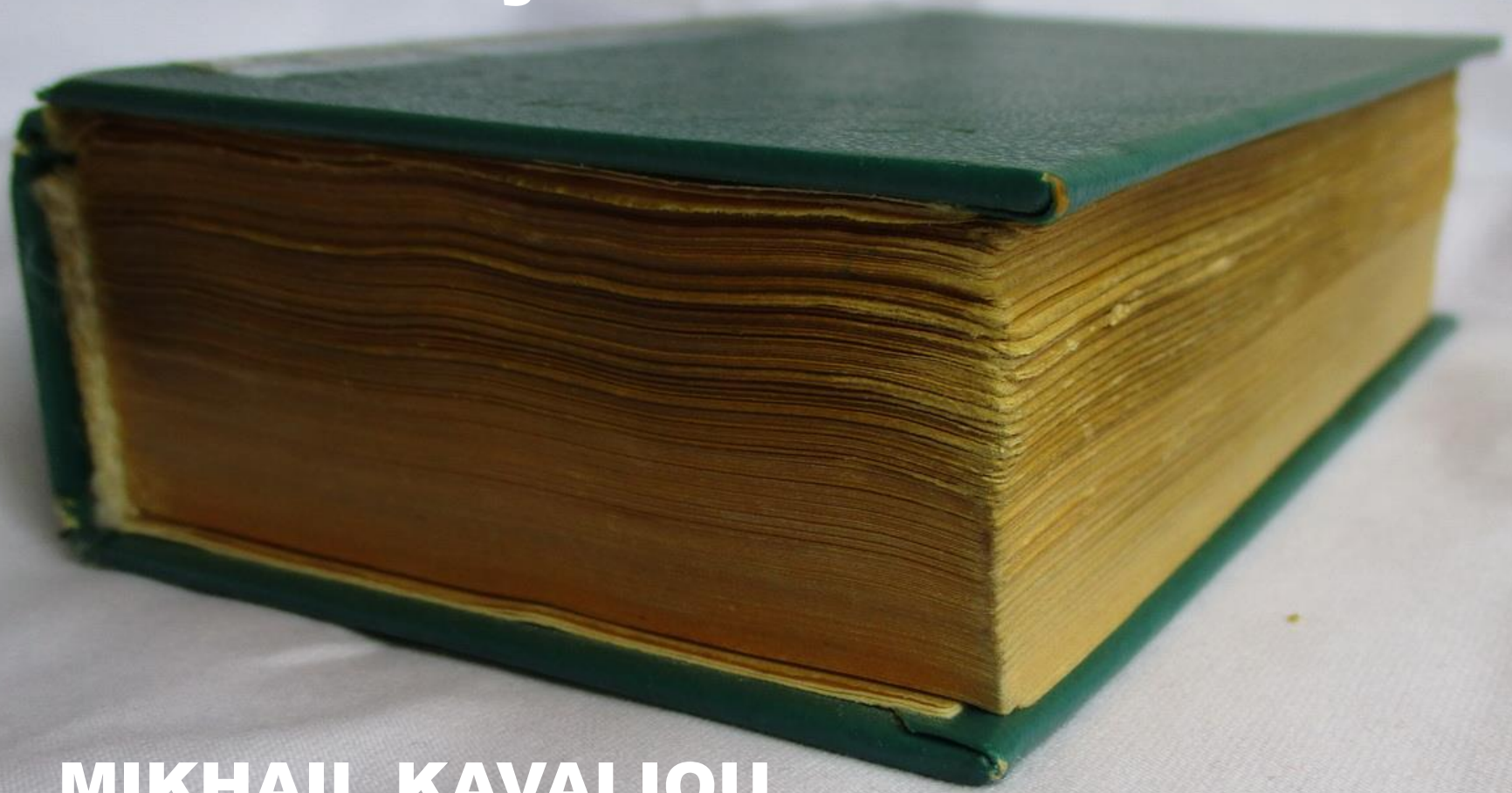
LEARNING INTELLIGENCE
the history of mine



MIKHAIL KAVALIOU

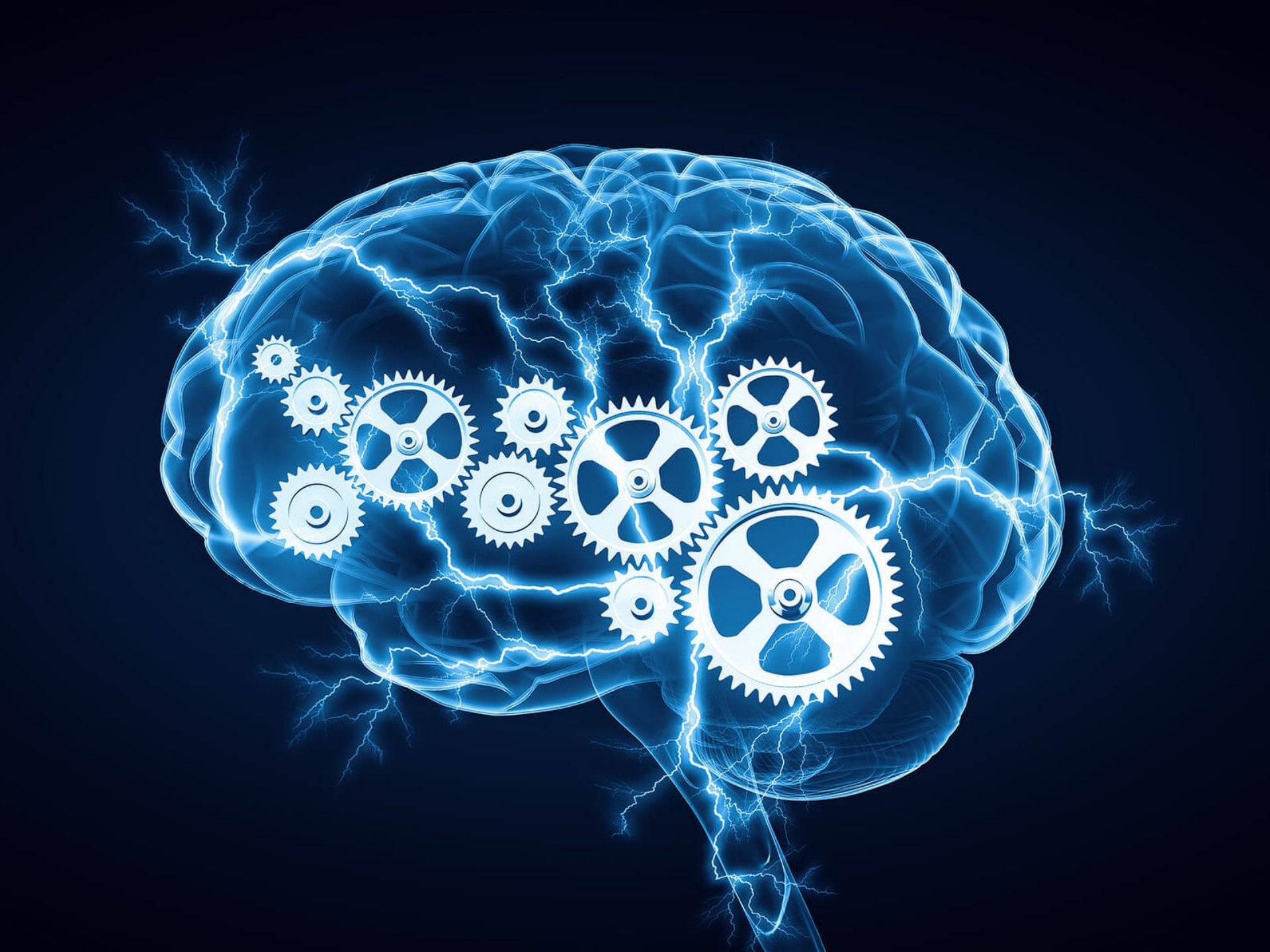
September, 2019

Mikhail Kavaliou

Senior Java Developer

6+ years in Java
3+ years in EPAM

MinuteForShop.com



Learning Styles

Audio

Visual

Kinaesthetic

container. The `BeanFactory` interface provides an advanced configuration mechanism capable of managing any type of object. `ApplicationContext` is a sub-interface of `BeanFactory`. It adds:

- Easier integration with Spring's AOP features
- Message resource handling (for use in internationalization)
- Event publication
- Application-layer specific contexts such as the `WebApplicationContext` for use in web applications.

In short, the `BeanFactory` provides the configuration framework and basic functionality, and the `ApplicationContext` adds more enterprise-specific functionality. The `ApplicationContext` is a complete superset of the `BeanFactory` and is used exclusively in this chapter in descriptions of Spring's IoC container. For more information on using the `BeanFactory` instead of the `ApplicationContext`, see The `BeanFactory`.

In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. Otherwise, a bean is simply one of many objects in your application. Beans, and the dependencies among them, are reflected in the configuration metadata used by a container.

1.2. Container Overview

The `org.springframework.context.ApplicationContext` interface represents the Spring IoC container and is responsible for instantiating, configuring, and assembling the beans. The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata. The configuration metadata is represented in XML, Java annotations, or Java code. It lets you express the objects that compose your application and the rich interdependencies between those objects.

Several implementations of the `ApplicationContext` interface are supplied with Spring. In stand-alone applications, it is common to create an instance of `ClassPathXmlApplicationContext` or `FileSystemXmlApplicationContext`. While XML has been the traditional format for defining configuration metadata, you can instruct the container to use Java annotations or code as the metadata format by providing a small amount of XML configuration to declaratively enable support for these additional metadata formats.

In most application scenarios, explicit user code is not required to instantiate one or more instances of a Spring IoC container. For example, in a web application scenario, a simple eight (or so) lines of boilerplate web descriptor XML in the `web.xml` file of the application typically suffices (see Convenient ApplicationContext Instantiation for Web Applications). If you use the Spring Tool Suite (an Eclipse-powered development environment), you can easily create this boilerplate configuration with a few mouse clicks or keystrokes.

The difference between using `FileSystemXmlApplicationContext` and `ClassPathXmlApplicationContext` is that `FileSystemXmlApplicationContext` will look for foo.xml in a specific location within the file system, whereas `ClassPathXmlApplicationContext` will look for foo.xml anywhere in the classpath (including JAR files).

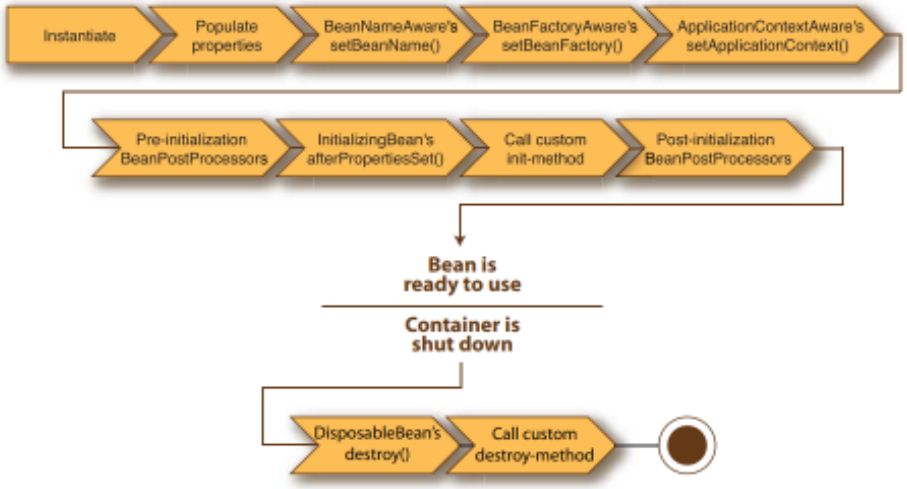


Figure 1.5 A bean goes through several steps between creation and destruction in the Spring container. Each step is an opportunity to customize how the bean is managed in Spring.

With an application context in hand, you can retrieve beans from the Spring container by calling the context's `getBean()` method.

Now that you know the basics of how to create a Spring container, let's take a closer look at the lifecycle of a bean in the bean container.

1.2.2 A bean's life

In a traditional Java application, the lifecycle of a bean is simple. Java's `new` keyword is used to instantiate the bean (or perhaps it's deserialized) and it's ready to use. Once the bean is no longer in use, it's eligible for garbage collection and eventually goes to the big bit bucket in the sky.

Аналогично выполняется загрузка контекста приложения из библиотеки классов приложения с помощью `ClassPathXmlApplicationContext`:

```
ApplicationContext context = new
        ClassPathXmlApplicationContext("foo.xml");
```

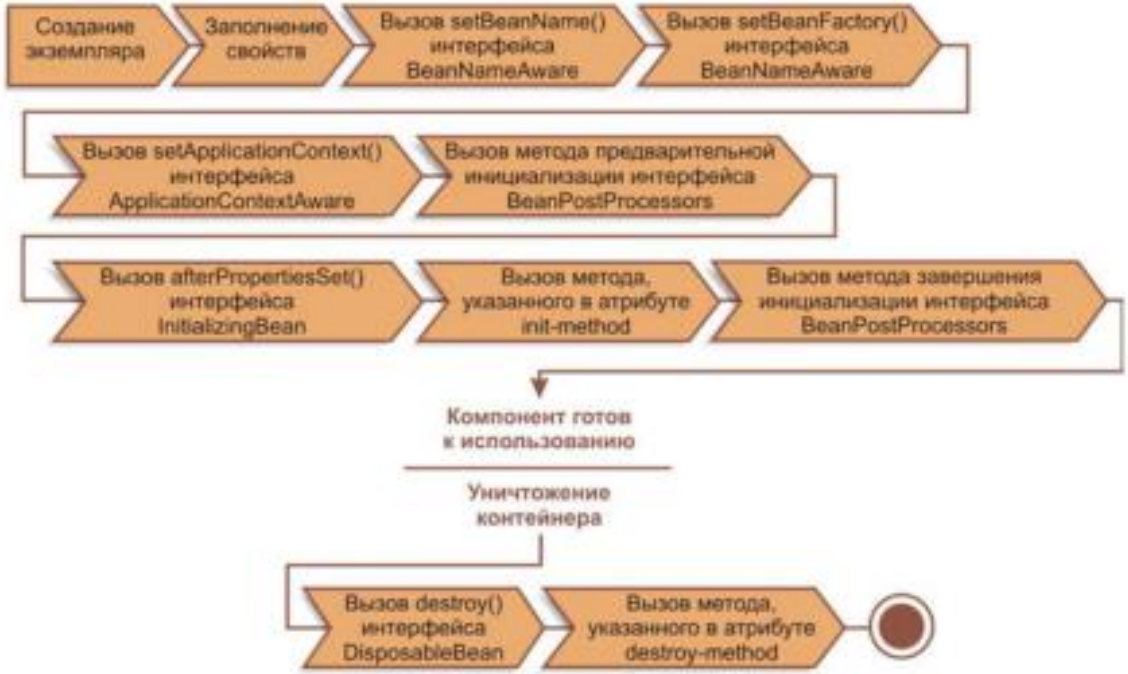


Рис. 1.5. От создания до уничтожения в контейнере Spring компонент преодолевает несколько этапов. Spring позволяет настроить выполнение каждого из этапов

Евгений Борисов — Spring-потрошитель, часть 2

69 289 просмотров

1,2 ТЫС. 3 ПОДЕЛИТЬСЯ СОХРАНИТЬ

JUG .ru
Опубликовано: 3 июл. 2014 г.

ПОДПИСАТЬСЯ 27 ТЫС.

Java-конференция Joker 2019:
25-26 октября, Санкт-Петербург.
Подробности и билеты: http://bit.ly/2uO0mzJ
ЕЩЁ

27 комментариев УПОРЯДОЧИТЬ

Оставьте комментарий

Igor Petrov 3 года назад
"Какая разница между Spring, String и Swing?")))
29 ОТВЕТИТЬ
Показать ответ

Phasers

A Phaser provides functionality similar to the CyclicBarrier and CountDownLatch synchronizers. It provides the following features:

Phaser is reusable.

The number of parties to synchronize on a Phaser can change dynamically. In a CyclicBarrier, the number of parties is fixed at the time the barrier is created.

A Phaser has an associated phase number, which starts at zero. When all registered parties arrive at a Phaser, the Phaser advances to the next phase and the phase number is incremented by one. The maximum value of the phase number is Integer.MAX_VALUE. After its maximum value, the phase number restarts at zero.

A Phaser has a termination state. All synchronization methods called on a Phaser in a termination state return immediately without waiting for an advance.

A Phaser has three types of parties count: a registered parties count, an arrived parties count, and an unarrived parties count.

The registered parties count is the number of parties that are registered for synchronization. The arrived parties count is the number of parties that have arrived at the current phase of the phaser.

The unarrived parties count is the number of parties that have not yet arrived at the current phase of the phaser.

When the last party arrives, the phaser advances to the next phase.

Optionally, a Phaser lets you execute a phaser action when all registered parties arrive at the

- **int arriveAndDeregister()** — сообщает о завершении всех фаз стороной и снимает ее с регистрации. Возвращает номер текущей фазы;

- **int awaitAdvance(int phase)** — если phase равно номеру текущей фазы, приостанавливает вызвавший его поток до её окончания. В противном случае сразу возвращает аргумент.

```
arriveAndAwaitAdvance();
arrive();
awaitAdvance(i);
arriveAndDeregister();
register();
```

Phaser

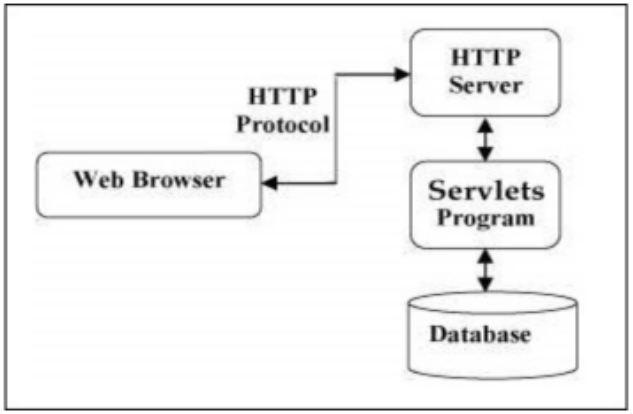
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

Servlets Architecture

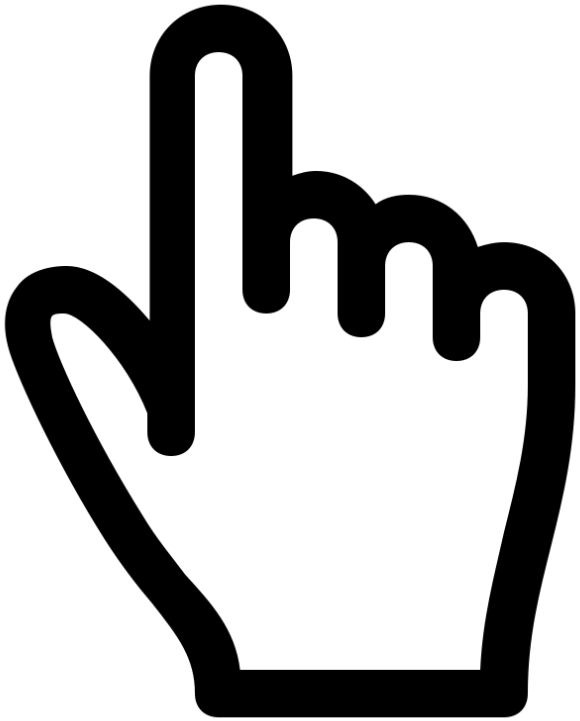
The following diagram shows the position of Servlets in a Web Application.



Servlets Tasks

Servlets perform the following major tasks −

https://github.com/MikhailKavaliou/LearningIntelligence




Never
stop
learning