Table of Contents

概述

本文包含对 EAD 应用设计开发平台的技术架构设计的综合阐述。包括前后端技术架构说明，平台开发使用关键编程技术和思想介绍。阐明系统完全面向数据模型的设计理念，以数据定义软件，一切皆资源，面向微服务和 API 的设计思想。

术语定义

EAD

企业云应用设计器，EAD 应用设计开发平台的核心配置模块，存储和管理配置开发的流程和数据，对引擎运行生命周期中的中间件、参数、视图和动作通用驱动和自定义驱动的调度。

Base

基础抽象，主要代表 EAD 平台中内置的基础抽象类数据模型或 EAD 引擎程序代码中的基础抽象类程序。

Sys

通用系统功能，主要代表 EAD 平台中全局通用的业务功能模块，主要包括账户、日志、工作流、通知、消息、动态、日志等通用系统模块。

Restful

具象状态传输，REST 通常基于使用 HTTP，URI，和 JSON、XML 以及 HTML 这些现有的广泛流行的协议和标准。资源是由URI来指定。对资源的操作包括获取、创建、修改和删除资源，操作对应 HTTP 协议提供的 GET、POST、PUT 和 DELETE 方法。通过操作资源的表现形式来操作资源。资源的表现形式则是 JSON、XML 或者 HTML，取决于读者是机器还是人，是消费 Web 服务的客户软件还是 Web 浏览器。在目前 EAD 的架构体系中，资源的主要表现形式为 JSON 对象。

EAD 引擎（EAD Engine）

EAD 服务端 API 引擎，基于 J2EE 软件开发平台的 Spring Cloud 框架体系构建，以 AOP 架构思想对数据持久化、视图驱动、动作驱动以及应用及资源进行管理和调度，对外统一输出基于 Restful 风格的 API，供 Rainbow 或者其它客户端程序调用和渲染。

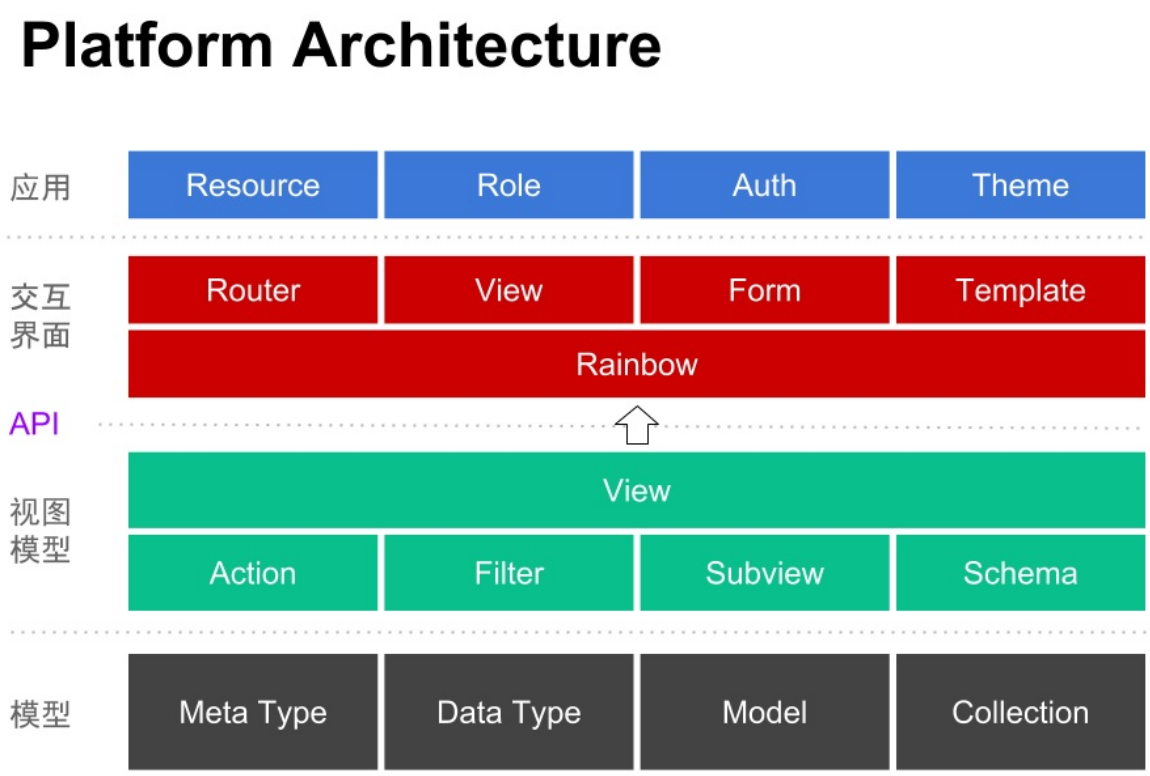
Rainbow

EAD 平台前端解析引擎，基于 HTML5、CSS3 和 JavaScript 对 EAD 服务引擎提供的 Rest Service API 进行解析和渲染，动态生成 Web 用户交互界面的前端应用程序框架。

参考资料

- 《EAD 引擎架构设计说明》；
- 《Rainbow 架构设计说明》；
- 《维基百科》；

平台架构



模型层

模型层为平台的基础数据定义层，负责业务逻辑的定义和描述，处于 EAD 引擎。

元数据（Meta Type）

EAD 引擎定义的平台元数据类型，平台根据不同的数据源映射对应的数据类型，在 EAD 运行时生命周期内遵循数据库不可知论原则，参考如下元数据类型：

- **Boolean**，布尔值;
- **String**，字符串;
- **Text**，文本;
- **Integer**，整数;
- **Number**，数字;
- **Time**，时间;

- **JSON** JSON 对象;
- **Array** JSON 数字;
- **Key**，键值;
- **Binary**，二进制;

数据类型（Data Type）

基于元数据类型扩展的基本数据类型，以满足业务模型数据校验规则，允许开发者自定义数据类型。EAD 平台已内置 30 余种常用数据类型，详细的数据类型列表请参考《平台附录参考》。

模型（Model）

单元数据定义，基于 OOP 完全面向对象的编程设计思想，将所有业务单元按照对象模型进行设计。模型可以添加不同数据类型的属性或已经创建的模型作为其成员。通过模型对象来定义业务单元，模型是整个 EAD 平台的核心业务接口定义，基于模型实现实体（一般指数据库表）创建、数据查询、数据操作、视图 API 提供等核心模块，各个模块依赖于模型进行互相之间的匹配和协作。

数据集模型（Collection）

模型可以以集合模式添加为其它模型的子成员，以支持一对多或多对多关系模型的设计。

视图模型层

视图模型层作为平台的 API 服务提供层，负责数据访问权限、数据操作、数据过滤器的配置和管理。视图模型层完成数据逻辑的组织后将模型范式、动作配置、过滤器配置、业务数据以 API 的形式提供给访问者。视图模型层处于 EAD 引擎。

范式（Schema）

模型属性详细描述数据，提供给客户端作为数据和操作表单动态渲染和校验的依据。

动作（Action）

基于视图进行绑定模型的数据操作，通过驱动设置、规则设置和 HTTP 方法实现复杂场景的数据操作配置开发。

过滤器（Filter）

基于视图属性组合设置实现数据过滤，支持排序和条件类型，不同类型条件（相等、包含、区间、不等于、相似）支持，设置合适的表单交互控件。

子视图（Subview）

支持递归嵌套方式访问子视图，以实现多级嵌套业务的导航和管理。

API

平台采用完全面向 Restful 风格的 API 进行设计开发，服务引擎与用户交互界面完全分离，Rainbow 通过 EAD 引擎 API 提供的数据动态渲染用户交互界面。API 属于 EAD 引擎与 Rainbow 的对接层，由 Rainbow 采用 Ajax 的方式向 EAD 引擎发起数据访问或操作请求，EAD 引擎以 JSON 的形式返回响应数据或消息。

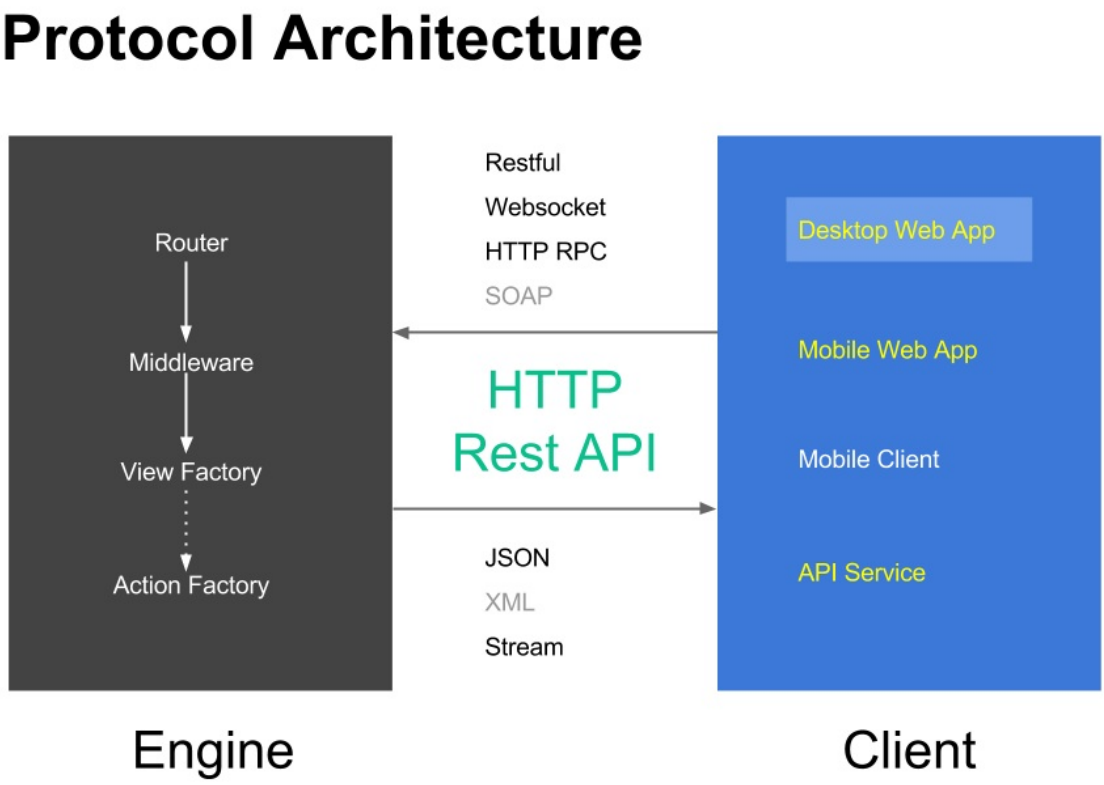
视图层

视图层处于 Rainbow，根据 Ajax 请求返回的 API 数据进行应用通用视图渲染、客户端路由导航，根据模型定义和主题模板动态渲染业务模块视图、表单等用户交互界面。

应用资源

应用层处于 EAD 引擎，支持多实例创建应用，按照各应用进行多视图的资源化配置管理，分应用创建应用角色，进行角色资源授权和用户角色分配。根据应用层角色进行用户鉴权和 API 对外提供。

协议架构



EAD 平台完全面向 Restful 风格的 API 进行设计开发，实现服务端与客户端的完全分离，采用目前业界最广泛应用的无状态 HTTP/HTTPS 协议为 Rainbow、移动客户端、第三方应用等各种形式的客户端提供数据服务。

HTTP / HTTPS

超文本传输协议（英文：HyperText Transfer Protocol，缩写：HTTP ）是互联网上应用最为广泛的一种网络协议。设计 HTTP 最初的目的是为了提供一种发布和接收HTML页面的方法。通过 HTTP 或者 HTTPS 协议请求的资源由统一资源标识符（ Uniform Resource Identifiers，URI ）来标识。EAD 完全依托于 HTTP 协议进行服务的提供和访问。

Restful

具象状态传输，REST 通常基于使用 HTTP，URI，和 JSON、XML 以及 HTML 这些现有的广泛流行的协议和标准。资源是由URI来指定。对资源的操作包括获取、创建、修改和删除资源，操作对应 HTTP 协议提供的 GET、POST、PUT 和 DELETE 方法。通过操作资源的表现形式来操作资源。资源的表现形式则是 JSON、XML 或者 HTML，取决于读者是机器还是人，是消费 Web 服务的客户软件还是 Web 浏览器。在目前 EAD 的架构体系中，资源的主要表现形式为 JSON 对象。

Rest RPC

EAD 允许其它第三方程序使用 Rest RPC 方式以用户身份进行所有 API 的远程访问和操作。

Websocket

WebSocket 是 HTML5 开始提供的一种在单个 TCP 连接上进行全双工通讯的协议。WebSocket 通讯协议于 2011 年被IETF定为标准 RFC 6455，WebSocketAPI 被 W3C 定为标准。在现代浏览器和主流移动设备中 EAD 采用 Websocket 进行即时类消息服务的推送。

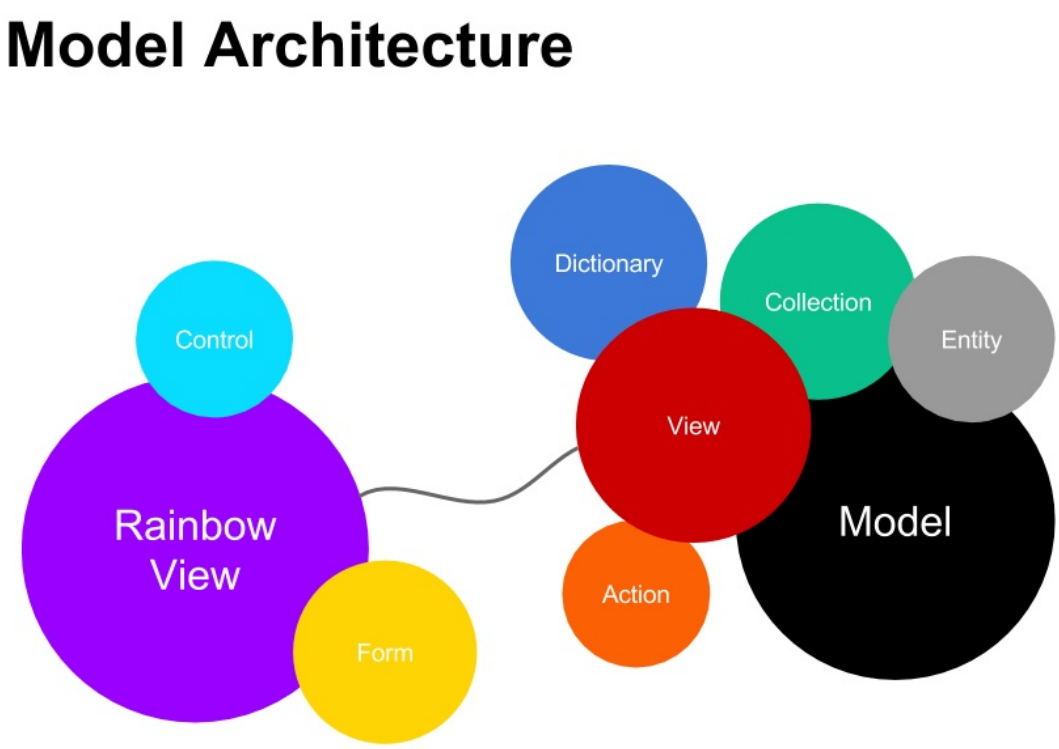
JSON / JSONP

JSON（JavaScript Object Notation）是一种由道格拉斯·克罗克福特构想设计、轻量级的数据交换语言，以文字为基础，且易于让人阅读。尽管JSON是Javascript的一个子集，但JSON是独立于语言的文本格式，并且采用了类似于C语言家族的一些习惯。EAD 采用 JSON 进行服务端和客户端的主要数据交换。

Stream

EAD 以流的方式对 GridFS 中的静态文件资源进行的访问，并对外以流的方式输出在线、附件或流媒体形式静态文件。

模型架构



模型作为整个业务定义的原点和基础，类似于面向对象编程中的类，基于模型实现实体创建，扩展定义视图、数据集，最终实例化为视图 API、数据字典、视图动作。 EAD 秉持一切皆模型（数据）的设计理念， 通过数据定义软件功能。

OOP

EAD 的配置开发遵循 OOP 思想，所有业务单元的定义都以对象形式进行定义，通过定义对象的属性、动作和过滤器完成业务单元的配置开发。

MVC

EAD 平台遵循经典 MVC 的架构基于模型（M）实例化扩展实现视图（V）和视图动作（C）。

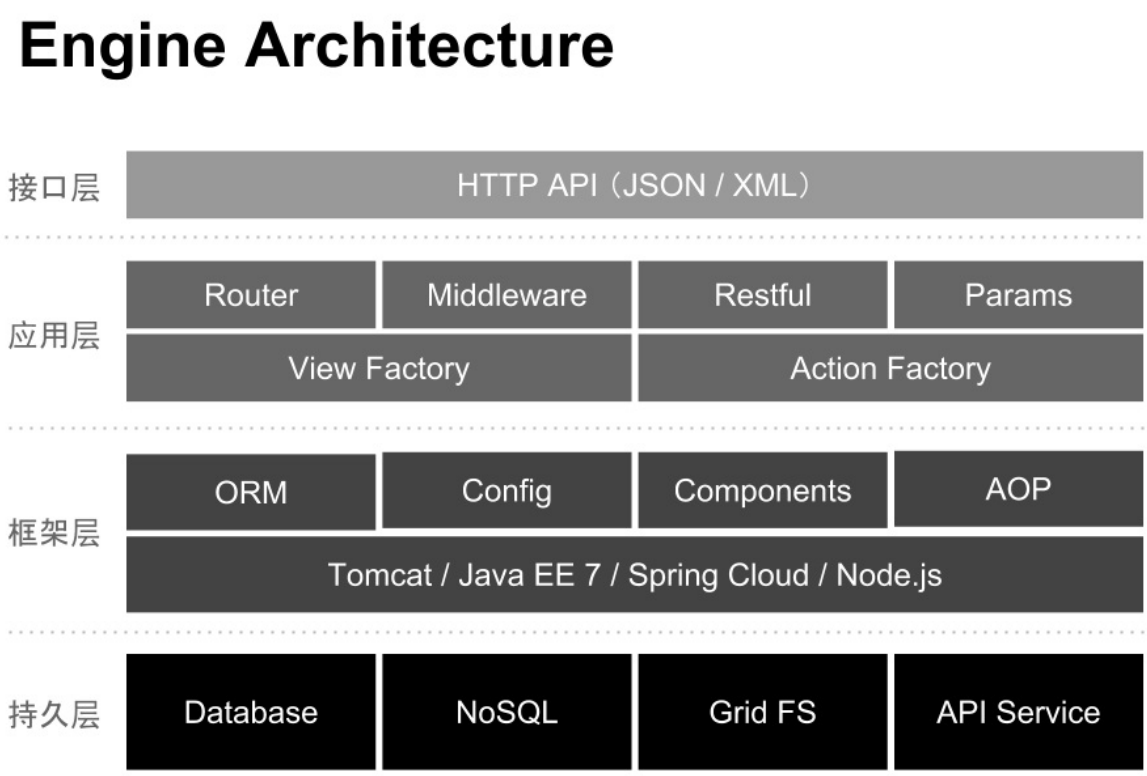
MVVM

Rainbow 基于 API 模型动态渲染用户交互视图，通过动态渲染的数据表单对视图模型进行更新，利用 Backbone 实现服务端和客户端数据同步交互。

数据定义软件

EAD 平台采用模型定义的方式进行应用功能的配置开发，除自定义驱动以外的所有应用逻辑都以数据的形式进行存储，使得应用功能的定义不过分依赖于程序代码。应用功能的逻辑可以通过实时修改配置数据实现变更，以满足业务规则多变场景中的快速响应。

引擎架构



1. 核心设计思想

EAD 引擎基于 J2EE 开发平台，依赖于 Spring Cloud 框架进行构建。遵循 AOP 程序设计思想实现整个引擎生命周期的调度和管理。以微服务的形式抽象服务组件，统一为引擎提供通用服务接口。引擎遵循 Restful 风格完全面向 API 的开发模型，只负责业务数据 API 的组织和渲染输出，即保持了引擎开发的专注和简单，又可以使引擎成为更加通用的数据 API 服务提供平台。

Spring Cloud

Spring Cloud 提供的工具用于开发人员快速构建一些常见的模式在分布式系统（如配置管理，服务发现，路器隔断，智能路由，微代理，控制总线，一次性令牌，全局锁，领导竞选，分布式会话，群集状态）。

AOP

AOP为Aspect Oriented Programming的缩写，意为：面向切面编程，通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。AOP是OOP的延续，是软件开发中的一个热点，也是Spring框架中的一个重要内容，是函数式编程的一种衍生范型。利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。

微服务

微服务的基本思想在于考虑围绕着业务领域组件来创建应用，这些就应用可独立地进行开发、管理和加速。在分散的组件中使用微服务云架构和平台使部署、管理和服务功能交付变得更加简单。

2. 引擎架构层级

持久层

通过 JDBC 支持主流关系型数据库进行业务数据持久化存储和查询，为引擎提供数据源。支持 NoSQL、GridFS 进行静态文件存储。支持使用 API Service 的形式（扩展驱动）将数据持久化到第三方应用系统。

框架层

基于 J2EE 开发平台，依赖于 Spring Cloud 框架，遵循 AOP 编程思想够建包含配置管理、 ORM、应用服务组件的引擎运行时框架。

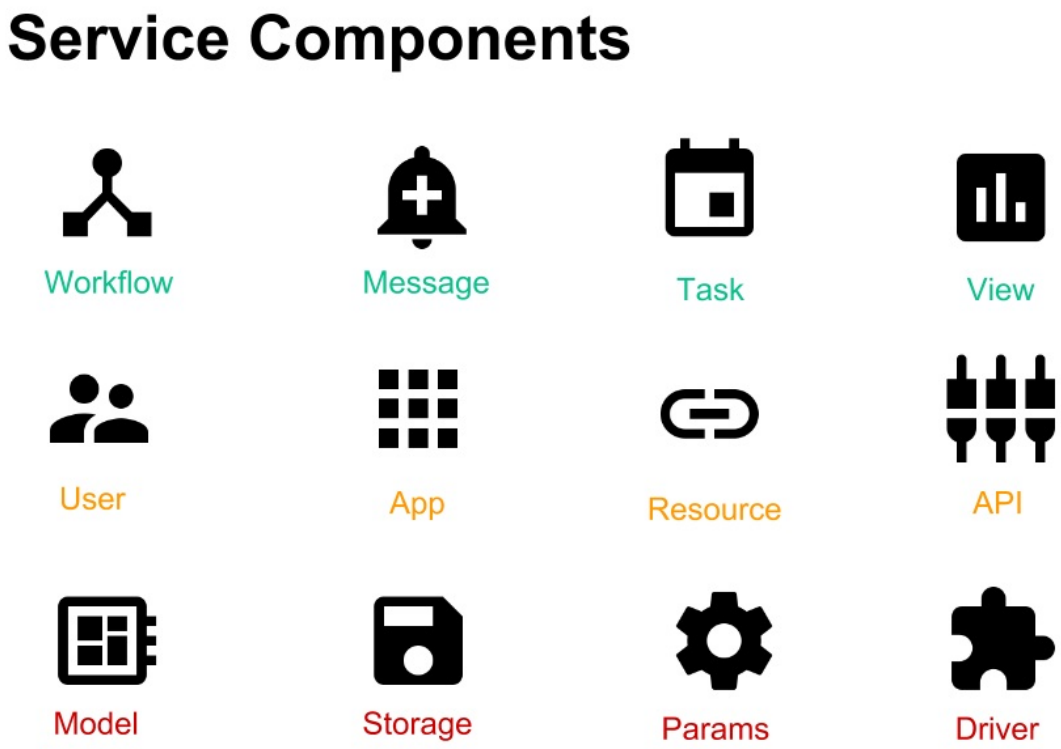
应用层

基于框架层遵循 Restful 架构风格依次通过路由（Router）、服务中间件（Middleware）、参数组件（Param）、视图工厂组件（View Factory）、动作工厂组件（Action Factory）完成普通业务应用的请求响应。

接口层

EAD 引擎完全遵循 Restful 风格对外提供 JSON 格式的数据视图以及流形式的静态文件。

服务组件



EAD 引擎采用微服务和 AOP 的架构进行设计开发，将引擎通用模块抽象为服务能力组件。 各服务组件各司其职，减少代码的冗余重复，各组件以服务的方式提供对外调用的方法和接口。

1. 基础组件

基础类组件为 EAD 引擎的核心组件，所有业务必须依赖于基础组件进行管理和调度或服务提供才可以实现 EAD 平台的正常运行。

模型（Model）

数据管理组件，通过对模型属性、方法等的定义实现业务数据的定义描述，并根据规则实现数据的查询和操作管理。

存储（Storage）

文件存储组件，利用 GridFS 可以实现 EAD 静态文件的分片、分布式架构存储。利用 MD5 Hash 统一实现重复文件校验，减少碎片化文件存储，节省存储资源。组件提供统一的文件写入和读取接口，降低文件存储的实现难度。通过适配器方式对接存储引擎，允许以插件的形式接入不同类型的存储引擎。

参数（Param）

参数管理组件，支持直接量、会话、驱动等形式的系统参数定义。提供参数访问的接口，可以在平台引擎各个配置参数节点访问系统参数，最大限度提升系统的动态可配置性。

驱动（Driver）

驱动管理组件，对开发者自定义扩展的驱动程序进行注册管理和访问接口提供。通过依赖注入的方式动态进行驱动程序的调度和生命周期管理。

2. 系统组件

系统类组件为 EAD 平台通用业务模块，系统组件负责用户、应用、角色、资源等通用业务的提供，也可以根据业务需要对通用业务进行扩展开发。

用户（User）

用户组件，对系统账户、组织机构、权限、登录认证进行管理，提供用户字典和认证服务接口。

应用（App）

应用组件，对视图进行资源化管理，支持多实例应用创建，对应用资源、角色和资源授权进行管理，提供用户应用资源 API。

资源（Resource）

资源管理，通过 URL 对应用和 API 进行调度和响应，支持多种形式的 API、文件资源化访问。

API

通过 JSON 的方式对外提供统一范式的视图数据接口。

3. 应用模块组件

应用组件，是抽象度较高且非普通数据管理类应用功能模块定义的组件，可以根据具体的需求灵活选择配置。

工作流（Workflow）

工作流组件，基于工作流模型和流转步骤规则完成流程业务的流转，包含审批、退回、条件判断等流程操作和规则流转的功能。支持工作流模型、流转步骤和规则的的可视化配置。

消息（Message）

消息组件，根据业务场景需要，对于需要用户及时关注的事件支持多种形式（系统、短信、邮件）的消息推送，提供消息通知接口供其它组件调用。

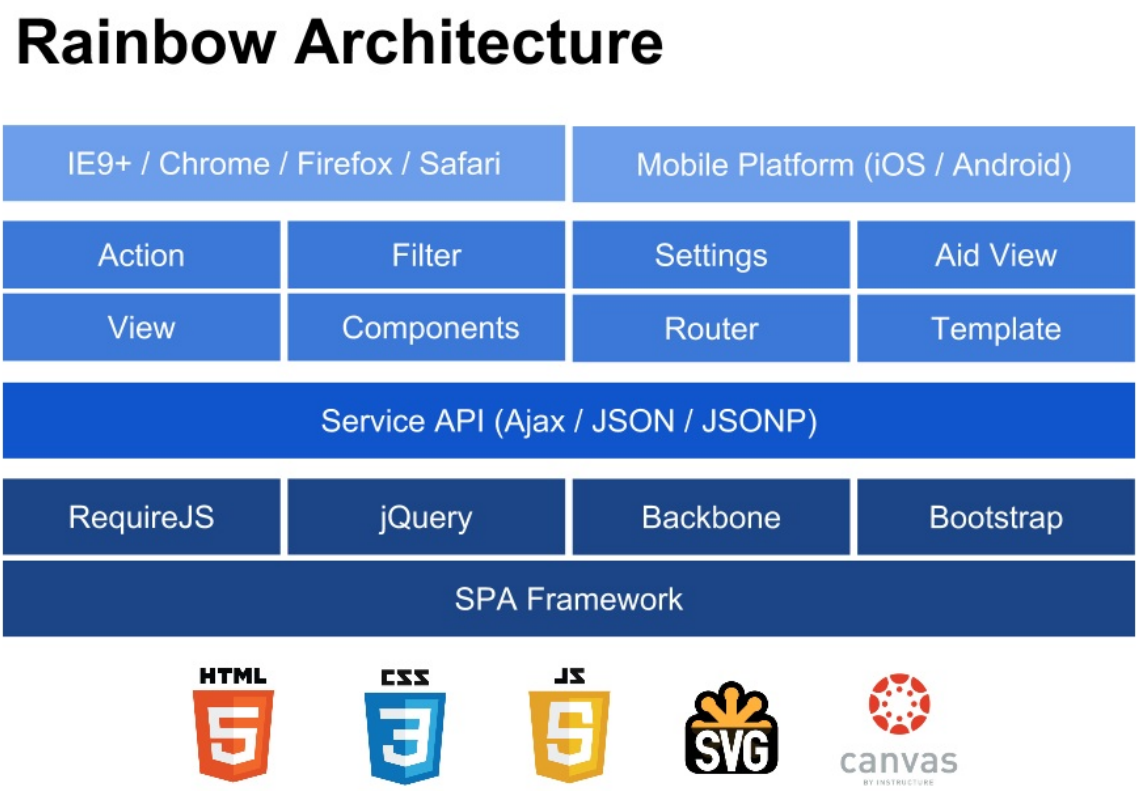
任务（Task）

任务组件，对 EAD 系统定时任务管理和触发。

视图（View）

视图组件，基于模型组件实现视图管理和组织输出视图 API 数据。

Rainbow 架构



AMD

Framework

采用 SPA（单页 Web 应用框架）以 Ajax 为核心的服务请求模式，利用 RequireJS 基于 AMD 规范进行模块化开发和依赖管理，使用 jQuery 作为 DOM 操作类库，使用 Backbone 作为前端 MVC 框架，结合 Bootstrap 构建前端 UI 组件形成根据服务 API 动态渲染交互视图的 Web App 框架。

API

以 JSON / JSONP 的方式与服务端引擎 API 发起请求并异步响应返回数据接口。

浏览器

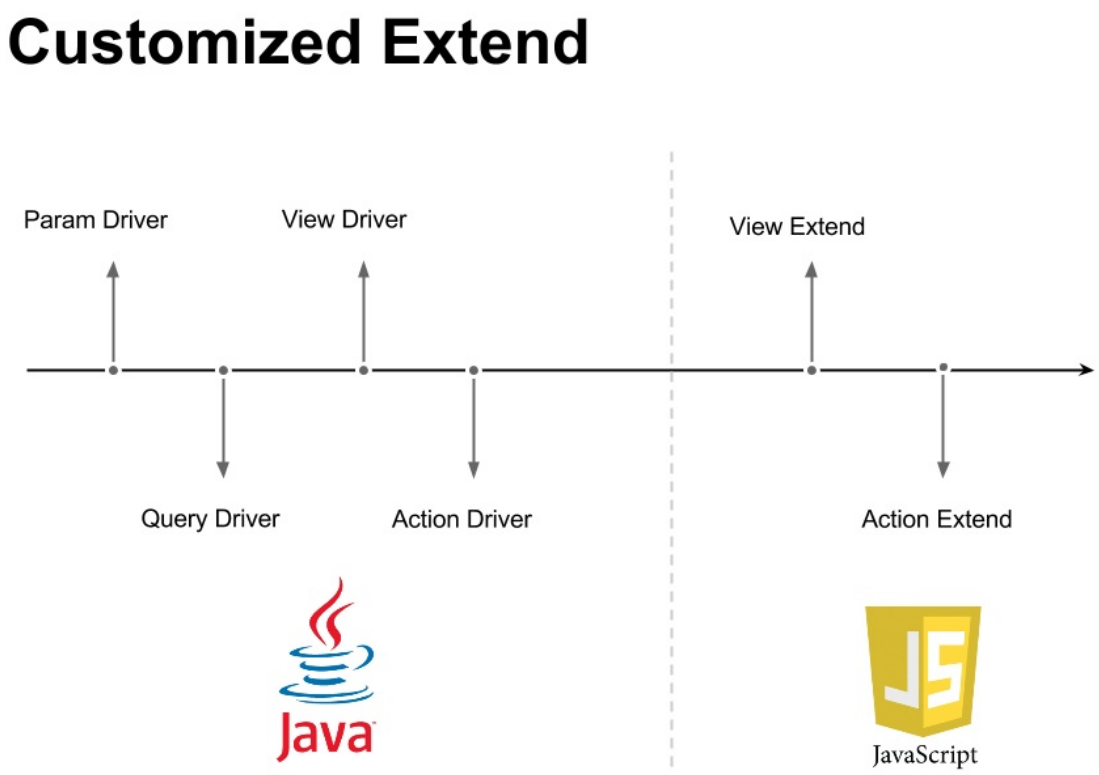
桌面（Desktop）

IE 10+ 、 Chrome、Firefox、Safari

移动端（Mobile）

iOS、Android、Windows Phone

扩展驱动



EAD 专注于解决 80% 简单重复且可配置的应用开发问题，在运行生命周期的各个关键节点都允许开发者自定义开发扩展驱动，以应对各种可能出现的复杂需求和应用场景。

如上图所示，从服务端到客户端的数据传输环节一共有 6 个节点允许自行开发扩展驱动，在 EAD 引擎和 Rainbow 中分别以依赖注入和闭包回调的方式动态加载和执行扩展驱动。

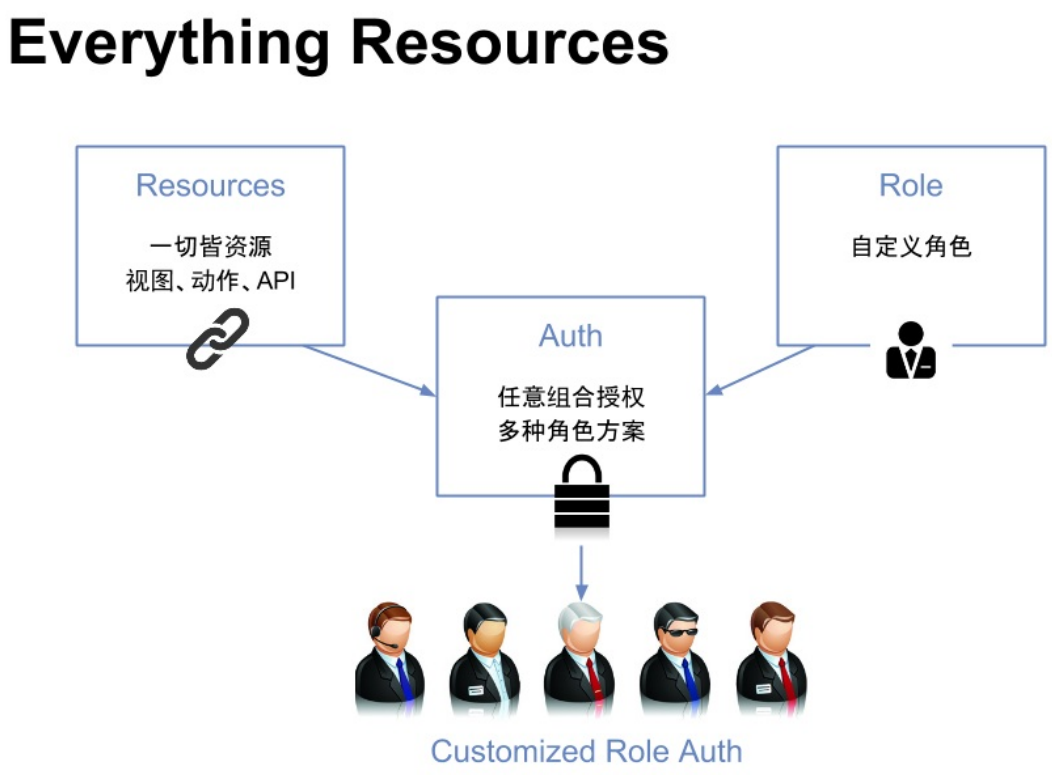
EAD 引擎

EAD 引擎端允许根据需求和场景利用 Java 语言扩展开发系统变量驱动、查询驱动、视图驱动、视图动作驱动，通过驱动管理组件注册后可以在开发者应用中心配置选择。

Rainbow

Rainbow 核心任务是负责视图用户交互界面的渲染和动作交互界面的渲染，允许开发者利用 JavaScript 自定义视图扩展驱动和动作扩展驱动，将扩展 JS 文件放置到指定目录即可在开发者配置中心以文件命名空间的形式配置动作执行自定义扩展驱动。

一切皆资源



EAD 引擎参考 Restful 架构风格，所有服务提供皆以资源 URI 方式提供。一切皆资源的架构风格，使配置开发无需过多考虑资源提供的命名和规范即可提供统一直观可快速理解的 URL。

业务资源化，减少服务开发语言和平台特征，有利于平台未来扩展升级的技术选型。

资源化 API 服务提供，统一资源的入口，可以更好的利用 AOP 实现更加灵活的资源中间件扩展和依赖注入，为平台未来扩展更丰富的资源形式规划好规范蓝图。

视图资源

视图资源为 EAD 平台提供服务的主要形式，通过视图资源为 Rainbow 或 其它形式的客户端提供业务服务 API的数据访问类资源。

动作资源

动作资源是基于视图资源提供数据管理操作和 数据 API 的操作类资源。

通用资源

通用资源是为 EAD 平台正常运行提供登录、注销、应用资源列表等通用能力的资源。

外部资源

EAD 允许以重定向或内嵌（iFrame）的方式引入自定义或第三方应用的 Web 资源。

开发者中心

EAD 平台主要通过开发者中心进行应用的设计开发，按照模型、数据集、视图、应用、资源、角色、授权的顺序依次进行业务模块的设计开发和资源初始化及访问授权。以下是对开发者中心的开发流程和核心功能点的概要说明，如需详细功能的了解和学习请参考《配置开发手册》。

1. 模型（开发 > 模型）

通过开发中心模型模块主题域管理、数据类型管理、模型管理进行业务模型的定义和属性管理。使用实体创建、视图复制、自动配置和模型批量导入模板（Excel格式）等快捷方式可以快速实现数据库表创建、视图复制、模型视图资源快速创建、模型批量创建，提升配置开发的效率。

2. 数据集（开发 > 数据集）

基于已经创建成功的模型定义数据集合，可以对一个具有实体的模型进行查询构建，也可以基于没有实体映射的虚拟模型构建查询。关系型数据库支持配置原生 SQL 语句配置查询，保持集合字段名称和模型属性名称一致即可自动实现数据映射。

如果使用原生数据库查询无法满足业务需求，还可以开发扩展查询驱动，发布注册后配置绑定即可满足复杂业务场景。

3. 视图（开发 > 视图）

基于已经创建成功的模型定义视图，从模型属性中选择允许访问的属性即可完成一个数据服务 API。通过视图属性的权限、表单、格式化规则等个性化配置可以构建丰富的数据视图界面。

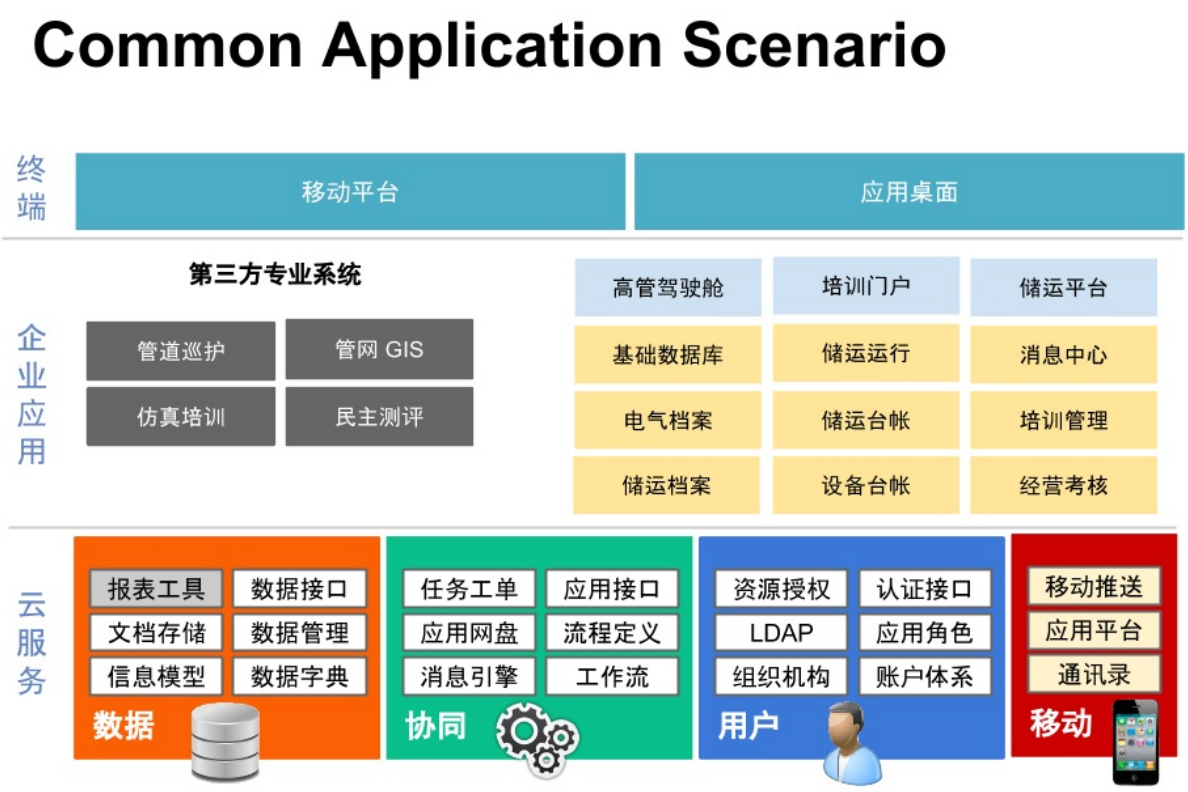
通过与同一类模型构建的数据集合进行绑定，通过数据集合查询返回的数据为服务请求者提供内容。

基于视图可以快速构建符合 Restful 风格的数据创建、更新、删除等操作，也可以开发扩展动作驱动，发布注册后配置绑定以支持复杂业务场景的数据操作需求。

4. 应用（应用 > 应用）

构建好业务功能视图后（非必须前置要求，也可以事先创建应用），创建应用后将业务视图添加为资源，通过角色资源授权完成业务功能视图的发布。

典型用户场景



EAD 企业应用设计开发平台，建议企业基于 EAD 构建面向云服务的企业信息化通用平台。在统一云服务的平台之上快速构建多实例应用矩阵，使得企业应用之间可以共享基础数据、无缝业务集成、共用并弹性扩展服务资源。

EAD 也支持第三方应用系统接入 EAD 基础服务，实现企业统一应用架构，降低用户使用和学习成本。

部署架构

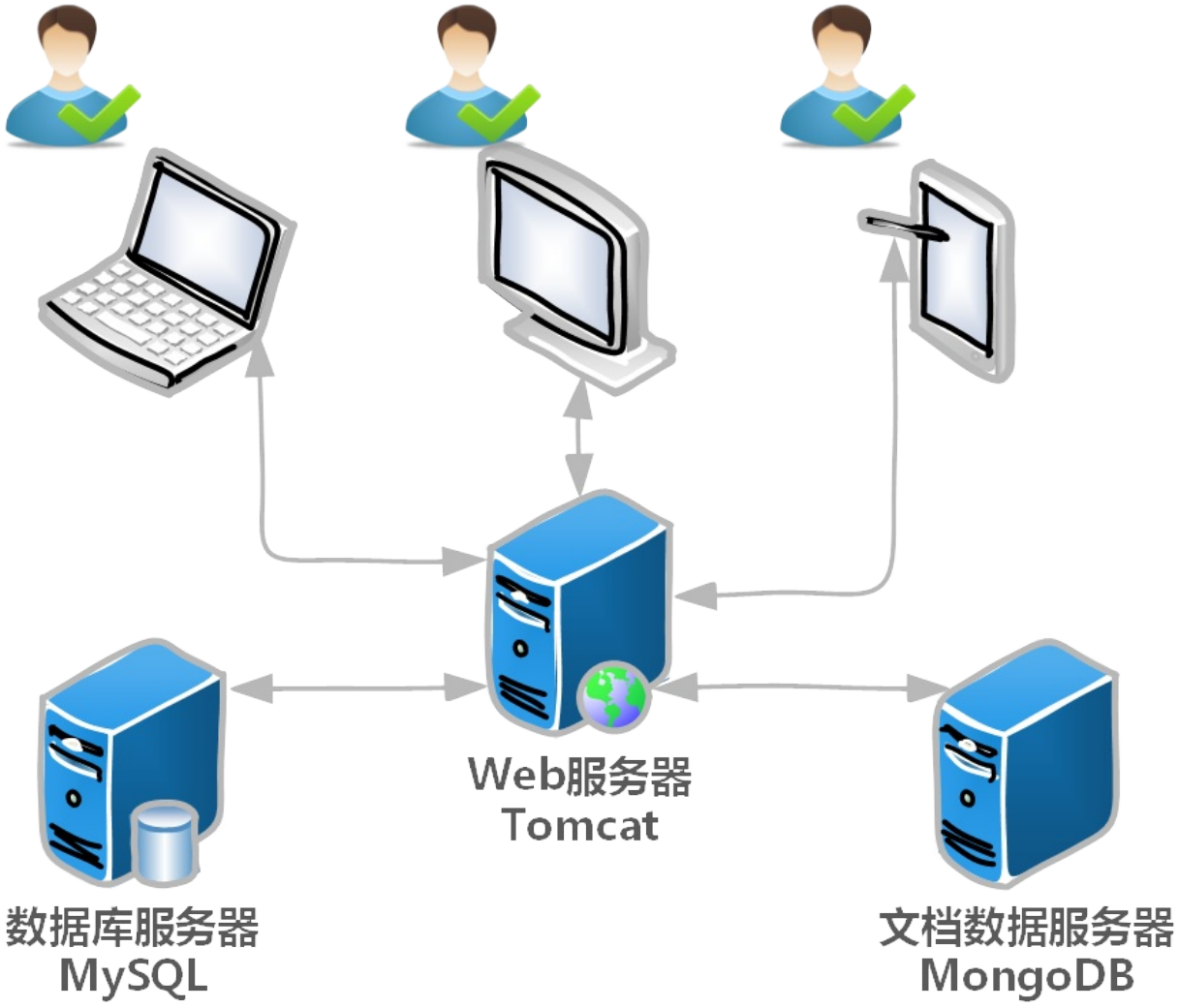
EAD的硬件部署方案可以非常灵活，支持传统主机部署、私有云部署和公有云部署等多种硬件部署架构和方案。不管是哪一种部署方式，对服务器配置和性能的要求基本是相同的。

因此，EAD的部署架构主要是根据用户访问的规模和对性能和可靠性的要求，分为单服务器部署架构和分布式部署架构两种方式。

1 集中式部署架构

集中式部署，又叫单例部署，是EAD最常见的部署方式，部署简单，对硬件要求低，只需要最多三台服务器就可以完成部署。

集中式部署方式如下图所示：

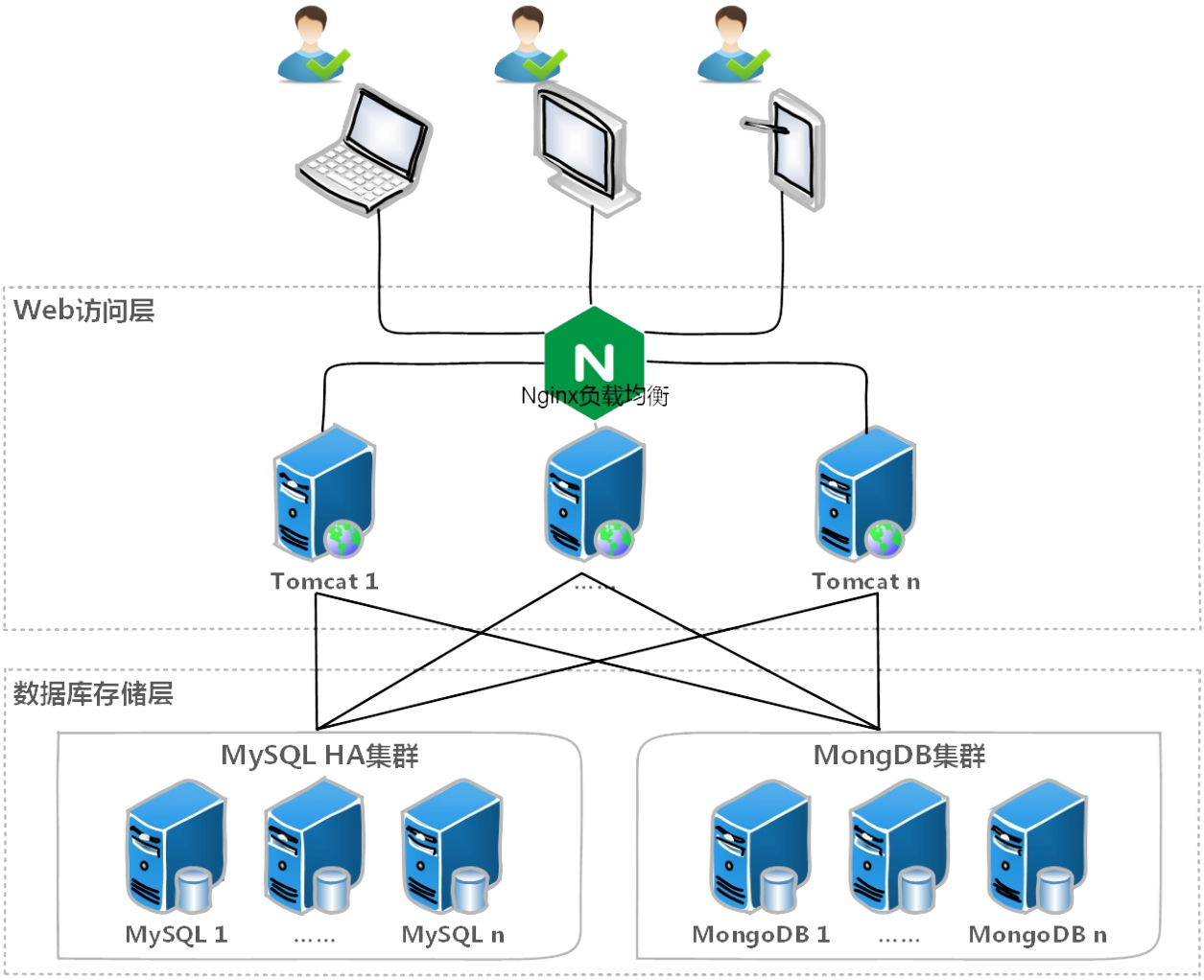


我们选用MySQL作为关系数据库存储，部署在一台服务器上，选用MongoDB作为文档数据服务器，EAD的核心应用和Tomcat部署在一台专用的Web服务器上。如果服务器资源紧张，这三个服务器也可以公用。

2 分布式部署架构

分布式部署多用在大型企业应用领域，适用于数据量大，用户数多，并发访问压力大的业务场景。

分布式部署方式如下图所示：



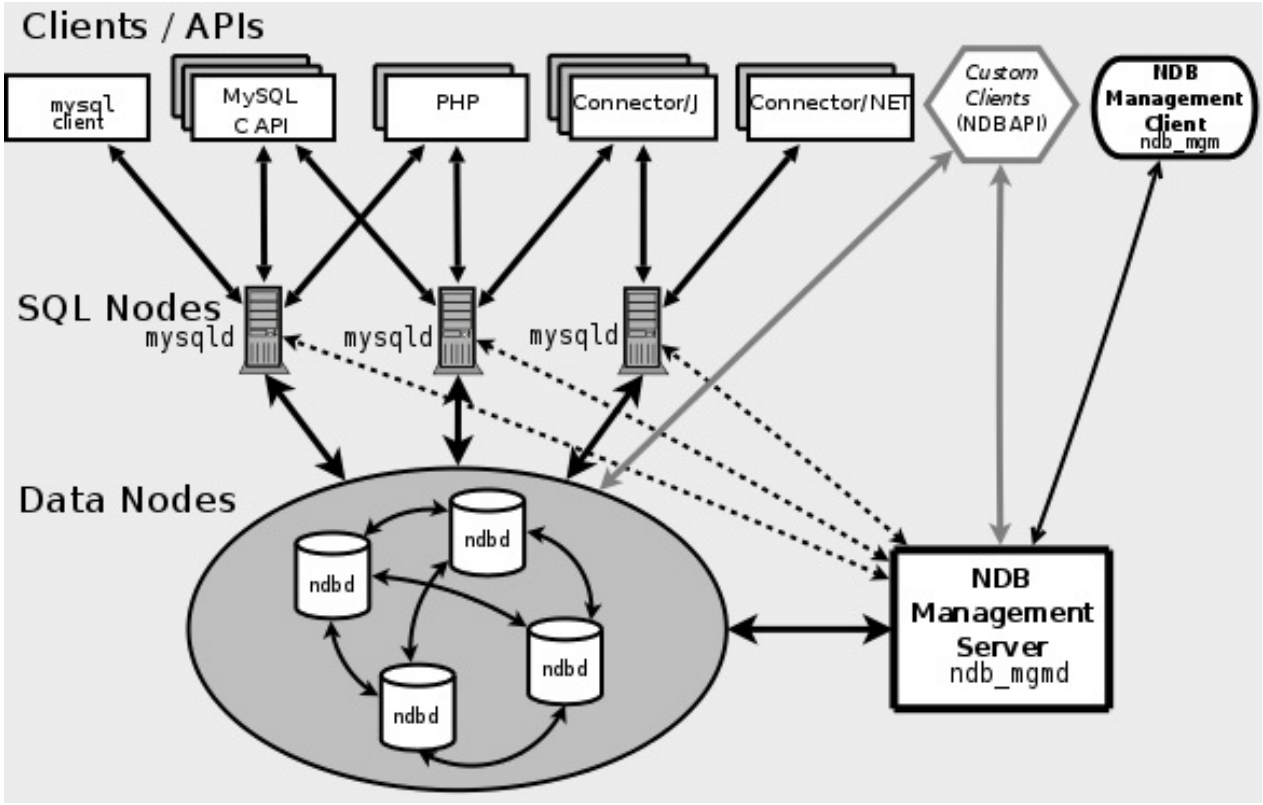
数据库存储层采用集群的方式来保证数据库服务的高可用性，Web访问层采用Nginx做负载均衡，通过反向代理的方式挂载多台Tomcat服务器对外提供Web服务。当然，也可以采用F5等通过硬件来做负载均衡，提高Web服务器性能的方式。

MySQL集群搭建

官方文档 from Orcale.com

MySQL Cluster是一个高性能、可扩展、集群化数据库产品。MySQL Cluster是一个基于**NDB Cluster**存储引擎的完整分布式数据库系统，采用无共享的数据存储技术，实时同步且支持快速故障切换、事务。NDB是一种in-memory的存储引擎，它具有可用性高和数据一致性好的特点。

MySQL Cluster可由多台服务器组成的、同时对外提供数据管理服务的分布式集群系统。通过合理的配置，可以将服务请求在多台物理机上分发实现**负载均衡**；同时内部实现了**冗余机制**，在部分服务器宕机的情况下，整个集群对外提供的服务不受影响，从而能达到**99.999%**以上的高可用性。



如上图所示，MySQL Cluster分为三类节点：

数据节点（Data Nodes）：用于存储集群的数据。实现底层数据存储的功能，保存Cluster的数据。每一个NDB节点保存完整数据的一部分（或者一份完整的数据，视节点数目和配置而定），在MySQL Cluster 里面叫做一个fragment。而每一个fragment，正常情况来讲都会在其他的主机上面有一份（或者多份）完全相同的镜像存在。这些都是通过配置来完成的，所以只要配置得当，MySQL Cluster 在存储层**不会出现单点**的问题。数据节点是用命令`ndbd`启动的。

SQL节点（SQL Nodes）：向外提供一个标准的SQL语言编程接口。SQL节点负责向数据节点传送访问请求，具体集群过程以及数据库底层均对外透明。

SQL节点提供用户SQL指令请求，解析、连接管理，query优化和响应、cache管理等、数据merge、sort，裁剪等功能，当SQL节点启动时，将向管理节点同步架构信息，用以数据查询路由。SQL节点作为查询入口，需要消耗大量cpu及内存资源，可使用分布式管理

节点，并在SQL节点外封装一层请求分发及HA控制机制可解决单点及性能问题，其提供了线性扩展功能。SQL节点是使用命令 `mysqld -ndbcluster` 启动的，或将 `ndbcluster` 添加到“my.cnf”后使用“mysqld”启动。

管理节点（NDB Management Server）：负责整个Cluster 集群中各个节点的管理工作，包括集群的配置，启动关闭各节点，以及实施数据的备份恢复等。管理节点会获取整个Cluster 环境中各节点的状态和错误信息，并且将各Cluster集群中各个节点的信息反馈给整个集群中其他的所有节点。通常只需配置一个管理节点；然而为了排除单点故障需要，有可能的话，尽量增加管理节点的数量。MGM节点是用命令 `ndb_mgm` 启动的。

Mysql集群软件：mysql-cluster-gpl-7.2.8-linux2.6-x86_64.tar.gz 下载地址

详细配置步骤 from SegmentFault

MongoDB集群搭建

MongoDB在处理非结构化数据时有以下优势：

大数据量，可以通过廉价服务器存储大量的数据，轻松摆脱传统mysql单表存储量级限制。

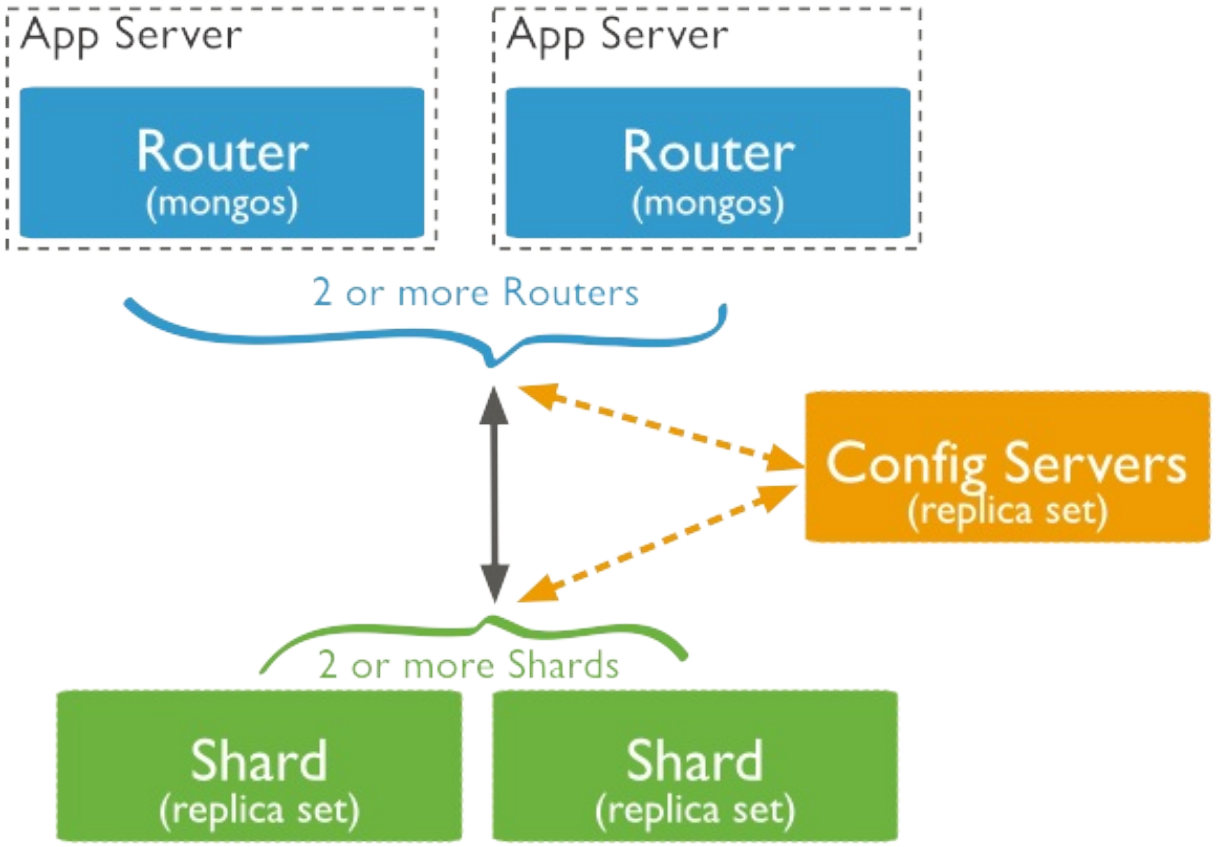
高扩展性，MongoDB去掉了关系数据库的关系型特性，很容易横向扩展，摆脱了以往老是纵向扩展的诟病。

高性能，MongoDB通过简单的key-value方式获取数据，非常快速。还有MongoDB的Cache是记录级的，是一种细粒度的Cache，所以MongoDB在这个层面上来说就要性能高很多。

灵活的数据模型，MongoDB无需事先为要存储的数据建立字段，随时可以存储自定义的数据格式。而在关系数据库里，增删字段是一件非常麻烦的事情。如果是非常大数据量的表，增加字段简直就是一个噩梦。

高可用，MongoDB在不太影响性能的情况，就可以方便的实现高可用的架构。MongoDB可以通过mongos、mongo分片就可以快速配置出高可用配置。

MongoDB也可以通过集群的方式实现高可用，MongoDB的集群配置包括以下组件，如图所示： 资料来源：mongodb.org



配置服务器Config Servers： 在最新的MongoDB3.2中，分片集群的配置服务器可以部署为副本集。副本集的配置服务器必须运行WiredTiger 存储引擎。单一的分片集群必须独占使用其配置服务器。如果您有多个分片集群，每个集群必须拥有专有的的副本集配置服务器。

两个或更多副本集作为分片服务器： 这些副本集是碎片。随后会详细介绍如何创建副本集。

一个或多个查询路由器(mongos)： mongos实例是为集群的路由器。通常情况下，每个应用服务器需要部署一个mongos实例。 也可以部署一组的mongos实例并在应用和mongos之间采用代理/负载均衡器。在这些部署中，您必须配置负载平衡器客户端的相似性，以便从单个客户端的每个连接都能指向相同的mongos。因为游标和其他资源是特定于单个mongos实例的，每个客户端只能与同一个mongos实例进行交互。

MongoDB副本（主从节点）配置

下面看一下怎么一步步搭建一个mongodb的主从复制节点：操作参考

1 准备两台机器 192.168.0.1 和 192.168.0.2。 192.168.0.1 当作主节点， 192.168.0.2作为从节点。

2 分别下载mongodb安装程序包。在192.168.0.1上建立文件夹/data/mongodbtest/master，192.168.0.2建立文件夹/data/mongodbtest/slave。

3 在192.168.0.1启动mongodb主节点程序。注意后面的这个 “ –master ”参数，标示主节点。

`mongod –dbpath /data/mongodbtest/master –master` 输出日志如下，成功!

```
[initandlisten] MongoDB starting : pid=18285 port=27017 dbpath=/data/mongodbtest/master maste
```

日志显示主节点参数

```
[initandlisten] options: { dbpath: “/data/mongodbtest/master”, master: true }
[initandlisten] waiting for connections on port 27017
```

4 在192.168.0.2启动mongodb从节点程序。关键配置，指定主节点ip地址和端口 –source 192.168.0.1:27017 和 标示从节点 –source 参数。

```
mongod –dbpath /data/mongodbtest/slave –slave –source 192.168.0.1:27017
```

输出日志如下，成功!

```
[initandlisten] MongoDB starting : pid=17888 port=27017 dbpath=/data/mongodbtest/slave slave=
```

日志显示从节点参数

```
[initandlisten] options: { dbpath: “/data/mongodbtest/slave”, slave: true, source: “192.168.0
[initandlisten] waiting for connections on port 27017
```

日志显示从节点 从主节点同步复制数据

```
[replslave] repl: from host:192.168.0.1:27017
```

成功!