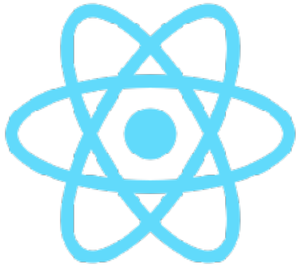
```
Dataloader
|> uncover()
|> apply()
```

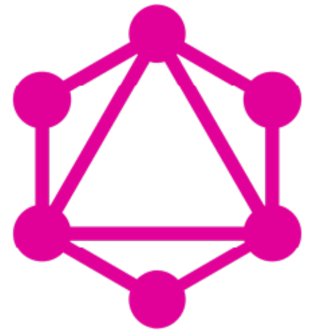
derrick.zhang@letote.cn

About me

```
me = %{
  name: "Derrick Zhang",
  company: "LE TOTE",
  languages: [
    "Ruby",
    "Node.js",
    "Elixir"
  ],
  links: %{
    "WeChat ID": "zillou",
    "GitHub": "https://github.com/zillou"
  }
}
```

LE TOTE IS POWERED BY:





Ruby

Describe your data

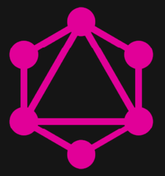
```
type Post {
  title: String!
  content: String!
  comments: [Comment]
}
```

Ask for what you want

```
{
  posts {
    title
    content
    comments {
      content
    }
  }
}
```

Get predictable results

```
{
  "data": {
    "posts": [
      {
        "title": "Pattern Matching in Elixir",
        "content": "Elixir is awesome",
        "comments": [
          {
            "content": "PHP 才是最好的编程语言"
          },
          {
            "content": "老子学不动了！"
          }
        ]
      }
    ]
  }
}
```



https://graphql.org/

```
[debug] QUERY OK source="posts" db=4.0ms queue=0.3ms
SELECT p0."id", p0."title", p0."content", p0."posted_at", p0."user_id", p0."inserted_at", p0."updated_at" FROM "posts" AS p0 []
[debug] QUERY OK source="users" db=2.0ms
SELECT u0."id", u0."name", u0."inserted_at", u0."updated_at" FROM "users" AS u0 WHERE (u0."id" = $1) [1]
[debug] QUERY OK source="comments" db=1.8ms
SELECT c0."id", c0."content", c0."posted_at", c0."user_id", c0."post_id", c0."inserted_at", c0."updated_at" FROM "comments" AS c0 WHERE (c0."post_id" =
$1) [1]
[debug] QUERY OK source="users" db=0.8ms
SELECT u0."id", u0."name", u0."inserted_at", u0."updated_at" FROM "users" AS u0 WHERE (u0."id" = $1) [3]     😟😟
[debug] QUERY OK source="users" db=2.1ms
SELECT u0."id", u0."name", u0."inserted_at", u0."updated_at" FROM "users" AS u0 WHERE (u0."id" = $1) [3]
[debug] QUERY OK source="users" db=3.2ms queue=0.1ms
SELECT u0."id", u0."name", u0."inserted_at", u0."updated_at" FROM "users" AS u0 WHERE (u0."id" = $1) [2]     😰😰😰
[debug] QUERY OK source="users" db=1.2ms
SELECT u0."id", u0."name", u0."inserted_at", u0."updated_at" FROM "users" AS u0 WHERE (u0."id" = $1) [2]
[debug] QUERY OK source="users" db=3.6ms decode=0.1ms
SELECT u0."id", u0."name", u0."inserted_at", u0."updated_at" FROM "users" AS u0 WHERE (u0."id" = $1) [2]
[debug] QUERY OK source="comments" db=0.6ms queue=0.4ms
SELECT c0."id", c0."content", c0."posted_at", c0."user_id", c0."post_id", c0."inserted_at", c0."updated_at" FROM "comments" AS c0 WHERE (c0."post_id" =
$1) [2]
[debug] QUERY OK source="users" db=1.4ms queue=0.1ms
SELECT u0."id", u0."name", u0."inserted_at", u0."updated_at" FROM "users" AS u0 WHERE (u0."id" = $1) [1]
[debug] QUERY OK source="users" db=2.6ms
SELECT u0."id", u0."name", u0."inserted_at", u0."updated_at" FROM "users" AS u0 WHERE (u0."id" = $1) [1]
[debug] QUERY OK source="comments" db=1.1ms
SELECT c0."id", c0."content", c0."posted_at", c0."user_id", c0."post_id", c0."inserted_at", c0."updated_at" FROM "comments" AS c0 WHERE (c0."post_id" =
$1) [3]
```

N + 1 problem

N + 1 problem 😟😟😟😟😟😟

Dataloader save the world

uncover(Dataloader)

```elixir
source = Dataloader.Ecto.new(MyApp.Repo)

# setup the loader
loader = Dataloader.new |> Dataloader.add_source(:db, source)

# load some things
loader =
  loader
  |> Dataloader.load(:db, Organization, 1)
  |> Dataloader.load_many(:db, Organization, [4, 9])

# actually retrieve them
loader = Dataloader.run(loader)

# Now we can get whatever values out we want
organizations = Dataloader.get_many(loader, :db, Organization, [1,4])
```

https://github.com/absinthe-graphql/dataloader#usage

```elixir
import Absinthe.Resolution.Helpers, only: [dataloader: 1]

object :author do
  @desc "Author of the post"
  field :posts, list_of(:post), resolve: dataloader(Blog)
end
```

https://hexdocs.pm/absinthe/1.4.13/ecto.html#dataloader

```
[debug] QUERY OK source="posts" db=2.7ms decode=2.2ms
SELECT p0."id", p0."title", p0."content", p0."posted_at", p0."user_id", p0."inserted_at", p0."updated_at" FROM "posts" AS p0 []
[debug] QUERY OK source="users" db=2.0ms
SELECT u0."id", u0."name", u0."inserted_at", u0."updated_at", u0."id" FROM "users" AS u0 WHERE (u0."id" = ANY($1)) [[1, 2]]
[debug] QUERY OK source="comments" db=2.1ms queue=0.1ms
SELECT c0."id", c0."content", c0."posted_at", c0."user_id", c0."post_id", c0."inserted_at", c0."updated_at", c0."post_id" FROM "comments" AS c0 WHERE
(c0."post_id" = ANY($1)) ORDER BY c0."post_id" [[3, 2, 1]]
[debug] QUERY OK source="users" db=2.0ms
SELECT u0."id", u0."name", u0."inserted_at", u0."updated_at", u0."id" FROM "users" AS u0 WHERE (u0."id" = ANY($1)) [[1, 2, 3]]
```

Mimic of Dataloader’s KV source

```elixir
test "load/2 puts the load key to the queue" do
  loader = Loader.load(loader, 1)
  assert MapSet.member?(loader.queue, 1)
end

test "run/1 executes the load function and caches the result" do
  loader = loader
           |> Loader.load(1)
           |> Loader.load(2)
           |> Loader.run()

  assert loader.results == %{ 1 => 1, 2 => 2 }
end

test "get/2 reads result from the cache" do
  loader = loader
           |> Loader.load(1)
           |> Loader.load(2)
           |> Loader.run()

  assert Loader.get(loader, 1) == 1
end
```

```elixir
defmodule BlogWeb.Loader do
  alias __MODULE__

  defstruct [:load_function, queue: MapSet.new(), results: %{}]

  def new(load_function) do
    %Loader{load_function: load_function}
  end

  def load(loader, key) do
   update_in(loader.queue, fn queue ->
      MapSet.put(queue, key)
    end)
  end

  def run(loader) do
   %{ loader |
      queue: MapSet.new(),
      results: loader.load_function.(loader.queue)
      }
  end

  def get(loader, key) do
    Map.get(loader.results, key)
  end
end
```

```elixir
loader = Loader.new(fn user_ids ->
  query = from u in User, where: id in ^user_ids, select: {u.id, u}
  query
  |> Repo.all()
  |> Map.new
end)
```

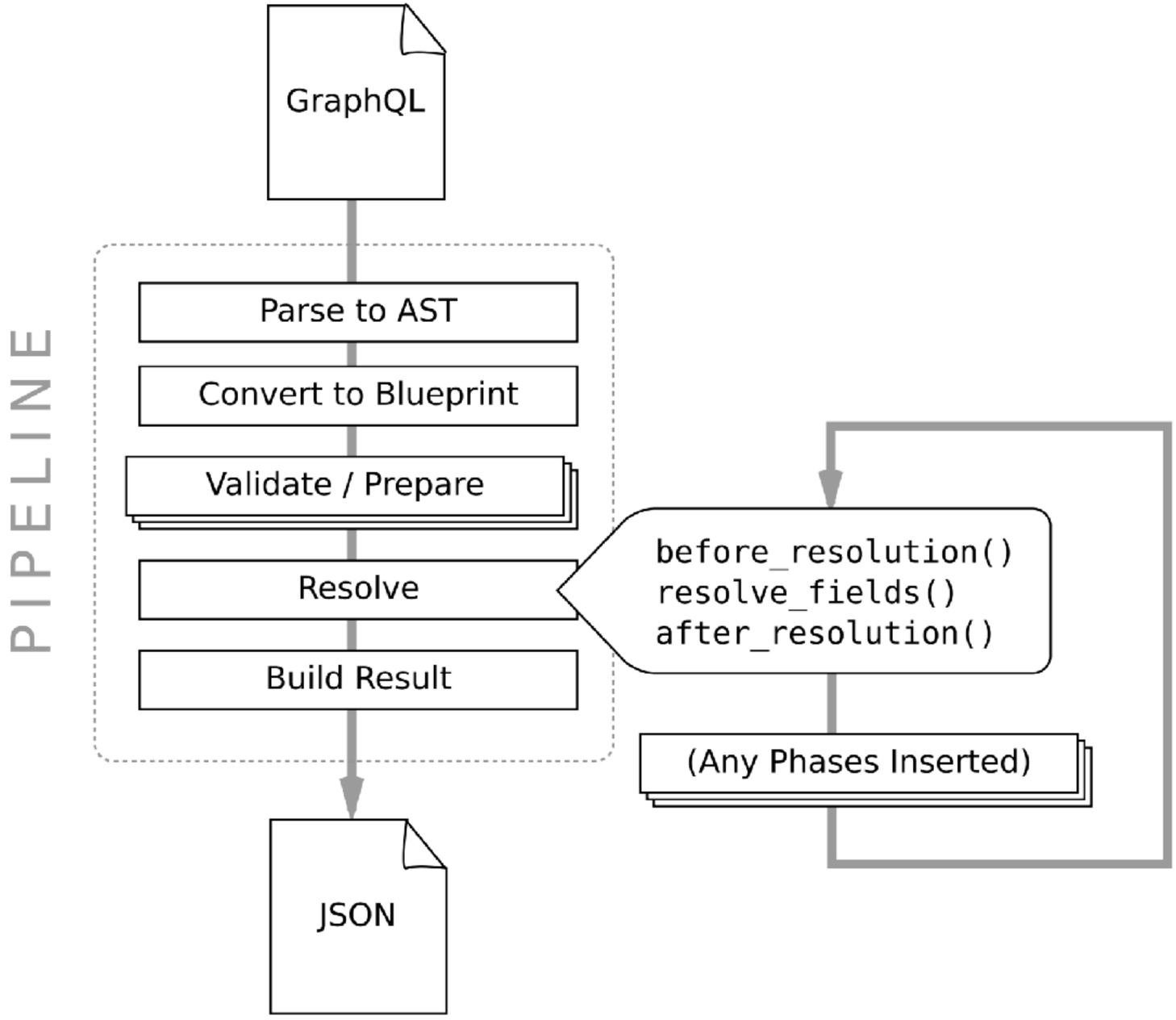
```elixir
loader
|> Loader.load(1)
|> Loader.load(2)
```

```elixir
loader = Loader.run(loader)
```

```elixir
Loader.get(loader, 1)
```

突然发现，我的心无处安放

```
document
|> Pipeine.Parse.run(schema: MyApp.Schema)
|> Pipeine.Blueprint.run()
|> Pipeine.Document.Validation.XXXX.run()
|> ... ...
|> Phase.Document.Execution.Resolution.run()  # Execution
|> Phase.Document.Result.run()                # Format Result
```



Excerpt From: Bruce Williams, Ben Wilson. Craft GraphQL APIs in Elixir with Absinthe. https://pragprog.com/book/wwgraphql/craft-graphql-apis-in-elixir-with-absinthe

Our new saviors

Absinthe.Plugin

Absinthe.Middleware

Reinvent dataloader

https://github.com/Pragmatic-Elixir-Meetup/dataloder-intro/commit/98e8b9

apply(Dataloader)

~~apply(Dataloader)~~ Tips

Tip 1: Loading one record from database

1. Setup dataloader context and plugin
2. Define the dataloader source
3. Add dataloader source
4. Load data with dataloader

```elixir
defmodule BlogWeb.Schema do
  use Absinthe.Schema

  def context(ctx) do
    dataloader = Dataloader.new()

    Map.put(ctx, :loader, dataloader)
  end

  def plugins do
    [Absinthe.Middleware.Dataloader] ++
      Absinthe.Plugin.defaults()
  end
end
```

https://github.com/Pragmatic-Elixir-Meetup/dataloder-intro/commit/eded219

Tip 1: Loading one record from database

1. Setup dataloader context and plugin
2. Define the dataloader source
3. Add dataloader source
4. Load data with dataloader

```elixir
defmodule BlogWeb.Dataloader.Database do
  def new do
    Dataloader.Ecto.new(Blog.Repo,
      query: &query/2)
  end

  def query(queryable, _params) do
    queryable
  end
end
```

https://github.com/Pragmatic-Elixir-Meetup/dataloder-intro/commit/eded219

Tip 1: Loading one record from database

1. Setup dataloader context and plugin
2. Define the dataloader source
3. Add dataloader source
4. Load data with dataloader

```
defmodule BlogWeb.Schema do
  use Absinthe.Schema

  def context(ctx) do
    dataloader =
      Dataloader.new()
+     |> Dataloader.add_source(:db,
+       BlogWeb.Dataloader.Database.new())

    Map.put(ctx, :loader, dataloader)
  end
end
```

https://github.com/Pragmatic-Elixir-Meetup/dataloder-intro/commit/eded219

Tip 1: Loading one record from database

1. Setup dataloader context and plugin
2. Define the dataloader source
3. Add dataloader source
4. Load data with dataloader

```elixir
import Absinthe.Resolution.Helpers, only: [on_load: 2]

field :user, non_null(:user) do
  resolve fn post,
             _,
             %{context: %{loader: loader}} ->
    loader
    |> Dataloader.load(:db, User, user)
    |> on_load(fn loader ->
      user = Dataloader.get(loader, :db, User, user)
      {:ok, user}
    end)
  end
end
```

https://github.com/Pragmatic-Elixir-Meetup/dataloder-intro/commit/eded219

Tip 2: Loading many records from database

```
{
  posts {
    title
    comments {
      content
    }
  }
}
```

```
resolve fn post, _, _ ->
  query = Ecto.assoc(post, :comments)
  {:ok, Blog.Repo.all(query)}
end
```

```
[debug] QUERY OK source="comments" db=0.8ms
SELECT c0.* FROM "comments" AS c0 WHERE (c0."post_id" = $1) [1]
[debug] QUERY OK source="comments" db=0.9ms queue=0.1ms
SELECT c0.* FROM "comments" AS c0 WHERE (c0."post_id" = $1) [2]
[debug] QUERY OK source="comments" db=0.9ms queue=0.1ms
SELECT c0.* FROM "comments" AS c0 WHERE (c0."post_id" = $1) [3]
```

Tip 2: Loading many records from database

```
field :comments, list_of(:comment) do
  resolve fn post, _, %{context: %{loader: loader}} ->
    loader
    |> Dataloader.load(:db, {:many, Comment}, post_id: post.id)
    |> on_load(fn loader ->
      comments = Dataloader.get(loader, :db, {:many, Comment}, post_id: post.id)
      {:ok, comments}
    end)
  end
end
```

```
[debug] QUERY OK source="comments" db=0.9ms
SELECT c0.* FROM "comments" AS c0 WHERE (c0."post_id" = ANY($1)) [[1, 2, 3]]
```

https://github.com/Pragmatic-Elixir-Meetup/dataloder-intro/commit/29afee1

Tip 3: Loading aggregated data with KV source

```
{
  posts {
    title
    content
    user {
      name
    }
    commentsCount
  }
}
```

```
field :comments_count, :integer do
  resolve fn %{id: post_id}, _, _ ->
    import Ecto.Query
    query = from(c in Comment, where: c.post_id == ^post_id)
    {:ok, Repo.aggregate(query, :count, :id)}
  end
end
```

```
SELECT count(c0."id") FROM "comments" AS c0 WHERE (c0."post_id" = $1) [1]
[debug] QUERY OK source="comments" db=0.3ms
SELECT count(c0."id") FROM "comments" AS c0 WHERE (c0."post_id" = $1) [1]
[debug] QUERY OK source="comments" db=2.3ms
SELECT count(c0."id") FROM "comments" AS c0 WHERE (c0."post_id" = $1) [2]
[debug] QUERY OK source="comments" db=2.9ms
SELECT count(c0."id") FROM "comments" AS c0 WHERE (c0."post_id" = $1) [3]
```

Tip 3: Loading aggregated data with KV source

1. Define the dataloader source
2. Add dataloader source
3. Load data with dataloader

```elixir
defmodule BlogWeb.Dataloader.CommentsCount do
  import Ecto.Query
  def new(), do: Dataloader.KV.new(&load/2)

  def load(_batch_key, post_ids) do
    query = from c in Blog.Comment,
      where: c.post_id in ^MapSet.to_list(post_ids),
      group_by: c.post_id,
      select: {c.post_id, fragment("count(*)")}

    query
    |> Blog.Repo.all()
    |> Map.new()  # %{ 1 => 4, 2 => 1 }
  end
end
```

https://github.com/Pragmatic-Elixir-Meetup/dataloder-intro/commit/18f0e05

Tip 3: Loading aggregated data with KV source

1. Define the dataloader source
2. Add dataloader source
3. Load data with dataloader

```
defmodule BlogWeb.Schema do
  use Absinthe.Schema

  def context(ctx) do
    dataloader =
      Dataloader.new()
+     |> Dataloader.add_source(:comments_count,
+        BlogWeb.Dataloader.CommentsCount.new())

    Map.put(ctx, :loader, dataloader)
  end
end
```

https://github.com/Pragmatic-Elixir-Meetup/dataloder-intro/commit/18f0e05

Tip 3: Loading aggregated data with KV source

1. Define the dataloader source
2. Add dataloader source
3. Load data with dataloader

```
field :comments_count, :integer do
  resolve fn %{id: post_id}, _, %{context: %{loader: loader}} ->
    loader
    |> Dataloader.load(:comments_count, :post, post_id)
    |> on_load(fn loader ->
      count = Dataloader.get(loader,
                :comments_count, :post, post_id) || 0
      {:ok, count}
    end)
  end
end
```

```
[debug] QUERY OK source="comments" db=5.4ms queue=0.1ms
SELECT c0."post_id", count(*) FROM "comments" AS c0 WHERE (c0."post_id" = ANY($1)) GROUP BY c0."post_id" [[1, 2, 3, 4]]
```

https://github.com/Pragmatic-Elixir-Meetup/dataloder-intro/commit/18f0e05

Tip 4: Using Absinthe’s dataloader helper

```
loader
|> Dataloader.load(source, batch_key, key)
|> on_load(fn loader ->
  result = Dataloader.get(source, batch_key, key)
  {:ok, result}
end)
```

Tip 4: Using Absinthe's dataloader helper

```
def dataloader(source) do
  fn parent, args, %{context: %{loader: loader}} = res ->
    resource = res.definition.schema_node.identifier

    loader
    |> Dataloader.load(source, {resource, args}, parent)
    |> on_load(fn loader ->
      result = Dataloader.get(loader, source, {resource, args}, parent)
      {:ok, result}
    end)
  end
end
```

Tip 4: Using Absinthe's dataloader helper

```
import Absinthe.Resolution.Helpers, only: [on_load: 2]

field :user, non_null(:user) do
  resolve fn user, _, %{context: %{loader: loader}} ->
    loader
    |> Dataloader.load(:db, User, user)
    |> on_load(fn loader ->
      user = Dataloader.get(loader, :db, User, user)
      {:ok, user}
    end)
  end
end
```

Tip 4: Using Absinthe's dataloader helper

```elixir
import Absinthe.Resolution.Helpers, only: [dataloader: 1]

field :user, non_null(:user), resolve: dataloader(:db)
```

```elixir
import Absinthe.Resolution.Helpers, only: [on_load: 2]

field :user, non_null(:user) do
  resolve fn user, _, %{context: %{loader: loader}} ->
    loader
    |> Dataloader.load(:db, User, user)
    |> on_load(fn loader ->
      user = Dataloader.get(loader, :db, User, user)
      {:ok, user}
    end)
  end
end
```

https://github.com/Pragmatic-Elixir-Meetup/dataloder-intro/commit/f720de8

Thank you