Web Assembly for Java Developers

Thomas Darimont
Identity Tailor GmbH

@thomasdarimont

Sep 27th 2024

Thomas Darimont

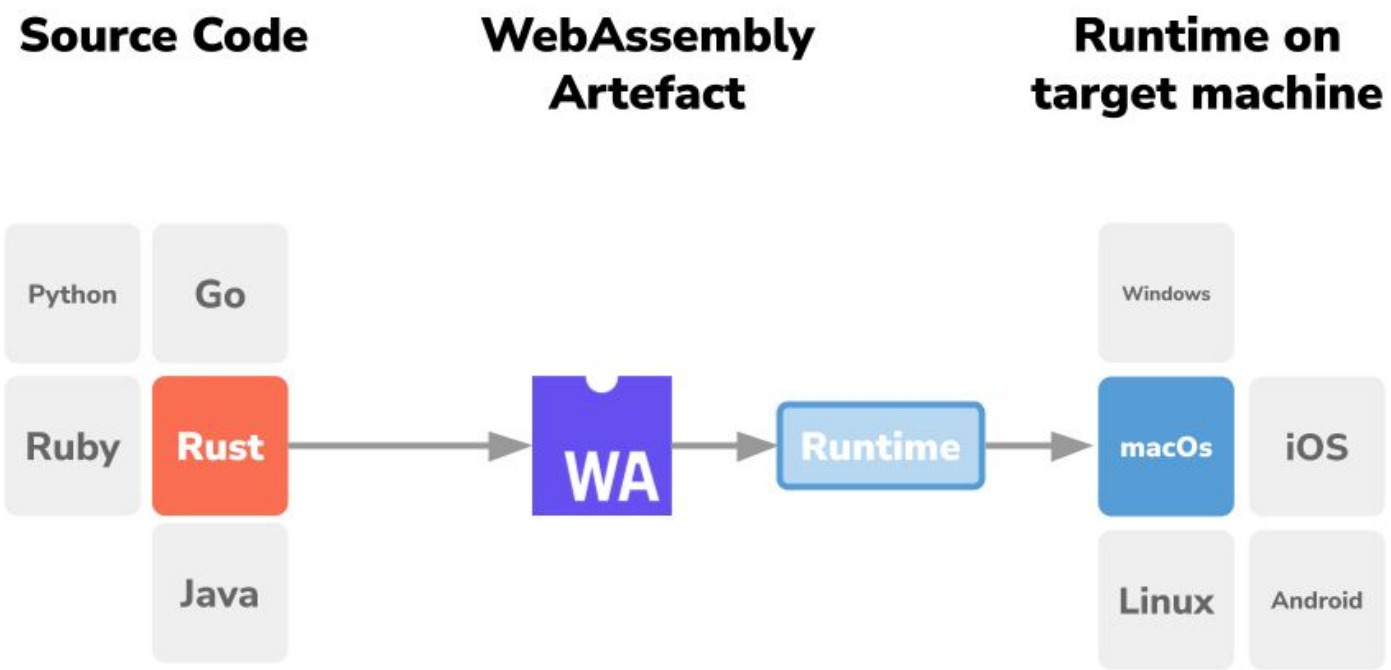
- Managing Director | Identity Tailor GmbH
- Focus on Digital Identities
- Open Source Enthusiast
- Spring Team Alumni
- Official Keycloak Maintainer
- OpenID Foundation Certification Team
- Java User Group Saarland Organizer
- Web Assembly Fanboy

@thomasdarimont
thomas@identity-tailor.de


WA
WEBASSEMBLY

*"**WebAssembly** or **Wasm** is a **binary** instruction **format** for a stack-based **"Virtual-machine."***

*"Wasm is **designed** as a **portable compilation target** for programming languages, enabling **deployment** on the web for **client** and **server** applications."*



Source: (Slightly adjusted) https://b-nova.com/en/home/content/how-containerless-works-thanks-to-web-assembly-runtimes

Source (C)

```c
int add(int a, int b) {
    return a + b;
}
```

```
(module
 (func $add (param $a i32) (param $b i32) (result i32)
   local.get $a
   local.get $b
   i32.add)
  ...
)
```

WAT (Web Assembly Text)

wat2wasm add.wat

```
00000000  00 61 73 6d 0b 00 00 00  04 74 79 70 65 87 80 80  |.asm.....type...|
00000010  80 00 01 40 02 01 01 01  01 08 66 75 6e 63 74 69  |...@......functi|
00000020  6f 6e 82 80 80 80 00 01  00 06 6d 65 6d 6f 72 79  |on........memory|
00000030  85 80 80 80 00 80 02 80  02 01 06 65 78 70 6f 72  |...........expor|
00000040  74 86 80 80 80 00 01 00  03 61 64 64 04 63 6f 64  |t........add.cod|
00000050  65 8c 80 80 80 00 01 86  80 80 80 00 00 14 00 14  |e...............|
00000060  01 40 04 6e 61 6d 65 86  80 80 80 00 01 03 61 64  |.@.name.......ad|
00000070  64 00                                            |d.|
```

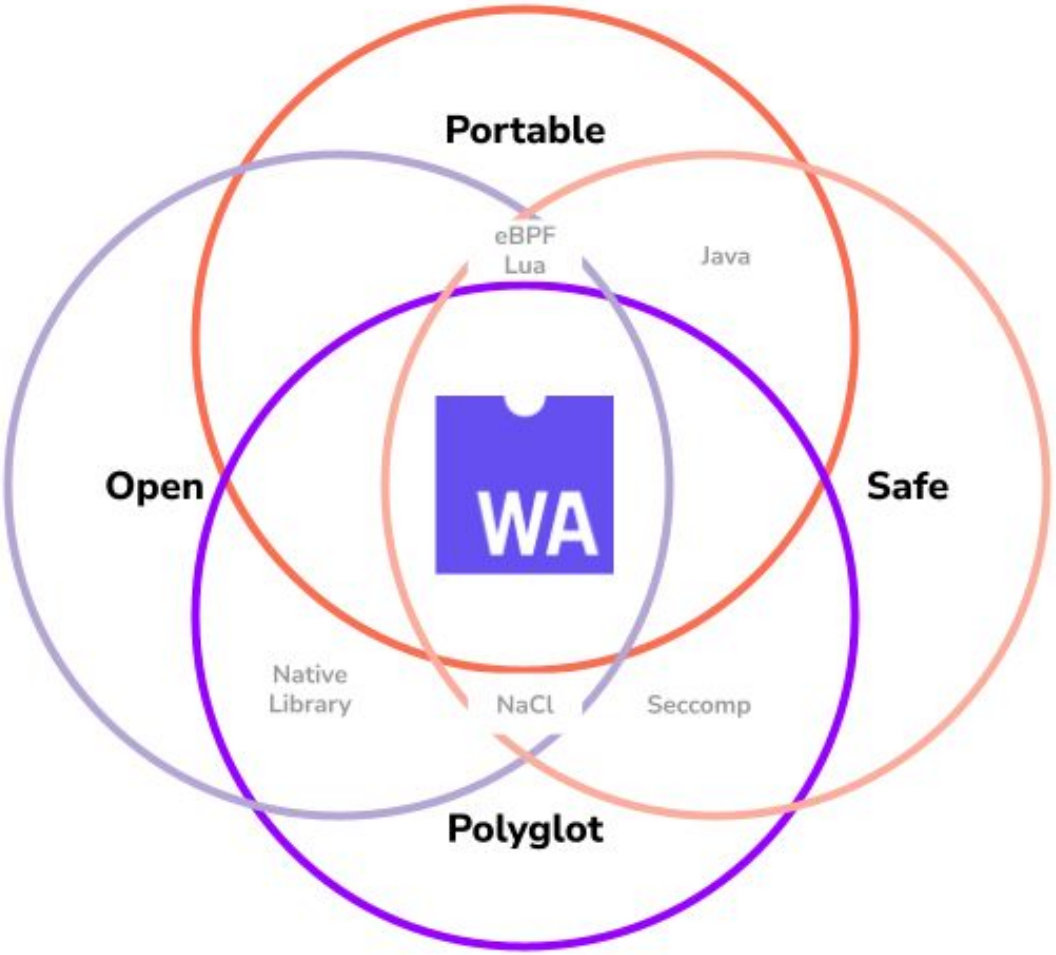
WASM (Web Assembly)

add.wasm

Benefits of Web Assembly

Runs on “every” Platform

Open Standard
Wide adoption



Sandbox
Confined memory
Limit execution
Capability config

Many languages compile to Web Assembly

Source: (Slightly adjusted) https://b-nova.com/en/home/content/how-containerless-works-thanks-to-web-assembly-runtimes

BYTECODE ALLIANCE

About the Bytecode Alliance

The Bytecode Alliance is a nonprofit organization dedicated to creating secure new software foundations, building on standards such as WebAssembly and WebAssembly System Interface (WASI).

The Bytecode Alliance is committed to establishing a capable, secure platform that allows application developers and service providers to confidently run untrusted code, on any infrastructure, for any operating system or device, leveraging decades of experience doing so inside web browsers.

We have a vision for a secure-by-default WebAssembly ecosystem for all platforms.

Web Assembly Use Cases *

Language Interoperability
Write library once; use with other languages
Figma, Google Earth, Adobe Photoshop

*) outside the Browser

Web Assembly Use Cases *

Language Interoperability	**Plugin Systems**	**Embedded Sandboxing**	**Containerization**	**Serverless**
Write library once; use with other languages	Flexible & secure plugin systems	Guard yourself against bugs in 3rd-party libraries	Universal Runtime, capability based security model	Minimal startup time, maximum isolation, Low resource usage
Figma, Google Earth, Adobe Photoshop	Envoy / Istio, Kubewarden, Minecraft, MS Flight Simulator, Nginx, Extism	Firefox, HttpServers	Kurstlet, Hippo, WasmCloud, WasmEdge, Spin Kube	CloudFlare Workers, AWS Lambda, Fastly, Fermyon Spin, Shopify

*) outside the Browser

What’s in for (Java) Developers?

- **Polyglot Programming**
  - Run code written in other languages in your JVM
  - Allow programmers to provide features in preferred language
- **Plugin Systems**
  - Make existing programs extensible
  - Replacement for Scripting
- **Efficient and fast Execution**
  - Fast cold start
  - Interpreter / JIT / AOT
- **Security Built-in**
  - Sandbox model
  - Execution with bounded Memory and CPU
  - Explicit capability mapping (FS / NET / OS)

Java on Web Assembly

Compile “Java” to WebAssembly

- Bytecoder
  - cross-compiles Java to WebAssembly that can be executed in the browser
- TeaVM
  - has experimental support for browser-based WebAssembly
- JWebAssembly
  - Translates JVM bytecode to WebAssembly, including Groovy, Clojure, and Kotlin
- CheerpJ
  - CheerpJ is a WebAssembly-based Java Virtual Machine for the browser.
- Kotlin Wasm
  - WebAssembly compilation target for Kotlin

Web Assembly on Java

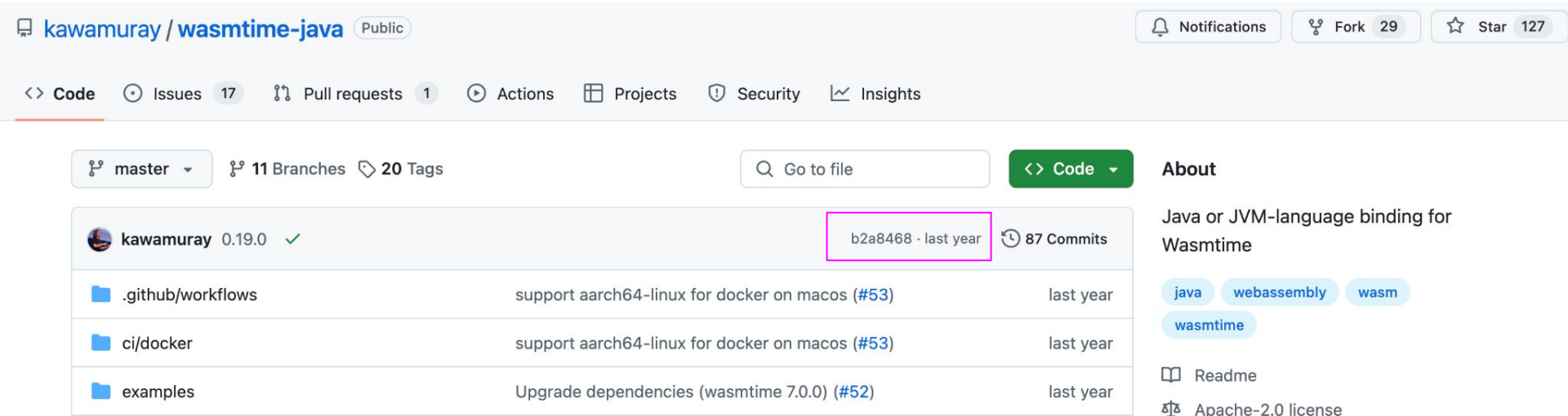
Wasmtime

github.com/kawamuray/wasmtime-java

A fast and secure runtime for WebAssembly

A Bytecode Alliance project

- wasmtime-java unofficial Java Support
- Calls wasmtime (rust) native library via Java Native Interface (JNI)
- Low-level interface



wasmtime-java Demo

```
;; call via: wasmtime sum.wat --invoke calc 1 2
(module

  ;; implementation of add function
  (func $add (param $a i32) (param $b i32) (result i32)
    local.get $a ;; push $a param on the stack
    local.get $b ;; push $b param on the stack
    i32.add) ;; pop $a and $b from the stack, compute sum, and push $result on s

  ;; expose function "calc" which calls our "add" function
  (export "calc" (func $add))
)
```

```
String wasmLocation = WasmIO.locateWatFromClasspath("sum.wasm").toFile().getAbsolutePath();

try (var store = Store.withoutData(); //
     var engine = store.engine(); //
     var module = Module.fromFile(engine, wasmLocation); //
     var instance = new Instance(store, module, Collections.emptyList()); //
     var func = instance.getFunc(store, name: "calc").orElseThrow()) {

    var results = func.call(store, WasmValType.I32.toWasmVal(3), WasmValType.I32.toWasmVal(4));

    var result = results[0];

    System.out.printf("Result: %s", result.i32());
}
```

Extism Universal Plug-in System

extism.org

Call code from your apps.

The cross-language framework for building with WebAssembly

Read the docs

Java SDK

Quickly embed into officially supported languages:

Easy to Use

Leveraging the power and portability of WebAssembly, Extism is an off-the-shelf plug-in system just a library import away. Ship in days, not weeks or months.

Secure by Default

Don't worry about what some plug-in code might do to your program. Extism is built with security as a core principle, and fully sandboxes the execution of all plug-in code.

Available Everywhere

Our flexible architecture uniquely allows Extism to run almost anywhere, with idiomatic Host SDKs for Python, Node, Ruby, Rust, Go, PHP, C/C++, OCaml, & more.

Extism

- **PDK (Plugin SDK)**
  Wrapper around wasmtime via JNA
  Support for WASI (Web Assembly System Interface)

- **Flexible Data-Exchange**
  Glue code for Data-exchange between Host and WASM module as JSON

- **Easy to Use**
  Leveraging the power and portability of WebAssembly, Extism is an off-the-shelf plug-in system just a library import away. Ship in days, not weeks or months.

- **Secure by Default**
  Don't worry about what some plug-in code might do to your program. Extism is built with security as a core principle, and fully sandboxes the execution of all plug-in code.

Use-cases of a plug-in system

- Adding functionality to command-line tools
- Enabling users to "mod" a game
- Simplify "webhooks" to run event-driven logic on the server
- User-defined functions in a database
- No-code application extensions
- Content management system extensions

Extism Plugin Java SDK Demo

```java
public class ExtismGoVowels {  Thomas Darimont *

    public static void main(String[] args) {  Thomas Darimont *

        var source = new PathWasmSource( name: "code", path: "demos/extism-demo/wasm/vowels/go/vowels.wasm", hash: null)

        try (var plugin = new Plugin(new Manifest(source), withWASI: true, functions: null)) {

            String output = plugin.call( functionName: "count_vowels", input: "JugSaxony");
            System.out.println(output);
        }
    }
}
```

code.wasm

Compiled with tinygo to WASM

```go
import (
    "strconv"
    "github.com/extism/go-pdk"
)

// build via
// tinygo build -o code.wasm -target wasi code.go

//export count_vowels
func count_vowels() int32 {
    input := pdk.Input()

    count := 0
    for _, a := range input {
        switch a {
        case 'A', 'I', 'E', 'O', 'U', 'a', 'e', 'i', 'o', 'u':
            count++
        default:
        }
    }

    output := `{"count": ` + strconv.Itoa(count) + `}`
    mem := pdk.AllocateString(output)

    pdk.OutputMemory(mem)
    return 0
}

func main() {}
```

github.com/dylibso/chicory

Chicory Runtime

Tests Interpreter 27877 success, 0 skip, 0 failures, 0 errors, 27877 total
Tests AOT 27877 success, 0 skip, 0 failures, 0 errors, 27877 total
Tests WASI 49 success, 0 skip, 0 failures, 0 errors, 49 total

zulip join chat

Chicory is a JVM native WebAssembly runtime. It allows you to run WebAssembly programs with zero native dependencies or JNI. Chicory can run Wasm anywhere that the JVM can go. It is designed with simplicity and safety in mind. See the development section for a better idea of what we are trying to achieve and why.

> *Reach out to us*: Chicory is very early in development and there will be rough edges. We're hoping to talk to some early adopters and contributors before we formally announce it a beta to the world. Please join our team Zulip chat with this invite link if you're interested in providing feedback or contributing. Or just keeping up with development.

Chicory

- Pure JVM native Web Assembly Runtime
- Initiated by the folks behind the Extism project
- Inspired among others by wazero (zero deps Go Wasm RT)
- Many attempts to bring WASM to the JVM
  - Implementing the WebAssembly Spec is hard work!
  - Chicory = Joint forces of many WASM & JVM enthusiasts

Chicory - a pure Java WASM Runtime ... but why?

- Native WebAssembly Runtimes
  - e.g. wasmtime, wasmedge, wasmer
  - Extremely performant (fast start, small memory footprint)
  - Only for a specific platform
  - Require native library
- JVM based “native” Runtimes
  - Can leverage JVM JIT
  - Require no native libraries
  - Provide Sandbox within JVM boundaries
  - Easier to interface with

Chicory Goals / Non-Goals

- **Goals**
  - Be as safe as possible
  - Willing to sacrifice performance for safety and simplicity
  - Wasm execution in every JVM environment without native code
  - Fully support the core Wasm spec
  - Make integration with Java (and other languages) easy and idiomatic

- **Non-Goals**
  - Be a standalone runtime
  - Be the fastest runtime
  - Be the right choice for every JVM project

Chicory Current State

- Current Version: 0.0.12
- Highly motivated & determined Team :)
- Functional Web Assembly Interpreter
- 27k+ (!!!) Web Assembly 1.0 Spec Tests passing
- Support for WAT (text) and WASM (binary)
- Initial support for AOT Compilation to JVM Bytecode
- … it can already run some complex applications
- API still in flux

Executing WASM Functions with Chicory

```java
import com.dylibso.chicory.runtime.ExportFunction;
import com.dylibso.chicory.runtime.Module;
import com.dylibso.chicory.wasm.types.Value;

import java.nio.file.Path;

public class ChicoryAddDemo {

    public static void main(String[] args) {

        var wasmPath = Path.of( first: "demos/chicory-demo/wasm/add/rust/add.wasm");
        var instance = Module.builder(wasmPath).build().instantiate();

        ExportFunction addFunc = instance.export( name: "add");

        Value[] input = {Value.i32( data: 3), Value.i32( data: 5)};
        Value[] output = addFunc.apply(input);

        System.out.println(output[0].asLong());
    }
}
```

Calling Java Function from WASM with Chicory

Allow WASM to call a Java Method!

```java
public class ChicoryHostFunctionDemo {

    public static void main(String[] args) {

        // this is called from web assembly!
        var func = new HostFunction((Instance instance, Value... inputs) -> {
            // read message from WASM instance
            var len = inputs[0].asInt();
            var offset = inputs[1].asInt();
            var message = instance.memory().readString(offset, len);

            System.out.printf("### %s%n", message);
            return null;
        }, moduleName: "console", fieldName: "log", List.of(ValueType.I32, ValueType.I32), List.of());

        var instance = Module.builder(Path.of( first: "demos/chicory-demo/wasm/log/logger.wasm")).build()
                .withHostImports(new HostImports(new HostFunction[]{func})) // expose func to WASM runtime
                .instantiate();

        var logIt = instance.export( name: "logIt");

        // this calls a web assembly function
        logIt.apply(Value.i32( data: 5)); // Prints the message hello world 5 times
    }
}
```

“That’s all quite low-level… can you do more than adding numbers or logging strings?”

Evaluating Open Policy Agent Policies in Java

- Open Policy Agent (OPA)
- CNCF Project
- Open Source is a Policy Engine written in Go
- Rego Language for defining Policies
- Policy Evaluation (usually) via Sidecar / API calls
- …but policies can be compiled to WASM 🤓
- … how about executing policies directly in the JVM?
- PoC by the great Andrea Peruffo

Execute WASM OPA Policies with Chicory

README

CI passing JitPack main-9931be414b-1

github.com/andreaTP/opa-chicory

Open Policy Agent WebAssembly Java SDK (experimental)

This is an SDK for using WebAssembly (wasm) compiled Open Policy Agent policies with Chicory, a pure Java Wasm interpreter.

Initial implementation was based on Open Policy Agent WebAssemby NPM Module and Open Policy Agent Ebassembly dotnet core SDK

Why

We want fast in-process OPA policies evaluations, and avoid network bottlenecks when using opa-java.

Meet Chicory, exploit the power of WebAssembly on the server side!

TOOLS-IN-ACTION

(INTERMEDIATE LEVEL)

Monday from 18:20 – 18:50

Room 6

WebAssembly is a rapidly emerging technology that enables the execution of code written in various languages while providing strong sandboxing and safety guarantees. Initially developed for the web to enhance browser capabilities, developers soon recognized the potential of reusing Wasm modules in server-side applications. wazero, a native Go runtime for Wasm, played a pivotal role in showcasing the versatility and power of this solution. With its widespread adoption and integration into diverse applications, wazero demonstrated the value of using Wasm modules beyond the web environment.
Inspired by the goals of wazero, we launched Chicory, a pure Java interpreter, with zero dependencies, for Wasm. Chicory empowers developers to load and execute Wasm modules with fine-grained control over their interactions with the system and memory allocation. Notably, Chicory seamlessly integrates with barebone JVM runtimes, eliminating any system dependencies.
In this presentation, we will explore the exciting possibilities that Chicory offers for the JVM ecosystem. Through practical, real-world examples, we will showcase how Chicory can be seamlessly integrated into your application, enabling you to run Wasm programs within minutes. Additionally, we will discuss the various approaches to designing integrations, exploring the trade-offs associated with each option.

ANDREA PERUFFO 𝕏
Red Hat

With nearly two decades of coding experience, I'm fueled by passion as I continue to type away daily.
As a Principal Software Engineer at Red Hat, I actively contribute to diverse Open Source projects, driven by both personal fulfillment and professional advancement. My not-so-secret passion lies in programming languages, developer tools, compilers, and beyond. Come and spot me on a project near you!

RELATED

- A better Jupyter Experience for Java Developers - JTaccuino unveiled
- Run your favorite games everywhere with WASM: the BlazorDoom use case
- Supercharge your Java Applications with Python!
- Test Automation with Selenium 5 and Java

Can Chicory run DOOM?

Can Chicory run DOOM? Oh yes!

- Rust Doom Port
- Compiled to WASM
- Demo for running Doom in the browser
- Interfaces with Host Environment (Browser)
  - Attach to Browser Window
  - Handle Key events
  - Draw raw Framebuffer onto Canvas
- How about replacing the Browser bindings with…
  - JFrame
  - Swing EventListener
  - Draw to a Buffered Image

WASM Doom on Chicory last year…

WASM Doom on Chicory now with AOT

Doom on Chicory

https://github.com/thomasdarimont/doom-chicory/tree/poc/aot

```
mvn clean verify
```

```
java -Dchicory.aot=true -jar ./target/doom-*-SNAPSHOT-jar-with-dependencies.jar
```


Doom on Chicory
FPS: 36
DOOM
New Game
Options
Load Game
Save Game
Read This!
Quit Game
id
SOFTWARE
PROVIDED BY id FREE OF CHARGE • SUGGESTED RETAIL PRICE $9.00 • id SOFTWARE ©1993


Doom on Chicory
FPS: 35
180
AMMO
100%
HEALTH
2 3 4
5 6 7
ARMS
200%
ARMOR
BULL 180 / 200
SHEL 50 / 50
RCKT 50 / 50
CELL 300 / 300

Summary

- Web Assembly support on the JVM is here to stay!
- Many powerful integration options available
- JVM native implementations on the rise!
- Better developer experience will be a game changer
- Web Assembly has potential beyond the browser!

Web Assembly for Java Developers

Thank you!

Code & Slides

github.com/thomasdarimont/javawasm-talk

@thomasdarimont

Sep 26th 2024