Spring Boot

Shihab Rahman
Senior Software Engineer
Infolytx Bangladesh Limited

IoC and DI

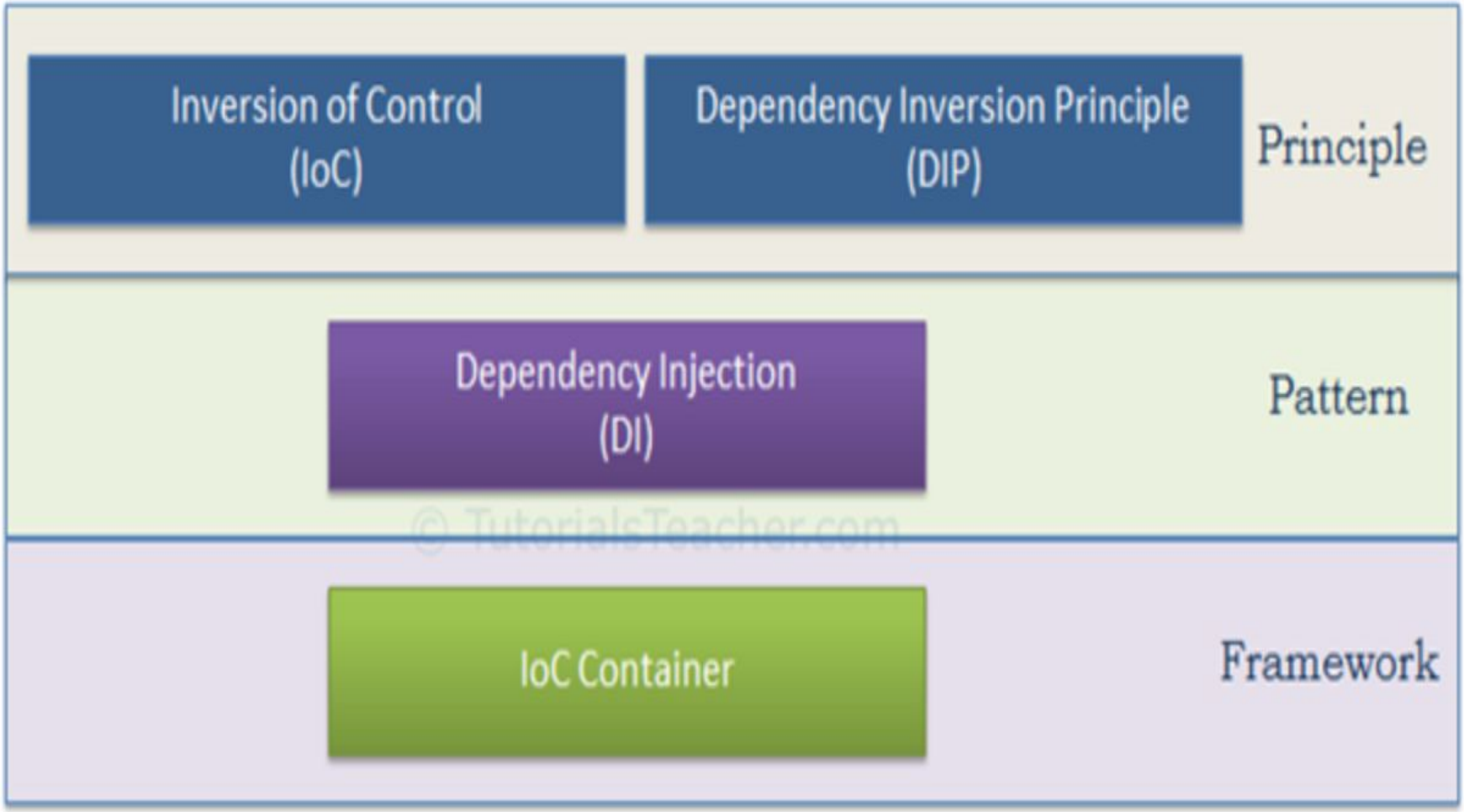
```java
//Hardcoded dependency
public class MyClass {
    private MyDependency myDependency = new MyDependency();
}
//Injected dependency
public class MyClass {
    private MyDependency myDependency;
    public MyClass(MyDependency myDependency){
        this.myDependency = myDependency;
    }
}
```

```java
public class MyClass3 {
    public void doSomething(){}
}

//MyClass2 depends on MyClass3
public class MyClass2 {
    private MyClass3 myClass3;
    public MyClass2(MyClass3 myClass3){
        this.myClass3 = myClass3;
    }
    public void doSomething(){
        myClass3.doSomething();
    }
}

//MyClass1 depends on MyClass2
public class MyClass1 {
    private MyClass2 myClass2;
    public MyClass1(MyClass2 myClass2){
        this.myClass2 = myClass2;
    }
    public void doSomething(){
        myClass2.doSomething();
    }
}

public class Main {
    public static void main(String[] args) {
        //All dependencies need to be managed by the developer
        MyClass3 myClass3 = new MyClass3();
        MyClass2 myClass2 = new MyClass2(myClass3);
        MyClass1 myClass1 = new MyClass1(myClass2);
        myClass1.doSomething();
    }
}
```

```java
public class MyClass2 {
    private MyClass3 myClass3;
    private MyClass4 myClass4;
    public MyClass2(MyClass3 myClass3, MyClass4 myClass4){
        this.myClass3 = myClass3;
        this.myClass4 = myClass4;
    }
    public void doSomething(){
        myClass3.doSomething();
        myClass4.doSomething();
    }
}
public class Main {
    public static void main(String[] args) {
        MyClass4 myClass4 = new MyClass4();
        MyClass3 myClass3 = new MyClass3();
        MyClass2 myClass2 = new MyClass2(myClass3, myClass4);
        MyClass1 myClass1 = new MyClass1(myClass2);
        myClass1.doSomething();
    }
}
```

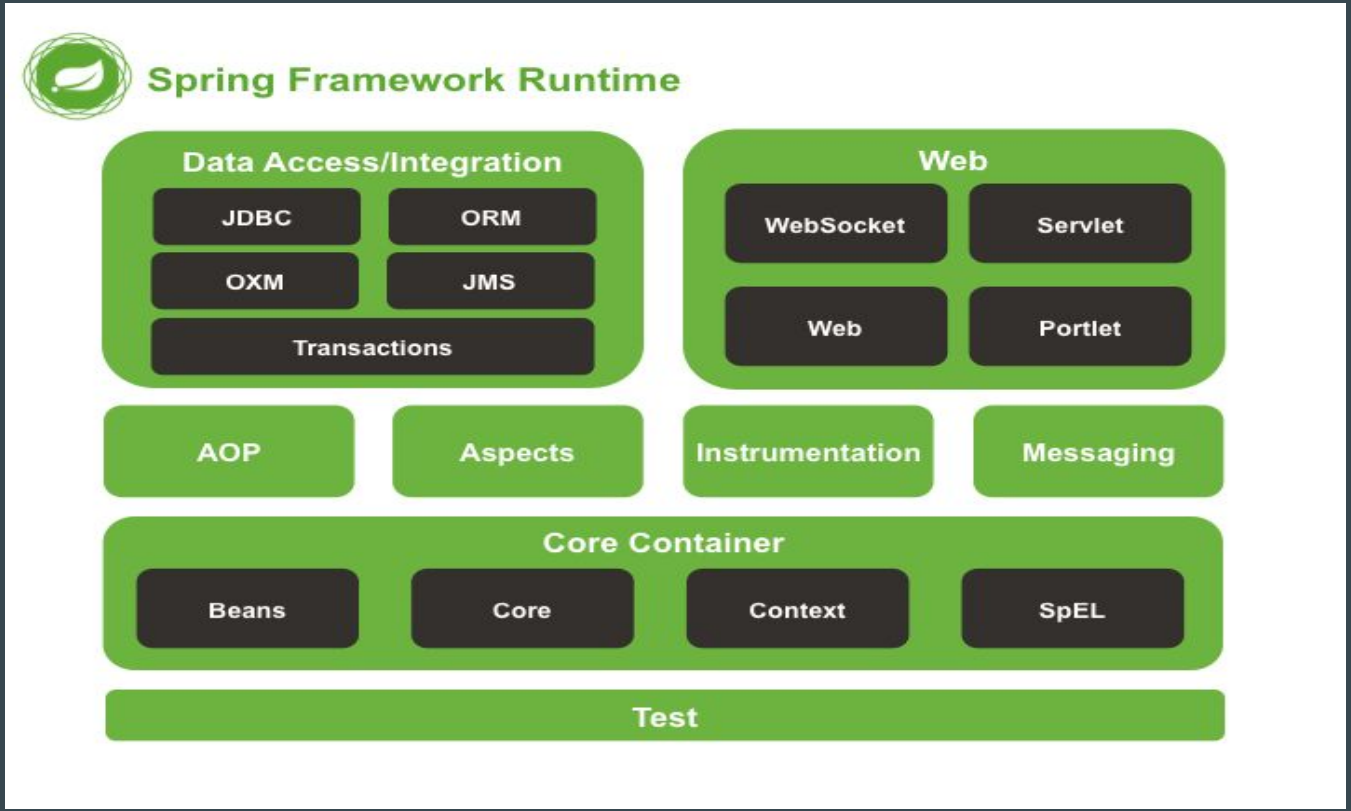
When you have someone else create objects for you and also manages those.

Someone else : IoC container


Inversion of Control
(IoC)
Dependency Inversion Principle
(DIP)
Principle
Dependency Injection
(DI)
Pattern
© TutorialsTeacher.com
IoC Container
Framework

```java
public class MyClass1 {
    @Autowired
    private MyClass2 myClass2;
    public void doSomething(){
        myClass2.doSomething();
    }
}
public class MyClass2 {
    @Autowired
    private MyClass3 myClass3;
    @Autowired
    private MyClass4 myClass4;
    public void doSomething(){
        myClass3.doSomething();
        myClass4.doSomething();
    }
}
```

Spring Architecture



IoC container responsibility

- Creation
- Destruction
- Invoking callbacks.

IOC CONTAINER	MANAGED OBJECT NAME	MANAGED OBJECTS DEFINITION
Spring Container	Bean	Classes defined with annotations/XML configuration
Servlet Container	Servlet	Classes implementing interface Servlet
Actor System	Actor	Classes extending trait Actor

@Autowired

@Autowired on properties

```
@Component("fooFormatter")
public class FooFormatter {

    public String format() {
        return "foo";
    }
}
```

```
@Component
public class FooService {

    @Autowired
    private FooFormatter fooFormatter;

}
```

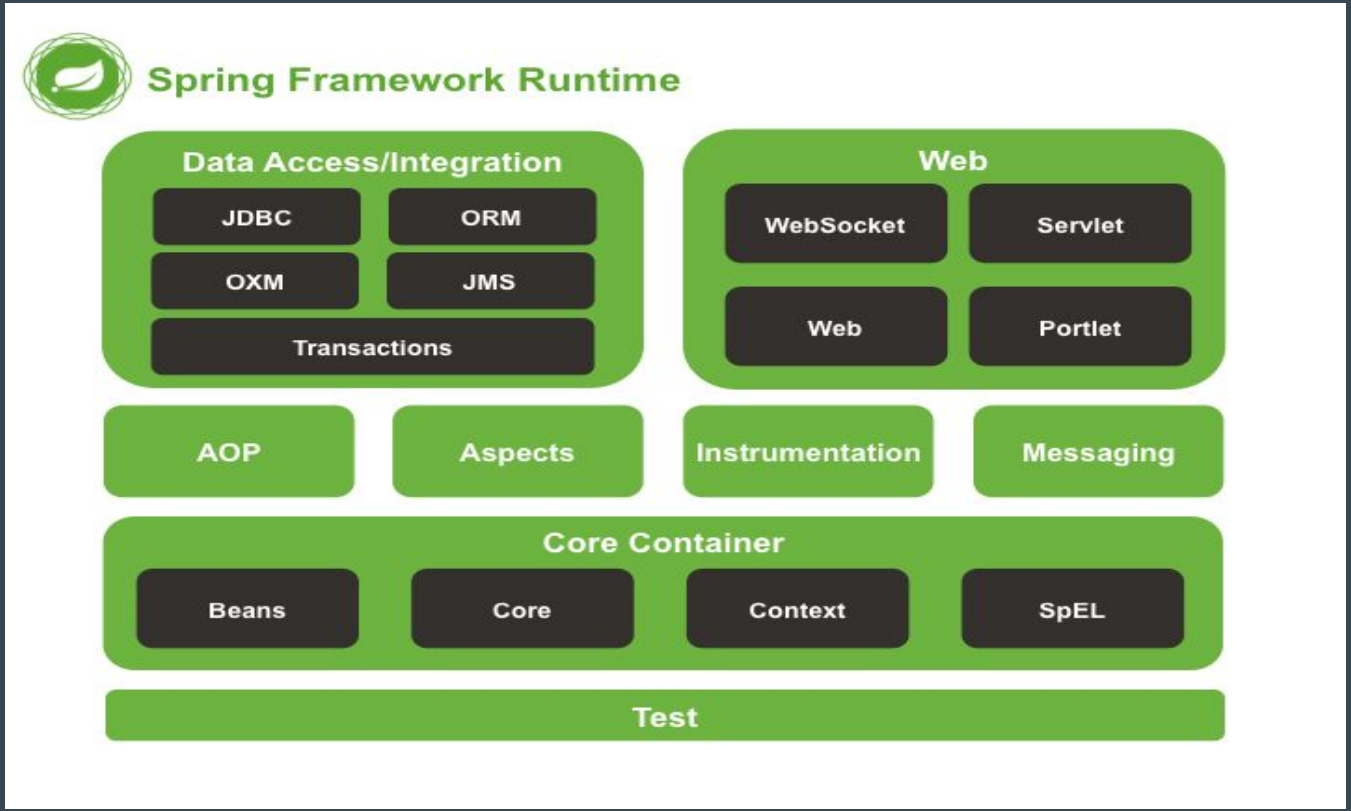
@Autowired on setter

```
1  public class FooService {
2
3      private FooFormatter fooFormatter;
4
5      @Autowired
6      public void setFooFormatter(FooFormatter fooFormatter) {
7              this.fooFormatter = fooFormatter;
8      }
9  }
```

@Autowired on constructor

```
1  public class FooService {
2
3      private FooFormatter fooFormatter;
4
5      @Autowired
6      public FooService(FooFormatter fooFormatter) {
7          this.fooFormatter = fooFormatter;
8      }
9  }
```

Spring Architecture



@Repository

@Repository

- @Repository - indicates decorated class is a repository.
- 3 types of repository interfaces :
  - *CrudRepository*
  - *PagingAndSortingRepository*
  - *JpaRepository*
- Generic Interface - *Repository*
  - *<CrudRepository* extends Repository>
  - *<PagingAndSortingRepository* extends *CrudRepository>*
  - *<JpaRepository extends PagingAndSortingRepository>*

Repository

```
public interface Repository<T, ID> {

}
```

Crud Repository

```java
public interface CrudRepository<T, ID> extends Repository<T, ID> {

    /**...*/
    <S extends T> S save(S entity);

    /**...*/
    <S extends T> Iterable<S> saveAll(Iterable<S> entities);

    /**...*/
    Optional<T> findById(ID id);

    /**...*/
    boolean existsById(ID id);

    /**...*/
    Iterable<T> findAll();

    /**...*/
    Iterable<T> findAllById(Iterable<ID> ids);

    /**...*/
    long count();

    /**...*/
    void deleteById(ID id);

    /**...*/
    void delete(T entity);

    /**...*/
    void deleteAll(Iterable<? extends T> entities);

    /**
     * Deletes all entities managed by the repository.
     */
    void deleteAll();
}
```

Paging and Sorting Repository

```
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {

    /**...*/
    Iterable<T> findAll(Sort sort);

    /**...*/
    Page<T> findAll(Pageable pageable);
}
```

JPA Repository

Modifier and Type	Method and Description
void	**deleteAllInBatch**() Deletes all entites in a batch call.
void	**deleteInBatch**(**Iterable**<**T**> entities) Deletes the given entities in a batch which means it will create a single **Query**.
List<**T**>	**findAll**()
List<**T**>	**findAll**(**Iterable**<**ID**> ids)
List<**T**>	**findAll**(**Sort** sort)
void	**flush**() Flushes all pending changes to the database.
T	**getOne**(**ID** id) Returns a reference to the entity with the given identifier.
<S extends **T**> **List**<S>	**save**(**Iterable**<S> entities)
<S extends **T**> S	**saveAndFlush**(S entity) Saves an entity and flushes changes instantly.

Error handling with @ExceptionHandler & @ControllerAdvice

Let’s build some API

`GET /birds/{birdId}`	Gets information about a bird and throws an exception if not found.
`GET /birds/noexception/{birdId}`	This call also gets information about a bird, except it doesn’t throw an exception in case that the bird is not found.
`POST /birds`	Creates a bird.

Success Result


```
{
 "scientificName": "Common blackbird",
 "specie": "Turdus merula",
 "mass": "aaa",
 "length": 4
}
```

But with Exception thrown.


```
{
 "timestamp": 1500597044204,
 "status": 400,
 "error": "Bad Request",
 "exception": "org.springframework.http.converter.HttpMessageNotReadableException",
 "message": "JSON parse error: Unrecognized token 'three': was expecting ('true', 'false' or 'null'); nest
 "path": "/birds"
}
```

@ExceptionHandler

```
public class FooController{

    //...
    @ExceptionHandler({ CustomException1.class, CustomException2.class })
    public void handleException() {
        //
    }
}
```

@HandlerExceptionResolver

```java
@Component
public class RestResponseStatusExceptionResolver extends AbstractHandlerExceptionResolver {

    @Override
    protected ModelAndView doResolveException
      (HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) {
        try {
            if (ex instanceof IllegalArgumentException) {
                return handleIllegalArgument((IllegalArgumentException) ex, response, handler);
            }
            ...
        } catch (Exception handlerException) {
            logger.warn("Handling of [" + ex.getClass().getName() + "]
              resulted in Exception", handlerException);
        }
        return null;
    }

    private ModelAndView handleIllegalArgument
      (IllegalArgumentException ex, HttpServletResponse response) throws IOException {
        response.sendError(HttpServletResponse.SC_CONFLICT);
        String accept = request.getHeader(HttpHeaders.ACCEPT);
        ...
        return new ModelAndView();
    }
}
```

@ControllerAdvice

```java
import ...

@ControllerAdvice
public class CustomizedResponseEntityExceptionHandler extends ResponseEntityExceptionHandler {

    @ExceptionHandler(Exception.class)
    public final ResponseEntity<ErrorDetailsDTO> handleAllExceptions(Exception ex, WebRequest request) {
        ErrorDetailsDTO errorDetails = new ErrorDetailsDTO(new Date(), ex.getMessage(),
                request.getDescription( b: false));
        return new ResponseEntity<>(errorDetails, HttpStatus.INTERNAL_SERVER_ERROR);
    }

    @ExceptionHandler(EaiResultNotFoundException.class)
    public final ResponseEntity<ErrorDetailsDTO> handleEaiResultNotFoundException(EaiResultNotFoundException ex, WebRequest request) {
        ErrorDetailsDTO errorDetails = new ErrorDetailsDTO(new Date(), ex.getMessage(),
                request.getDescription( b: false));
        return new ResponseEntity<>(errorDetails, HttpStatus.NOT_FOUND);
    }


    @ExceptionHandler(EmptyResultDataAccessException.class)
    public final ResponseEntity<ErrorDetailsDTO> handleDeleteEaiResultException(EmptyResultDataAccessException ex, WebRequest request) {
        ErrorDetailsDTO errorDetails = new ErrorDetailsDTO(new Date(), ex.getMessage(),
                request.getDescription( b: false));
        return new ResponseEntity<>(errorDetails, HttpStatus.NOT_FOUND);
    }


    @ExceptionHandler(CannotCreateTransactionException.class)
    public final ResponseEntity<ErrorDetailsDTO> handleConnectException(CannotCreateTransactionException ex, WebRequest request) {
        ErrorDetailsDTO errorDetails = new ErrorDetailsDTO(new Date(), ex.getMessage(),
                request.getDescription( b: false));
        return new ResponseEntity<>(errorDetails, HttpStatus.INTERNAL_SERVER_ERROR);
    }

}
```

Serverless Spring Boot.

Improving cold start

- ✓ Static Handler.
- ✓ Avoid Component Scan (@ComponentScan)
- ✓ Avoid Constructor Injection By name. (Use @ConstructorProperties)

Thanks