开发者在项目中该如何定位自己？

从毕业到现在一直工作在所在的瑞士投资外企公司，当初刚刚进入公司也是作为一个普通的前端开发者，负责当时我们公司海外平台的开发和维护。和最初的进入职场的人一样，开始也是各种技术各种不会。但是我是一个从不服输的人，也正是自己的韧性和努力坚持走到了今天。可能和其他人一样都是在不断做项目和开发中提高自己的。但是我在进入公司的第一天给自己的定位不仅仅是前端开发。而是把自己定位一个开发者.

为何是开发者

开发者在公司项目中不是一个固定的职位，是以解决问题为目的的，不论你是前端开发还是后端开发，只要遇到了问题你就去解决，虽然在这个过程中你可能遇到你不会的，超出你能力范围的问题，但是正是这个阶段不断锻地炼了你的思维和提升了你自己发现问题解决问题的能力。这样你可以参与到后端表模型创建，容器镜像使用，微服务的构建等等。正是这些你的参与项目和学习机会，才会使得自己进步的！

前端开发者的任务

前端开发主要工作是用web技术（如html,css,js）设计和渲染页面开发，实现UI页面效果和数据交互。
第一次在公司做项目是用的angularjs。当时也用过和了解过angularjs。

Angularjs是一款开源的javascript库，最初是由google维护，用来协助单一页面应用程序运行的，它的目标是通过MVC模式(MVC)功能增强基于浏览器的应用，使开发和测试变得

更加容易。Angular在呈现和数据中间，可以简单创建双向的数据绑定。一旦创建双向绑定，用户输入，会由Angular自动传到一个变量中，再自动读到所有绑到它的内容，更新它。效果上就是立即的数据同步。在代码中修改变量，也会直接反应到呈现的外观上。配合使用还有它的路由ui.router组成的SPA页面，效率不仅快而且可以实现无刷新加载数据。

当时基于这些优点我们的平台开发就选定了angularjs，对于开发大型车辆后台控制系统的时候使用angularjs可以快速的进行数据的渲染。在前端直接用 $http指令封装它的ajax异步请求，然后前端模拟数据用的mock,这样不仅提高了开发的效率，而且在快速开发的过程中我们也可以用后端的脚手架生成一个前端对应的模板，实现快速开发。可能这样和它的理念有关系，**Angularjs的创建的理念**就是：即声明式编程应该用于构建用户界面以及编写软件构建，而命令式编程非常适合来表示业务逻辑，框架采用并扩展了传统的HTML，通过双向的数据绑定来适应动态内容，双向的数据绑定允许模型和视图之间的自动同步。这样做避免了DOM的操作，也避免了很多冗余的js代码。

这样的好处可以实现：

1. 将应用逻辑与对DOM的操作解耦。这会提高代码的可测试性。
2. 将应用程序的测试看的（得）跟应用程序的编写一样重要。代码的构成方式对测试的难度有巨大的影响。
3. 将应用程序的客户端与服务器端解耦。这允许客户端和服务器端的开发可以齐头并进，并且让双方的复用成为可能。
4. 指导开发者完成构建应用程序的整个历程：从用户界面的设计，到编写业务逻辑，再到测试 Angular遵循软件工程的MVC模式，并鼓励展现，数据，和逻辑组件之间的松耦合。通过依赖注入（ dependency injection ），Angular为客户端的Web应用带来了传统服务端的服务，例如独立于视图的控制。这样前端开发也避免了很多负担，产生了更轻的Web应用。

下面试我封装的$http的ajax异步请求方法：

```
function Ajax() {
      this.$get = function ($q, $http) {
          var self = this;
          return {
              GET: function (url) {
                  var d = $q.defer();
                  $http({
                      method: 'GET',
                      url: UrlLocation+url,
                      cache: false
                  }).success(function (data) {
                      d.resolve(data);
                  }).error(function (err) {
                      d.reject(err);
                  });
```

```
            return d.promise;//返回一个promise对象
        },
        POST: function (url, data) {
            var d = $q.defer();
            $http({
                method: 'POST',
                url: UrlLocation+url,
                data: data,
                cache: false
            }).success(function (data) {
                // Wunderground API返回
                //嵌套在forecast.simpleforcast属性内的
forecasts对象
                d.resolve(data);
            }).error(function (err) {
                d.reject(err);
            });
            return d.promise;
        },
        PUT: function (url, data) {
            var d = $q.defer();
            $http({
                method: 'PUT',
                url: UrlLocation+url,
                data: data,
                cache: false
            }).success(function (data) {
                d.resolve(data);
            }).error(function (err) {
                d.reject(err);
            });
            return d.promise;
        },
        DELETE: function (url) {
            var d = $q.defer();
            $http({
                method: 'DELETE',
                url:UrlLocation+ url,
                cache: false
            }).success(function (data) {
                d.resolve(data);
            }).error(function (err) {
                d.reject(err);
            });
            return d.promise;//返回一个promise对象
        },
        FILEUPLOAD: function (url, data) {
            var d = $q.defer();
            $http({
                method: 'POST',
                url: UrlLocation+url,
                data: data,
```

```
                    headers: {'Content-Type': undefined},
                    cache: false
                }).success(function (data) {
                    d.resolve(data);
                }).error(function (err) {
                    d.reject(err);
                });
                return d.promise;//返回一个promise对象
            }
        }
    };
}
```

在我们开发的过程中，让我们最舒服的就是它的数据双向绑定这块，因为这个是一个很大的便利，对于一些复杂的逻辑的电商Sku和Spu数据量非常大，通常一个请求返回的数据机构都是json套对象，套json等的好多层级，这简直是开发者使用起来最愉悦的事情。还有数据交互很庞大的车辆型号的后台系统。特别是对于电商和一些金融后端网站，也会涉及到很多的权限问题和用户角色，类似的是特定的角色才能有特定的权限，这样用angularjs来处理也是很棒的，因为angular的指令directives用来实现这样的效果会有优势，根据不同的级别设定特定的权限，利用自定义指令，这样在开发过程中会帮助开发者实现效率的提高，也方便了开发者进行快速开发。

前端开发者如何更好的去融入后端开发

作为一个前端开发者需要更好的去理解后端，如果有能力可以参与到后端的开发中，在我们团队当中我们没有把一个人的位置固定死，比如就是认定谁是前端开发者，或者谁是后端开发者。我们团队的成员都是开发者。**何为开发者呢**？就是只要遇到问题就去解决，不分你是做前端还是做后端。前端要会后端的语言，理解后端的开发和定义接口，以及定义的结构体（ps：我们公司后端都是golnag开发）。可能大家会说，团队中都是开发者，前端和后端分割不就没有啥明显区别的了吗？实际不是的，我们团队开发人员除了每个人掌握的专项技能之外，这些前端的或者后端的都是在工作中随着不断地做项目和业务过程中去磨合，也在不断的深化和拓展自己的知识面，和开发技能。**因为我们公司的理念就是，为了所有人的发展，让他人变得的伟大。这样你就会更加的伟大！**都是互相帮助开发的，有问题了大家就去解决。这样也是对一个人职业和知识面的拓展，业务虽然是公司的，但是学到的知识和本领全是自己的。

举个例子：我们团队成员有段时间一个做项目的前端人员生病了最后的关键，还有3个模块没有实现，我们后端的接口写的都差不多了，但是前端的做的系统界面还有没有做完的。这个时候可能其他人会说团队项目的前端人员没有了这个怎么做。但是我们的团队的后端开发就会很快填补这个空缺，我们要求就是前端要会后端的开发语言，后端也要会前端的开发技能，这样大家都是开发者，当其他人生病或者请假了，团队的人可以快速替代帮忙开发。这样在开发的过程中有了问题的话，前端直接回给出一个很明显的信息反馈给后端，哪个地方接口没有定义好，后端直接就会改好。或者有时候如果后端在忙，前端开发者就会直接自己去改下后端接口，然后直接说问题，包括解决方法。TODO做好提交项目分支代码，会有专门检查，然后负责检查的人看了没问题就会Done掉这个

问题。这样一个流程下来，团队的开发效率和工作的能力，包括对业务的理解能力都会在不断的上升。其实这样看来，这样的成长对一个团队和人来说都是很快的，几个项目下来，团队的开发者的能力的提升都是很显著的。

如何做好一个开发者

但是可能有人要问了作为一个前端开发者如何去做到从前端做到后端，到真正的成为一个开发者。
只有在你们公司内部真正的在掌握你擅长的基础上融合了前端和后端的技术，不断的去在做项目和用技术的过程中去领悟，和探索才能做好一个开发者，但是这个正是我们的成长。

那么前端如何学习后端开发语言，快速成长呢？

在我介绍这个问题之前我先给大家讲述下我看到的一个国外的文章对于程序开发人员的五个进化阶段:

- 第一个阶段：小白阶段，刚刚学习了一门语言。可以写短的代码片段。
- 第二个阶段：探索者阶段，可以写一个完整的程序，但不懂一些更高级的语言特征，和语言特性。
- 第三个阶段：你能熟练的使用开发语言，能够用开发语言去解决，生产环境中的一个具体和完整的问题，已经形成了一套自己的惯用法和常用代码库，在你的编码方案中，你有一套自己的逻辑思维和运用开发的工具。
- 第四个阶段：大神阶段.，清楚你所用的语言的设计，选择和背后的动机。能用你自己的极其简洁的语言和可组合性的哲学去讲述和分享。
- 第五个阶段：布道师，积极第和他人分享关于你对开发语言的知识，和你对开发语言的理解。在各种合适的场所发出自己的声音，参与邮件列表，做专题报告，参加语言标准开发版本规则的制定。

上面这个也有的是我的体会，其实看到这个仿佛就可以看到自己对语言的认识和理解。如果你在一个语言开发过程中有了你自己的理解达到了上述的五个阶段的中的某个阶段，你再去用你自己的方式去学习一门新的语言的话，时间应该会很快的，因为你的能力已经有了，剩下的就是看你自己对语言语法的熟悉和使用了。

那么作为前端开发者到底如何去学习go呢？因为我们公司用的是go后台开发，包括微服务的架构，所以我用go来作个比喻。可以根据你们自己公司的后端语言去实现。

Go是谷歌开发的一种静态类型，编译型，并发型，并具有垃圾回收功能的编程语言。Go语言的语法接近C语言，但对于变量的声明有所不同。Go语言支持垃圾回收功能，C++相比，Go语言并不包括如异常处理、继承、泛型、断言、虚函数等功能，但增加了 Slice 型、并发、管道、垃圾回收、接口（Interface）等特性的语言级支持，go2.0目前仍正在

讨论是否应该支持泛型，其态度还是很开放的，但在该语言的常见问题列表中，对于断言的存在，则持负面态度，同时也为自己不提供类型继承来辩护。

不同于Java，Go语言内嵌了关联数组（也称为哈希表（Hashes）或字典（Dictionaries）），就像字符串类型一样。

使用go开发的人对它的第一印象都是简洁，优雅，语法简单，上手快。

对go有了简单的认识之后你需要开始的去了解下go的语法和规则，接着就是开启你的go之旅了。
学习一门语言最好的办法就是去用，去实践。

在你接触go语言之后你就要开始去参与项目了，如果你们公司后端是其他语言开发，或者你学的其他语言，是一样的道理，你就要以你所用的去参与了。记住一点在公司你能学到的都是你主动去争取的，别人不会亲自来教授的。

可能很多人都喜欢写博客啊，写网站来做个总结的等等都可以的。你可以用go做个自己的网站，基础语法你会了之后你就是需要在实践使用go开发了。我推荐的go做的web项目使用的技术可以从这里开始 **go + gin + gorm**，这样的类型的组合，可以做出你想要的很炫酷的网站，而且性能也很好。

Go的语法和基础知识：命名，在go语言中函数名，变量名，常数名，类型名，语句标号和包名，都遵循一个简单的命名规则：一个名字必须以一个字母或下滑线开头，后面可以跟任意数量的字母，数字或下滑线。在go语言中类似if和switch的关键字有25个：关键字不能用于定义名字。

break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

Go的声明方式： 在go中主要有四种类型的声明语句： var，const，type，func 这四个类型分别对应的是变量，常量，类型和函数实体对象的声明。

```go
type AuthorityController struct {
    ApplicationController
}

const (
    AuthorityKey = "ALLIANCE::AUTHORITY"
)

func (ctrl AuthorityController)New(c *gin.Context) {
    ctrl.Render.HTML(c.Writer, http.StatusOK, name: "authority/index", binding: nil, render.HTMLOptions{Layout: "authority/templates"})
}

func (ctrl AuthorityController)All(c *gin.Context) {
    resp := ctrl.NewResponse()
    allianceRedis := redis_authories.Alliance{Redis:ctrl.RedisPool, Key:AuthorityKey}
    authories := make([]redis_authories.Authority, 0)
    s:=allianceRedis.GetAll()
    err := json.Unmarshal([]byte(s), &authories)
    if err != nil {
        resp.ErrorCode = 1
        resp.ErrorMessage = err.Error()
        c.JSON(http.StatusOK, resp)
        return
    }
    resp.ErrorCode = 0
    resp.Data = authories
    c.JSON(http.StatusOK, resp)
}
```

接着学习go的赋值语句，类型，包和文件，基础数据类型(整形，浮点数，复数，布尔类型，字符串，常量），复合数据类型(数组，slice，map，结构体，json，文本和html模板)，函数（函数声明，异常处理和panic异常等）最后在看下方法和Interface，这样基本你已经入门了go。

接下来我想重点讲的是如何用go +gin +gorm做个项目，这样前端开发就真正的融入到golang的后端开发了。

gin 是一个go语言的一个web框架，它类似Martini。不懂的可以看下github上关于gin使用：

https://github.com/gin-gonic/gin

gin通过精心的设计与技术实现，集成了大部分稳定开发组件。

Gin的特点：

1. 功能丰富 ：支持大部分服务器组件，支持API Doc。
2. 得心应手 ：简单的实例，非常容易上手。
3. 深度整合 ：深度整合memcache，redis，mysql，beego ，gin 框架。
4. 性能优先 ：为啥不直接使用beego那，因为gin是目前httprouter性能最高，占用资源最少的框架，而beego的 orm是目前最接近django的框架。

它的安装使用很简单：只需要：go get gopkg.in/gin-gonic/gin.v1

然后绕在你的项目中导入：import “gopkg.in/gin-gonic/gin.v1”

可以先用gin实现一个简单的hello world

```go
package main

import (
    "github.com/gin-gonic/gin"
    "net/http"
)

func main(){
    router := gin.Default()
    router.GET( relativePath: "/", func(c *gin.Context){
        c.String(http.StatusOK, format: "Hello World")
    })
    router.Run( addr: ":3000")
}
```

然后运行go run mian.go就可以看到你用gin实现的了。

它的接口API的使用也是很简单：

```go
func main() {
    router := gin.Default()
    router.GET("/Get", getting)
    router.POST("/Post", posting)
    router.PUT("/Put", putting)
    router.DELETE("/Delete", deleting)
    router.PATCH("/Patch", patching)
    router.HEAD("/Head", head)
    router.OPTIONS("/Options", options)
    router.Run()
}
```


有没有觉得写的很优雅呢？

Gin不仅可以设置APi的路由配置，还可以用来设置middleware中间件代码以及生成的api的文档代码，此外gin还可以在你的控制器层面写你的APi接口，有没有感觉到很棒啊！！因此gin是你写go web项目的不错的选择。

Gin的代码块的实现：

```
func (ctrl UsersController) Show(c *gin.Context) {
    var user models.User
    err := ctrl.DB.First(&user, c.Param( key: "id")).Error
    if err != nil {
        ctrl.Render.Text(c.Writer, http.StatusOK, err.Error())
    } else {
        ctrl.Render.HTML(c.Writer, http.StatusOK, name: "use/show", gin.H{
            "title":"show",
            "user":user,
        }, render.HTMLOptions{Layout: "template_layout"})
    }
}
```

这个是gin结合gorm进行的查询数据表user，这也是为何推荐gin+gorm的原因。

那么GORM又是什么呢？

GORM 全程是 Grails' object relational mapping (ORM) 是 Grails 的 ORM 实现，也就是对象关系映射（object-relational mapper，ORM）gorm 地址：https://github.com/jinzhu/gorm。

grails 中的gorm框架是基于 hibernage，就是在hibernate基础上进行了一层薄薄的封装。

GORM使用的好处是：

1. 全功能ORM（几乎）关联包含一个，包含多个，属于，多对多，多种包含）Callbacks（创建/保存/更新/删除/查找之前/之后）预加载（急加载）事务复合主键等等。

2. 在项目中使用orm的好处还是很多可以防止直接拼接sql语句引入，sql注入漏洞，方便对modle进行统一管理，专注于业务，加速快速开发的效率。

那么接下来就是快速开始了：

```go
import (
    "github.com/jinzhu/gorm"
    _ "github.com/jinzhu/gorm/dialects/sqlite"
)

type Product struct {
    gorm.Model
    Code  string
    Price uint
}

func main() {
    db, err := gorm.Open( dialect: "sqlite3",  args: "test.db")
    if err != nil {
        panic( v: "连接数据库失败")
    }
    defer db.Close()

    // 自动迁移模式
    db.AutoMigrate(&Product{})

    // 创建
    db.Create(&Product{Code: "L1212", Price: 1000})

    // 读取
    var product Product
    db.First(&product,  where: 1)                 // 查询id为1的product
    db.First(&product,  where: "code = ?", "L1212") // 查询code为l1212的product

    // 更新 - 更新product的price为2000
    db.Model(&product).Update( attrs: "Price", 2000)

    // 删除 - 删除product
    db.Delete(&product)
}
```


使用gin和gorm框架去定制你自己的前端项目，然后积累经验，参与到后端的开发，完成一次自己的提升！

这样学习下来你先从做一个简单的go后端项目，而且是很标准化的企业开发的模式和框架使用，很快你就会掌握和使用golang开发，从一个前端开发人员做到后端开发，然后成长为一个真正的开发者，在项目团队中遇到问题你就可以快速的去解决的！

微服务

微服务是一种架构风格，一个大型复杂软件应用由一个或多个微服务组成。系统中的各个微服务可被独立部署，各个微服务之间是松耦合的。每个微服务仅关注于完成一件任务并很好地完成该任务。在所有情况下，每个任务代表着一个小的业务能力。 微服务的概念源于2014年3月Martin Fowler所写的一篇文章“Microservices”(http://martinfowler.com/articles/microservices.html)。

尽管“微服务”这种架构风格没有精确的定义，但其具有一些共同的特性，如围绕业务能力组织服务、自动化部署、智能端点、对语言及数据的“去集中化”控制等等。

为什么需要微服务架构

1. 使用传统的整体式架构(Monolithic

   Architecture)应用开发系统，如CRM、ERP等大型应用，随着新需求的不断增加，企业更新和修复大型整体式应用变得越来越困难。

2. 随着移动互联网的发展，企业被迫将其应用迁移至现代化UI界面架构以便能兼容移动设备，这要求企业能实现应用功能的快速上线。

3. 许多企业在SOA投资中得到的回报有限，SOA可以通过标准化服务接口实现能力的重用，但对于快速变化的需求，受到整体式应用的限制，有时候显得力不从心。

4. 随着应用云化的日益普及，生于云端的应用具有与传统IT不同的技术基因和开发运维模式。

此外从技术来看，云计算以及一些互联网公司大量开源请两级技术，互联网更加的成熟，包括新的方法和工具以及Restful Api 接口，轻量级消息机制还有基础设施容器化平台等等这一切都是催生了新的架构设计风格，微服务架构的出现。

微服务架构的优点

1. 每个服务都比较简单，只关注于一个业务功能。

2. 微服务架构方式是松耦合的，可以提供更高的灵活性。

3. 微服务可通过最佳及最合适的不同的编程语言与工具进行开发，能够做到有的放矢地解决针对性问题。每个微服务可由不同团队独立开发，互不影响，加快推出市场的速度。

4. 微服务架构是持续交付(CD)的巨大推动力，允许在频繁发布不同服务的同时保持系统其他部分的可用性和稳定性。

5. 服务整合可以快速的整合和使用。

微服务使用

1. 在一定的条件下，合适的项目，合适的团队，可以采用微服务架构。

2. 微服务架构有很多吸引人的地方，但在拥抱微服务之前，也需要认清它所带来的挑战。

3. 微服务架构引入，也要考虑公司的业务模式和开发，以及在架构的过程中逐步探索和积累微服务架构经验。

这些是微服务的简介但是不同的语言实现微服务的方法不同，go的实现我们是用的go-kit来做的：https://gokit.io/这个微服务我们可以在交流的时候好好在聊，因为更多的是关于实现的，这里就不再累述。