**健康管理平台架构设计**

1. **上下文**



1. **质量属性**
   1. **性能**

在资源数据量为1000万，注册用户数为500万，QPS(每秒请求数)为1万的情况下，首页用户请求的响应时间小于3秒，资讯类页面请求的响应时间小于3秒。展示心电图页面在QPS为1000时响应时间小于3秒。

测试的环境：

负载均衡lvs，2个Ngnix做主备，16个Tomcat(apr模式)实例，Fastdfs：2 Tacker + 2 Storage，Oracle rack数据库，3台分布式服务Dubbox的服务器（2 provider + 1 zookeeper），2台缓存redis服务器（主备），2台静态页面Ngnix服务器。

* 1. **可用性**

可接受99.99%.

升级不停服务：

小升级使用依次升级和重启web应用和dubbo服务提供者的方式。

大升级（界面变化大）使用灰度发布的方式，逐步把用户引导到新系统上。

* 1. **安全性**
* 作为项目部署使用http协议，作为互联网应用需局部使用https：登录页面。
* 对用户使用基于角色的访问控制来保证安全。
* 数据权限

1 配置权限规则：

需使用部门，角色等配置规则。

数据规则表：类名，规则，增，删，改，查。

资源字典表：资源名，类名，表名，



2 使用MyBatis拦截器拦截StatementHandler的prepare方法，动态拼接根据权限规则得到的SQL条件。

配置复杂，在目前的健康管理平台中不是必需的。建议不使用。

* API服务安全

对外部系统(包括公司其他系统)使用API 密钥的方式来保证安全。

在API调用时按照指定规则使用API 密钥对请求参数进行签名，服务提供方收到请求时会进行签名验证，既可以界定身份也可以防止其他人通过某种手段恶意篡改请求数据。

服务消费者签名规则：参数中加上固定的apikey, timestamp(时间戳)，nonce(随机字符串)，signature（签名），signature的计算规则：把secret(密钥), timestamp, nonce排序，把排序后的字符串数组拼接为一个字符串，对该字符串进行sha1加密得到的字符串就是signature。

服务提供者签名验证：根据apikey查询数据库得到secret与最近一次调用的参数timestamp，若本次参数timestamp<=最近一次timestamp，则是伪造调用，否则再用与签名同样的方法得到signature，看是否等于参数中的signature，若相等，则是合法请求，否则非法。

* 网络安全

防范横向攻击的策略：通过API来调用的系统都需进行安全认证。

跨站点攻击（XSS）：前端解决。

恶意访问：使用过滤器对恶意访问拦截，如封IP或账户5分钟。恶意访问判断标准：请求频率，错误频率（http状态码）等。

敏感数据（如id）防篡改：后端把敏感数据加密后传到前端，前端把数据传到后端后，后端先解密再处理。

* 1. **国际化**

用户界面支持中文和英文。界面配成中文。数据库的数据（如：下拉框的选项）目前不必支持。

* JSP

在类路径中放各种语言和默认的文字属性文件，在页面用如下语句显示相应语言：

ResourceBundle myResourcesBundle = ResourceBundle.getBundle("me.gacl.i18n.resource.myproperties",request.getLocale());

<%=myResourcesBundle.getString("username")%>

* 静态页面

使用开源js库：jquery.i18n.properties-1.0.9.js

在静态资源文件夹（如i18n）下放各种语言和默认的文字属性文件，使用如下方法显示中文：

jQuery.i18n.properties(..).

如：

<script type="text/javascript">

$(function(){

jQuery.i18n.properties({

name : 'strings', //资源文件名称

path : '/i18n/', //资源文件路径

mode : 'map', //用Map的方式使用资源文件中的值

language : 'zh',

callback : function() {//加载成功后设置显示内容

$('#button-login').html($.i18n.prop('Login'));

$('#label-username').html($.i18n.prop('User Name'));

$('#label-password').html($.i18n.prop('Password'));

}

});

});

</script>

* 1. **浏览器兼容性**

由于面向互联网，目前主流浏览器都需要支持。具体包括：IE8以上，Firefox，Chrome，360浏览器，QQ浏览器，搜狗浏览器，猎豹浏览器。

1. **约束**

* 预算
* 时间
* 技术

JavaEE；

使用HTTP REST协议兼容已有的对外部系统的接口；

部署到Linux服务器上；

Oracle JDK7；

CentOS 7；

spring Mvc；

spring。

* 人员

人数：

技术水平：3年以上经验

1. **原则**
   1. **开发原则**

* 遵守统一的代码规范
* 代码中执行SQL语句使用预处理方式：PreparedStatement，目的是防止SQL注入攻击。
* 单元测试

controller方法需做。

* 接口规范

需要正常模式和debug模式，在调试阶段使用debug模式可以获取更多的错误提示信息。

需为测试人员开发接口测试程序，方便对接口的测试。

* 日志

向前端返回错误信息时，使用错误码和简明的提示信息，不需返回完整的错误信息。在日志中需打出错误码和完整的错误信息备查。

* 构建

分两种版本：集群版和非集群版。

非集群版系统由2个Tomcat服务器（web后端服务器和接口服务器）和Oracle服务器组成。因此需要把集群版的所有服务的配置文件（除provider配置文件和MyBatis mapper文件外）合并，连同集群版的前端资源文件和所有服务的jar文件及其依赖的jar文件打到web应用的war文件中，web.xml中需配置加载合并后的spring应用上下文配置文件，不需配置consumer配置文件。代码上，要把对文件的存取做成两种实现：分布式文件方式和本地文件方式，非集群版使用本地文件方式，在Spring应用上下文中配置使用的方式。

服务的jar文件打包方法：选择输出JAR file，而不是Runnable JAR file。只打包sr/main/java，由于根路径上的资源文件已手工合并到war中了，就不用打包它们了。打包时要选择选项：Add directory entries，这样就可以使用jar中的资源文件了。

* 1. **架构原则**
* 分层策略

分为三层：展现层（Spring MVC），服务层(Spring)和数据库访问层(MyBatis)。在服务层有持久化对象PO，在展现层有视图对象VO（可选），在展现层进行二者的转换。

|  |
| --- |
| 展现层  （Spring MVC 4.2.8） |
| 服务层  (Spring 4.2.8) |
| 数据库访问层  (MyBatis 3.3.1) |

数据库连接池用druid 1.0.26。

* 全文检索目前不需要。
* 模块高内聚，模块间低耦合

使用三方库都需要封装后使用，这样可以解耦，便于以后对其替换。

* Http session使用

使用统一的分布式session库来处理session。使用shiro的redis缓存库。见“9.1.分布式session库”。

* 按组件(或模块)建包

组件包内部再分层。

* 依赖注入

优先使用Spring注解装配方式，必要时使用xml配置。

* 使用微服务即分布式服务来实现服务层

服务可以组合。为了使用统一的事务控制，组合在服务层实现，展现层对每个请求只能调用1个服务。服务分为单一模块的服务和跨模块的服务，对前者使用JDBC的事务控制，对后者需在服务内编码实现事务控制。



* 前后端分离

视图模板引擎使用前端js模板引擎，这样就可以把页面文件都放到Ngnix服务器上来支持高并发访问。

* 支持多种数据库

目前需支持两种数据库Oracle和MySQL。以后数据量增加到1千万条（MySQL的极限）时需支持使用NoSQL数据库MongoDB。

为支持Oracle和MySQL，需建两个mybatis-config.xml文件：mybatis-config-oracle.xml和mybatis-config-mysql.xml，分别支持这两种数据库。数据库访问层有两套mapper实现：MySQL和Oracle。每个mapper文件都把两种实现所公用的方法放在文件上部并加注释说明，这样便于基于一种实现开发另一种实现。在Spring上下文配置文件中，根据要使用的数据库来配置对应的mybatis-config.xml文件。



* 对外提供的API版本升级时要向下兼容
* 字符编码

为避免编码不一致导致的乱码问题，从页面到数据库的字符编码都统一使用UTF-8。

具体来说，包括以下方面：

1. 项目源码都使用UTF-8；

2. 数据库字符集使用UTF-8；

3. Ngnix中需设置charset为UTF-8；

4. Tomcat的server.xml 中设定 useBodyEncodingForURI="true"，URIEncoding="utf-8"；

5. html文件的header中加上：<meta charset="utf-8">；

6. 前端AJAX发起请求时，对URL或者数据中的非ASCII字符使用javascript的encodeURIComponent进行编码；

7. 在后端controller中，对要返回响应的方法加上设置：charset=UTF-8。

* API接口debug模式与正常模式

在API工程中配置properties文件，设置是否使用debug模式。在debug模式下，系统会把字段校验等具体错误信息以错误消息的方式返回给调用方。

1. **软件架构**
   1. **容器图**
      1. **集群版**



* + 1. **非集群版**



1. **决策日志**
   1. **分布式session库**

候选方案有：

* shiro-redis分布式缓存库

与shiro session可以无缝集成，因本身就是shiro缓存的一种实现。

* Spring session库

是Spring开源项目。把所有session存入redis缓存中。应用服务器中立，设计简单。与shiro session有冲突。

* tomcat-redis-session-manager

把所有session存入redis缓存中。个人项目，已经不维护了。发布的jar有bug，不能直接用，需手工编译源代码。与Tomcat绑定。设计复杂，维护难度大。

从便于维护角度考虑，建议选择shiro-redis库。

* 1. **前后端分离**

候选技术：

* react.js

facebook主导，轻量级，性能好，编程模型简单：使用界面组件的方式组合出界面，不需使用模板，降低复杂应用的编写难度。将来能与手机APP统一。社区非常活跃，可能是将来的主流前端框架。

成功案例：Facebook，Paypal，Instagram，Netflix，BBC，Yahoo的Email，支付宝新一代的框架基于React，淘宝订单页面，知乎的评论功能前端等，详见： <https://github.com/facebook/react/wiki/Sites-Using-React>。

* AngularJS

Google主导，编程模型复杂，大而全，性能较差。

* artTemplate

腾讯主导，简单易用，成功案例：QQ个人空间。

* baiduTemplate

百度主导，成功案例：百度贴吧。

* velocity.js

阿里主导，版本<1.0。成功案例：淘宝。

从长远考虑，选择react.js比较好，但目前项目时间紧，建议选择使用简单的artTemplate。

* 1. **数据库访问层的实现**

不能使用如下方案：对每个DAO接口，公用的实现放在一个mapper文件中，数据库特有的实现放在另一个mapper文件中。因为Mybatis缓存不支持一个DAO接口对应多个mapper文件。

* 1. **数据库选型**

互联网平台选用MySQL,大客户倾向于用oracle，小客户如社区使用MySQL。集群版MySQL需使用Percona XtraDB Cluster,对公司来说是新技术。新版本数据库迁移是从Oracle迁移到新数据库，如果一次性迁移到XtraDB上技术风险和工作量都很大，因此新版本使用Oracle数据库，该版本完成后再单独进行一次数据库迁移：迁到Percona XtraDB Cluster上。

* 1. **源码防止反编译**
* 加密:

需使用C++实现JVMTI（Java 虚拟机工具接口）和 Agent，无免费开源工具，复杂，有技术风险：对三方库可能会有问题；

* 混淆：

可使用ProGround对指定类和方法进行混淆，简单，成熟稳定，有成功案例。

建议使用混淆。

* 1. **分布式日志架构**
* [Facebook Scribe](http://www.baidu.com/link?url=Iu7IDneYyyBpdfkeRtnnojFuXDmApiAIrgPzWDIyRGjevlQJvx2s7T_PEBH2JrpRsQYEVntzl4kv0e1BQ_p8uZaGHjlDm6UfZzneDhxcddO)
* Apache Flume
* ELKStack
* apache的chukwa
* linkedin的kafka

建议选择ELKStack。

1. **遗留问题**
   1. **持续集成**