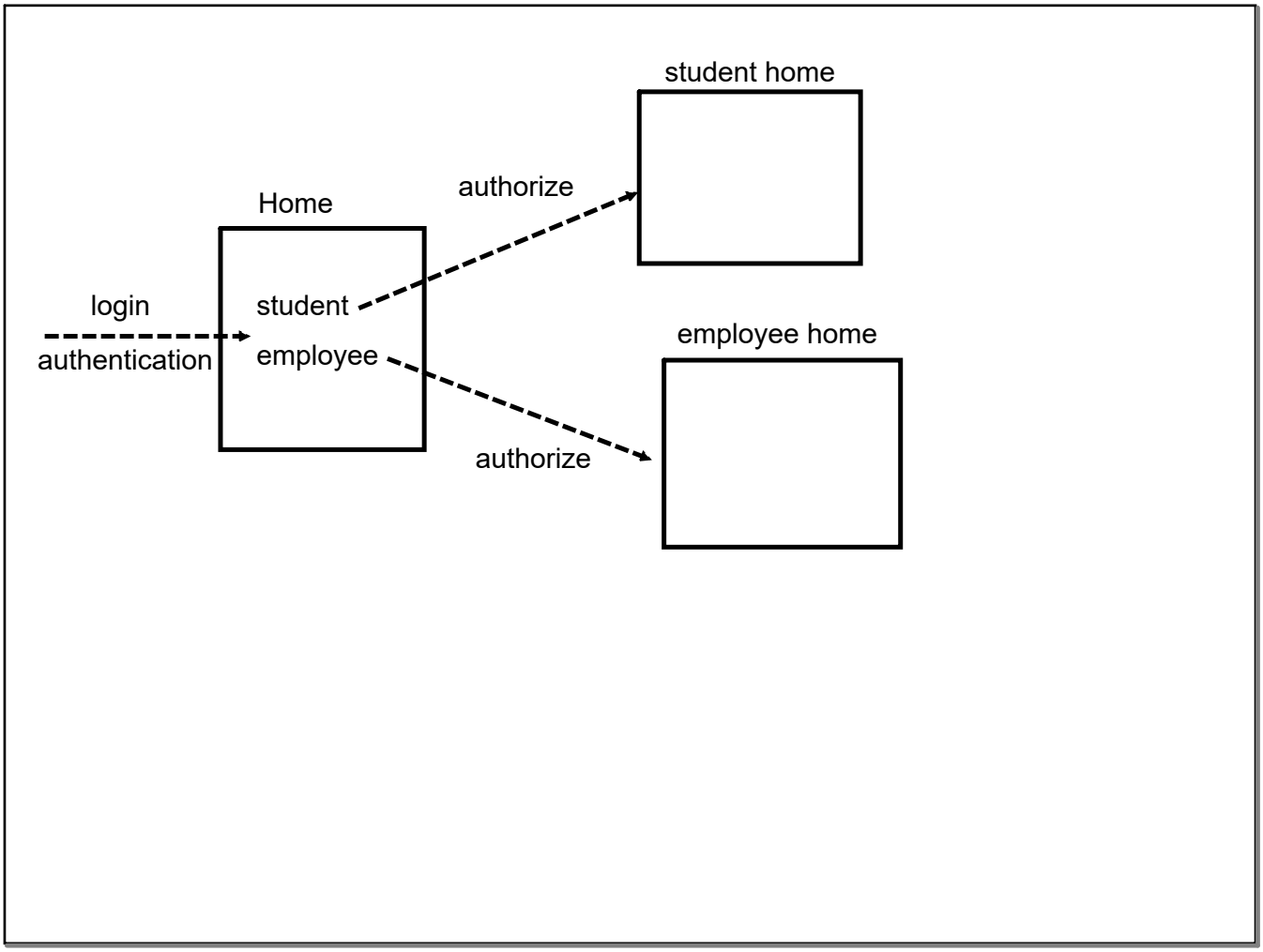
authentication(valid) + authorization (role based)

Dependency:

spring-security-web
spring-security-config
spring-security-taglibs

Spring Security Module


Client
Filter/proxy
Application
Secured Resource
Security
Filter
intercepts
Security-config


student home
authorize
Home
login
student
authentication
employee
employee home
authorize

1.Add Dependency

2. a class to activate security filter

3. a class to add config

default security (all resources are secured) :

provide an inbuilt login form

3 authentication ways

1. httpBasic : not recommended
2. formLogin (inbuilt)
3. formLogin (custom)

form : spring-form

security over CSRF attack

spring security for every form , will pass a security token, while submission same token needed to sent back, checked by security filter

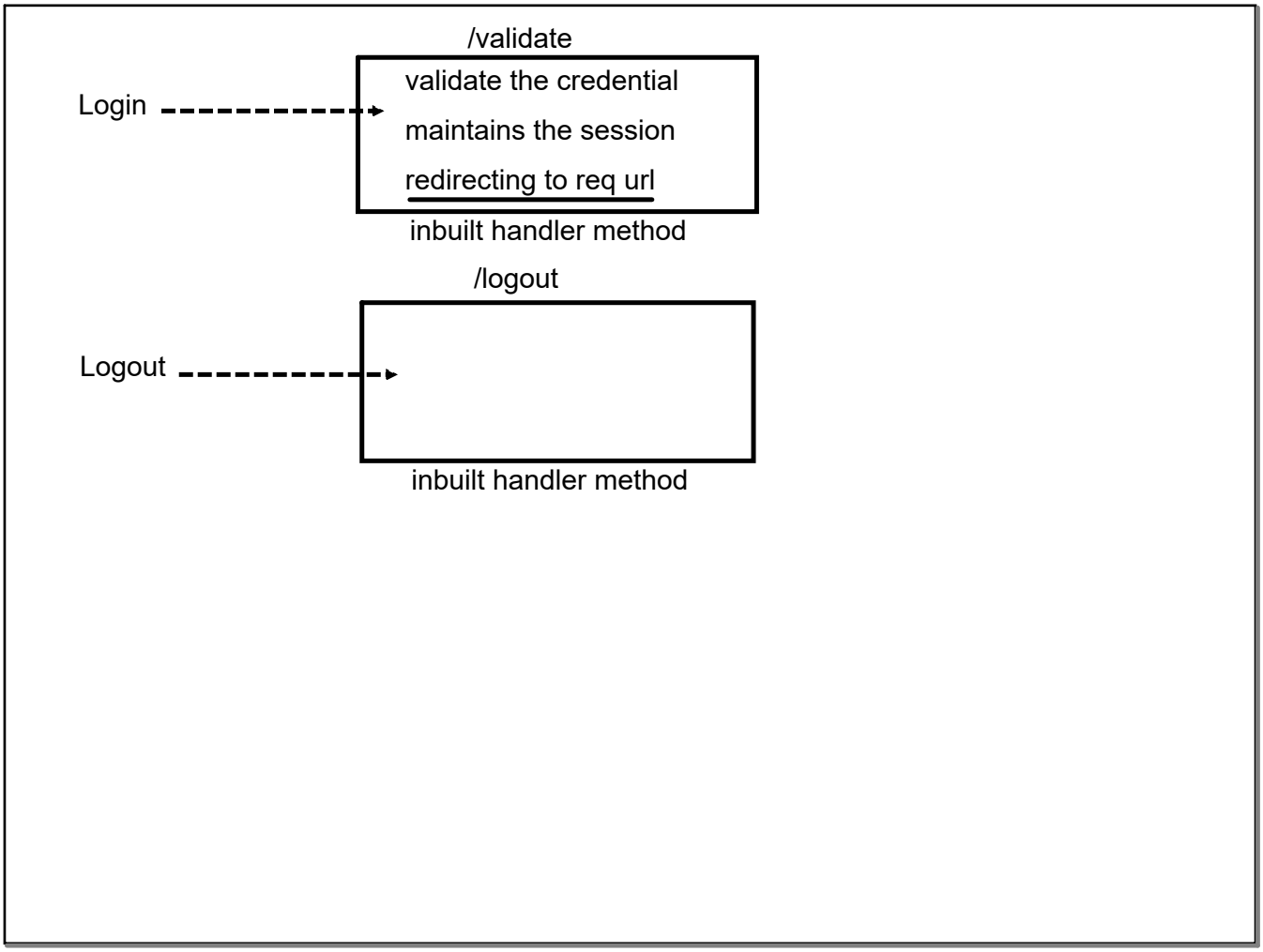
Every form must be spring-form

Custom login (spring-form)

login : username

password : password

method : POST


/validate
validate the credential
maintains the session
redirecting to req url
Login
inbuilt handler method
/logout
Logout
inbuilt handler method

taglib of security

1. get some info about logged in user

Spring-Boot Framework (Tool)

make the development of Spring app easier

Performance outcome remains same

Spring - boot

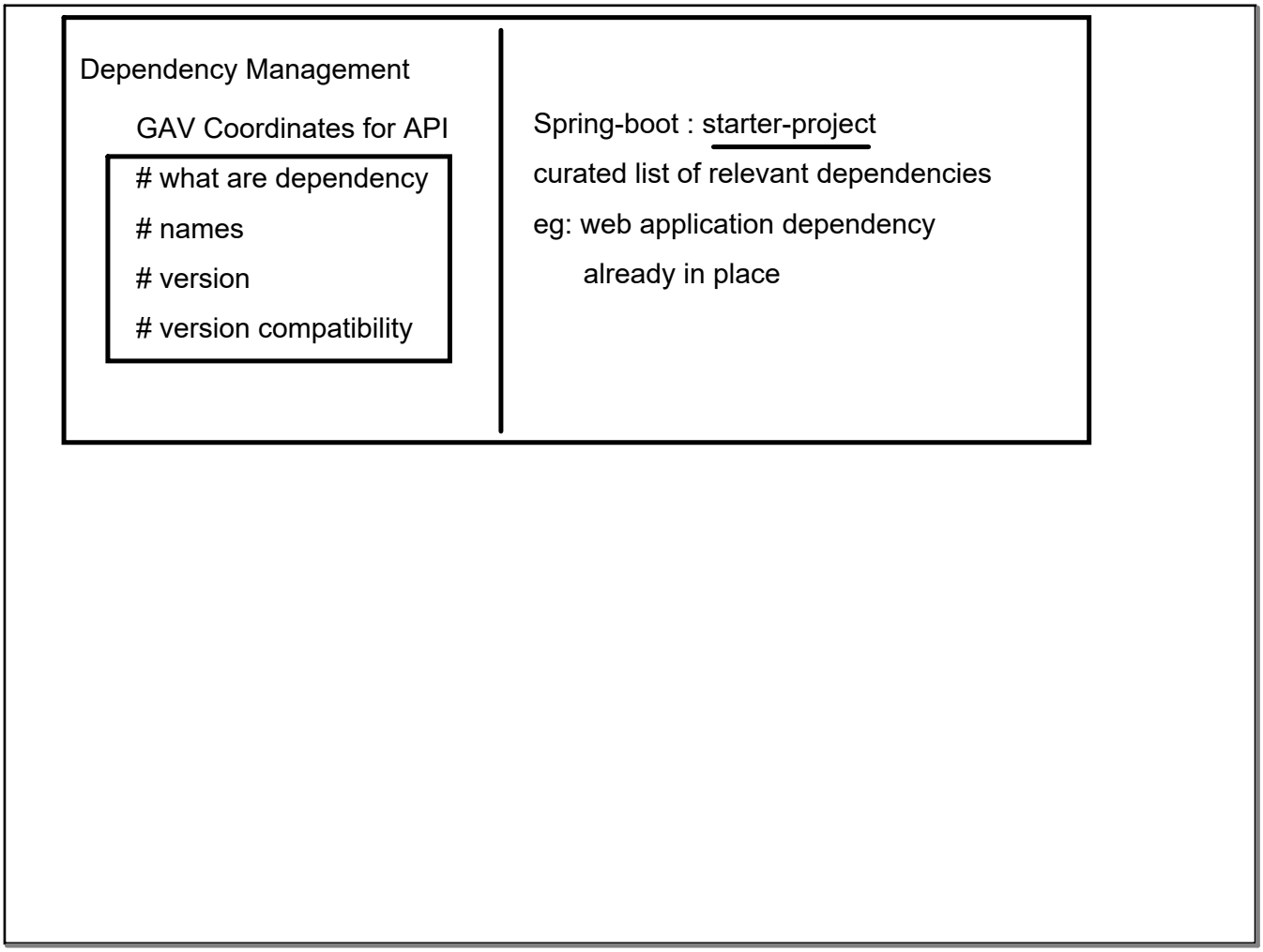
relevant , in sync with current industry std.

Dependency management

Auto Configuration

Reduction in boiler-plate code for custom config

Self - Sufficient in nature


Dependency Management
GAV Coordinates for API
what are dependency
names
version
version compatibility
Spring-boot : starter-project
curated list of relevant dependencies
eg: web application dependency
already in place

Auto Configuration

=> Dependency : looking at dependencies in pom.xml , default & std config auto done

=> Specialized annotation : auto config

Custom Config : Reduction in boiler-plate code

heavily rely on properties

key-value pair

list of predefined key (spring-boot) : possible value

Self-Sufficient

eg: spring web application using maven tool

=> Maven installed/maven plugin

=> IDE support

=> Tomcat Server

=> Deploying the app

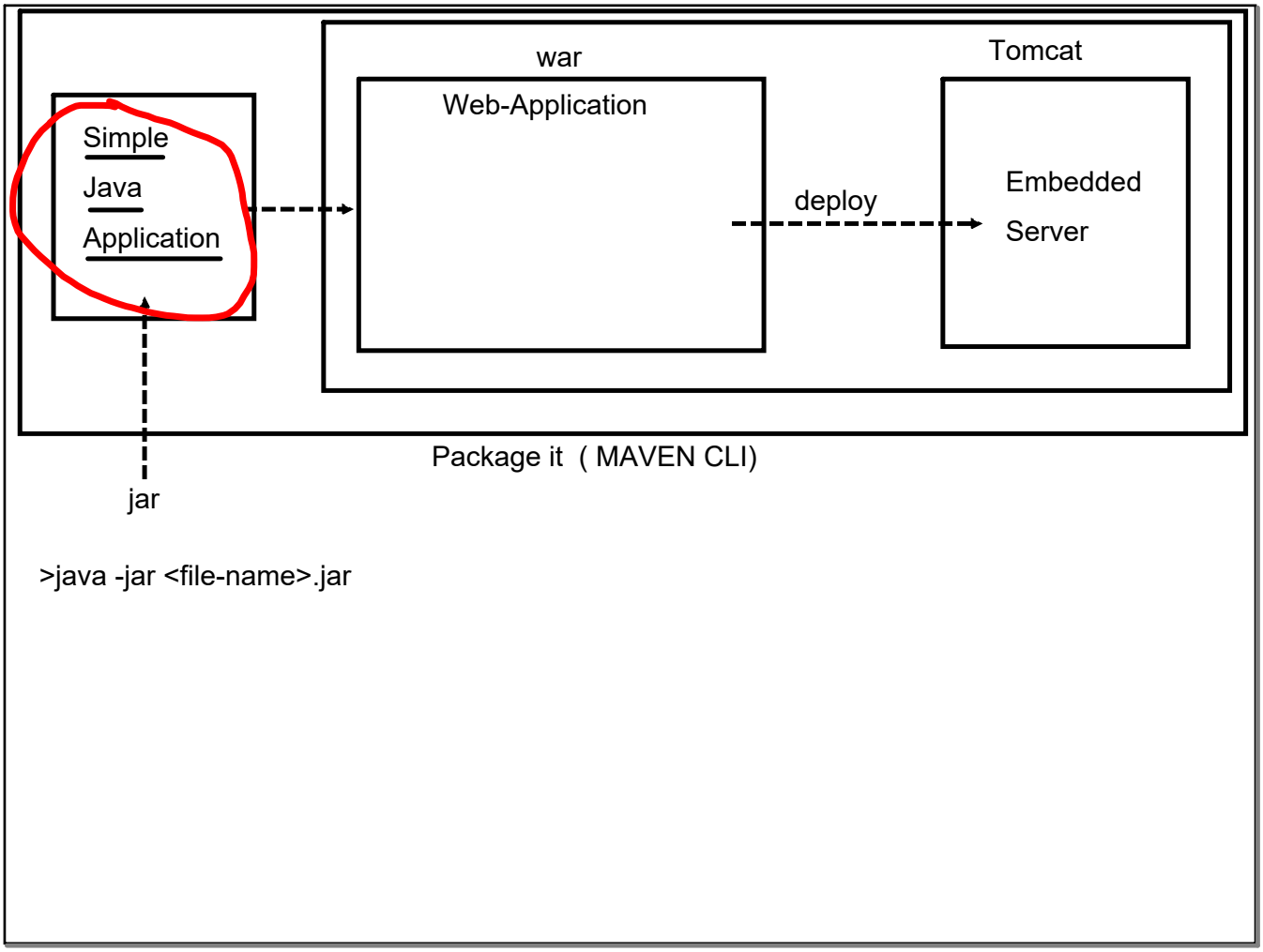
All requirements embedded in project

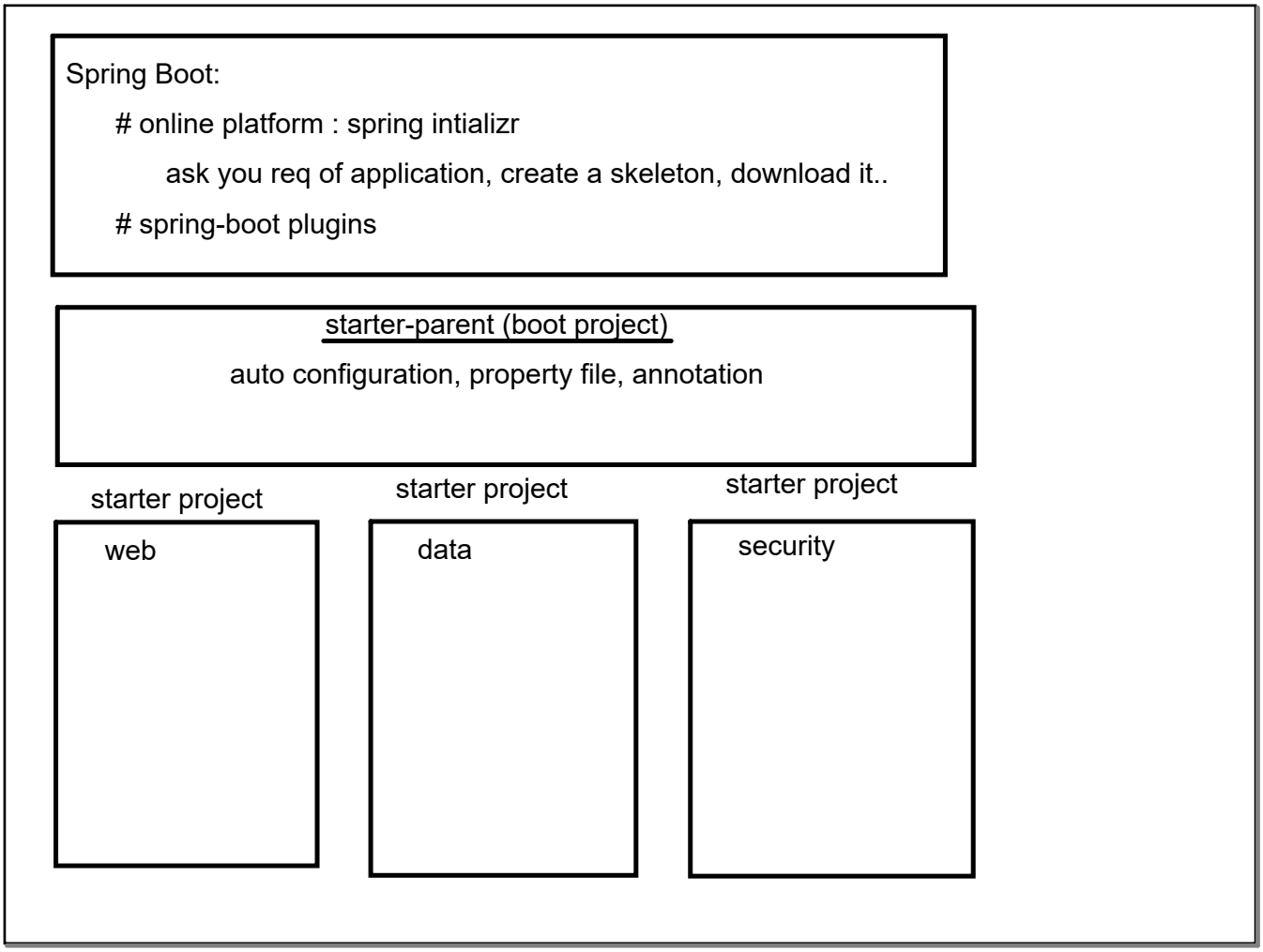
embed the maven cli command file

Text editor (no plugin dependency)

embedded tomcat server

add support of deploying app


war
Web-Application
Tomcat
Simple
Java
Application
deploy
Embedded
Server
Package it (MAVEN CLI)
jar
>java -jar <file-name>.jar


Spring Boot:
online platform : spring intializr
ask you req of application, create a skeleton, download it..
spring-boot plugins
starter-parent (boot project)
auto configuration, property file, annotation
starter project
web
starter project
data
starter project
security

no webapp :

Spring-boot applications by default does not support jsp+jstl view template

By default support of Thymeleaf (just required to add thymeleaf)

static : images etc

template : home folder for view pages (no view resolver needed to configure)

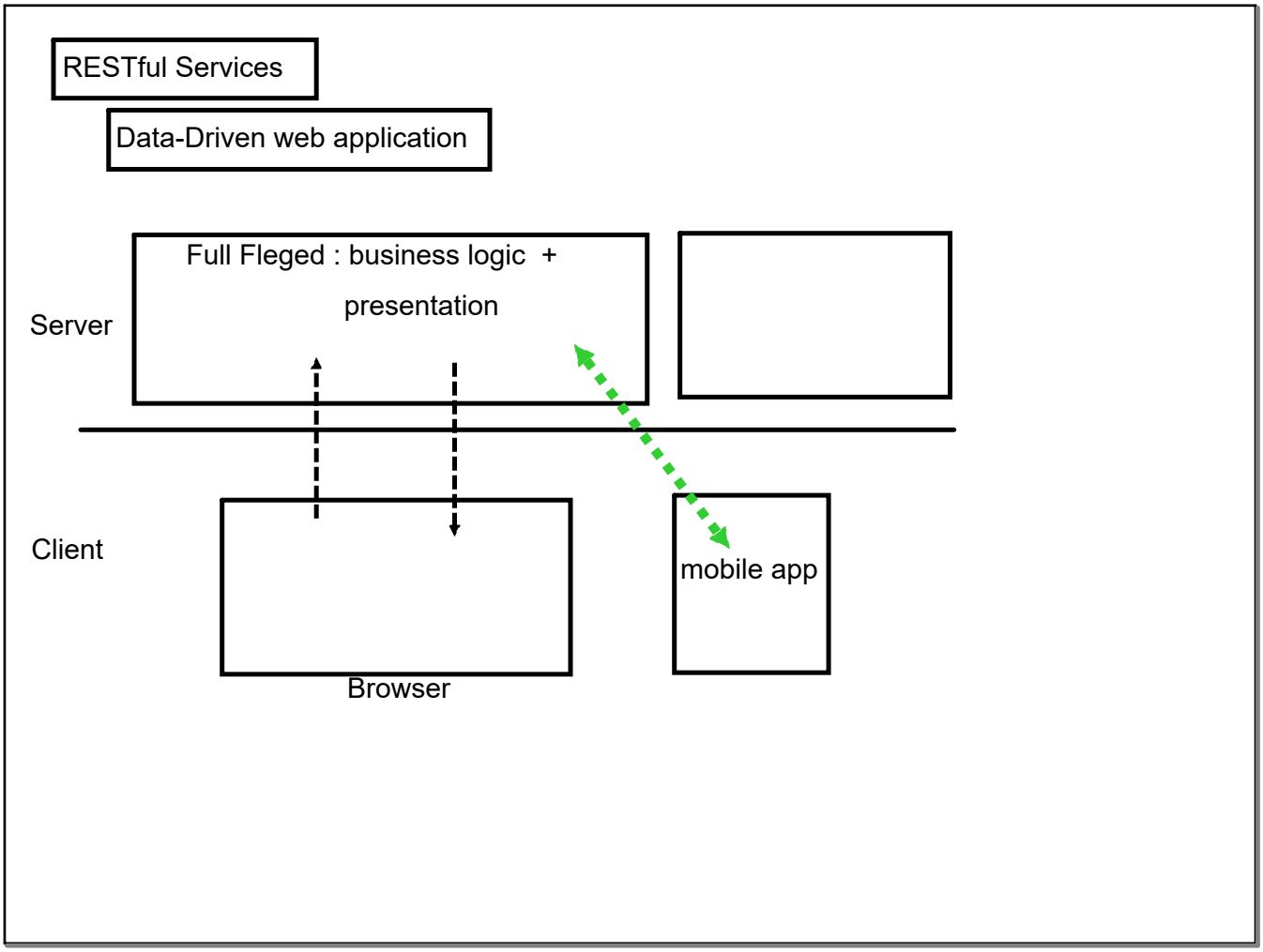
application.properties : file for custom configuration

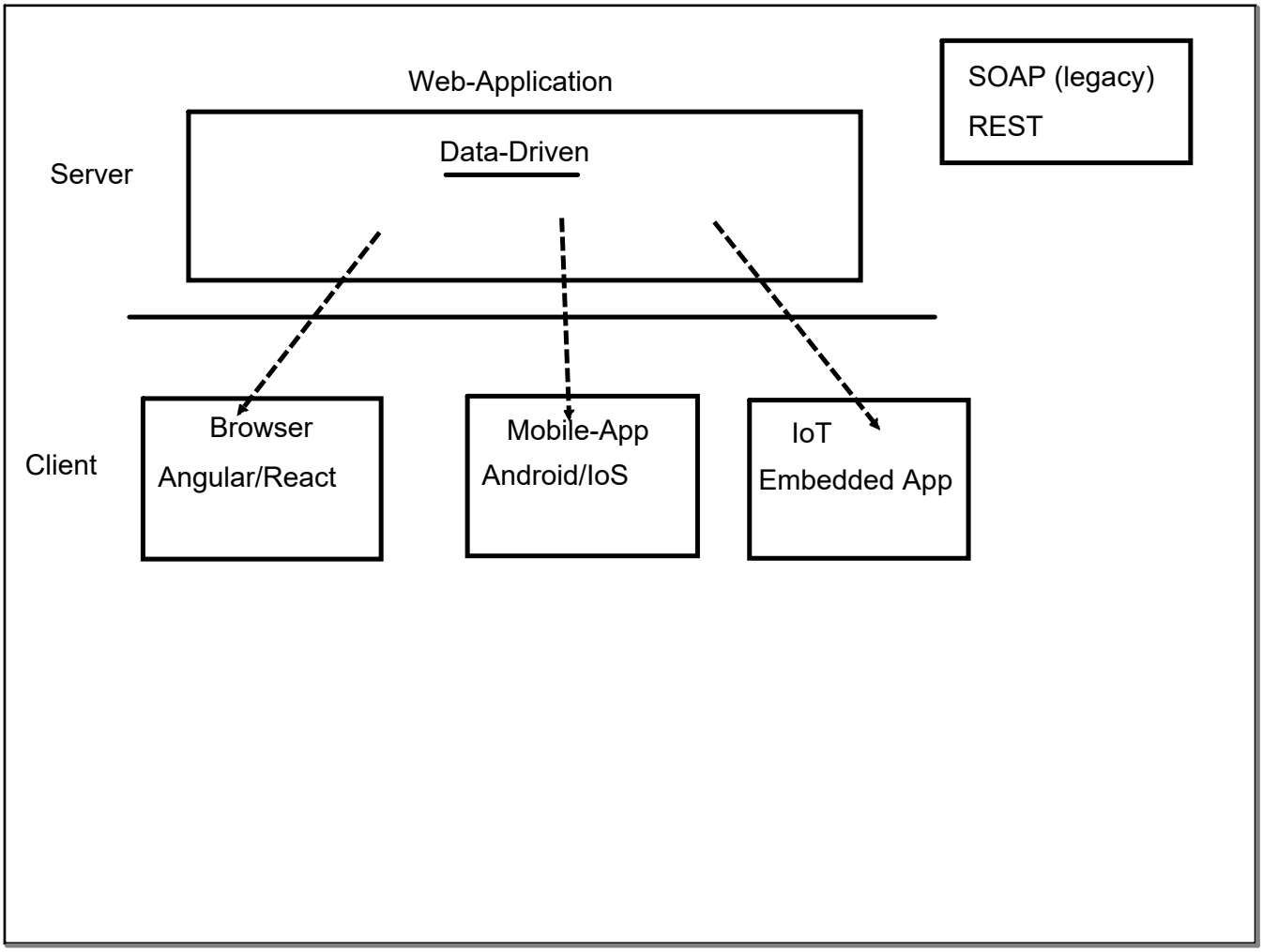
JSP-JSTL

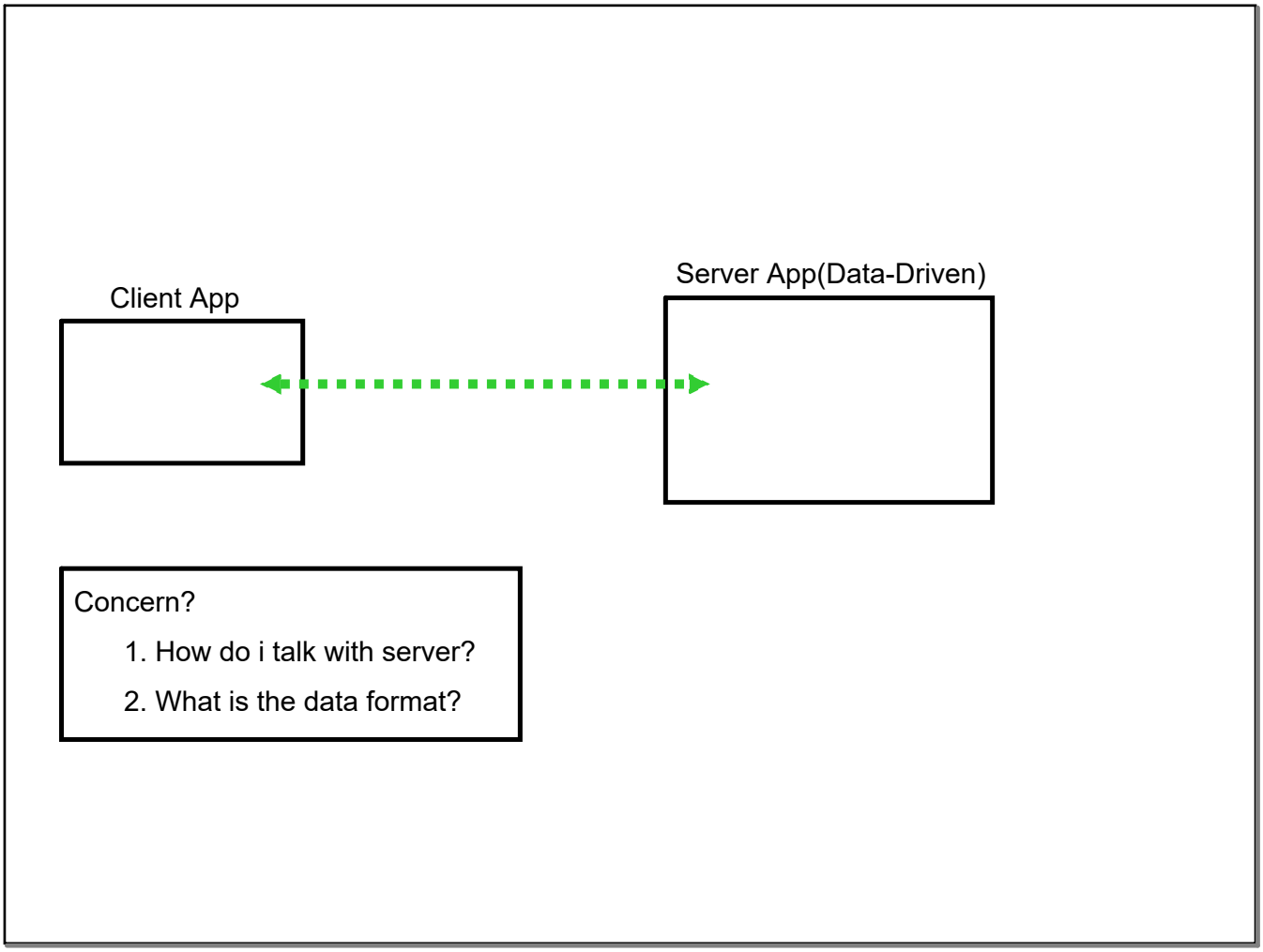
1. Add external dependency
   - jsp compilation
   - jstl
2. add webapp folder structure explicitly
3. add jsp files
4. custom config : View Resolver : properties file

Adding support of security

1. add dependency of starter project for security : security filter would be initialized auto
   (external dependency for security taglibs)
2. Add dependency for starter project for validators
3. security config file needed to be added (custom)
4. Copy the resources


RESTful Services
Data-Driven web application
Full Fleged : business logic +
presentation
Server
Client
mobile app
Browser


Web-Application
SOAP (legacy)
REST
Server
Data-Driven
Client
Browser
Angular/React
Mobile-App
Android/IoS
IoT
Embedded App


Client App
Server App(Data-Driven)
Concern?
1. How do i talk with server?
2. What is the data format?

SOAP :

Functional implementation: exposes API

Client need to download API

Client need to interact using object and method of API

Object Oriented interaction for data

eg:

Server : PHP

Client Java : PHP----->JAVA

Object conversion

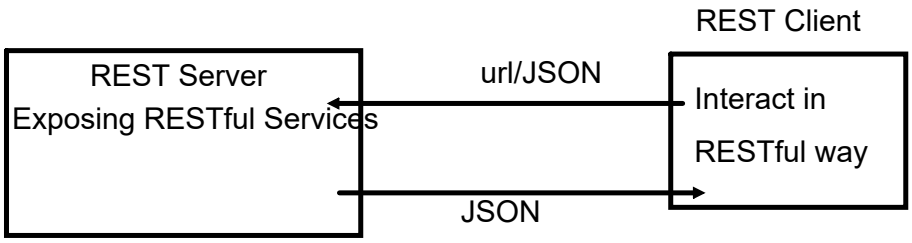
Complex in nature

High level implementation

Performance outcome low

REST:

1. url based interaction + HTTP Verbs
2. text based : JSON(popular), XML, text, html...



Mapping JAVA<---------->JSON : API : jackson-databind project,gson
uses getter - setter method

REpresentation State Transfer : [REST]

1. Stateless : no state maintainance(uniqueness of client is not maintained)
   (rely on client side platform)
2. Inherently not secured
   (rely on backend platform)

RESTful Spring Application

Controller design will change

Use-Case:

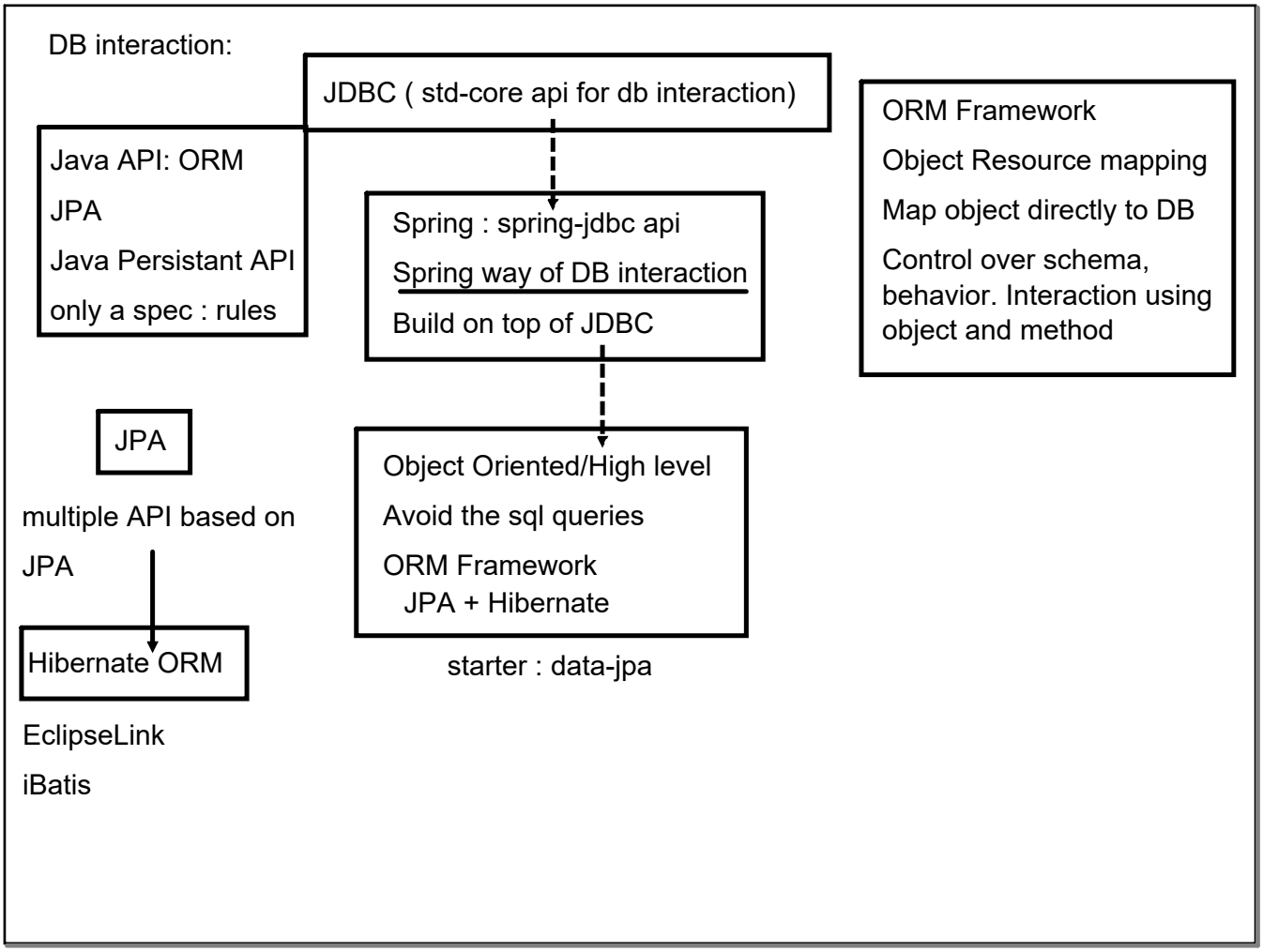
Employee : RESTful application expose all crud functionality

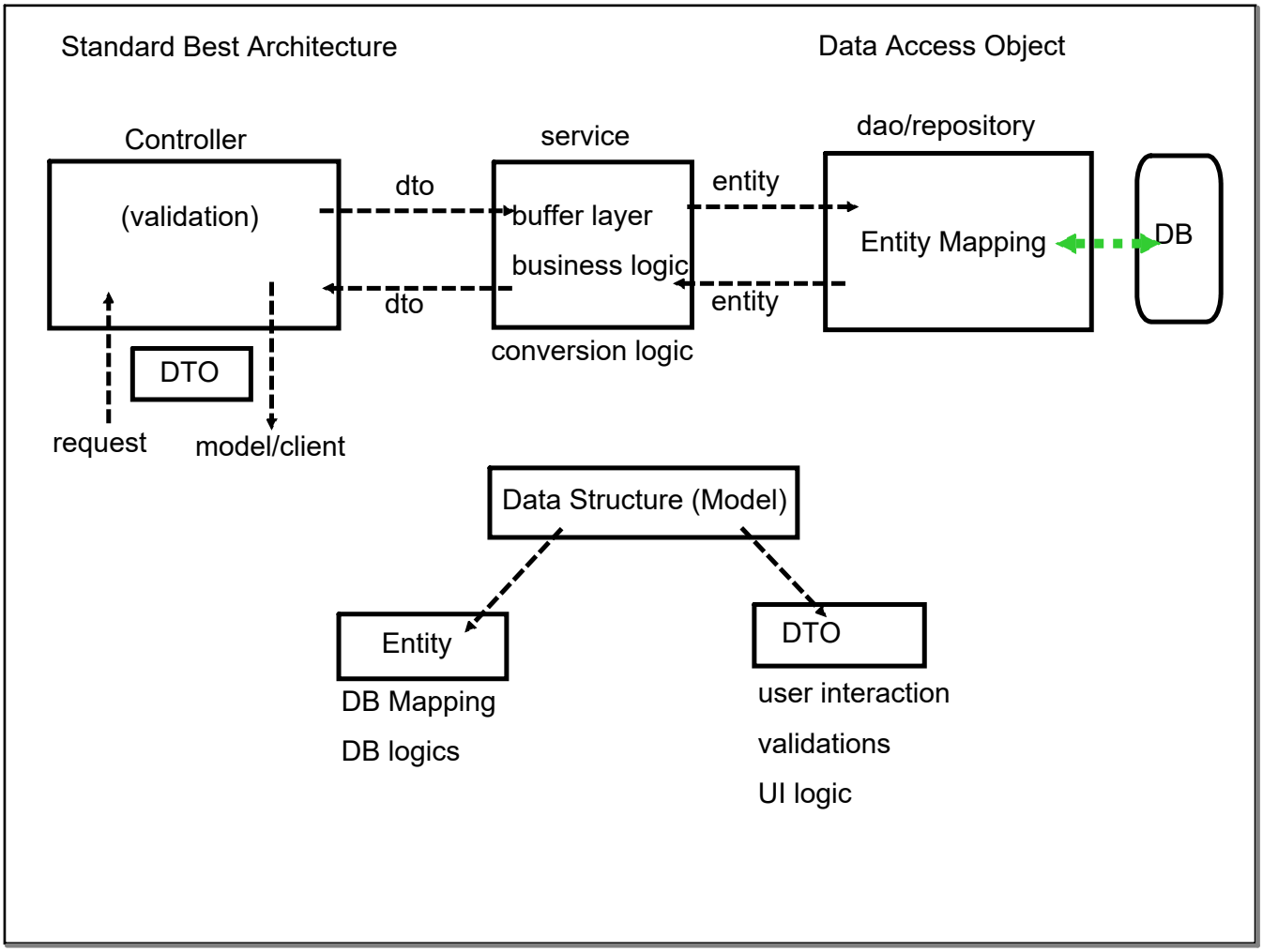
- add new record
- delete a record
- edit a record
- fetch all record
- fetch a particular record

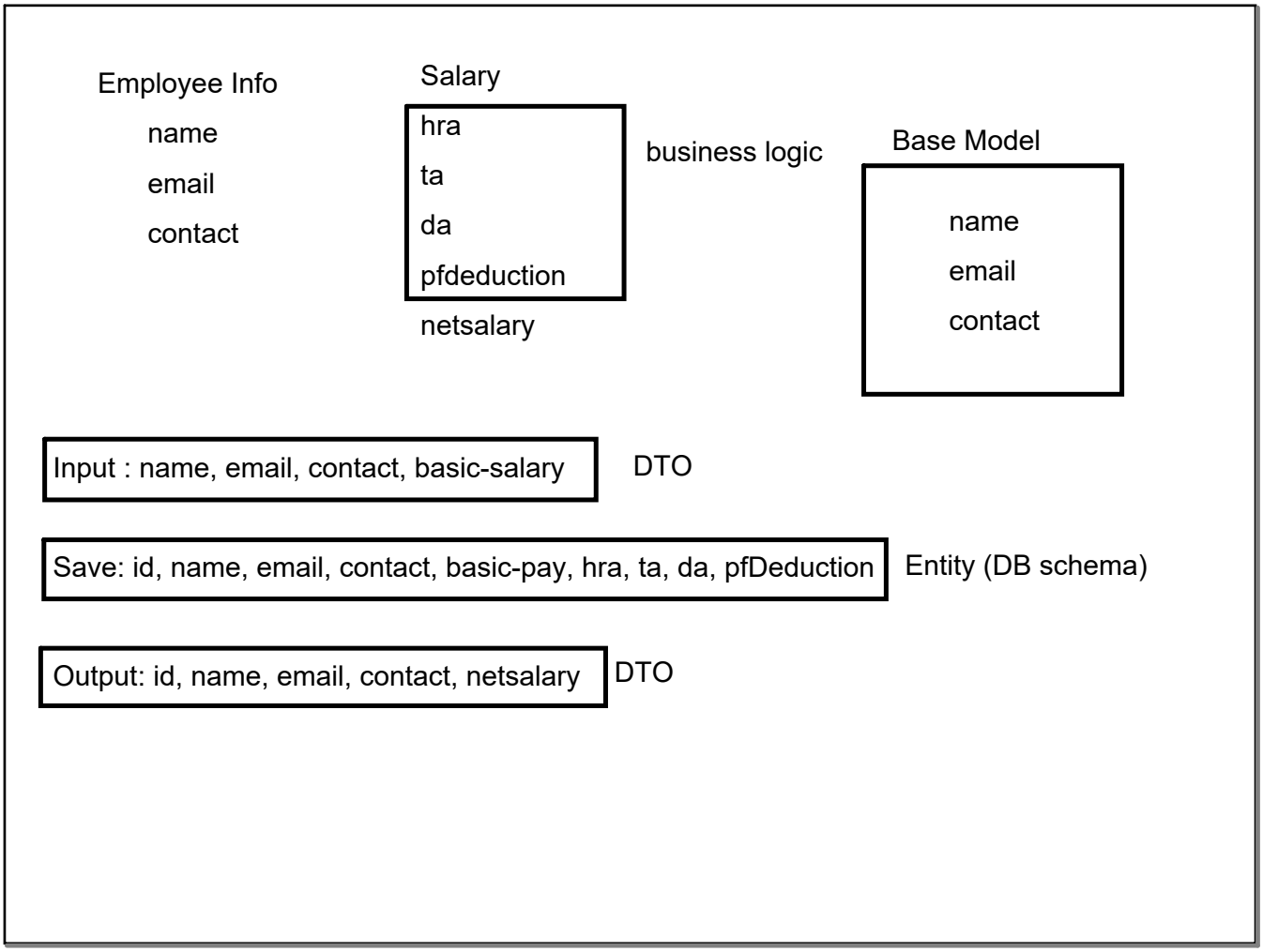
maintain in data base : Data Module of Spring

JAVA

- SPRING
- Java EE
- Struts
- EJB


DB interaction:
JDBC (std-core api for db interaction)
Java API: ORM
JPA
Java Persistant API
only a spec : rules
Spring : spring-jdbc api
Spring way of DB interaction
Build on top of JDBC
ORM Framework
Object Resource mapping
Map object directly to DB
Control over schema, behavior. Interaction using object and method
JPA
multiple API based on JPA
Hibernate ORM
EclipseLink
iBatis
Object Oriented/High level
Avoid the sql queries
ORM Framework
JPA + Hibernate
starter : data-jpa


Standard Best Architecture
Data Access Object
Controller
(validation)
service
buffer layer
business logic
conversion logic
dao/repository
Entity Mapping
DB
dto
dto
entity
entity
DTO
request
model/client
Data Structure (Model)
Entity
DB Mapping
DB logics
DTO
user interaction
validations
UI logic


Employee Info
name
email
contact
Salary
hra
ta
da
pfdeduction
netsalary
business logic
Base Model
name
email
contact
Input : name, email, contact, basic-salary
DTO
Save: id, name, email, contact, basic-pay, hra, ta, da, pfDeduction
Entity (DB schema)
Output: id, name, email, contact, netsalary
DTO

Best Practices for REST Endpoints

/<plural form of entity> : HTTP verbs

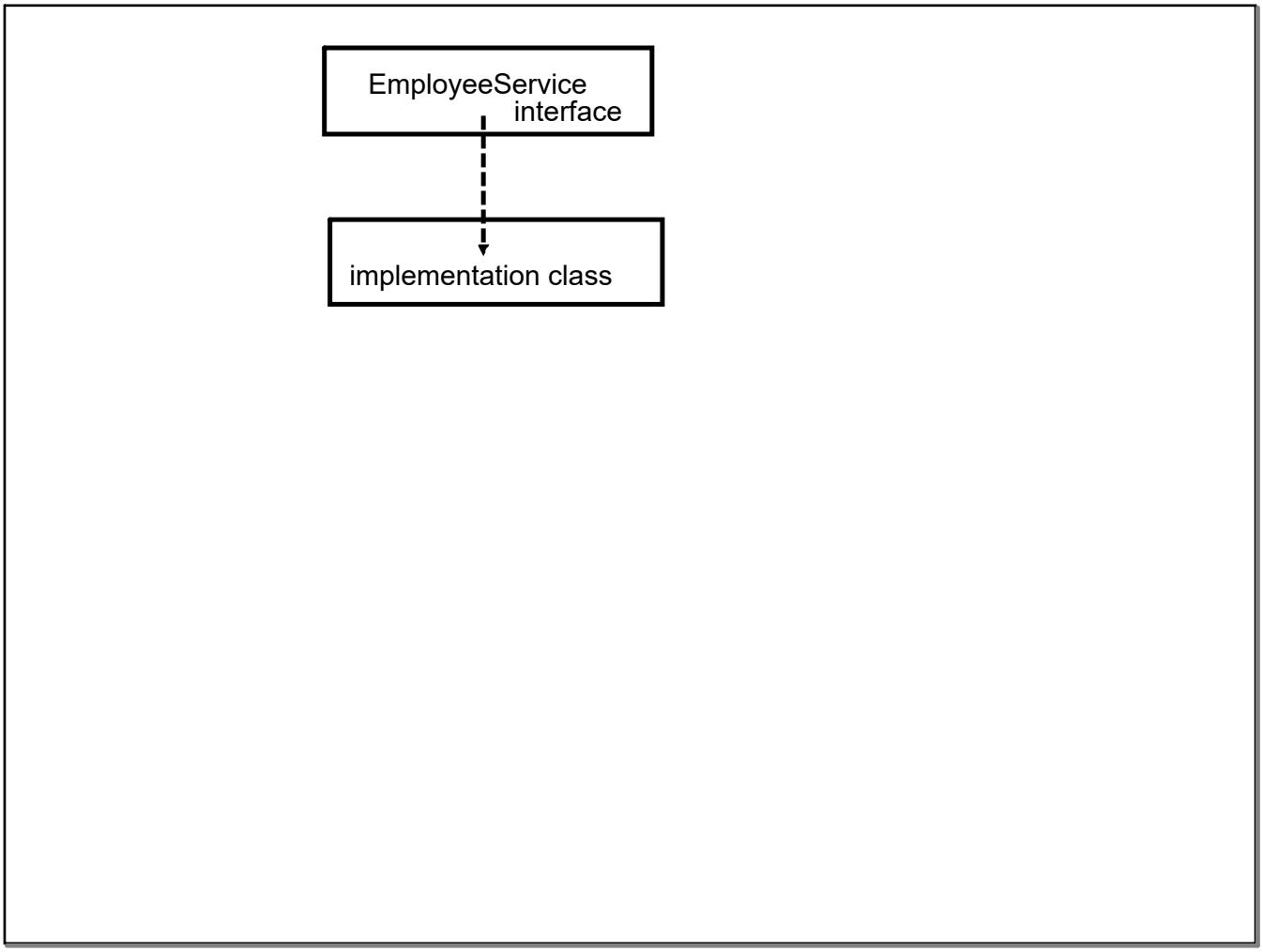
add : (/addEmployee) : /employees : POST

delete: (/deleteEmployee) : /employees/{id} : DELETE

edit : /employees/{id} : PUT

fetch all : /employees : GET

fetch a record based on id : /employees/{id} : GET


EmployeeService
interface
implementation class