REST APIS WITH MICRONAUT

Stanciu Neculai, The Republic

AGENDA

- Challenges Facing Java
- Introduction to Micronaut
- Micronaut http server

JAVA AND SERVERLESS

- Challenges to using Java in Serverless / Microservices scenarios
- Existing Tools and Frameworks Not Optimized for Cold Starts / Low Memory
- Go, Node etc. better in this regards
- Tim Bray (Amazon/AWS) and others not recommending Java

https://youtu.be/IPOvrK3S3gQ?t=1109

JAVA'S PROBLEMS FOR FRAMEWORKS

- Limited Annotation API
- Type Erasure
- Slow Reflection
- Reflective Data Caches
- Classpath Scanning
- Slow Dynamic Class Loading

JAVA'S PROBLEMS

- Greatly Exaggerated (Java has been dead forever)
- Java can be Fast! (see Android and Micronaut)
- However Most Existing Tools are based around
    - Reflection
    - Runtime Proxies
    - Runtime Byte Code Generation (bytebuddy/cglib)

WHY IS REFLECTION A PROBLEM?

- **All** Reflective data initialized on first access and held in soft references (yes every field, method etc.)
- Won't be GC'ed until your application is low on memory!

AVOID REFLECTION!

- Reflection usages increases memory usage
- Using reflection relatively slow
- Problem is most modern server-side frameworks and specifications are built on reflection
- Some of the Jarkarta specifications even mandate reflection

JUST LIKE THE ANDROID WORLD

- The Android Community already solved the problem
- Ahead of Time Compilation used extensively
- Google Dagger 2.x for DI
- Most Android Frameworks avoid reflection to avoid paying the memory / performance cost

MICRONAUT

- A Microservices and Serverless Focused framework (hence the branding)
- Also a Complete Application Framework for any type of Application
- Dependency Injection, Aspect Oriented Programming (AOP), Configuration Management, Bean Introspection and more..

AUTHOR

- Graeme Rocher
  - Creator of Grails and Micronaut
  - Principal Engineer at **Object Computing**
  - 2018 Oracle Groundbreaker Award Winner
  - Java Champion

WITH MICRONAUT YOU CAN BUILD

- Microservices
- Serverless Applications
- Message-Driven Applications with Kafka/Rabbit
- CLI Applications
- Even Android Applications
- Anything with `static void main(String..args)`

SO WHAT IS MICRONAUT? REALLY?

- An Application Framework for the Future
- Reflection Free, Runtime Proxy Free, No Dynamic Classloading (in 1.1)
- Ahead of Time (AOT) Compilation AOT APIs
- ... oh and let's you build Microservices

FAST STARTUP TIME — LOW MEMORY CONSUMPTION

Reflection-based IoC frameworks load and cache reflection data for every single field, method, and constructor in your code, whereas with Micronaut, your application startup time and memory consumption are not bound to the size of your codebase.

MICRONAUT'S SOLUTIONS

Problem	Solution
Limited Annotation API	Precomputed `AnnotationMetadata`
Type Erasure	Precomputed `Argument` Interface
Slow Reflection	Eliminate Reflection
Reflective Data Caches	Zero Data Caches
Classpath Scanning	No Classpath Scanning
Slow Dynamic Class Loading	No Dynamic Class Loaders

A CONTROLLER IN MICRONAUT

```
// build.gradle key dependencies
 annotationProcessor enforcedPlatform("io.micronaut:micronaut-bom:$micronautVersion")
 annotationProcessor "io.micronaut:micronaut-inject-java"
 implementation enforcedPlatform("io.micronaut:micronaut-bom:$micronautVersion")
 implementation "io.micronaut:micronaut-inject"
```

```
import io.micronaut.http.MediaType;
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;

@Controller("/hello") //#1
public class HelloController {

    @Get(produces = MediaType.TEXT_PLAIN) //#2
    public String index() {
        return "Hello World"; //#3
    }
}
```

RUN

```
$ ./gradlew run
> Task :run
[main] INFO  io.micronaut.runtime.Micronaut - Startup completed in 972ms. Server Running: htt
```

ROUTING EXAMPLES

Template	Description	Matching URI
/books/{id}	Simple match	/books/1
/books/{id:2}	A variable of 2 characters max	/books/10
/books{/id}	An optional URI variable	/books/10 or /books
/book{/id:[a-zA-Z]+}	An optional URI variable with regex	/books/foo
/books{?max,offset}	Optional query parameters	/books?max=10&offset=10
/books{/path:.*}{.ext}	Regex path match with extension	/books/foo/bar.xml

@Introspected

```
import io.micronaut.core.annotation.Introspected;

@Introspected
public class Person {

    private String name;
    private int age = 18;

    public Person(String name) {
        this.name = name;
    }
  // ... getters and setters
}
```

- AOT Reflection-free replacement for `java.beans.Introspector`
- Set/get bean properties, create instances
- Includes `AnnotationMetadata`

BEANINTROSPECTOR EXAMPLE

```
final BeanIntrospection<Person> introspection = BeanIntrospection.getIntrospection(Person.cla
        Person person = introspection.instantiate("John");
        System.out.println("Hello " + person.getName());

        final BeanProperty<Person, String> property = introspection.getRequiredProperty("name
        property.set(person, "Fred");
        String name = property.get(person);
        System.out.println("Hello " + person.getName());
```

FILTER SUPPORT

The Micronaut HTTP server supports the ability to apply filters to request/response processing in a similar, but reactive, way to Servlet filters in traditional Java applications.

WRITING A FILTER

```
import io.micronaut.http.*;
import io.micronaut.http.annotation.Filter;
import io.micronaut.http.filter.*;
import org.reactivestreams.Publisher;

@Filter("/hello/**")
public class TraceFilter implements HttpServerFilter {
    private final TraceService traceService;

    public TraceFilter(TraceService traceService) {
        this.traceService = traceService;
    }

    @Override
    public Publisher<MutableHttpResponse<?>> doFilter(HttpRequest<?> request, ServerFilterCha
        return traceService.trace(request)
                           .switchMap(aBoolean -> chain.proceed(request))
                           .doOnNext(res ->
                                res.getHeaders().add("X-Trace-Enabled", "true")
                           );
    }

}
```

ERROR HANDLING

LOCAL ERROR EXAMPLE

```
@Error(exception = ConstraintViolationException.class)
public Map<String, Object> onSavedFailed(HttpRequest request, ConstraintViolationException
  final Map<String, Object> model = new HashMap<>();
  model.put("errors", messageSource.violationsMessages(ex.getConstraintViolations()));
  return model;
}
```

You can check a full example in surveys-api here

GLOBAL ERROR HANDLER

```
@Error(global = true)
public HttpResponse<JsonError> error(HttpRequest request, Throwable e) {
  JsonError error = new JsonError("Bad Things Happened: " + e.getMessage())
      .link(Link.SELF, Link.of(request.getUri()));

  return HttpResponse.<JsonError>serverError()
      .body(error);
}
```

You can check a full example in surveys-api here

API VERSIONING

The following example demonstrates how to version an API:

```
import io.micronaut.core.version.annotation.Version;
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;

@Controller("/versioned")
class VersionedController {

    @Version("1")
    @Get("/hello")
    String helloV1() {
        return "helloV1";
    }

    @Version("2")
    @Get("/hello")
    String helloV2() {
        return "helloV2";
    }
```

Request a specific version:

```
$ curl http://localhost:8080 -H "X-API-VERSION:1"
```

QUESTIONS ?

DOCUMENTATION

- Introduction to Micronaut Framework
- Micronaut deep dive