# The Spring Framework: Transforming Java Development

**SHUBHAM MATHUR[1] , Dr. VISHAL SHRIVASTAVA[2], Dr. AKHIL PANDEY[3], Mr. AMIT KUMAR TEWARI[4]**

[1]B.TECH. Scholar,[2,3]Professor, [4]Assistant Professor
Computer Science & Engineering
Arya College of Engineering & I.T. India, Jaipur

[1]mathur27shubham@gmail.com , [2]vishalshrivastava.cs@aryacollege.in, [3]akhil@aryacollege.in, [4]amittewari.cs@aryacollege.in

***Abstract***- The Spring Framework has brought a new face to Java development by providing a unified programming and configuration model for contemporary Java applications. From the very beginning, Spring has focused on the fact that enterprise Java is inherently complex and offered the simple, flexible modular framework across the most diverse applications, from prototype web applications to large-scale enterprise applications and even micro services architectures. This paper looks at the development history of the Spring Framework, its structure, main components, and advantages. It also demonstrates the continual influence of the framework on today's progressive development in the Java environment, specifically through the creation of Spring Boot and Spring Cloud projects that help simplify the development of highly dependable micro services. We describe what issues are relevant to Spring and what may be expected from it in the context of the constant evolution of the software development process.

***Index Terms- Java, Spring, Spring Boot, application development, micro services, dependency injection, inversion of control, Spring MVC, Spring Data, Spring Cloud***

## I. INTRODUCTION

Enterprise-level has been implemented through Java since the mid-1990s when Java became a popular language among developers. But then Java developer begin to face some Enterprise Application development issues such as; excessive amount of boiler plate codes, complicated configuration management and integration with other framework. The Spring Framework initiated in 2003 by Rod Johnson was intended to solve these problems making the infrastructure more flexible, narrow in weight, and more familiar to the developer.

Spring was first to introduce the idea of Dependency Injection (DI) alongside Inversion of Control (IoC) to make application designs simple as compared to complex to develop, maintain and enhance. In recent years, Spring has integrated several projects, for example, Spring MVC for web applications, Spring Data for working with databases, Spring Boot – for creating standalone applications, and Spring Cloud as an application suite for developing micro service architectures.
This paper focuses on the Spring Framework and Java development discussing its architecture, fundamental properties, advantages, and use in micro services architecture. It also contains an understanding of developers' difficulties and solutions proposed by the Spring Framework.

## II. EVOLUTION OF JAVA DEVELOPMENT

Java development has from its inception undergone the same cycle of evolution as most of the modern languages.

In the time prior to the advent of the Spring Framework, Java developers had Enterprise JavaBeans (EJB) at their disposal for constructing business applications. EJB was a part of the Java EE that tried to ensure and facilitate distribution, transactions, security and persistence in distributed applications. However, EJB had numerous problems that affected the developers' productivity and applications maintainability

### A. Challenges with Enterprise JavaBeans (EJB):

1. **Complexity**: EJB involves a multitude of configuration and deployment steps that made the overall process complicated.

2. **Boilerplate Code**: One of the problems that were associated with creation of EJB applications was the presence of large amount of code that was reused in various forms but nevertheless required deeper writing, and thus augmented the overall development time.

3. **Tight Coupling**: EJB components had very strong dependencies amongst them; each individual component could not be tested in isolation.

4. **Performance Overhead**: WebLogic's optimization of IIOP created performance bottlenecks attributable to EJB's heavyweight nature especially in applications that required extremely concurrency

**B. The Need for a New Framework:**

Here it is important to mention that the problems with EJB exposed the need for a more lightweight and flexible framework. Reimagine how configuration is done, as well as making deployment much easier. The design should encourage loose coupling so that the system can be easily tested and modified. Enhance performance as well as capability of handling large volume of traffic challenges with EJB highlighted the need for a more flexible and lightweight framework. Developers sought a solution that would:

- Simplify configuration and development.
- Reduce boilerplate code.
- Promote loose coupling for easier testing and maintained.
- Improve performance and scalability.

**C. The Emergence of Spring:**

In his book, Expert One-on-One J2EE Development without EJB, written in 2002, Rod Johnson expanded on the idea of POJOs as a solution to Java enterprise application development. Another important development emerged in 2002, following this approach paved way to a new framework named Spring Framework that was released in 2003. Spring provided an alternative to EJB by offering:

- **Lightweight Containers**: The Spring framework's IoC container handled Java object creation and instantiation and their setting up as well.
- **Dependency Injection**: Made dependencies easier to wire, so that code was less cluttered and easier to work on.
- **Aspect-Oriented Programming (AOP)**: Allowed for a level of composition of utility concerns such as logging, security, etc.

**D. Major Development in Evolution of Spring:**

1. **Spring 1.0 (2004):** They also put into use basic ideas such as IoC and DI.
2. **Spring 2.5 (2007)**: Introduced plugin annotation-based configuration and extend other plugin configuration source to via annotation, which decreased the use of XML configuration.
3. **Spring 3.0 (2009):** They added support for Restful web services.
4. **Spring 4.0 (2013):** Increase in compatibility with Java 8 and improvement of features in building an application for an enterprise.
5. **Spring Boot (2014):** Here, it is easy to install application interfaces and launch them with automatically configured and integrated server systems.
6. **Spring Cloud (2015):** Gave methods for designing Disperse systems and Micro services techniques.

## III. KEY FEATURES OF SPRING FRAMEWORK

This paper aims at identifying the key features of the Spring Framework which include the following:

Java developers will find that Spring has a number of facilities designed to ease application development and enhance the underlying architecture for enterprise applications.

**A. Dependency Injection (DI):**

In its basic form, Dependency Injection (DI) is a design pattern that focuses on the ability of the developer defining the dependencies of an object instead of providing them within the object. This leads to the principles of loose coupling, which also make the tests more effective.

**1. Types of Dependency Injection (DI)**

- **Constructor Injection**: Dependencies are injected through the class constructor
- **Setter Injection**: Dependencies are injected via setter methods.
- **Field Injection**: Dependencies are injected directly into class fields using annotations like `@Autowired`.

**2. Benefits of Dependency Injection (DI)**

- **Reduced Boilerplate Code:** It helps to minimize the complexity of creating and dealing with dependence relationships
- **Increased Testability:** For the unit test, dependencies can be easily mocked or replace.
- **Flexibility:** Enables making configuration changes without the need to rewrite new code.

**B. Inversion of Control (IoC):**

Inversion of Control (IoC) is a principle whereby the framework rather controls the application's flow than the developer. When in Spring, the IoC container is concerned with how beans (Java objects) are created and how they live out their lives.

**1. IoC Containers in Spring**

- **BeanFactory:** A starting point IoC container that gives rather minimalistic DI functionality.
- **Application Context:** A sophisticated IoC container that embeds BeanFactory and has added functionality for event management and i18n features as well as integrated support for AOP.

**2. Configuration Methods**

- **XML Configuration:** This is a way of doing things in the traditional manner with the help of xml files where we define beans and their dependencies.
- **Java-Based Configuration:** Determining beans and using @Configuration classes and @Bean methods.
- **Annotation-Based Configuration:** Annotations like @Component, @Autowired @Service to declare and instantiate the beans.

**C. Aspect-Oriented Programming (AOP):**

This paradigm helps to modularize the cross cutting concerns like logging, security, and transaction management and many more. In spring framework, the aspects are developed by using @Aspect annotation and by including AspectJ library in Spring application.

**1. Core Concept of AOP**

- **Aspect:** A module that contains code that performs cross cutting concerns.
- **Advice:** That which was done by an aspect (before, at, after).
- **Pointcut:** An expression that corresponds with join points (points in code where advice can be given).
- **Join Point**: An instance of program control, usually at the level of a method call.
- **Weaving:** The hours in which the different aspects are applied to target code.

**2. Types of Advice**

- **Before Advice:** Called before a method call.
- **After Returning Advice**: Performs when a method does returns normally or succeeds in its operations.
- **After Throwing Advice**: Performs if a method has an exception.
- **Around Advice:** Used around a method call and provides functionality of defining behaviour before and after the method execution.

## IV. SPRING COMPONENTS AND PROJECTS

Spring Frameworks has a number of projects under its hood, which helps bring about the easy flow for integration of different technologies in a spring application, like integration cloud, securities, database and others.

### A. Spring MVC

Spring MVC is the MVC architecture framework for the development of web applications. It means that application-building elements such as application logic, user interface, and data management are isolated into different layers. Essentials of Spring MVC are:

- **Controllers:** Execute HTTP requests and give back answers. @Controller or @RestController annotations are used in controllers.
- **Views:** Refer to the view and can be created with different web-application Templating Engines such as JSP, Thymeleaf, FreeMarker and many more.
- **Model**: Holds the application's data and can be supplied to views for display.
- **Request Mapping:** @RequestMapping – Is a stereotype that maps HTTP request to handler methods in controllers.
- **Form Handling:** There are some enhancements of form handling in Spring MVC use @Model Attribute and Binding Result
- **Exception Handling:** Supports @ExceptionHandler for convenient of exception handling.

### B. Spring Data

Database access is made easier by Spring Data by offering the data repository abstraction that supports both relational and non-relational database systems. It eliminates redundancy and makes data access operation more efficient and easy to read compared to when using the database connection. Key Spring Data Projects:

- **Spring Data JPA:** Use of interfaces and annotations to enhance the simplified data access on JPA. It enables developers to open, view and manipulate, edit and delete records with little or largely no programming.
- **Spring Data MongoDB:** Offers base objects and context to work with MongoDB out of the box to interact with the NoSQL data store.
- **Spring Data Redis:** Provides ambiguous coverage for Redis based data access and caching which is helpful in applications that need fast, in-memory data storage.
- **Spring Data JDBC:** Simpler and more direct than JPA that delivers only JDBC mainly for data access.

### C. Spring Boot

The Spring Boot framework aims at making the development and configuration of Spring applications fast and easy as well as adding features of server availability. It reduces the amount of default material that needs to be written as well as avoiding having to set up environment variables to build for a production environment and so is very useful when it comes to developing apps suited for a production environment in a short span of time. Key features of Spring Boot:

- **Auto-Configuration:** Allows automatic wiring of Spring components based on the classpath and properties of the application hence minimizing wiring.
- **Embedded Servers:** Contains embedded Tomcat, Jetty and Undertow servers to avoid using external applications servers during build and runtime processes.
- **Starter Dependencies:** It delivers predefined dependency sets to ease the problem of managing dependencies. These starters group related dependencies in order to ensure compatibility.
- **Spring Boot Actuator:** Offers application health monitoring, application health checks, and application metrics in production.

### D. Spring Cloud

Spring Cloud is a framework for developing applications in a micro service architecture based on distributed computing. Key features of Spring Cloud:

- **Spring Cloud Config:** Allows for centralized administration of configuration for large multi node infrastructures. It provides the possibility to make configuration stored within a single source and dynamically modify it for the services.

- **Spring Cloud Netflix:** Compatible with most-used Netflix OSS libraries: Eureka for service discovery, Ribbon for client-side load balance, Hystrix for handling failure with circuit breaker and, finally, Zuul for API gateway and routing.
- **Spring Cloud Gateway:** An API gateway for routing requests and for transforming request and response messages. It offers real time routing, traffic dispersion and security facilities.
- **Spring Cloud Sleuth**: It offers distributed tracing for the tracking of interactions of the micro services. It works with tools such as Zipkin and Jaeger for visualization
- **Spring Cloud OpenFeign:** A simple declarative REST client for micro services and their interactions. It provides a more simple way to write HTTP client code as it uses an annotation based interface.

**E. Spring Security**

Spring Security is a framework that offers wide coverage of security for Java applications. This includes authentication, authorization and prevention of generic security risks. Key features of Spring Security are:

- **Authentication:** It supports different kinds of authentication kinds of authentication, namely, user name/password, OAuth, LDAP and SSO.
- **Authorization:** Demonstrates both role-oriented and method-based access control.
- **CSRF Protection**: Preserves the Cross-Site Request Forgery attacks by creating tokens and checking out for the valid token.

## V. FUTURE PATTERN AND CHALLENGES

The Spring Framework and its subprojects are in the process of transitioning to new patterns formed by the cloud and PaaS, serverless computing, and containerization. As these innovations serve development, issues such as scalability, security, and maintainability remain an issue of concern.

**A. New Trends in Spring Surroundings:**

- **Micro services**: Spring Cloud supports Micro services; however, micro services require a sophisticated pattern like API Gateway and Event Driven Architecture for managing the logical layered architecture regarding service discovery and load balancing.
- **Serverless Computing**: Mircoservice architecture with support of Spring Cloud Function allows developers working with code only and integrating with such platforms like AWS Lambda regarding such services as cost-optimization.
- **Reactive Programming:** Spring WebFlux enables the development of efficient, asynchronous applications which are crucial in utilizing high numbers of concurrent connectivity and real-time web.

**B. Future Of Spring Framework**

Spring Framework provides definite projections about how the framework will evolve in the future.

- **AI and ML Integration:** The capability will mature to incorporate AI and ML services into applications in spring.
- **Cloud-Native Enhancements:** The planned work for Spring in improve the integration mechanism for Kubernetes and serverless to enhance the development mode in native cloud.
- **Developer Experience:** Improved tooling and eased configuration will place an emphasis on the productivity of the developers.
- **Ecosystem Expansion:** The main elements of the Spring ecosystem will remain relevant and develop, further extending development to new technologies and architectures suitable for modern, highly scalable applications

## V. CONCLUSION

Now with features such as Dependency Injection (DI) and Aspect-Oriented Programming (AOP), The Spring Framework has taken a leading role of reducing complexity of enterprise applications in Java development. With Spring Boot and Spring Cloud it has made application setup, deployment and micro services easier to develop. New problems, such as dealing with complexity and achieving performance, still exist, but the sophisticated environment of Spring's ecosystem and beloved by developer's community meet them halfway. Other features such as the capabilities of the continued trend like serverless computing along with reactive programming and AI make Spring the essential tool of constructing scalable, efficient, and high quality applications which will make it relevant in modern software environments.

REFERENCES

[1] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion. "Ix: A protected dataplane operating system for high throughput and low latency." In *OSDI '14*, pages 49–65, 2014.

[2] Q. Cai, H. Zhang, G. Chen, B. C. Ooi, and K.-L. Tan. "Memepic: Towards a database system architecture without system calls." Technical report, NUS, 2015.

[3] C. Cascaval, C. Blundell, M. Michael, H. W. Cain, P. Wu, S. Chiras, and S. Chatterjee. "Software transactional memory: Why is it only a research toy?" *Queue*, 6(5):40–46, Sept. 2008.

[4] Chelsio. "RoCE at a crossroads." Technical report, Chelsio Communications Inc., 2014.

[5] S. Damaraju, V. George, S. Jahagirdar, T. Khondker, R. Milstrey, S. Sarkar, S. Siers, I. Stolero, and A. Subbiah. "A 22nm IA multi-CPU and GPU system-on-chip." In *ISSCC '12*, pages 56–57, 2012.

[6] J. DeBrabant, A. Joy, A. Pavlo, M. Stonebraker, S. Zdonik, and S. R. Dulloor. "A prolegomenon on OLTP database systems for non-volatile memory." In *ADMS '14*, pages 57–63, 2014.

[7] Z. Feng, E. Lo, B. Kao, and W. Xu. "Byteslice: Pushing the envelope of main memory data processing with a new storage layout." In *SIGMOD '15*, 2015.

[8] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker. "OLTP through the looking glass, and what we found there." In *SIGMOD '08*, pages 981–992, 2008.

[9] S. Jha, B. He, M. Lu, X. Cheng, and H. P. Huynh. "Improving main memory hash joins on Intel Xeon Phi processors: An experimental approach." In *PVLDB '15*, pages 642–653, 2015.

[10] E. P. C. Jones, D. J. Abadi, and S. Madden. "Low overhead concurrency control for partitioned main memory databases." In *SIGMOD '10*, pages 603–614, 2010.

[11] A. Kalia, M. Kaminsky, and D. G. Andersen. "Using RDMA efficiently for key-value services." In *SIGCOMM '14*, pages 295–306, 2014.

[12] A. Kemper and T. Neumann. "Hyper: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots." In *ICDE '11*, pages 195–206, 2011.

[13] P.-A. Larson, S. Blanas, C. Diaconu, C. Freedman, J. M. Patel, and M. Zwilling. "High-performance concurrency control mechanisms for main-memory databases." In *PVLDB '11*, pages 298–309, 2011.

[14] J. Lee, Y. S. Kwon, F. Farber, M. Muehle, C. Lee, C. Bensberg, J. Y. Lee, A. H. Lee, and W. Lehner. "SAP HANA distributed in-memory database system: Transaction, session, and metadata management." In *ICDE '13*, pages 1165–1173, 2013.

[15] S. Lee, M. Kim, G. Do, S. Kim, H. Lee, J. Sim, N. Park, S. Hong, Y. Jeon, K. Choi, et al. "Programming disturbance and cell scaling in phase-change memory: For up to 16nm based 4F2 cell." In *VLSIT '10*, pages 199–200, 2010.

[16] V. Leis, A. Kemper, and T. Neumann. "Exploiting hardware transactional memory in main-memory databases." In *ICDE '14*, pages 580–591, 2014.

[17] F. Li, B. C. Ooi, M. T. Ozsu, and S. Wu. "Distributed data management using MapReduce." *ACM Computing Surveys*, 46(3):31:1–31:42, Jan. 2014.

[18] H. Li, X. Wang, Z.-L. Ong, W.-F. Wong, Y. Zhang, P. Wang, and Y. Chen. "Performance, power, and reliability trade-offs of STT-RAM cell subject to architecture-level requirement." *IEEE Transactions on Magnetics*, 47(10):2356–2359, Oct. 2011.

[19] D. Loghin, B. M. Tudor, H. Zhang, B. C. Ooi, and Y. M. Teo. "A performance study of big data on small nodes." In *PVLDB '15*, 2015.

[20] L. M. Maas, T. Kissinger, D. Habich, and W. Lehner. "Buzzard: A NUMA-aware in-memory indexing system." In *SIGMOD '13*, pages 1285–1286, 2013.

[21] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker. "Rethinking main memory OLTP recovery." In *ICDE '14*, pages 604–615, 2014.

[22] C. Mitchell, Y. Geng, and J. Li. "Using one-sided RDMA reads to build a fast, CPU-efficient key-value store." In *USENIX ATC '13*, pages 103–114, 2013.

[23] T. Neumann, T. Mühlbauer, and A. Kemper. "Fast serializable multi-version concurrency control for main-memory database systems." In *SIGMOD '15*, 2015.

[24] A. Pavlo, C. Curino, and S. Zdonik. "Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems." In *SIGMOD '12*, pages 61–72, 2012.

[25] K. Ren, A. Thomson, and D. J. Abadi. "Lightweight locking for main memory database systems." In *PVLDB '13*, pages 145–156, 2013.

[26] L. Rizzo. "Netmap: A novel framework for fast packet I/O." In *USENIX ATC '12*, pages 101–112, 2012.

[27] S. M. Rumble, A. Kejriwal, and J. Ousterhout. "Log-structured memory for DRAM-based storage." In *FAST '14*, pages 1–16, 2014.

[28] S. Sanfilippo and P. Noordhuis. "Redis." http://redis.io, 2009.

[29] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. "Speedy transactions in multicore in-memory databases." In *SOSP '13*, pages 18–32, 2013.

[30] T. Willhalm, N. Popovici, Y. Boshmaf, H. Plattner, A. Zeier, and J. Schaffner. "SIMD-scan: Ultra fast in-memory table scan using on-chip vector processing units." In *PVLDB '09*, pages 385–394, 2009.

[31] C. Yao, D. Agrawal, P. Chang, G. Chen, B. C. Ooi, W.-F. Wong, and M. Zhang. "DGCC: A new dependency graph based concurrency control protocol for multicore database systems." *ArXiv e-prints*, Mar. 2015.

[32] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker. "Staring into the abyss: An evaluation of concurrency control with one thousand cores." In *PVLDB '15*, pages 209–220, 2014.

[33] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang. "In-memory big data management and processing: A survey." *TKDE*, 27(7):1920–1947, July 2015.

[34] H. Zhang, G. Chen, W.-F. Wong, B. C. Ooi, S. Wu, and Y. Xia. "Anti-caching-based elastic data management for big data." In *ICDE '15*, pages 592–603, 2014.

[35] H. Zhang, B. M. Tudor, G. Chen, and B. C. Ooi. "Efficient in-memory data management: An analysis." In *PVLDB '14*, pages 833–836, 2014.