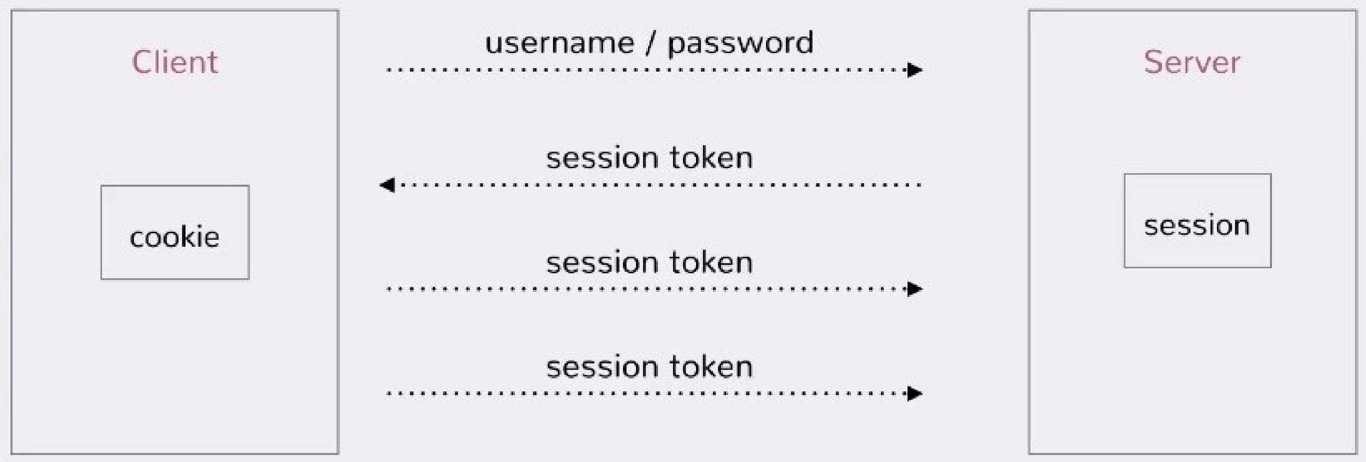
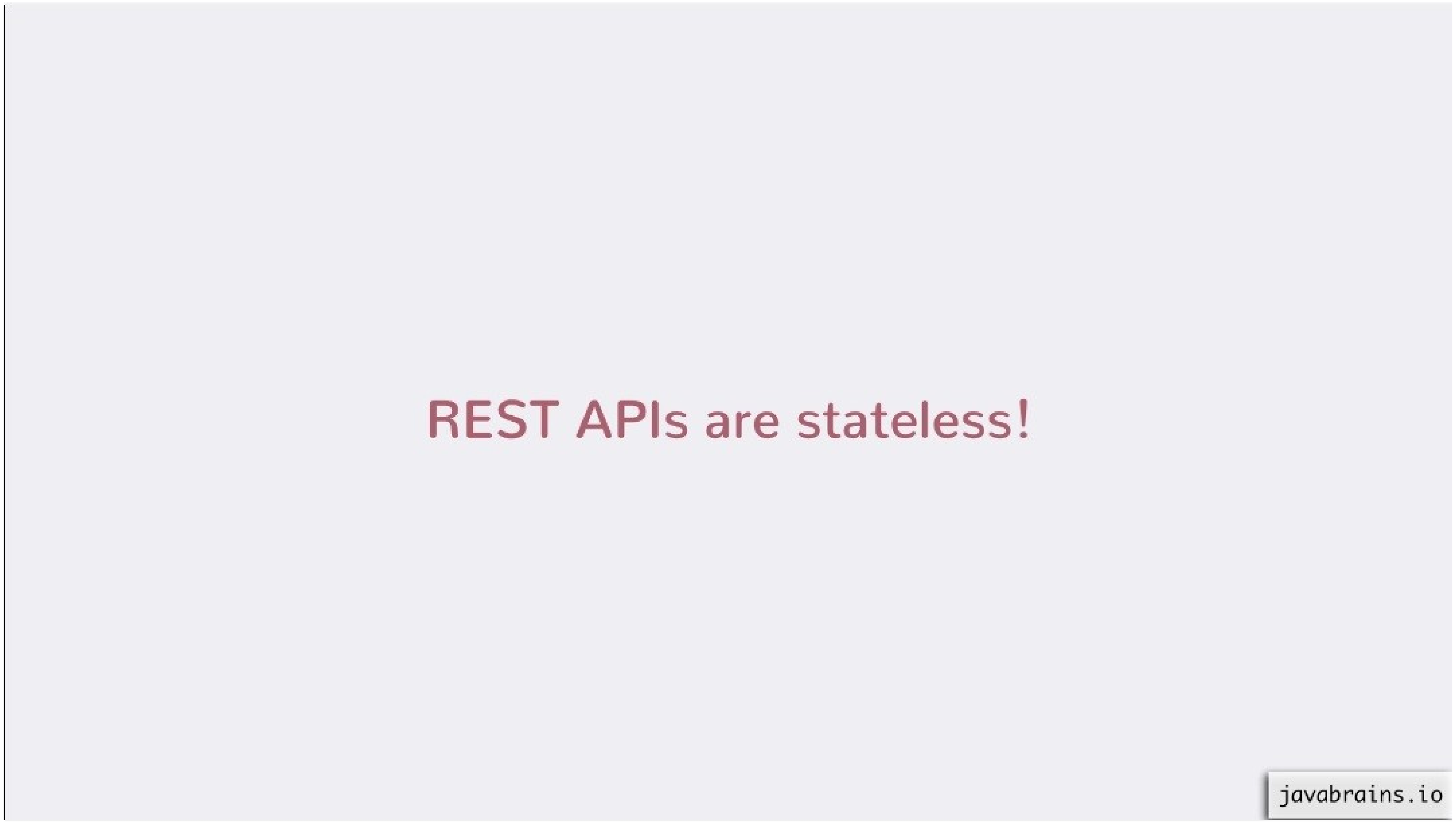
# REST API Authentication

Source: Java Brains
(javaBrains.io)

## Classic session-based authentication

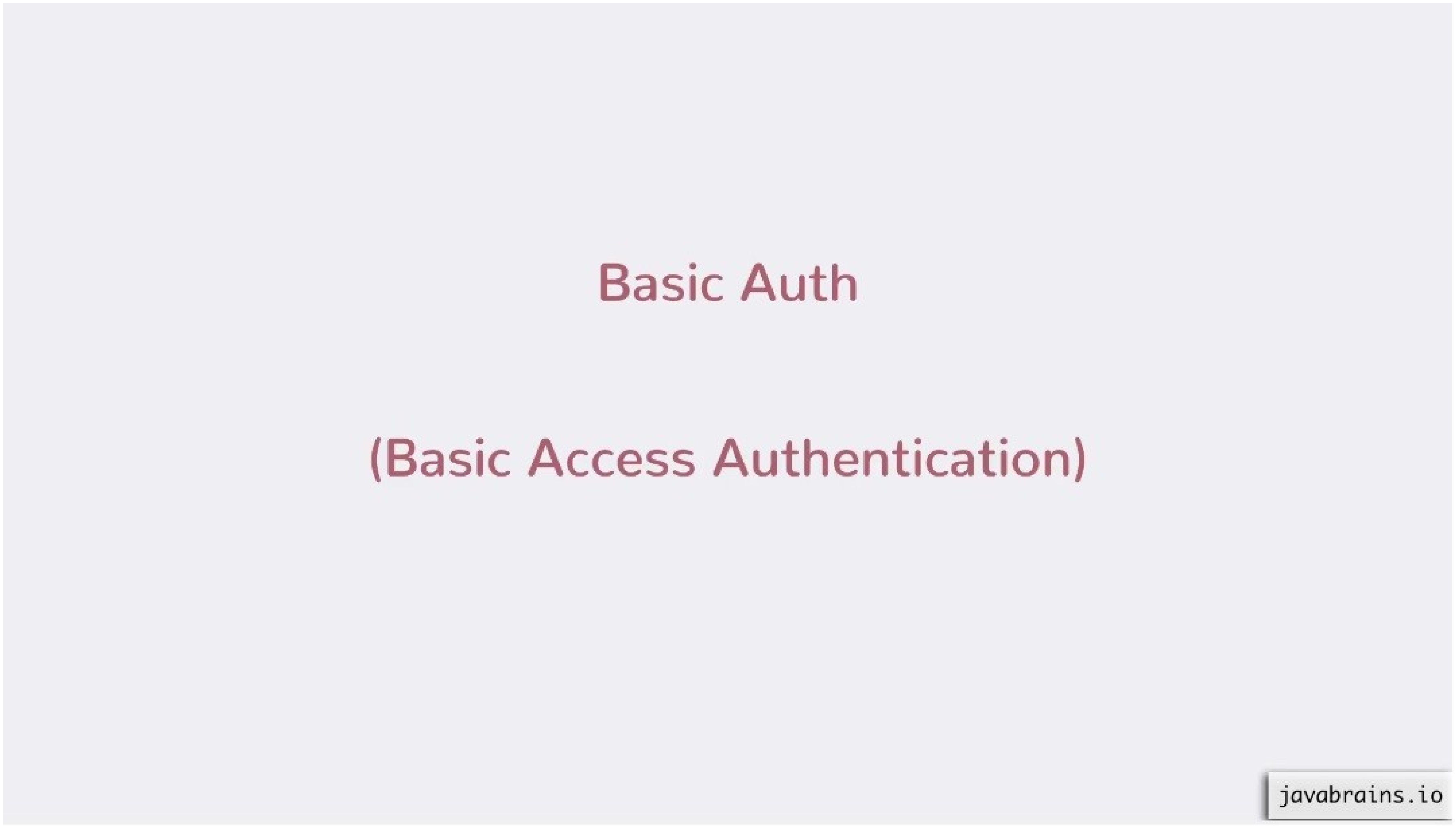


javabrains.io



REST APIs are stateless!

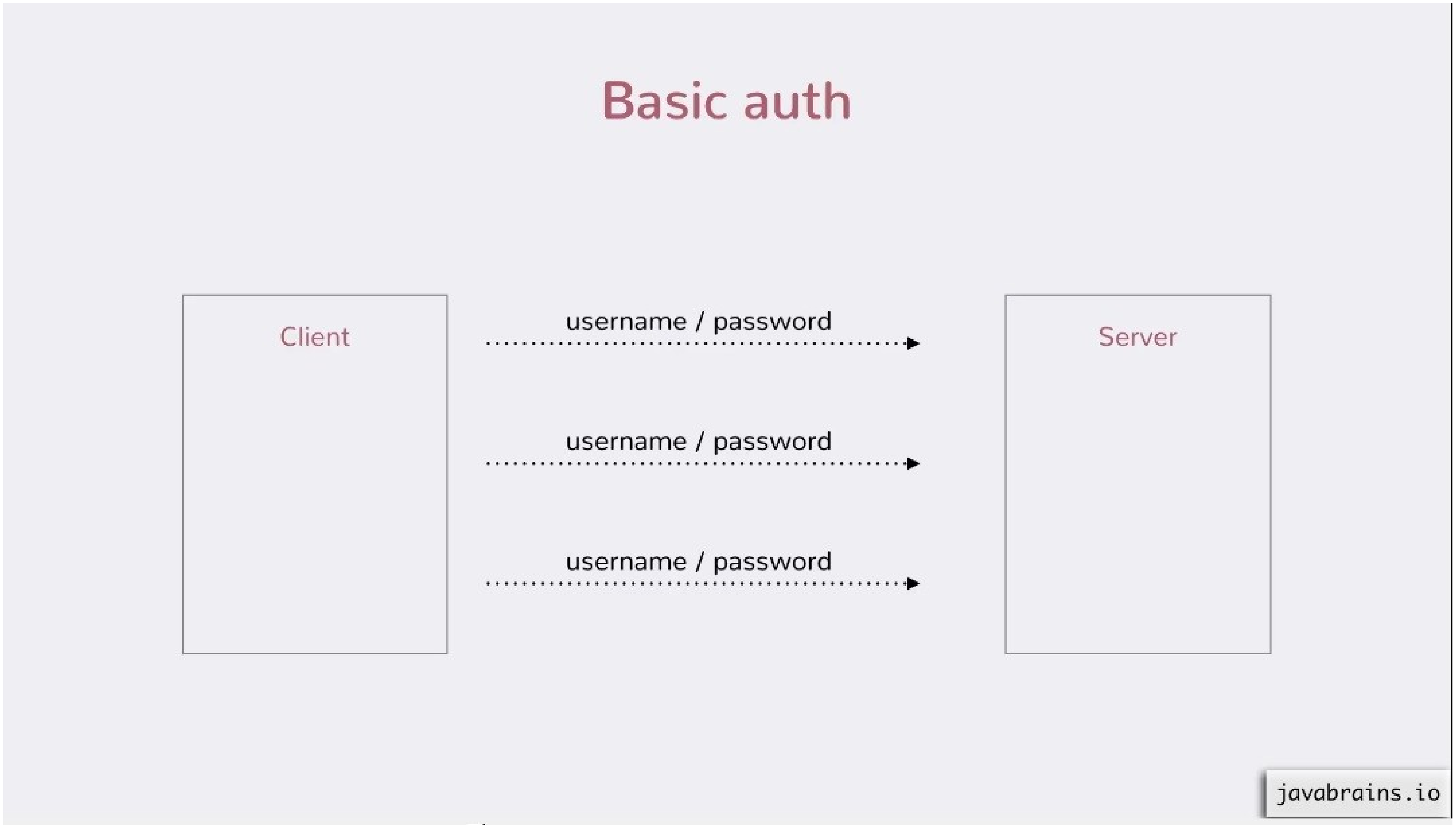
javabrains.io



Basic Auth

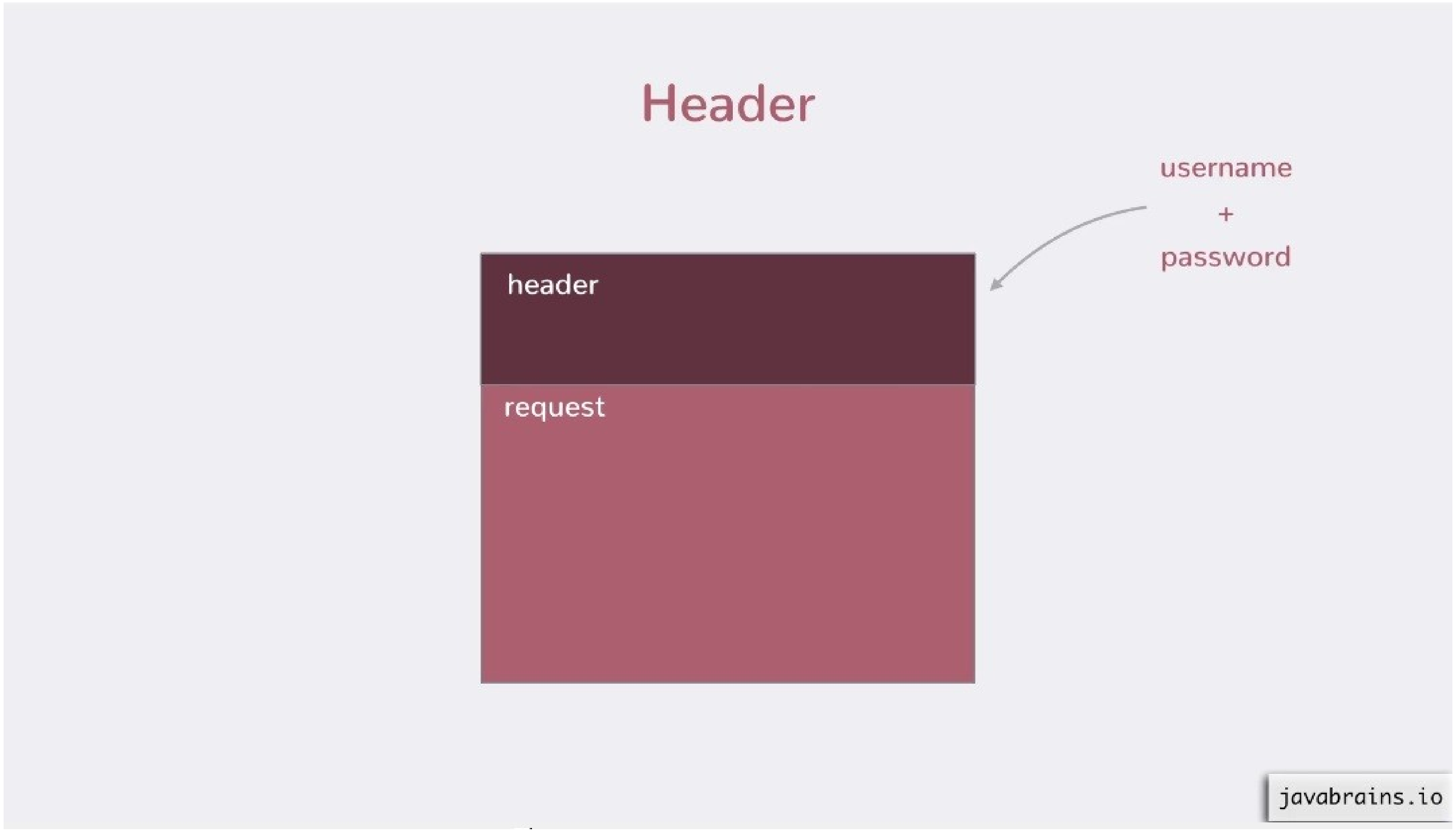
(Basic Access Authentication)

javabrains.io

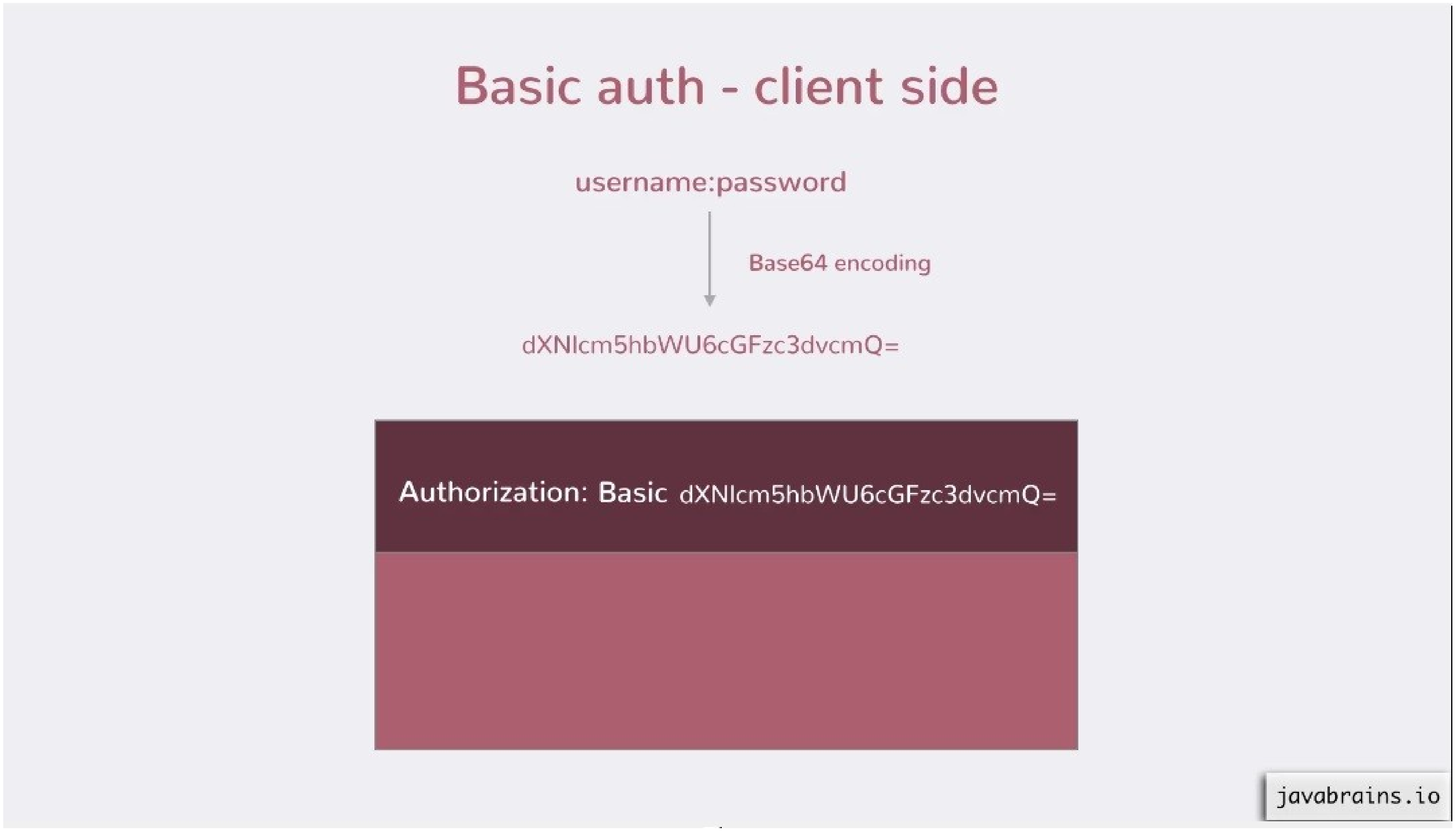


## Basic auth

Client
username / password
username / password
username / password
Server
javabrains.io

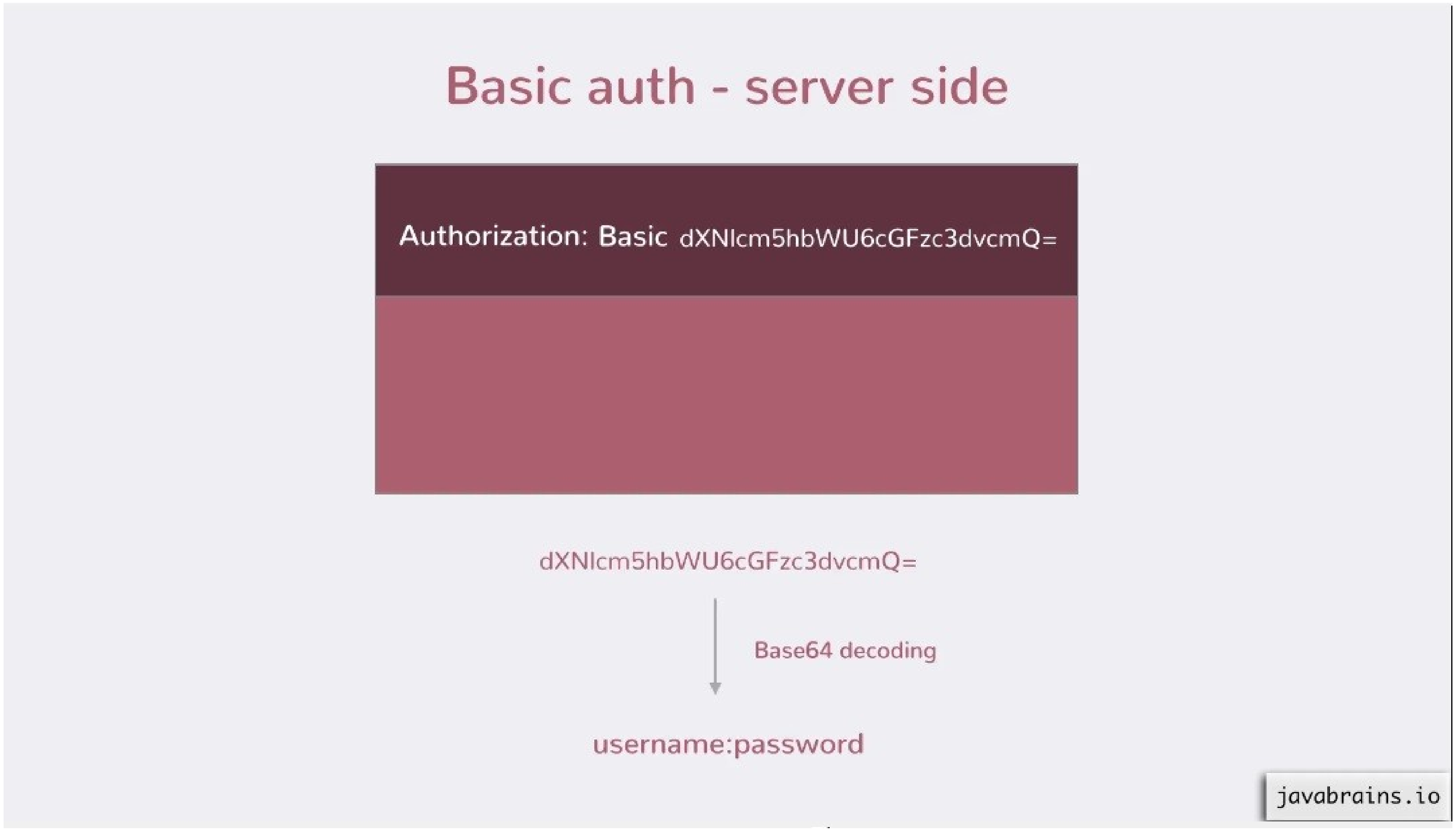


## Header



## Basic auth - client side

```
Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
```



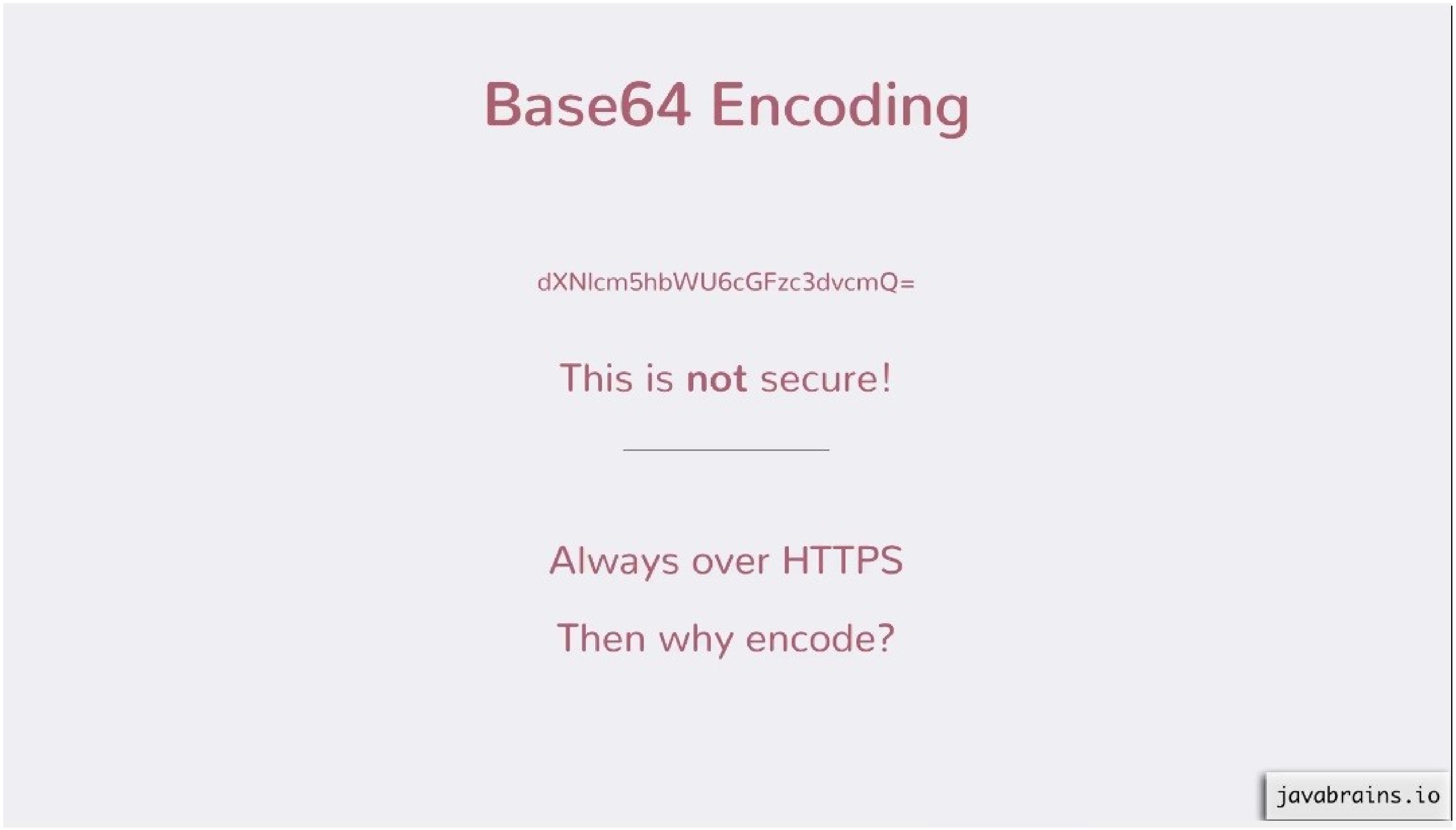
## Basic auth - server side

Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=

dXNlcm5hbWU6cGFzc3dvcmQ=

Base64 decoding

username:password



## Base64 Encoding

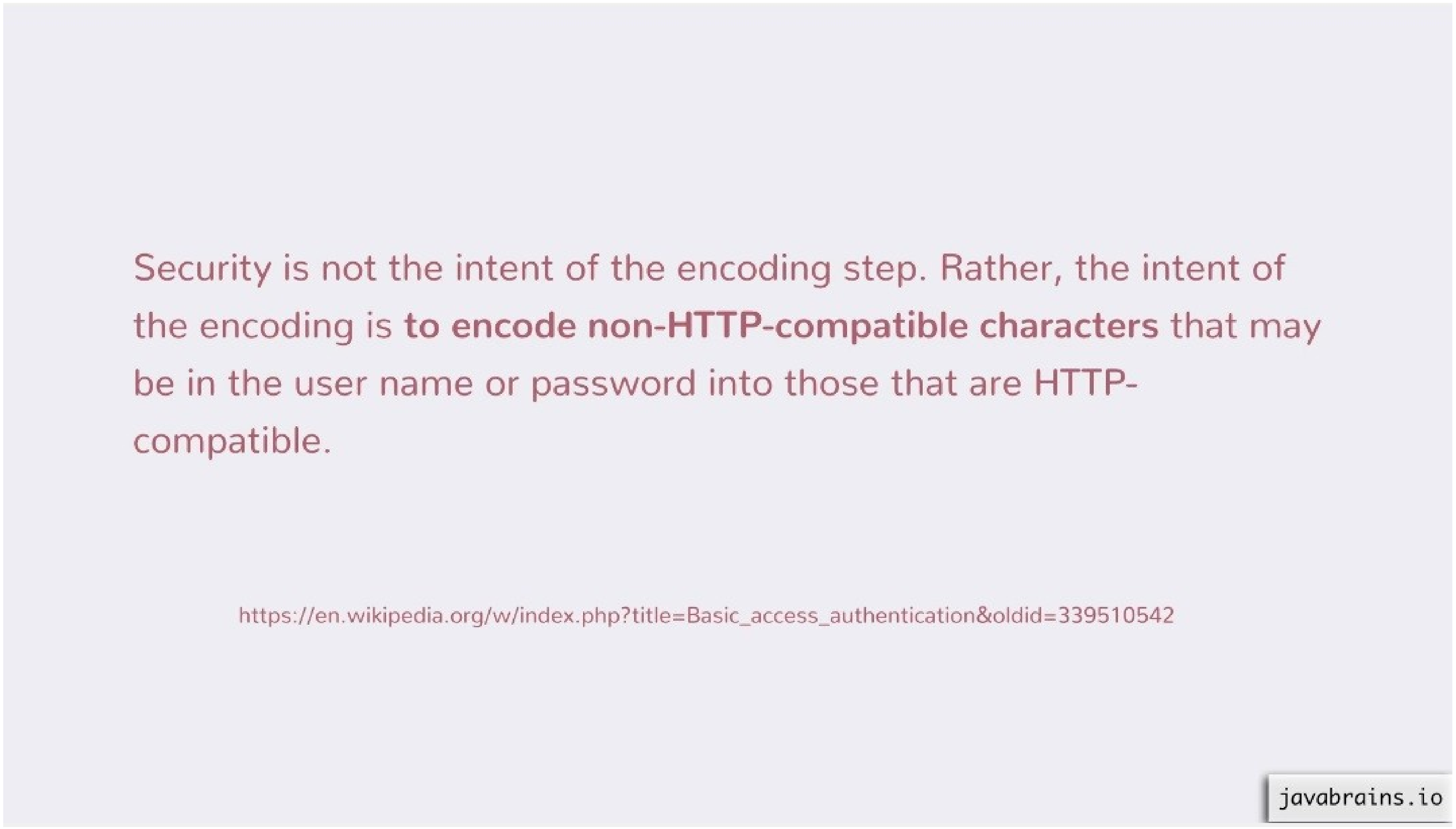
dXNlcm5hbWU6cGFzc3dvcmQ=

This is **not** secure!

Always over HTTPS

Then why encode?

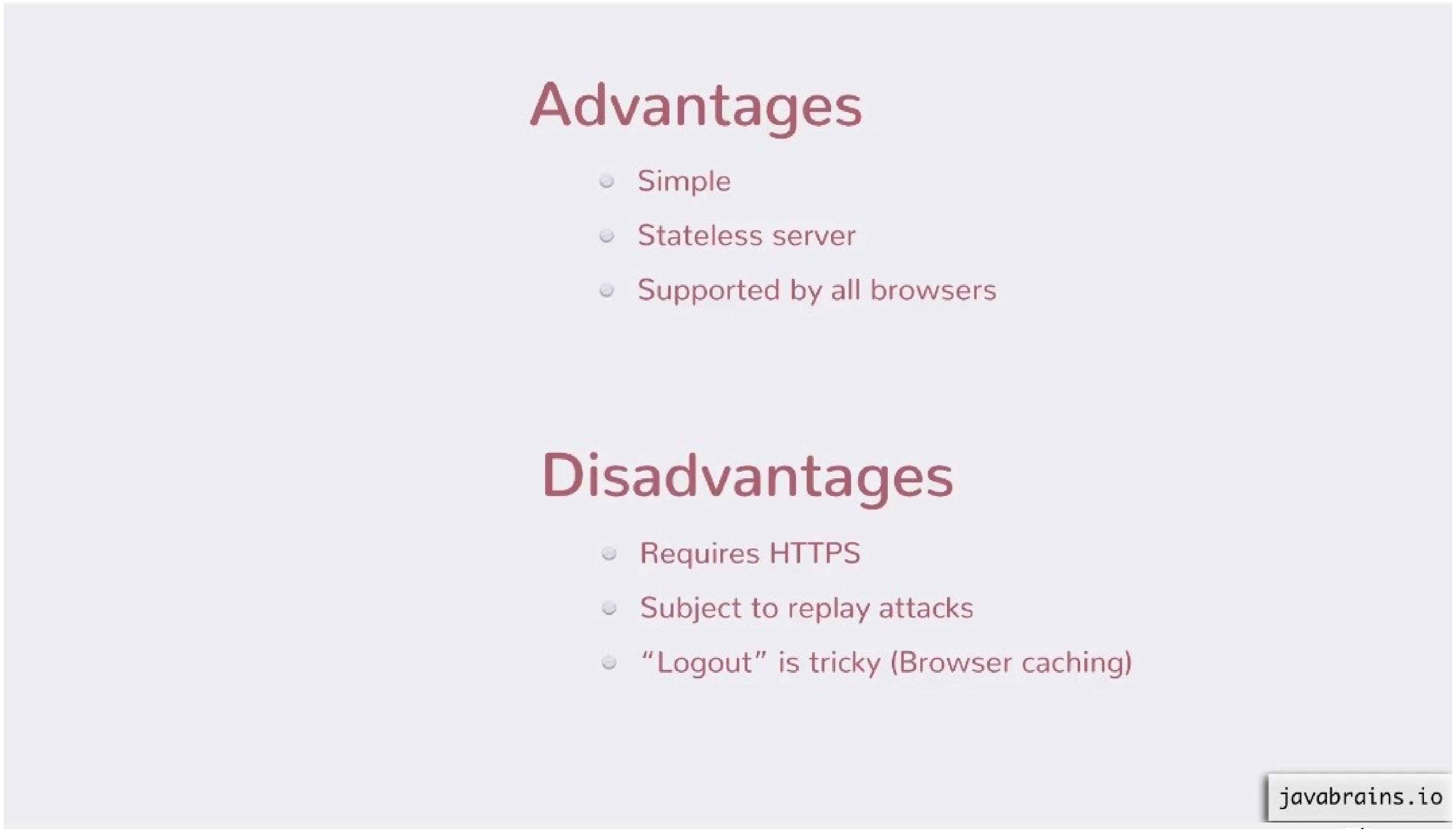
javabrains.io



Security is not the intent of the encoding step. Rather, the intent of the encoding is **to encode non-HTTP-compatible characters** that may be in the user name or password into those that are HTTP-compatible.

https://en.wikipedia.org/w/index.php?title=Basic_access_authentication&oldid=339510542

javabrains.io



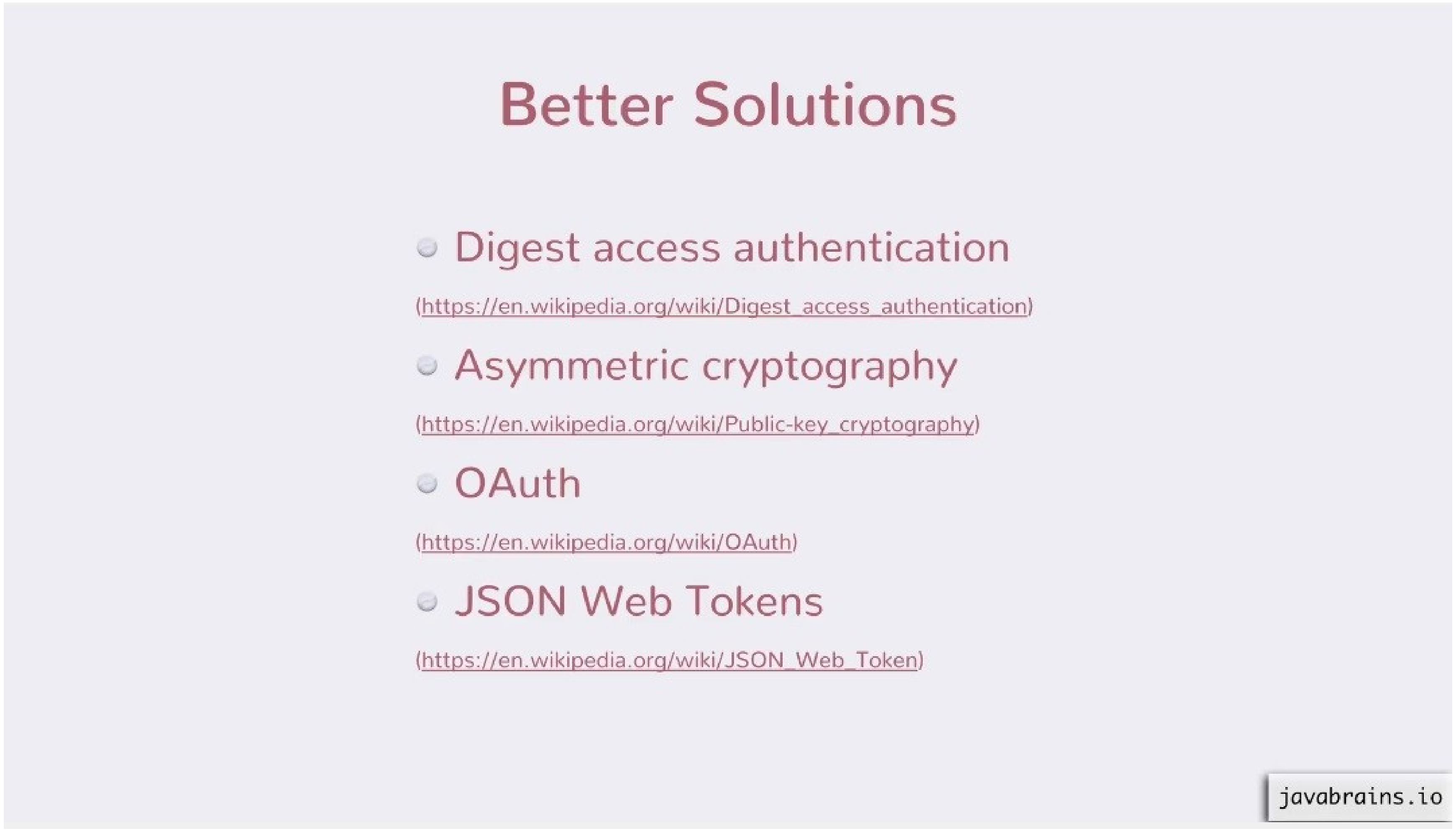
## Advantages

- Simple
- Stateless server
- Supported by all browsers

## Disadvantages

- Requires HTTPS
- Subject to replay attacks
- “Logout” is tricky (Browser caching)

javabrains.io



## Better Solutions

- Digest access authentication

(https://en.wikipedia.org/wiki/Digest_access_authentication)

- Asymmetric cryptography

(https://en.wikipedia.org/wiki/Public-key_cryptography)

- OAuth

(https://en.wikipedia.org/wiki/OAuth)

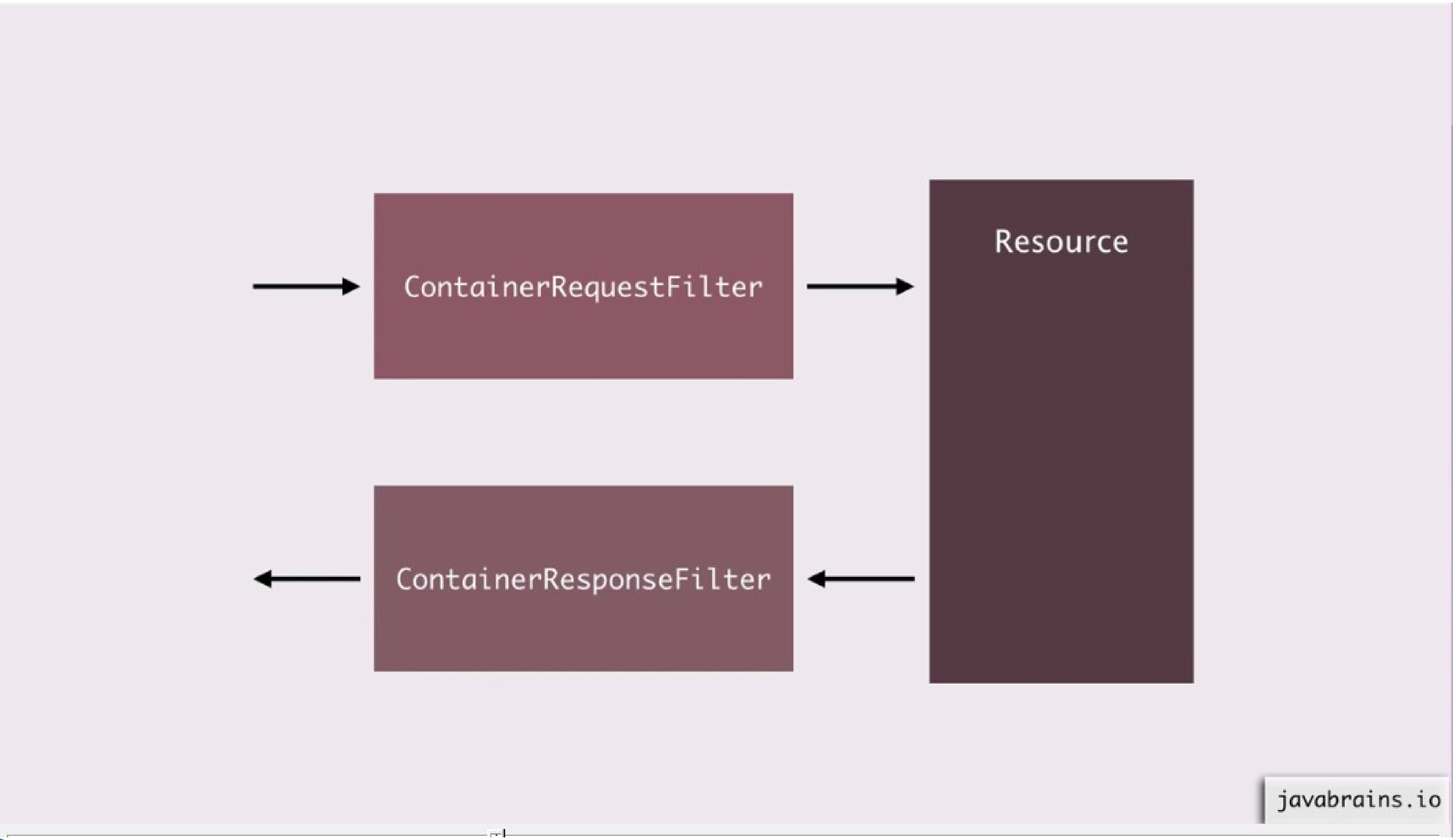
- JSON Web Tokens

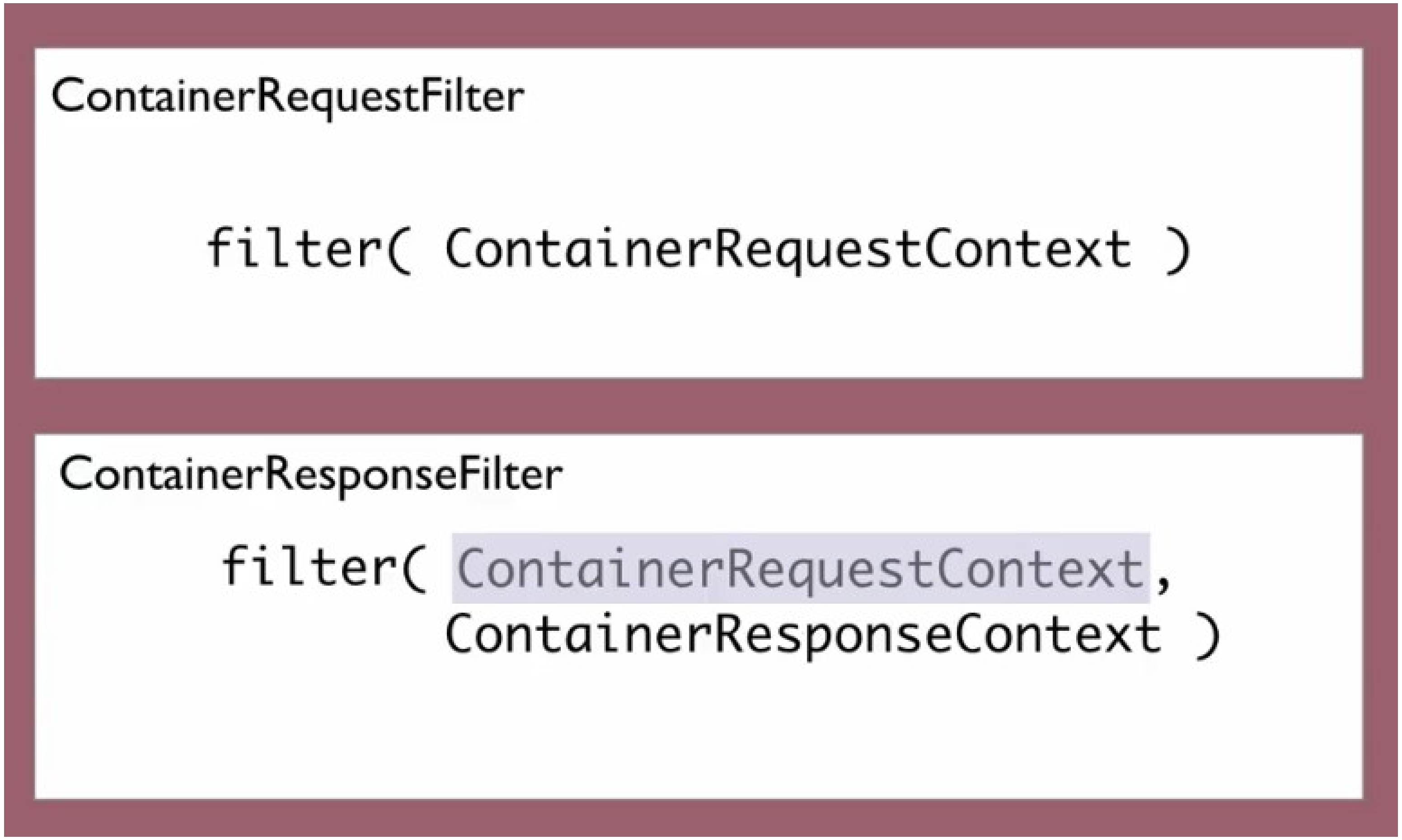
(https://en.wikipedia.org/wiki/JSON_Web_Token)

javabrains.io

## Filters and Interceptors

Source: Java Brains
(javaBrains.io)


ContainerRequestFilter
Resource
ContainerResponseFilter
javabrains.io

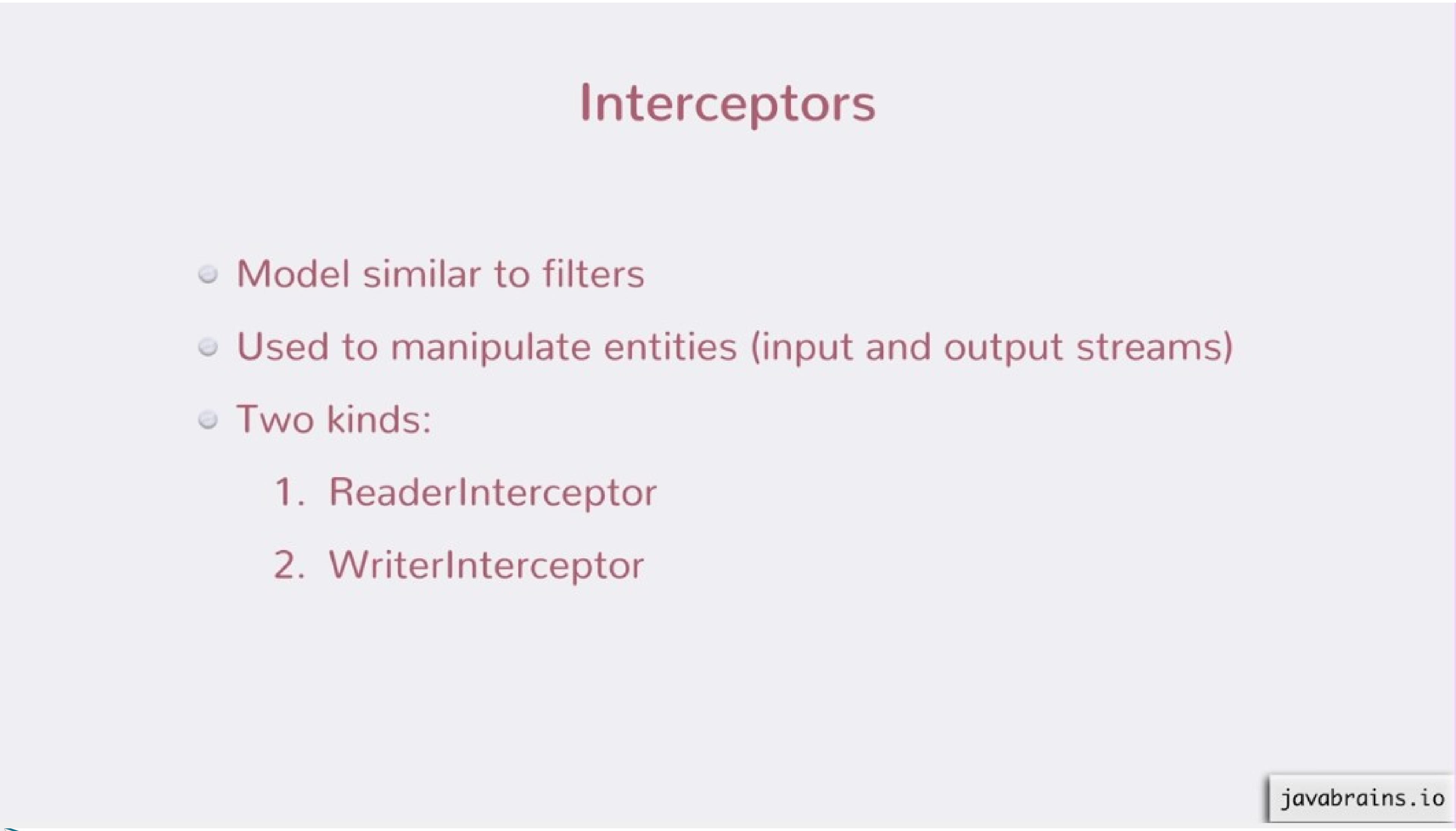


ContainerRequestFilter

```
filter( ContainerRequestContext )
```

ContainerResponseFilter

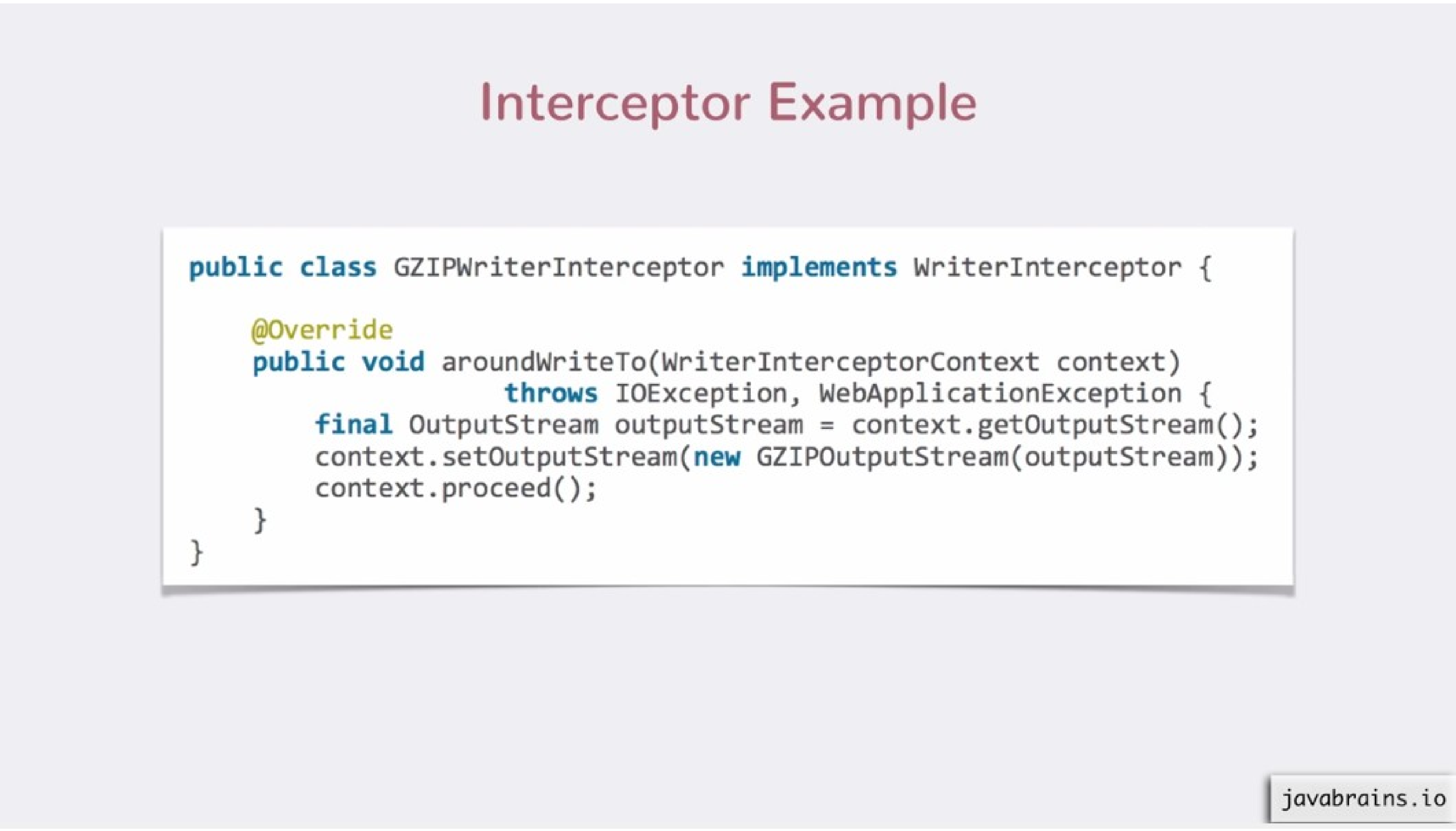
```
filter( ContainerRequestContext,
        ContainerResponseContext )
```



## Interceptors

- Model similar to filters
- Used to manipulate entities (input and output streams)
- Two kinds:
  1. ReaderInterceptor
  2. WriterInterceptor

javabrains.io

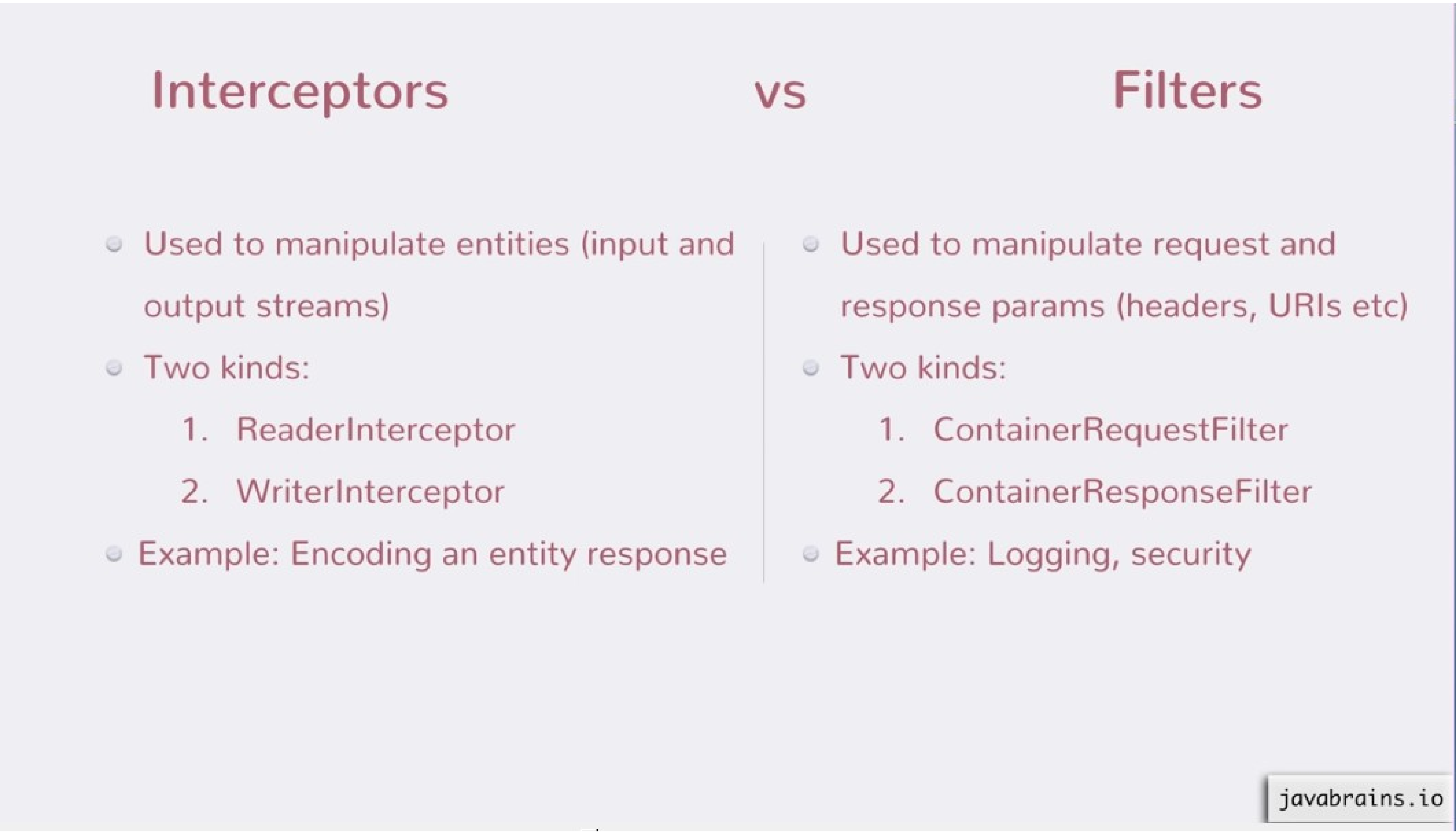


## Interceptor Example

```java
public class GZIPWriterInterceptor implements WriterInterceptor {

    @Override
    public void aroundWriteTo(WriterInterceptorContext context)
                    throws IOException, WebApplicationException {
        final OutputStream outputStream = context.getOutputStream();
        context.setOutputStream(new GZIPOutputStream(outputStream));
        context.proceed();
    }
}
```

javabrains.io



## Interceptors vs Filters

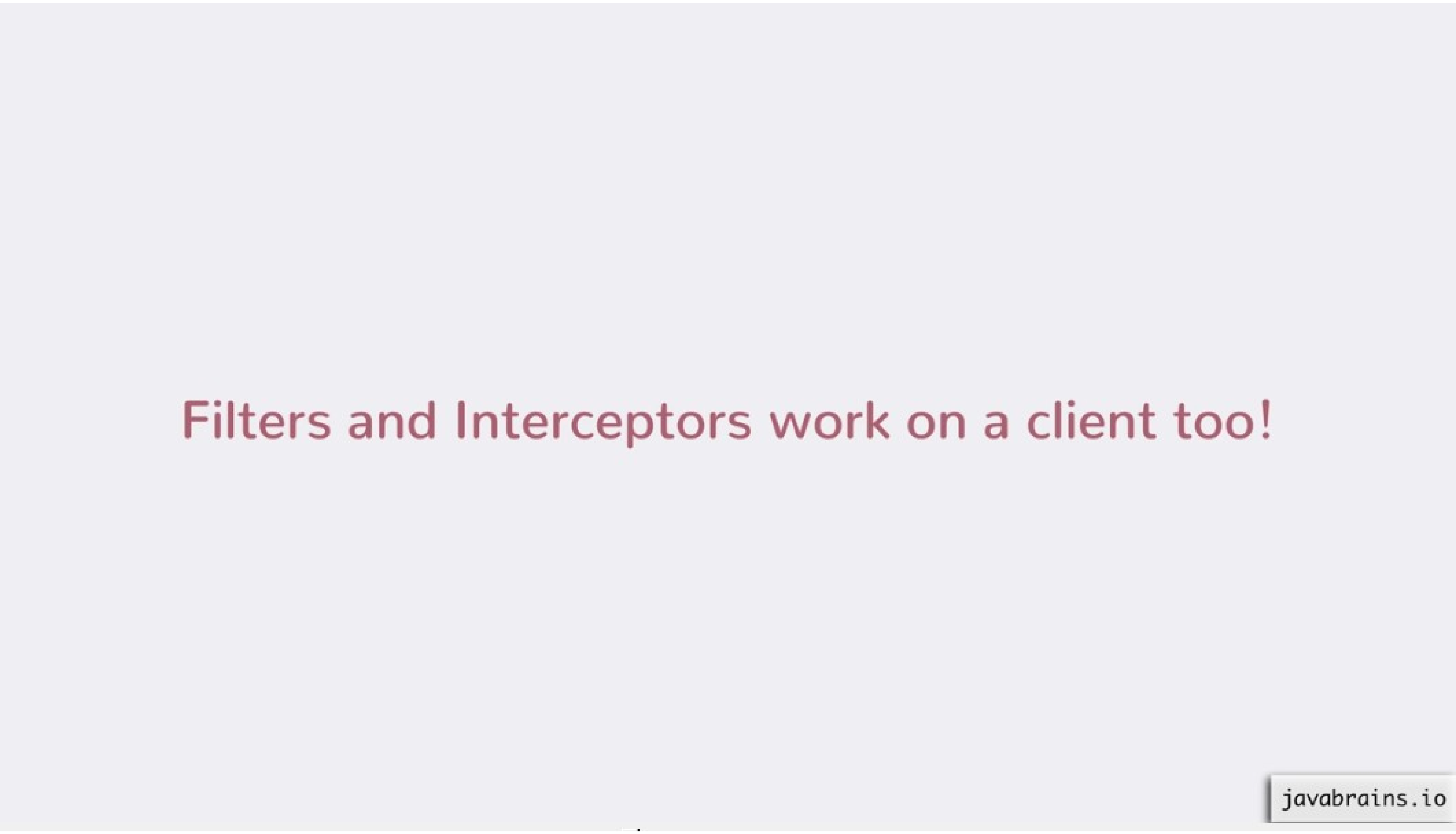
**Interceptors**

- Used to manipulate entities (input and output streams)
- Two kinds:
  1. ReaderInterceptor
  2. WriterInterceptor
- Example: Encoding an entity response

**Filters**

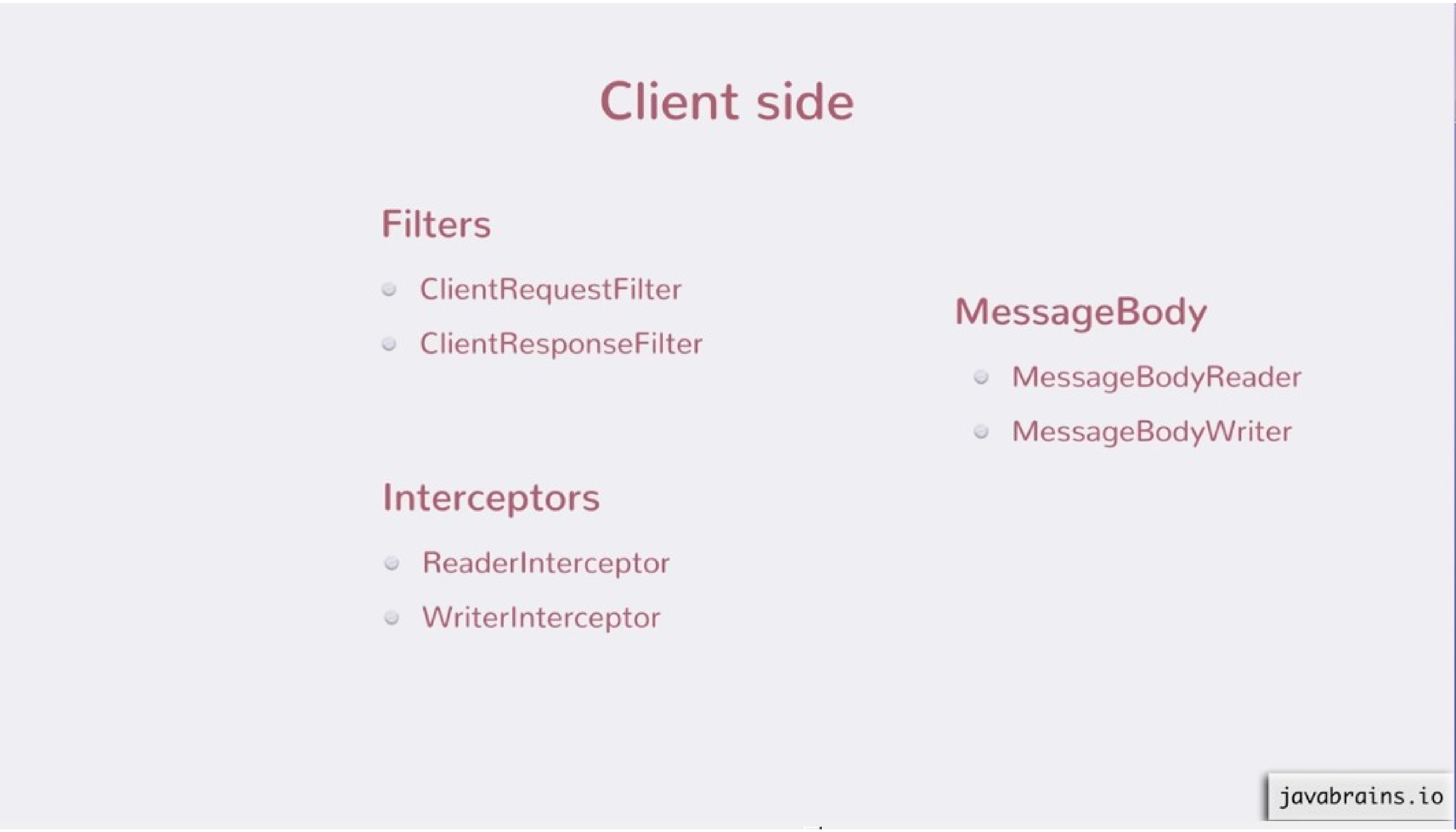
- Used to manipulate request and response params (headers, URIs etc)
- Two kinds:
  1. ContainerRequestFilter
  2. ContainerResponseFilter
- Example: Logging, security

javabrains.io



Filters and Interceptors work on a client too!

javabrains.io



## Client side

### Filters

- ClientRequestFilter
- ClientResponseFilter

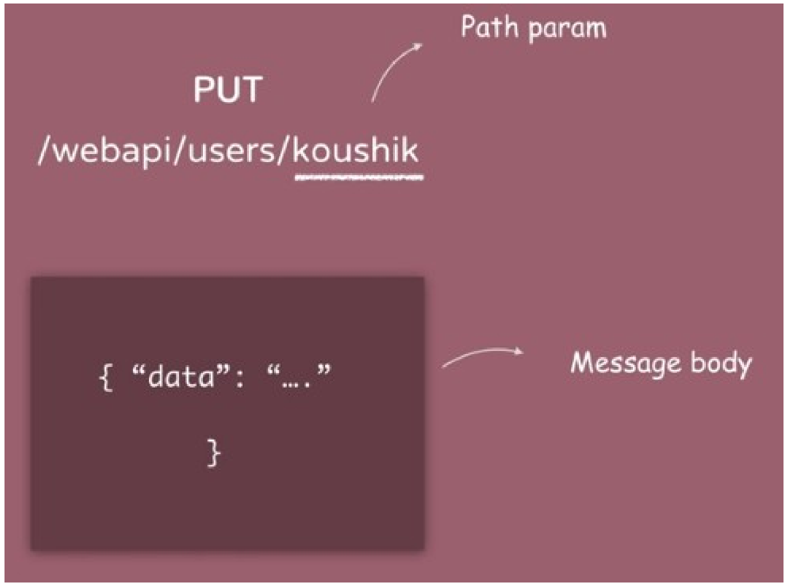
### Interceptors

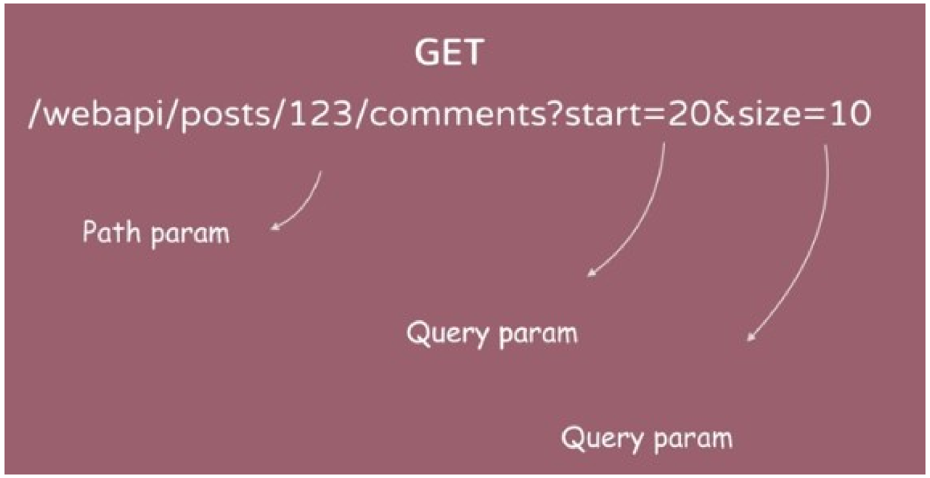
- ReaderInterceptor
- WriterInterceptor

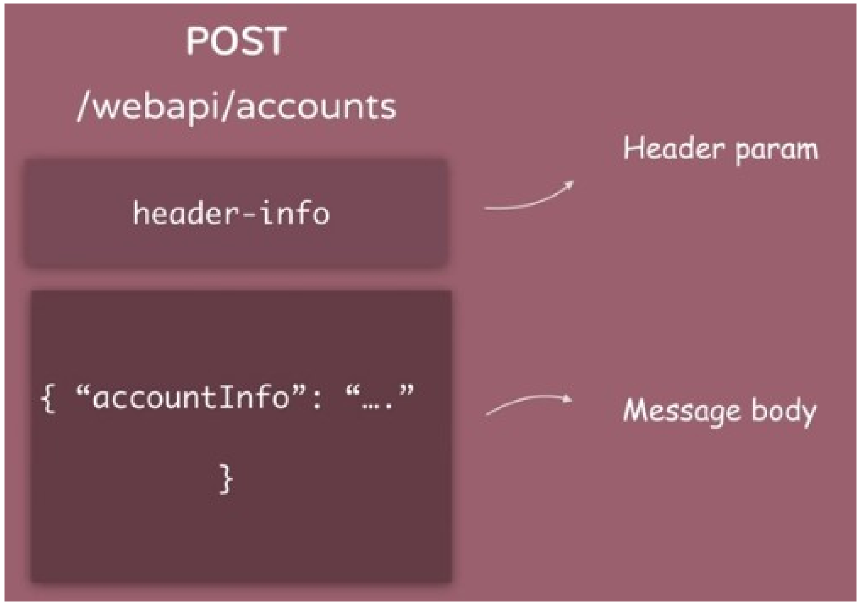
### MessageBody

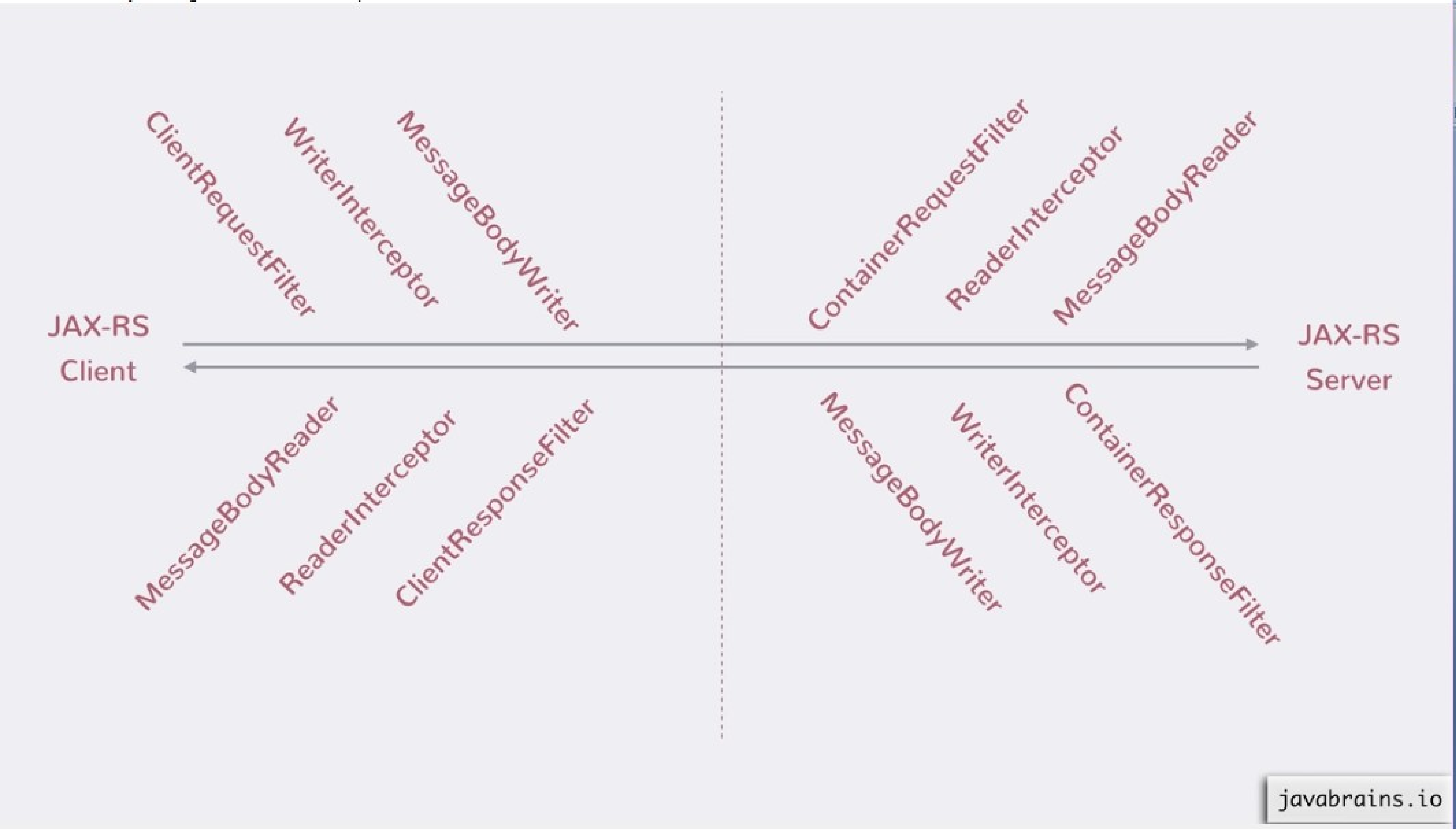
- MessageBodyReader
- MessageBodyWriter

javabrains.io


PUT
/webapi/users/koushik
Path param
{ "data": "…."
}
Message body


GET
/webapi/posts/123/comments?start=20&size=10
Path param
Query param
Query param


POST
/webapi/accounts
header-info
Header param
{ "accountInfo": "…."
}
Message body


ClientRequestFilter
WriterInterceptor
MessageBodyWriter
ContainerRequestFilter
ReaderInterceptor
MessageBodyReader
JAX-RS Client
JAX-RS Server
MessageBodyReader
ReaderInterceptor
ClientResponseFilter
MessageBodyWriter
WriterInterceptor
ContainerResponseFilter
javabrains.io