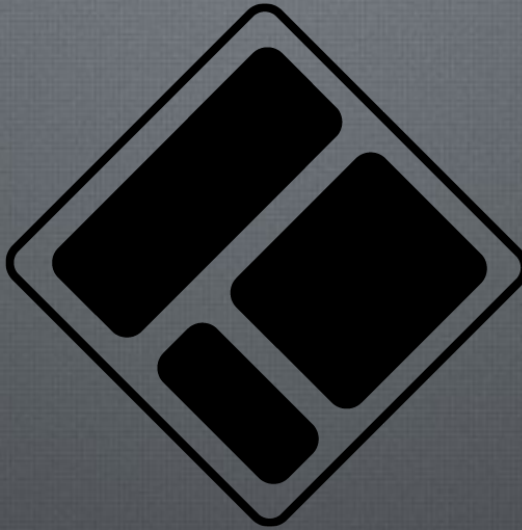


# JARS



The JavaScript RESTful Specification

# What is JARS?



JARS stands for  
**JavaScript API for RESTful Services.**

It is a working draft specification project which aims to provide a standard REST API to JEC.

- easy-to-use
- built for maintainability
- container independent
- built on top of the TypeScript decorators specification
- asynchronous and non-blocking

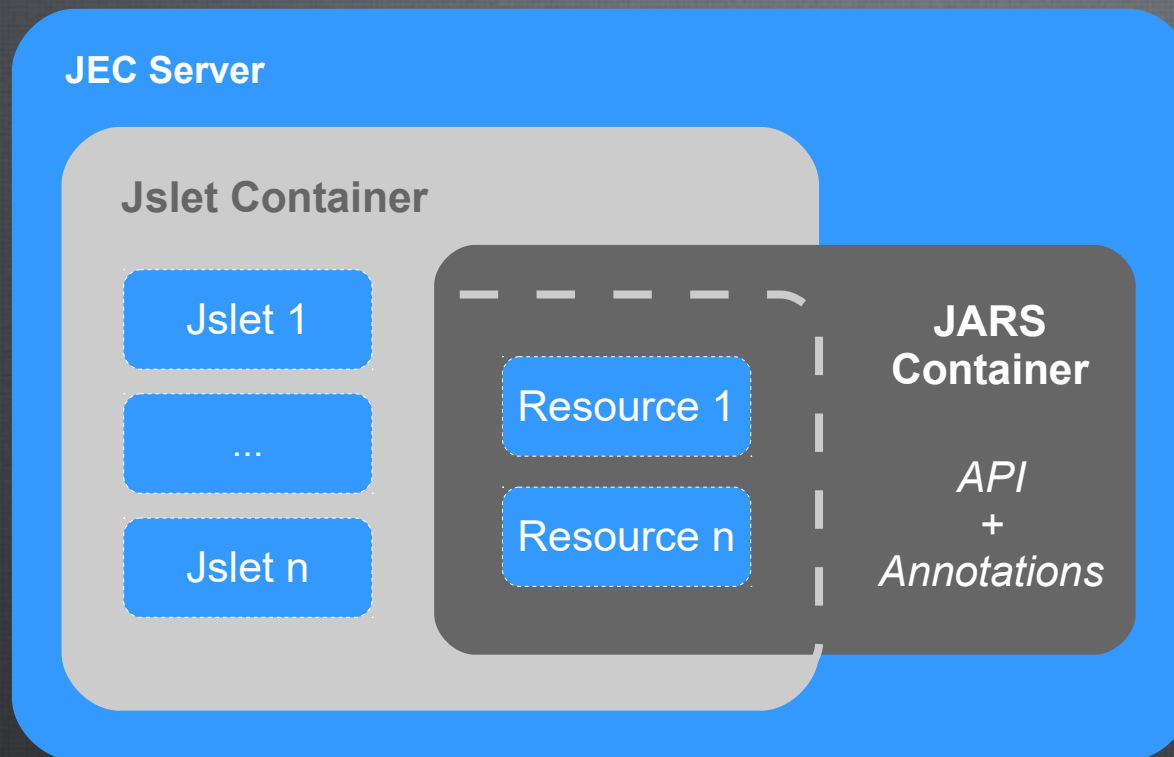
The Sandcat framework is the  
GlassCat default implementation of JARS.

# JARS Architecture



JARS is built over the JEC **jslet API**.

- it uses POJOs<sup>[1]</sup> to implement the corresponding Web resources
- it provides support for asynchronous and non-blocking I/O



[1] Plain Old JavaScript Objects

# The Modular Approach



Like all specifications that are part of the JCE Project, JARS is based on modular approach:

JARS implementations can be used independently of any container.

*Sandcat initialization script for autowiring support:*

```
import { BootstrapScript, Bootstrap, JecContainer } from "jec-commons";
import { SandcatBuilder } from "jec-sandcat";

@Bootstrap()
export class InitApp extends BootstrapScript {

  public run(container:JecContainer):void {
    new SandcatBuilder().build(container)
                        .process((err:any)=>{});
  }
}
```



# List of JARS Annotations



| Annotation   | Target | Description  |
|--|--------|--|
| <code>@GET, @HEAD, @POST, @PUT, @DELETE, @CONNECT, @OPTIONS, @TRACE</code> | Method | Represent the HTTP requests that can be handled by a method.                                   |
| <code>@ResourcePath</code>   | Type   | Specifies a relative path for a resource   |
| <code>@Exit</code>   | Field  | The reference to the callback function for non-blocking support                                |
| <code>@Init, @Destroy</code>   | Method | Provides access to the jslet initialization API.   |
| <code>@PathParam</code>  | Field  | The value of a method parameter, extracted from the URI paths.                                 |
| <code>@QueryParam</code>   | Field  | The value of a method parameter, extracted from the URI query parameters.                      |
| <code>@RequestParam</code>   | Field  | Provides access to the current HTTP request.   |
| <code>@RequestBody</code>  | Field  | Provides access to the body of the current HTTP request.                                       |
| <code>@RootPath</code>   | Type   | Specifies the resource-wide version path that forms the base URI of a set of resource classes. |
| <code>@RootPathRefs</code>   | Type   | Specifies the reference to all of the root paths for a resource class.                         |
| <code>@CookieParam</code>  | Field  | The value of a method parameter, extracted from the cookies.                                   |



JARS API is highly intuitive to learn and use.

It provides support for sub-routing, parameters extraction and MIME types treatment:

```
import { ResourcePath, GET, PathParam, Exit } from "jec-jars";

@ResourcePath("/hello")
export class Hello {

  @GET()
  public sayHelloWorld(@Exit exit:Function):void {
    exit("Hello World!");
  }

  @GET({
    route: "/*:username"
  })
  public sayHello(@PathParam username:string, @Exit exit:Function):void {
    exit(`Hello ${username}!`);
  }
}
```

# API Versioning



JARS provides support for REST APIs URL versioning.

*URL versioning declaration:*

```
@RootPath({
  path: "/versioned.api",
  ref: "v2.0",
  version: {
    prefix: "v",
    major: 2,
    minor: 0
  }
})
export class VersionedSampleApi_v_2_0 {}
```

*API references declaration:*

```
@ResourcePath("/search")
@RootPathRefs(["v1.0", "v2.0"])
export class Search {}
```

*Client usage:*

```
http://mydomain.com/myservices/versioned.api/v1.0/search
http://mydomain.com/myservices/versioned.api/v2.0/search
```

# Accessing Request Properties



JARS annotations allow access to all parts of a HTTP request and easily combine them with any parameter:

```
import { HttpRequest } from "jec-exchange";
import { ResourcePath, GET, RequestParam, PathParam, RequestBody,
      Exit } from "jec-jars";

@ResourcePath("/context")
export class Context {

  @GET({
    route: "/header-params/:param"
  })
  public getHeader(@Exit exit:Function, @RequestParam request:HttpRequest,
                  @PathParam param:string):void {
    exit(param + ": " + request.getHeader(param));
  }

  @GET({ route: "/body" })
  public getHeader(@Exit exit:Function, @RequestBody body:any):void {
    exit("request body: " + JSON.stringify(body));
  }
}
```



# Accessing Query Parameters



JARS query parameter use a name-based implementation to make search engines more easy to implement and maintain:

```
import { ResourcePath, RootPathRefs, GET, QueryParam, Exit } from "jec-jars";

@ResourcePath("/search")
@RootPathRefs(["v1.0", "v2.0"])
export class Context {

  @GET({
    route: "/users"
  })
  public getHeader(@Exit exit:Function, @QueryParam name:any,
                  @QueryParam age:any):void {
    let response:string = "/searching for users with ";
    if(name) response += "name='" + name + "'" + (age ? " and " : "");
    if(age) response += "age='" + age + "' ";
    exit(response);
  }
}
```

# Where to go from here?

