

Writing a Java library with better experience

Trustin Lee, LINE
Jul 2020

 @trustin

A three-part session

- Core principles
- Tips & tricks
 - Mostly Java-specific
 - Applicable to other languages
- How to nurture the community
 - ... for mutual benefit of users & yourself
 - May seem less important but is a crucial part

Where this came from



- 🌟 23.6k 👤 100
- 🐱 netty/netty
- 🐦 @netty_project

- 🌟 2.6k 👤 111
- 🐱 line/armeria
- 🐦 @armeria_project

Core principles

Be in user's shoes

- What will user's code look like?
- Imagine the best possible user code for the core problem.
 - Type like the API is already there.
 - Forget about how you'll implement it.
- Keep refining while adding increasingly complex use cases.
- Reach a certain level of confidence first.
 - Keep in mind – This is not a sprint but a journey.
 - No need to implement anything until ready.

Consistency

- ‘A’ is like this. Why is ‘B’ like that?
- Consistency in your API
 - e.g. This builder accepts `long`, but this accepts `Duration`.
- When in doubt, check others’ work:
 - Language SDK’s API
 - JDK, Kotlin SDK, ...
 - Other popular libraries
 - Guava, Spring, ...
- Don’t follow blindly.
 - Different context, new trends, mistakes, ...

Cognitive load · Learning curve

- Start with a small set of concepts.
- Build more complex solutions on top of them.
- Use familiar constructs
 - Builders and factories
 - Decorators and strategies
 - (Functional) Composition
- Don't expose too much at once.
 - e.g. 10 overloaded builder methods

User experience \gg Internal complexity

- Always choose UX.
 - Good UX is crucial for the virtuous cycle of project.
- Can tame internal complexity, but can't tame poor UX:
 - Confused users → Too many support tickets → Poor experience · Less time for dev →
 - Low-quality feed back · Burnout → Fix not at its best form → ...
- API with good UX often leads to better implementation:
 - Encapsulation → More freedom at implementation level
 - Composition → Less complexity at implementation level

Tips & tricks

Use Java, not other languages

- Non-Java library means:
 - Additional (BIG) runtime dependencies
 - Weird synthetic classes and methods
 - Lang A → Java & Lang B → Java may be easy, but Lang A → B may not.
- Keep the core in Java
- Provide language-specific layers, e.g. DSL
 - Use others' help if you are not good at those languages.

Keep core dependencies minimal

- Don't depend on another framework
- Let people choose
 - ... by providing integration layers
 - Spring Boot, Dropwizard, ...
- Shade utility dependencies
 - Guava, Caffeine, FastUtil, JCTools, Reflections, ...
 - Use ProGuard to trim unused shaded classes: 30 MB → 6 MB
 - Be aware of licenses

Go non-null by default

- Use `@Nullable`.
 - Do not use `j.l.Optional`.
- A language has its own `null` handling:
 - `Option` (Scala)
 - `Nullable` types (Kotlin)
- The caller can wrap the result:
 - `Optional.ofNullable()`
 - `Option()`
- Escape analysis doesn't always work.

```
class Server {
    ...
    @Nullable
    ServerPort activePort() { ... }
    ...
}

class CacheControlBuilder {
    ...
    CacheControlBuilder maxAge(
        @Nullable Duration maxAge) { ... }
    ...
}
```

@NonNullByDefault in package-info.java

```
/**
 * Indicates the return values, parameters and fields are non-nullable by default.
 * Annotate a package with this annotation and annotate nullable return values,
 * parameters and fields with {@link Nullable}.
 */
@NonNull
@Documented
@Target(ElementType.PACKAGE)
@Retention(RetentionPolicy.RUNTIME)
@TypeQualifierDefault({
    ElementType.METHOD,
    ElementType.PARAMETER,
    ElementType.FIELD })
public @interface NonNullByDefault {}
```

Nullness annotation libraries

- `javax.annotations` (jsr305)
 - Most popular
 - Not compatible w/ Java Modules
- `jspecify`
 - Joint effort to replace jsr305
 - Too soon too tell
- JetBrains annotations
 - IDE support
 - Contract annotations
 - `@Contract("null -> null")`
- Checker framework
 - Mature
 - More than just annotations

Validate early with message

- Which stack trace is easier to understand?

```
java.lang.NullPointerException: idleTimeout
    at java.util.Objects.requireNonNull( )
    at com.linecorp.armeria.server.ServerBuilder.idleTimeout( )
    at com.linecorp.armeria.server.ServerTest$1.configure( )
```

```
java.lang.NullPointerException: null 🤔
    at com.linecorp.armeria.server.ServerBuilder.initSocket( )
    at com.linecorp.armeria.server.ServerBuilder.build( )
    at com.linecorp.armeria.server.ServerTest$1.configure( )
```

Think cognitive load

- Raise an exception as soon as a bad value is given.
 - e.g. Don't validate builder properties at `build()`
- Provide a meaningful message
 - What is `null`?
 - Use `Objects.requireNonNull()`
 - Why is this bad and what is good?
 - Use `Preconditions.checkArgument()`

Consistent exception messages

- `NullPointerException`
 - “paramName”
- `IllegalArgumentException`
 - “paramName: badValue (expected: goodValueSpec)”
 - `idleTimeoutMillis: -1 (expected: >= 0)`
 - `filePath: ../my_file (expected: an absolute path)`
- Choose what works for your project
 - ... but the messages should be consistent and meaningful.

```
import static com.google.common.base.Preconditions.checkArgument;
import static java.util.Objects.requireNonNull;

public final class ServerBuilder {
    ...
    public ServerBuilder port(ServerPort port) {
        ports.add(requireNonNull(port, "port"));
        return this;
    }

    public ServerBuilder http2MaxStreamsPerConnection(
        long http2MaxStreamsPerConnection) {
        checkArgument(
            http2MaxStreamsPerConnection > 0 &&
            http2MaxStreamsPerConnection <= 0xFFFFFFFFL,
            "http2MaxStreamsPerConnection: %s (expected: a 32-bit unsigned integer)",
            http2MaxStreamsPerConnection);
        this.http2MaxStreamsPerConnection = http2MaxStreamsPerConnection;
        return this;
    }
    ...
}
```

Static factory methods over constructors

- Easier to hide implementations
- More naming options
 - `of()`, `ofDefault()`, `empty()`
- Chaining to a builder
 - `CorsService.builder(String... origins)`
 - `CorsService.builderForAnyOrigin()`
- Optimization opportunity
 - Create an instance of different type for different parameters

```
public interface EndpointGroup extends ... {
    static EndpointGroup empty() {
        // StaticEndpointGroup is package-private.
        return StaticEndpointGroup.EMPTY;
    }

    static EndpointGroup of(EndpointGroup... endpointGroups) {
        requireNonNull(endpointGroups, "endpointGroups");
        // Highly simplified pseudo code
        // that shows how a static factory can optimize:
        switch (actualNumberOfEndpoints) {
            case 0:
                return empty();
            case 1:
                return endpointGroups[0].first();
            default:
                // CompositeEndpointGroup is package-private.
                return new CompositeEndpointGroup(endpointGroups);
        }
    }
    ...
}
```

Avoid inner classes

- SomeClass.Builder

```
import com.example.SomeClass.Builder;  
Builder builder = SomeClass.builder();
```

- Whose “Builder” is it?
- What happens if we import OtherClass.Builder?
- It’s dev-centric to organize the builder impl in the same class.
 - User experience >>> Internal complexity

Avoid code generators in public API

- e.g. Lombok
 - Sometimes OK for generating internal classes, though.
- Public API requires a lot more human touch
 - Even a simple getter · setter needs hand-crafted Javadoc
 - When this method should be used
 - The default value when unused
 - Examples
- Often lowers the contribution barrier if not used

Hand-crafted setter in action

```
public final class CookieBuilder {
    /**
     * Sets the SameSite
     * attribute of the Cookie. The value is supposed to be one of
     * "Lax", "Strict" or "None". Note that this
     * attribute is server-side only.
     */
    public CookieBuilder sameSite(String sameSite) {
        this.sameSite = validateAttributeValue(sameSite, "sameSite");
        return this;
    }
}
```

Follow semantic versioning

Version 1.3.2

<major>.<minor>.<micro>

- Major bump → Breaking ABI changes
 - Dropping a new version of JAR may fail.
- Minor bump → New features
 - May add new classes
 - May add *default* or *static* methods
 - May add methods to `final` classes
- Micro bump → Bug fixes
 - No public API changes
- It's hard to follow semantic versioning strictly.
 - Experiment as much as possible in `0.x.y` days.
 - Use tools such as `JDiff`

Keep public API to minimum

- Think twice before adding 'public' or 'protected'.
 - Hide implementations and use static factory methods in interface.
 - If not possible because of inter-package references:
 - Move to the 'internal' packages.
 - Can be hidden or deprioritized by tools at least – Javadoc, IDE, ...
- Why?
 - API change often requires a major version bump.
 - Nobody likes breaking changes...
 - Implementation detail changes *often* in reality – bugs, performance, hindsight, ...

Make your classes & methods final

- Prefer composition over inheritance.
 - Accept Function, Consumer, Predicate, ...
- Why?
 - User mistakes:
 - Forgets to call super, Performs something totally unexpected
 - Good balance between UX and DX (developer experience)
- Discuss · Think *a lot* before removing final (= opening a can of worms!)
 - Users can often fork a problematic class.
- Of course, you can provide some abstract classes.

Use API stability annotations

- A new feature sometimes needs time to mature.
 - Use API stability annotations to balance between velocity & stability.
- Choose what works best for your community.
 - @Beta, @UnstableApi, @InternalApi, @MaturityLevel...
 - Apache Yetus audience annotations
 - @API Guardian
- Don't overuse it, though.

Usual engineering tips

- Prefer immutability.
- Implement `toString()` properly.
- Write awesome Javadoc · documentation.
- Keep your code clean
 - Consistent and beautiful code style
 - Technical debt must be handled on time.
- Automate for less tedium & higher quality:
 - Formatting, linting, static analysis, test coverage, release process, ...

Community growth

Don't give up

- It usually takes (long) time – years, not months
- Dogfood and keep improving, because people don't use when:
 - Stale activity
 - e.g. No releases, commits or site updates in last 6 months
 - Poor metrics
 - e.g. No test coverage, build failures, ...
 - Poor documentation
 - No need to write a book, but should be good enough to get started

Get the most out of interaction

- Set up a place to chat informally.
 - e.g. Slack, Gitter, ...
- Respect and appreciate.
 - Everyone is not good at textual communication.
 - Every single visitor is important especially at the early stage.
 - They could be your first customer or colleague!
- Engage proactively.
 - Create a new issue to detail the reported problem · feature · workaround.
- They will usually be happy to provide feed back, or even a pull request.



trustin  2 months ago

That'd be awesome. Thanks again, [@Tobias](#)



Tobias 2 months ago

It's weird that I'm using a product that you wrote, I'm asking for features that you and your friends/colleagues and other people "on the internet" are implementing, paying in nothing but a little praise and recognition, and you're the one thanking me.. Either way, thank you too for Armeria & Netty



Don't be shy but ask questions

- How did you find us?
- What features do you like about us?
- What is missing? What could be added or improved?
- Are you using it in your product?
 - If so, how? If not, what would help you decide?
- I created an issue for you. Do you have anything to add?
- Are you interested in sending a pull request?

Usual soft skills

- Be thankful.
- Assume good faith.
- Don't take it personally.
- You sometimes need to agree to disagree.
- Humors, emojis and GIFs 🤔😭😱🧑🏻🙏🎉
- Be careful of burnout.

You need a web site

- README .md is not enough.
 - Aesthetics matter!
 - No room for elevator pitch
 - Not always mobile friendly
 - No traffic analysis
- Find the site generator that works best for you.
 - Gatsby, Hugo, Sphinx, Jekyll, ...
 - <https://www.staticgen.com/>

Users

1.3K

↑ 40.0%

Sessions

2.2K

↑ 50.4%

Bounce Rate


48.88%

↓ 17.3%

Session Duration

3m 18s

↑ 6.3%



Armeria
0.99

- Setting up a project
- Writing a server
- Writing a client
- Advanced topics
- Release notes
- API documentation
- Source cross-reference
- Questions and answers
- Fork me on GitHub
- Contributing
- Stars 2.6k
- Follow 610
- Chat on slack
- build passing
- maven-central v0.99.6
- commit activity 56/month
- good first issues 6 open
- coverage 72%

Docs » Welcome to Armeria

[Edit on GitHub](#)

Welcome to Armeria

Armeria is an open-source asynchronous HTTP/2 RPC/REST client/server library built on top of [Java 8](#), [Netty](#), [Thrift](#) and [gRPC](#). Its primary goal is to help engineers build high-performance asynchronous microservices that use HTTP/2 as a session layer protocol.

It is open-sourced and licensed under [Apache License 2.0](#) by [LINE Corporation](#), who uses it in production.

Want a quick tour?

Check out [the introductory slides](#):



Armeria
A Microservice Framework
Well-suited Everywhere

Trustin Lee, LINE
Oct 2019

[@armeria_project](#) [LINE/armeria](#) [Enter fullscreen](#)

Features

HTTP/2

- Supports HTTP/2 on both TLS and cleartext connections
- Supports protocol upgrade via both [HTTP/2 connection preface](#) and [traditional HTTP/1 upgrade request](#)
- Fully compatible with existing HTTP/1 servers
- Integrated [PROXY protocol](#) support for interoperability with load balancers such as [HAProxy](#) and [AWS ELB](#).

 Armeria

[News](#) [Documentation](#) [Community](#) [Feedback](#) [Twitter](#)

Build a reactive microservice at your pace, not theirs.

Armeria is your go-to microservice framework for any situation. You can build any type of microservice leveraging your favorite technologies, including gRPC, Thrift, Kotlin, Retrofit, Reactive Streams, Spring Boot and Dropwizard.

* Brought to you by the creator of [Netty](#) and his colleagues at [LINE](#) *

[Learn more](#)

[Community](#)

gRPC, Thrift, REST, static files? You name it. We serve them all.

Let's embrace the reality – we almost always have to deal with more than one protocol. It was once Thrift, today it's gRPC, and REST never gets old. At the same time, you sometimes have to handle health check requests from a load balancer or even serve some static files.

Armeria is capable of running services using different protocols, all on a single port. No need for any proxies or sidecars. Enjoy the reduced complexity and points of failure, even when migrating between protocols!

```
Server
.builder()
.service(
"/hello",
(ctx, req) -> HttpResponse.of("Hello!"))
.service(GrpcService
.builder()
.addService(myGrpcServiceImpl)
.build())
.service(
"/api/thrift",
ThriftService.of(myThriftServiceImpl));
.service(
"prefix:/files",
FileService.of(new File("/var/www")))
.service(
"/monitor/l7check",
HealthCheckService.of())
.build()
.start();
```

Keep your eyes on “Referer” logs

- Who wrote about us?
- Who is using us?
- Engage!
 - Let users know we care and appreciate.
 - Update the ‘community resources’ page.
 - Ask questions.
 - Tweet about it.
 - Propose a better way to use.



Trustin Lee @trustin · May 2

@JaapCoomans Hi! Just found your "The ultimate microframework smackdown" and wanted to tell you @armeria_project is 1) based on @netty_project 2) supports both annotations and programmatic model. Thanks a lot for mentioning Armeria! Any other comments you wanted to say?



Trustin Lee @trustin · May 2

Replying to @trustin

I'm especially interested in what improvements could be made to make Armeria be one of the final top list. 😊



Jaap Coomans @JaapCoomans · May 2

Replying to @trustin

Thanks for reaching out. That's awesome! It's been a while, but I recall I found the programmatic model fairly limited. The focus seems to be on annotations. Also the anonymous inner classes in the docs are not my style.



Jaap Coomans @JaapCoomans · May 2

An improvement wrt the annotated model would be to automatically respond with 404 when a handler returns null or an empty Optional.

Furthermore greater adoption would have scored some extra points, but that's not something you can control directly.



Armeria Official Tweets (Please Retweet!) APP

Jun 1st at 10:19 PM

https://twitter.com/armeria_project/status/1267445703652937728

Armeria @armeria_project

+ = Baremaps, a vector tile server and pipeline for producing @Mapbox vector tiles from @openstreetmap and other data sources, built with #Armeria by @bchapuis et al <https://t.co/Mwml2qyUha>
Tutorial: <https://t.co/67y0FAAI82>

Twitter | Jun 1st



2 replies



Bertil Chapuis 27 days ago

Thanks a lot for the tweet 😊 I'm now in a rush to release the new version 😊 In the near future, the plan is to develop additional webservices based on armeria-grpc for localization, search, routing, etc. Thank you for your work! (edited)



Trustin Lee 26 days ago

Awesome! Looking forward to your feed back



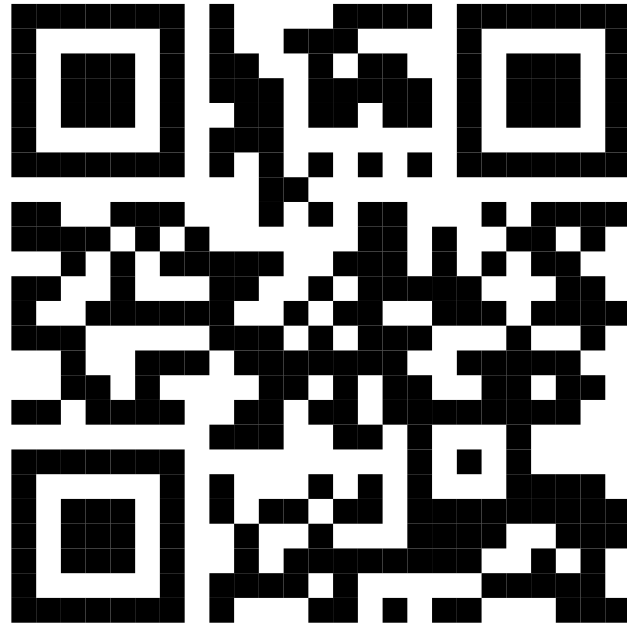
Prepare a good dev guide

- Today's user can be tomorrow's dev in a library.
- Spend your time for good onboarding experience.
 - Build requirements & How to build
 - How to set up with IDE
 - Conventions, e.g. Checkstyle rules
 - CLA (Contributor License Agreement) – <https://cla-assistant.io/>
- See <https://armeria.dev/community/developer-guide> for an example.

If you have more time...

- Speak in conferences.
 - <https://www.cfpland.com/>
- Host contributor events.
 - e.g. Open source coding sprint
 - Try with your colleagues at work first.
- Watch and learn, or let's work together!
 - <https://github.com/line/armeria>
 - <https://github.com/netty/netty>

Meet us at GitHub



ARMERIA.dev
github.com/line/armeria

 @trustin